Volume 31 — Number 1 — 2016 — ISSN 2190-6858

JLCL
Journal for Language Technology
and Computational Linguistics

Corpus Linguistic Software Tools

Herausgegeben von / Edited by
Marc Kupietz, Alexander Geyken

GSCL Gesellschaft für Sprachtechnologie & Computerlinguistik

Contents

Impressum

Herausgeber	Gesellschaft für Sprachtechnologie und Computerlinguistik (GSCL)
Aktuelle Ausgabe	Band 31 – 2016 – Heft 1
Gastherausgeber	Alexander Geyken, Marc Kupietz
Anschrift der Redaktion	Lothar Lemnitzer Berlin-Brandenburgische Akademie der Wissenschaften Jägerstr. 22/23 10117 Berlin lemnitzer@bbaw.de
ISSN	2190-6858
Erscheinungsweise	2 Hefte im Jahr, Publikation nur elektronisch
Online-Präsenz	www.jlcl.org

Alexander Geyken, Marc Kupietz

Editorial

With the growing availability and importance of (large) corpora in all fields of linguistics, the role of software tools is gradually moving from useful, possibly intelligent information-technological "helpers" towards scientific instruments that are as integral parts of the research process as data, methodology and interpretations. Both aspects are present in this special issue of JLCL on *corpus linguistic software tools*: The contributions address topics such as software tools for managing corpora, transforming corpus data, annotating and analyzing corpora as well as innovative ways of exploring and visualizing corpus data and analyses.

Thomas Krause, *Ulf Leser* and *Anke Lüdeling* present graphANNIS, a newly developed graph database for querying deeply annotated linguistic corpora. After discussing the pros and cons of existing solutions, they document implementation details, the data model, and query aspects, including the evaluation of the the proposed graph based implementation vs. the established RDBMS backend. It is reported that graphANNIS performs in the majority of the tested cases much faster than the relational equivalent.

Paul Rayson, *John Mariani*, *Bryce Anderson-Cooper*, *Alistair Baron*, *David Gullick*, *Andrew Moore* and *Stephen Wattam* address in their paper "towards interactive multidimensional visualisations for corpus linguistics" several important issues, including how iterative and exploratory corpus investigations can be supported by dynamic and interactive visualisation techniques and how to approach issues with current corpus retrieval tools by including interactive and multidimensional visualizations.

Christian Pölitz presents in his paper "data mining software for corpus linguistics with an application in diachronic linguistics" a freely available plugin for the RapidMiner software. This plug-in could be of particular relevance for researchers already using RapidMiner, or for computational linguists looking for a tool that is more user-friendly - through drag-and-drop - than e.g. the established R software. The author uses the example case of topic modelling over time to demonstrate the advantages of the new plugin, including the implemented topic models and coherence measures.

Nils Diewald and *Eliza Margaretha* present "Krill", the search component of the KorAP corpus search and analysis platform. This paper makes three important contributions to research and software engineering in the area of corpus indexing and query: It introduces a new open-source corpus indexing software based on Apache Lucene and describes how linguistic corpus search can be implemented on top of a full text search engine such as Lucene. Furthermore, as an implementation of the KoralQuery specification, Krill is an important milestone for the types of search pattern expected from a modern linguistic corpus query engine.

Maria Skeppstedt, *Carita Paradis* and *Andreas Kerren* present PAL, a standalone tool for pre-annotation and active learning. PAL can be trained using gold standard data

and it can incorporate manually annotated/corrected data created by the user within the BRAT annotation tool. PAL tries to optimize the process by using an active learning approach to select sentences to be annotated by the user. The tool focuses on "chunk"-based annotation tasks, e.g. for named entity detection.

Arne Neumann presents “discourse graphs”, a library and command-line application for merging and validating heterogeneous, multi-layered corpora. The resulting software tool consists of a python library and command-line applications for converting multi-level annotated data into different formats, merging independent annotations of the same text into one graph representation and validating heterogeneous annotation layers of the same text. This complex model is built upon the property graph model and can therefore benefit from the ecosystem built around modern graph libraries based on that paradigm.

Thomas Schmidt presents software tools and workflows for the construction and dissemination the FOLK corpus, a corpus of spoken interaction. The article covers the tools used in the individual steps of transcription, anonymization, orthographic normalization, lemmatization and POS tagging of the data, as well as the utilities used for corpus management. Furthermore, it presents the DGD (Datenbank für Gesprochenes Deutsch - Database of Spoken German) as means for distributing the completed research data and making it available for qualitative and quantitative analysis.

Ruprecht von Waldenfels and *Michał Woźniak* present SpoCo, an adaptable query and analysis system for spoken dialect corpora with aligned audio data encoded in ELAN. SpoCo is targeted at users of different levels of expertise and provides them with advanced concordancing functions, as well as the the possibility to edit and correct transcriptions. SpoCo takes a middle position between systems that are developed for the purposes of a specific dialect corpus, on the one hand, and general-use systems designed for a wide range of data and usage cases, on the other. It is used in a network of Slavic dialect projects that cooperate in tool development and data sharing.

Our gratitude goes to the colleagues who contributed to this special issue as external reviewers.

Berlin and Mannheim, May 2017

Alexander Geyken
Marc Kupietz

Thomas Krause, Ulf Leser, Anke Lüdeling

graphANNIS: A Fast Query Engine for Deeply Annotated Linguistic Corpora

Abstract

We present graphANNIS, a fast implementation of the established query language AQL for dealing with deeply annotated linguistic corpora. AQL builds on a graph-based abstraction for modeling and exchanging linguistic data, yet all its current implementations use relational databases as storage layer. In contrast, graphANNIS directly implements the ANNIS graph data model in main memory. We show that the vast majority of the AQL functionality can be mapped to the basic operation of finding paths in a graph and present efficient implementations and index structures for this and all other required operations. We compare the performance of graphANNIS with that of the standard SQL-based implementation of AQL, using a workload of more than 3000 real-life queries on a set of 17 open corpora each with a size up to 3 Million tokens, whose annotations range from simple and linear part-of-speech tagging to deeply nested discourse structures. For the entire workload, graphANNIS is more than 40 times faster, and slower in less than 3% of the queries. graphANNIS as well as the workload and corpora used for evaluation are freely available at GitHub and the Zenodo Open Access archive.

1 Introduction

Many linguistic research questions require the analysis of several different types of information attached to different substructures of a given linguistic corpus. For instance, it is necessary to collate parts of speech and lemmas for the computation of complexity measures for language acquisition studies (Lue, 2011), to consider referential chains and intonation for the analysis of information structure phenomena (Baumann and Riester, 2013), and to combine temporal information and inflectional forms to trace language change (Hilpert, 2008). To make the categorization that lies behind such studies explicit, it is usually attached to a corpus as annotation of the different substructures, like tokens, phrases, or sentences (Leech, 1997; Lüdeling, 2011). A number of models and formats for representing and exchanging such annotated corpora have been proposed in recent years (Carletta et al., 2003; Chiarcos et al., 2008), with a clear tendency towards multi-layer models, i.e., models which allow the concurrent annotation of different types of (possibly nested) substructures. During creation and usage in research, such multi-layer corpora can grow by adding more texts and by adding more layers of annotation; as it is often not foreseeable at the beginning of a project what the

corpus may look like in a more advanced project state, systems for supporting the management and analysis of multi-layer corpora should ideally allow for extensibility both in the corpus model and in the corpus size.[1] Such a system should furthermore offer a powerful query language to perform quantitative analysis of corpora, and an implementation of this query language that is fast enough for interactive usage and that scales well with growing corpus size and complexity.

In this paper, we present graphANNIS, a very fast and flexible main-memory based implementation of the well-established corpus query language AQL (Rosenfeld, 2010; Krause and Zeldes, 2016). AQL was developed in the ANNIS project (Krause and Zeldes, 2016), whose long-term goal is to create a system that allows for unified search across diverse kinds of annotations. ANNIS supports a large range of different types and structures of annotations and offers an intuitive web-based user interface with sophisticated visualizations. The system is used as a corpus management platform for a variety of projects ranging from historic to spoken language. There are at least 11 public installations of ANNIS and an unknown number of non-public instances. Our own, publicly available ANNIS server (`https://korpling.org/annis3`) hosts more than 150 corpora, has 96 registered users and serves roughly 1800 queries every month.

The query language AQL builds on a graph-based data model, which makes modeling and querying deeply nested structures simple. However, the current implementation of AQL, called relANNIS, uses a relational database as storage and query layer. To this end, any AQL query is translated into one or more SQL commands which are executed by the database server. The tabular results are then transformed back into the AQL exchange format and displayed for user interaction at the ANNIS front-end. This design, which delegates the seamless exchange of data between main memory and disk to the database server, was originally chosen because memory was expensive at that time, which meant that very large corpora could not be maintained in main memory. The downside is it makes query execution somewhat slow, due to the necessary query and result transformation steps, the unsuitability of an SQL back end for certain graph-like predicates in AQL, and the general overhead incurred by the multi-tier, multi-user, transactional memory management inside a relational database[2]. Despite considerable effort spent on tuning and tweaking the system (Rosenfeld, 2010), including extensive use of indexing and materialized views leading to a very large disk footprint, we frequently observe queries that run into time-outs even on smaller corpora with less than $100,000$ tokens.

Here, we present graphANNIS, a novel implementation of AQL that directly implements its underlying graph-based data model and that is purely main-memory based. Due the ever falling prices of computer memory, main-memory technologies are nowadays extremely popular both in database research (Zhang et al., 2015) and applications (Färber et al., 2012). At the same time, especially deeply structured corpora are typically generated manually by linguistic experts, which limits their size to a few

[1] Example corpus projects where additional annotations have been added can be found e.g. in Zeldes (2016b) page 1f.

[2] Note that AQL is a pure query language and has no operations to manipulate data.

(dozen) megabyte. For instance, the size of the largest corpus managed by the public ANNIS server, the "Parlamentsreden_Deutscher_Bundestag" corpus (Odebrecht, 2011), is 500 megabyte in size (uncompressed) and has only flat annotations at token level. In contrast, the largest corpus in this collection which has a more complex annotation structure, the "TueBa-DZ 6.0" corpus (Telljohann et al., 2009b), has a size of only 1.2 gigabytes. Holding such corpora in main memory is easily possible with current PCs.

Graph-based databases are not a new topic, and several research (Wood, 2012) and commercial implementations (e.g., Neo4J, DEX) exist. However, systems are typically either optimized for navigational queries (adjacent nodes, neighborhood queries) but lack efficient support for transitive predicates (such as reachability queries), or are specialized data structures to support certain complex graph operations but lack basic navigational features. Therefore, we decided to build graphANNIS from scratch, which mainly encompasses a query compiler, a graph store, a basic query optimizer, and special index structures and graph algorithms to support specific features of AQL[3]. In this paper, we describe all of these components in detail and give a quantitative comparison of the performance of graphANNIS to that of relANNIS. We use a workload of more than 3300 AQL queries against 17 corpora; this workload was created by logging real-life user queries on the public ANNIS server over a period of roughly 5 months. Note that, to the best of our knowledge, this is the by far greatest real-life workload of linguistic queries ever published. We show that graphANNIS is more than 40 times faster than relANNIS over the entire workload; specifically, it outperforms relANNIS in 95% of all queries, and especially in deeply nested queries.

The paper is structured as follows: Chapter 2 describes the query language AQL and its existing implementation relANNIS. A description of the concepts underlying graphANNIS is given in Chapter 3. Chapter 4 describes the concrete implementation of graphANNIS. An evaluation how this new system compares to the legacy implementation relANNIS is presented in Chapter 5. Section 6 reviews other linguistic query systems, and Chapter 7 concludes our paper.

2 The ANNIS System for Corpus Querying

This section provides necessary background information that is needed to understand the design decisions that were made for the graphANNIS implementation. We describe the query language AQL and its current implementation relANNIS. For more details on both topics, the reader is referred to Rosenfeld (2010) and Krause and Zeldes (2016).

2.1 AQL, the ANNIS Query Language

AQL is a linguistically motivated query language not limited to a certain tag-set. It has been influenced by other query languages, such as TIGERSearch (Lezius, 2002),

[3] Note that graphANNIS is not yet a complete implementation of AQL, as it lacks certain convenience features; see Section 2.1 for details.

and is described more formally in Rosenfeld (2010) and Rosenfeld (2012). An up-to date description of AQL is also given in the ANNIS User Guide (Zeldes, 2016a).

The basic model of ANNIS is a graph consisting of nodes and edges. Nodes represent linguistic substructures of a text, like tokens, phrases, or sentences, whereas edges represent the relationship between nodes, such as followed-by, part-of, dominates, or within-distance. The type of a node is given by an annotation (or label).

The building blocks of an AQL query are terms, which select nodes either via a label or via their text. For instance, the query

```
pos="NN"
```

selects all nodes having the annotation (or label) "pos" (a common category name for part of speech annotations) with the value "NN" (often used for nouns). The query

```
"the"
```

will select tokens that have the value "the" as spanned text. In both types of selections, it is possible to use regular expressions instead of string literals by using `/` instead of quotation marks.

Labels can also have a namespace to distinguish annotations with the same name from different sources. This namespace can be selected using the `namespace:name` expression. For instance, the query

```
treetagger:pos="NN"
```

would select all noun tokens, where the POS information stems from the namespace "treetagger". This way, a single structure can have multiple values of the same type, such as multiple POS tags generated by different taggers or following different linguistic theories.

Multiple terms can be defined in one query using the special symbol `&`. Such conjunctions are typically followed by a predicate which specifies the nature of the selected nodes. For instance, the following query

```
cat="S" & "the" & #1 >* #2
```

first selects all nodes of category "S" (sentences) and then all nodes whose text is "the". Next, the Cartesian product of these two sets of nodes is built and filtered for those pairs where the sentence dominates (i.e., contains) the respective token. Note that the token need not be directly contained in the sentence; it might as well be that a sentence consists of phrases consisting of subphrases, etc. consisting of tokens. This implies that the path between the node with annotation "S" and the node "the" can have an arbitrary length. AQL also supports a number of more convenient syntax rules; for instance, the above query may also be abbreviated as

```
cat="S" >* "the"
```

Several other operators exist for different constraints, like paths lengths, types, and edge annotations (like direct dominance in syntax trees or co-reference chains). Another class of operators compare the texts covered by two nodes, such as phrase structures that overlap, or tokens being contained in a sentence. We describe the most important relationships in the following paragraphs.

Pointing relation operator: "`->type`" Each pointing relation has a named type and the names must be explicitly given as argument to the operator. All edges of one type define a graph component which is required to be acyclic. The path can be constrained to have the length 1 when using the "direct pointing relation" form of the operator (`->type`) or any length when using the indirect form which is marked by a star (`->type*`). An explicit range $n \dots m$ of allowed path lengths can be given by using the form `->type,n,m`. Only edges belonging to the same type are allowed in the path definition and there is no possibility to mix different types of edges by using only one operator. Edges are allowed to have labels as annotation and the direct pointing relation operator can take an additional argument that constrains the labels. An exemplary use of this predicate is the query

```
pos="VVFIN" ->dep[func="sbj"] tok
```

which selects all finite verbs (part of speech is "VVFIN") which are connected to a token via a pointing relation of type "dep" (which marks dependency relations). This connected token is the subject of the finite verb (this is marked with the `func="sbj"` edge annotation).

Dominance operator: "`>`" This operator selects a path composed of so-called dominance edges, which imply inclusion dependencies at the textual level. The syntax for the dominance operator is the same as for the pointing relation, thus it is possible to specify a path of unspecified length (`>*`), ranged length paths (`>n,m`) or edge labels (`>[anno="value"]`). An exemplary use of this predicate is the query

```
cat & cat="S" & #1 >[func="SB"] #2
```

which searches a sentence/clause category (node `#2` having a "cat" annotation with value "S") which is also a subject clause (expressed by the `func="SB"` edge annotation) and is dominated by another syntactic category (node `#1` having the "cat" annotation without a specific value).

Precedence operator: "`.`" Two tokens are in a precedence relationship if they are next to each other in the text. As with the other operators the precedence operator can have the path length as an argument. Precedence is not only defined for tokens but for any node that has a text coverage relation to any set of tokens. In this case the precedence is defined between the right-most covered token of the left-hand side (LHS) operand and the left-most covered token of the right-hand side (RHS) operand. For instance, in the sentence "I am hungry" the "I" token precedes the "am' token and the

"am" token precedes the "hungry" token. This construction in AQL would be expressed as

```
"I" . "am" . "hungry"
```

Text coverage operators: "_=_" "_o_" and "_i_" Any node covers a certain token span. Text coverage operators describe the relations between two text spans covered by a node. The identical coverage operator `_=_` ensures that both sets are completely equal while the overlap operator `_o_` only requires that there is a non-empty intersection between two sets. These operands may be swapped in a query since they describe a commutative relation between both nodes. In contrast, the inclusion operator `_i_` is not commutative since it defines that all covered tokens of the RHS must also be covered by the LHS.

graphANNIS implements all of the features of AQL described in this section. Additional features of AQL are available in the existing relANNIS implementation, including filtering documents by metadata, negation of values, filtering for value identity and more operators. For a complete list of features see the ANNIS User Guide (Zeldes, 2016a).

2.2 A Relational AQL Implementation: relANNIS

The existing relANNIS query system offers a complete implementation of AQL on top of the relational PostgreSQL (https://www.postgresql.org/) database. Queries entered via the ANNIS front-end are translated to SQL, executed by the database engine, and results are transformed from the relational data model to a Salt graph (Zipser et al., 2010) before being displayed through various visualizers in the front-end. Additionally, the total number of results of a query is computed by executing a separate SQL-query.

This workflow leaves most of the responsibility for optimizing the query execution to the relational database implementation. It also means that the graph model data in ANNIS needs to be mapped to the relational model. For most cases, this is straightforward; the database scheme of relANNIS uses mostly the following tables:

- `node` for informations about the node itself,
- `node_annotation` for the different labels of a node,
- `component` lists connected components,
- `rank` contains all pre-/post-order entries and
- `edge_annotation` for the labels of each edge.

Note that this is a normalized schema. It has some drawbacks; if an AQL query uses node annotations and edge labels, even a simple AQL operator between two annotations will cause an SQL-join with 9 tables (`edge_annotation` is joined once, all other tables are joined twice). However, also queries joining five annotations and more are common,

which would result in a query of more than a dozen joins. To avoid such difficult-to-optimize queries, relANNIS uses several materialized views that combine specific information into a single table. Using these tables, the query compiler can usually reduce the number of SQL-joins to match the number of AQL operators, but it also drastically increases the required space and makes memory management much harder for the database server.

A number of AQL queries are not as easily mapped to SQL. This affects, in particular, relationships between nodes over a path of unknown length, which cannot be expressed directly in SQL[4]. To answer such queries efficiently, relANNIS uses pre-/post-order indexing (Grust et al., 2004), with which reachability information can be obtained with a single query. However, this only holds true for tree-shaped graphs. To also handle annotations shaped as Directed Acyclic Graphs (DAGs), which are common in ANNIS, we use a technique described in Rosenfeld (2010), which increases memory consumption but still allows comparably fast queries.

An additional problem occurring in relANNIS are the statistics required by the cost-based optimizer in modern RDBMS. Note that each AQL operator results in at least one join, often over more than one attribute. The concrete way of executing such a join (and sizing of buffers etc.) is determined by the query engine based on statistics on the value distributions in the join columns. However, RDBMS typically assume these distributions to be independent (which allows for very succinct statistics and low cost maintenance), an assumption which is breached regularly in relANNIS; for instance, the post-order of a node will always be larger than the pre-order. We observed that, due to this discrepancy, PostgreSQL very often underestimates the size of intermediate results, which leads to suboptimal choices during query optimization.

3 Basic Design of graphANNIS

An important goal for the design of the new graphANNIS system was to reuse the best parts of the existing relANNIS system, but to address the performance issues which arose from using a disk-based relational database. The graph-based data-model and the query language AQL proved to be very useful for searching in linguistic corpora[5] and thus the new system should support the largest possible subset of AQL. Also the experience with the relational database showed that there should not be a mapping of data-models if not necessary. This helps to avoid losing possibilities for optimization, like good statistics for better execution plans. Therefore we did not want to rely on XML databases or RDF triple stores, as these have different data models.

Experiences with the query planning from PostgreSQL, where we needed to "trick" the optimizer into using a more efficient plan, also confirmed that we needed explicit control of the query execution. Such control is not possible when using a query language

[4] Note that the SQL-3 standard defines recursive queries which can express such operations. However, the current implementations are notoriously ineffective.

[5] For a discussion about the pros and cons of different linguistic query languages see Frick et al. (2012).

from a graph database like Cypher from Neo4j (Robinson et al., 2013). Also, systems like Neo4j usually have one specific way of storing and querying the graph, e.g. only using graph traversal. Since linguistically annotated graphs tend to have very different kinds of structures, and we want to be able to have solutions that are optimized for these specific structures, we did not want to restrict ourselves to a single storage and querying strategy.

ANNIS is solely a search system and using an SQL database to do the actual search helped to take advantage of all the advanced search optimization a modern relational database has to offer. But databases do a lot more than only providing search capabilities: They typically organize the persistent storage of the data on the disk and provide transactions for persistence and isolation of database updates. A corpus search system is special since after a corpus is imported once by an administrator, it solely provides read-only access to normal users. Thus complicated and costly transactions are not needed. The bulk import also allows to add much more computationally intensive optimizations since importing a corpus is not done very often. E.g., very specific index structures can be used and even data types for values can be optimized since the corpus is static and will not change. These index structures are the only way the read-only data will be accessed. Since there are no data updates, there is no need to store an additional normalized representation (e.g. the database table holding the nodes) if all information is available in the indexes themselves. We also wanted to avoid overhead created by managing the transparent persistence of the data to the disk.

General availability of notebooks with at least eight gigabytes of RAM and server systems with more than 100 gigabytes makes a main memory only approach feasible even for larger corpora. Not having to worry about the representation of the data both on the physical disk (which may not even be a relatively fast SSD) and the main memory allows to optimize the data structures for main memory and CPU cache access only. We also expect much faster queries if we can guarantee that all needed indexes are already loaded in main memory and do not need to be fetched from a disk. Our goal is to use data structures that, in contrast to the materialized view used in relANNIS (see section 2.2), scale well with the size of the corpus.

In order to achieve this goal we designed an architecture that combines the generic graph-based data model with specific implementations for different types of graphs. These so-called Graph Storages (GSs) only implement the storage of edge components while a common node annotation store stores all node-relevant information. Different annotation layers will result in different types of components and the architecture described here allows for the selection of an optimized implementation to store and query edges for a component. When executing a query it is parsed into a generic JSON representation, an optimized plan is generated with help of statistics and this optimized plan is subsequently executed. Finding nodes that are reachable from a starting point is an essential substep in these plans. The query optimizer does not need to choose which indexes to use, the best GS implementation is already chosen automatically when a corpus is first imported. Strategies for finding reachable nodes are graph-traversal, pre-/post-order or other graph indexes and our architecture allows to easily add new

Type	Description
COVERAGE	Edges between a span node and tokens. Implies text coverage.
INVERSE_COVERAGE	Edges between a token and a span node.
DOMINANCE	Edges between a structural node and any other structural node, span or token. Implies text coverage.
POINTING	Edge between any node.
ORDERING	Edge between two tokens implying that the source node comes before the target node in the text-flow.
LEFT_TOKEN	Explicit edge between any non-token node and the left-most token it covers.
RIGHT_TOKEN	Explicit edge between any non-token node and the right-most token it covers.

Table 1: Component types used in graphANNIS.

types of GSs.

4 Implementation

While the original ANNIS ecosystem is mostly developed in the Java programming language, graphANNIS is written in C++ 11. This allows for a better control of the allocation, deallocation and structure of the data in main memory. The system is optimized for read-only access of the data, thus the corpora are imported once and normally not changed after they have been loaded into the query system. Since the system is read-only there is no need for a complex transaction management. graphANNIS is implemented as a library which can be used in other programs and it is planned to make graphANNIS a drop-in replacement for the existing relational database back-end used in the current ANNIS versions.[6] All source code is published as open source at `https://github.com/thomaskrause/graphANNIS`.

4.1 Data model

graphANNIS represents the linguistic annotations as a directed labeled graph. The edges are grouped into named components and each component has an explicit type. A list of all component types used in graphANNIS is listed in Table 1. For some linguistic annotations like co-reference chains or constituent trees the mapping to the graphANNIS data model is straightforward: e.g., for a constituent tree the clauses are modeled as nodes, the tokens are terminal nodes and the relations are expressed using edges as can be seen in Figure 1. The data model of graphANNIS is very similar to the Salt (Zipser et al., 2010) data model, differing slightly only to match the semantics of AQL as closely as possible. E.g., both Salt and graphANNIS use explicit edges between

[6] In order to use the C++ library from the existing Java-based back-end the JavaCPP library (`https://github.com/bytedeco/javacpp`) is used.

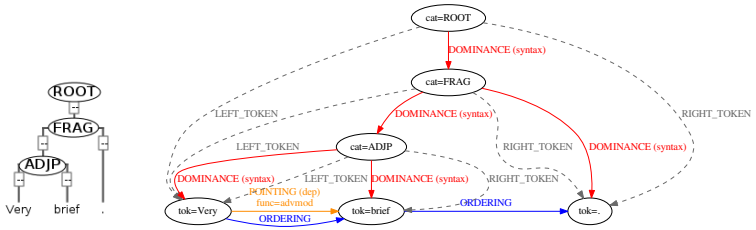


Figure 1: Example how a constituent tree is modeled in graphANNIS. Each token has the special annotation "tok" which contains the covered text as value and tokens are connected with explicit edges which define their ordering (in blue). So-called structural nodes of the constituent tree have their annotation as label and these are connected with edges of type DOMINANCE (in red) to either other structural nodes or tokens. In addition each non-token node has an explicit LEFT_TOKEN and RIGHT_TOKEN (the dashed lines) to the left- or right-most token they cover. There is also an additional dependency edge in this example which is modeled as a so-called pointing relation (in orange). This example is taken from the GUM corpus (Zeldes, 2016b) and is also available online in ANNIS (`https://korpling.org/annis3/?id=cc06121a-09b6-455f-aef4-f9eaee7a34f5`)

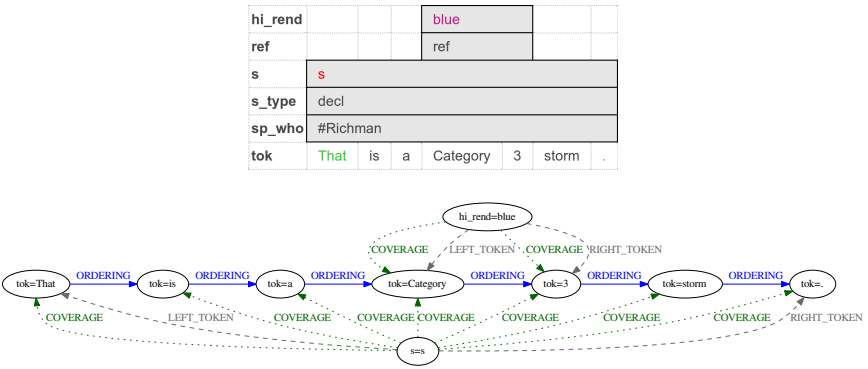


Figure 2: Example how spans are modeled in graphANNIS. Each span has an explicit edge to each token it covers (green COVERAGE edges). Only the "s" and "hi_rend" annotations have been included and the INVERSE_COVERAGE edges have been excluded to enhance readability. This example is taken from the GUM corpus and is also available online in ANNIS (`https://korpling.org/annis3/?id=d23627b4-3b30-4876-9ce5-7acb3406d3a4`)

the so-called span nodes (nodes that cover a range of tokens) from each span node to every token and this is compatible with the way AQL handles text coverage (see Figure 2 for an example). In AQL the leaf nodes in the annotation graph are always tokens whereas in Salt there are special nodes containing the complete text of a document and are connected with the tokens. The tokens in graphANNIS have a special label with the name "annis4_internal:tok" instead which contains the covered text.

4.2 Node labels

The graphANNIS implementation separates the storage of nodes and edges. There is one central map for storing node labels and several storages for different types of edges (see subsection 4.3 for details). Nodes are not explicitly stored but have a numeric ID to which node labels and edges refer. Every node has at least the special "annis4_internal:name" label, so it is always possible to iterate over all nodes even if they do not have a user-supplied annotation.

The query system should support corpora of all kinds of languages and thus needs to support Unicode in order to be able to represent the different scripts. In addition, strings representing linguistic annotation names or values might have a wide variation in their length. In certain tag sets the annotation names are very short like "pos" for "part of speech" and the possible annotation values are also abbreviations with a limited length. A good example for this kind of annotations is the Stuttgart-Tübingen-Tagset (STTS) (Schiller et al., 1999). As another extreme annotations can be free text comments with several hundreds of characters (e.g., the "Open-ended comments" in the CityU corpus from Lee et al. (2015)). graphANNIS uses a dictionary encoding where all strings get a unique ID and are only referenced by this ID. A global ID manager is responsible for adding new strings and getting the ID for a certain string or the string value of an ID. This has the advantage that an annotation has a constant size independently of the length of the strings it contains. For annotation categories with a limited number of possible values this approach might also reduce the needed memory, since repeated occurrences of a value or name only store the ID and not the real string. This only holds for annotation values or names whose memory representation is larger than the size of an ID (which is 4 bytes). The assumption does not hold for certain tag sets which contain very short identifiers representable with the ASCII encoding and will actually cause a slightly worse memory usage in this case. E.g., in the TIGER corpus (Brants et al., 2004) the phrase category annotations have values from 1 to 4, but typically only 2 characters.

Additionally to the actual values, the node label storage contains statistics about the values of different label categories. A label category is a combination of a namespace and name. These statistics include the total number of labels and equi-width histograms of the values for each label category. Thus it is possible to estimate the instance count of label categories for a specific value or a range of possible values both efficiently and accurately based on the actual distribution of label values in the corpus.

4.3 Specialized Graph Storages

The main task of the query system is to find nodes with certain labels and to check constraints on the path. Thus the node described by a RHS of the operator must be reachable by the node of the LHS. Finding nodes that are (indirectly) reachable by another node is therefore a very important subtask that must be implemented efficiently. There exist various graph indexes to query reachability (Grust et al., 2004; Seufert et al., 2013; Yildirim et al., 2010) but they typically do not work on any kind of graph (e.g., see section 2.2 for a description of the problems that pre-/post-order indexes have when used for a DAG instead of a tree). Since the annotation edges of different kinds of linguistic annotations are grouped in components, graphANNIS can exploit the specific structure of each component and use an optimized implementation for storing and querying edges.

A so-called Graph Storage (GS) is responsible to store the information about the edges of a component. It does not only allow to retrieve the directly connected edges for a node and their annotations, but provides more complicated functions for accessing the graph. Each GS has the functions (pseudo-code)

- `boolean isConnected(n1, n2, minDistance, maxDistance)`,
- `iterator findConnected(n1, minDistance, maxDistance)` and
- `integer distance(n1, n2)`.

`isConnected(...)` checks if node $n2$ is reachable from node $n1$ within a given (optional) distance. `findConnected(...)` will find all reachable nodes from a start node and `distance(...)` calculates the minimal path length between two given nodes. It is expected for each GS to have an optimized implementation for these three functions. As a result a GS cannot store any type of component but is limited to graphs which fulfill special constraints. It is the responsibility of the general database to only use an appropriate implementation for a specific component. There are three different kinds of graph storages implemented already.

4.3.1 Adjacency list

This implementation of a GS stores the edges as set of node ID pairs. The set implementation is based on B-trees and uses the Google C++ B-tree template classes.[7] It is guaranteed that the set is sorted and since the pair always starts with the source node it is possible to get all outgoing edges for a certain node in logarithmic time.

The three basic GS functions are implemented using a Depth-First-Search (DFS) graph traversal. If the function `isConnected(...)` is called for the exact distance of 1 a shortcut is used which only checks if the source/target node pair is contained in the set. Since the components in this implementations are allowed to contain cycles a cycle-safe implementation of DFS can detect cycles and will result in an error state if such a cycle is found. Using a cycle check makes the execution less efficient. The

[7] The library is available from `https://code.google.com/archive/p/cpp-btree/`

statistics of a component already contain the information whether a component is cyclic (see section 4.3.4) and this could be used to build a specialized variant of the adjacency list that can be used for read-only noncyclic components.

Since adjacency lists and cycle-safe DFS work for every graph regardless of its properties this implementation is used as a fall-back if no optimized implementation is available.

4.3.2 Pre- and post-order

As described in Section 2.2 the legacy ANNIS implementation is based on a relational database and uses a pre-/post-order based index to find reachable nodes efficiently. This GS implementation uses the same index to accelerate reachability queries. It does not support components that contain cycles. In order to store the hierarchical structure of the graph it maintains a map from a node ID to a list of triples containing the level and the pre- and post-order. There is also an inverse map for each distinct triple to the corresponding node ID. Both maps are based on B-trees.

4.3.3 Linear graphs

Linear graphs are graphs where each node can only have one ingoing and one outgoing edge at maximum. This is quite common in linguistic annotations, e.g. when the order of words is stored or in any other form of chains of references. When determining whether a node is a descendant of another node an ordering can be used. A component might have several root nodes and the order is always the distance of a node to its root node. The order of each node is stored in the map which maps the node ID to a pair containing the ID of the root node and the (relative) order of the node. Additionally there is a vector for each root node. Each entry in this vector consists of the ID of a connected node and the index inside the vector is the order of this node.

4.3.4 Automatic selection of best Graph Storage

Each component has different characteristics depending on which annotation scheme was used to create it. graphANNIS tries to exploit this and provide optimized implementations for certain typical structures that can be found in linguistic annotations. When an existing corpus is imported for the first time each component is stored in a fall-back implementation using adjacency lists. This implementation is able to calculate statistics on the component which helps to automatically decide which implementation can be used. Table 2 contains the fields of the collected statistics. After the import is finished each component is converted to an optimized implementation if possible.

The GS for linear graphs (see section 4.3.3) is chosen whenever the component is a forest of rooted trees (`rootedTree` is true) and the maximal fan-out (`maxFanOut`) is 1. Since we know the length of the longest path inside the component (`maxDepth`) and the order of a node inside a linear graph can be only as large as the longest path, we choose a memory-saving data type for representing the order. Depending on `maxDepth` a single

Field	Description
cyclic	True if the component contains cycles.
rootedTree	True if the component is a forest of rooted trees (tree having only one root node). This is checked by ensuring that the graph is not cyclic and that each node has maximally one incoming edge.
nodes	Number of nodes in this component.
avgFanOut	The average number of outgoing edges for each node.
maxFanOut	The maximal number of outgoing edges for each node.
maxDepth	If noncyclic, the length of the longest path.
dfsVisitRatio	If noncyclic, the ratio between the number of nodes and the number of visits when traversing each subcomponent in a Depth-First-Search.

Table 2: Information collected as statistics for each component.

order value is using either 8, 16 or 32 bytes of main memory. Since the linear graph GS is read-only there is no possibility that the maximal path size changes and does not fit any longer in the chosen data type.

When the component is a forest of rooted trees but the maximal fan-out is greater than 1, the pre- and post-order based GS is chosen. As with the linear graphs, a memory optimized implementation is chosen depending on the actual number of nodes and the maximal depth. The data type for the pre/post-order is limited by the number of nodes since every node can have only one pre/post-order in a rooted tree. In the case that the component is not a forest of rooted trees but still acyclic, it is checked whether the ratio between the number of nodes and the number of visits in a Depth-First-Search is greater than a constant (currently 1.03). This condition shall express the case that a component is "almost" a tree, thus the number of order entries for the nodes is expected to be low. If a component fulfills this condition the pre- and post-order GS is still used, but only the data type for the level of a node is optimized with the help of the `maxDepth` statistic.

The default adjacency list implementation is not only used as a fall-back but also in the case if the longest path of the component is 1. This is justified by the overhead other implementations have for storing and querying the graph. If the longest path is 1, the adjacency list can implement the three basic functions with a single map look-up and does not have the overhead of the other implementations.

4.4 Query Optimization

graphANNIS does not include its own parser for AQL queries. Instead, the original Java-based parser from relANNIS is used to create a intermediate representation of the query as JSON objects, which is then passed as argument to the graphANNIS library to construct and optimize an execution plan. An execution plan of an AQL query is a tree of operators like joins, path constraints, filters, or matching label values. Nodes with certain values in their labels can be searched using exact matches or regular expressions, and any search may make use of namespace information.

For each plan, graphANNIS tries to derive an abstract cost measure which correlates to the sum of processed tuples over all execution steps. Estimates are calculated by using statistics on the distribution of label values and edge types; the edge component statistics are the ones described in Table 2 and are used to estimate the outcomes of join and filter operations. For estimating the number of labels which fulfill a certain criterion histogram statistics are used (see section 4.2).

These cost estimates are used by a simple query optimizer to reorganize the original plan following the simple heuristic that plans creating smaller intermediate results are generally faster. In a first optimization step the operands of each commutative operator are switched if the estimated number of matches for the LHS is larger than the RHS, since a smaller LHS is beneficial for most joins. The second step is to optimize the order of the joins themselves. For queries with less than eight predicates we exhaustively enumerate all permutations to find the presumably best plan; for queries with more than 7 operators we use a simple genetic optimization algorithm to keep the search space at a reasonable size.

5 Evaluation

In this section, we compare the performance achieved by graphANNIS with that of relANNIS using a large real-life workload and a diverse set of different corpora. We also show that the main memory requirements for graphANNIS are acceptable for all multi-layer corpora assuming availability of a well-equipped desktop computer or of a smaller server.

5.1 Dataset used for benchmark

To test AQL implementations with a large, diverse and realistic set of queries, we anonymously collected AQL queries from our public ANNIS server. Note that this server provides both selected free corpora and access-restricted corpora (via a login system); for our tests, we only used queries against corpora which, in principle, are freely available[8]. The selection of corpora available on this server is based on actual research questions and we assume the queries executed on the system are relevant for the researchers using the server.

For each query only the query itself, the selected corpora, a time-stamp and the execution time were logged (but not the user). We collected data beginning in November 2015 and ended the collection in March 2016. In total 8584 unique queries were collected[9]; some data unfortunately was lost due to configuration problems. We filtered all queries which (1) targeted more than one corpus (AQL allows multi-corpus queries, (2) targeted a corpus which was used only rarely, or (3) used an AQL feature not yet available

[8] The licensing situation for corpora is not always clear. Some corpus creators disallow non-academic use and allow download of the original corpus files only after registration. Accordingly, not all corpora in our benchmark set are directly downloadable, but all are free for non-commercial use.

[9] By "unique" we mean syntactic equivalence.

	queries
BeMaTaC_L1_2013-02.1 (Sauer, 2013)	194
BeMaTaC_L2_2013-02.1 (Sauer, 2013)	87
DDD-Tatian (Donhauser et al., 2015)	201
falkoEssayL1v2.3 (Reznicek et al., 2012)	124
falkoEssayL2v2.4 (Reznicek et al., 2012)	360
FalkoWHIGL2v2.1 (Hirschmann et al., 2008)	28
Fuerstinnenkorrespondenz1.1 (Lühr et al., 2015)	131
HIPKON (Coniglio et al., 2014)	28
KAJUK (Ágel and Hennig, 2014)	31
kobaltL1v1.4 (Zinsmeister et al., 2012)	173
kobaltL2v1.4 (Zinsmeister et al., 2012)	360
Maerchenkorpus (Walter, 2015)	111
Parlamentsreden_Deutscher_Bundestag (Odebrecht, 2011)	734
pcc176 (Stede and Neumann, 2014)	465
RIDGES_Herbology_Version4.1 (Odebrecht et al., 2016)	244
tiger2 (Brants et al., 2004)	14
TueBa-DZ.6.0 (Telljohann et al., 2009a)	102
sum	3387

Table 3: Corpora used in the benchmark and the number of queries for each corpus.

in graphANNIS. Table 3 lists the names of the remaining corpora and the number of unique queries for each corpus. In total 3387 queries from 17 corpora were included in the benchmark.[10] Note that the workload contains each query only once; multiple executions are not measured, although they exist in the log files.

5.2 Benchmark setup

All runtime measurements were executed on a server with 16GB of DDR3 RAM (1333 MHz) and an Intel Xeon X3460 CPU clocked at 2.80GHz on an Ubuntu Linux system. Table 4 contains the versions of the used software.[11] PostgreSQL was configured to use a shared buffer with 8GB which is enough to hold the relevant tables and indexes of each corpus in main memory.

relANNIS includes an internal benchmark functionality where all queries for a single corpus are executed first in sequential order to make sure the data is in the database cache, and then executed again 5 times in random order. We used this feature and report the median of the last five executions as runtime of a query. The median was chosen to flatten out any outliers caused by tuples not yet included in the cache. Note

[10] From the original 8584 queries 25.7% were filtered out because the corpus was not included, 31,1% of the remaining queries did target more than one corpus and 22,8% of them were filtered out because they did use a not yet available AQL feature. The complete dataset including the queries and the benchmark results can be downloaded from `http://dx.doi.org/10.5281/zenodo.61807`. All open access corpora used in the benchmark can be downloaded from `http://dx.doi.org/10.5281/zenodo.154343`.

[11] The version of graphANNIS used for the benchmarks can be downloaded from `http://dx.doi.org/10.5281/zenodo.61811`.

Software component	Version
relANNIS	3.4.1
PostgreSQL	9.4.6
graphANNIS	benchmark-journal-2016-07-27

Table 4: Versions of the software components used in the benchmark.

	sum (in ms)
baseline	6637917
graphANNIS	161160

Table 5: Sum of workload execution times. This is the sum over the execution times of all the queries in the benchmark.

that this is a fairly RDBMS-friendly setup, as running all queries first allows filling the main memory caches with all relevant tuples. Queries were aborted after a 60 seconds time-out and aborted queries were counted as 60 seconds execution time.

graphANNIS also has an integrated benchmark mode using the Celero Benchmark library (`https://github.com/DigitalInBlue/Celero`). Each query in the dataset was converted to the JSON intermediate representation together with the execution time baseline from relANNIS. Then all queries were executed 5 times with the mean execution time as the result. Additionally the memory consumption for each corpus was measured. In contrast to relANNIS there is no warm-up phase necessary for graphANNIS (the complete data is already loaded in memory before the benchmark is started) and thus the mean execution time was chosen instead of the median.

5.3 Benchmark results

In Table 5 the sum of the execution times for the relANNIS baseline and graphANNIS are shown. For executing the entire benchmark, relANNIS requires 41.188 times more time than graphANNIS. The individual speed-ups are shown in Figure 3: 94.12% of the queries are faster in graphANNIS than when using relANNIS. Table 6 shows the quantiles of the speed-ups. For instance, 75% of the queries are at least 29.57 times faster in graphANNIS than in relANNIS.

While these figures clearly show the superiority of graphANNIS in terms of execution speed, one also has to consider main memory requirements, as graphANNIS must hold an entire corpus in main memory before being able to execute AQL queries. Table 7 lists the number of nodes and the memory usage of graphANNIS for each corpus in the benchmark set. Note that the used memory size does not only depend on the number of tokens but on the overall number of nodes and edges, which grow superlinearly in multi-layer corpora. Thus a corpus with a relatively small number of tokens like "falkoEssayL2v2.4" (144.619 tokens) can have a similar size to a corpus like "tiger2" (888.578 tokens) due to the fact that the annotation in "tiger2" is only based on tokens and a single syntax layer, while the Falko corpus contains a lot of spanning annotations.

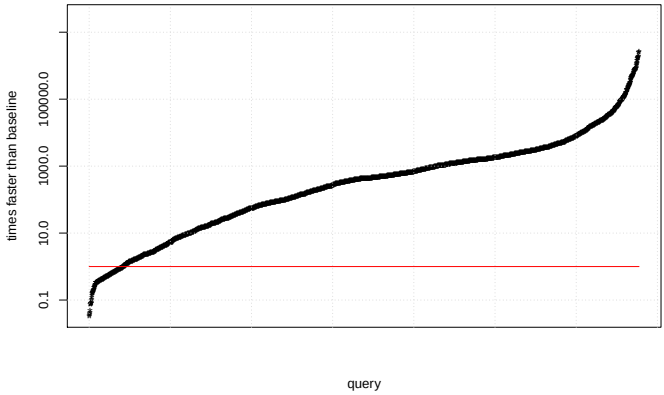


Figure 3: Distribution of the query execution time when using graphANNIS in comparison to the baseline relational database implementation. Each data point corresponds to a single query and how much time it took in relation to the baseline. Thus every data point below the red line (at 1.0) is slower than the baseline and the ones above are faster.

percent of queries	times faster than baseline
0%	2911623.49
25%	2097.62
50%	460.23
75%	29.57
100%	0.03

Table 6: Quantiles of the speedup from using graphANNIS instead of the baseline relational database implementation (larger is better).

Overall, even the largest corpus in this set, the TuebaD/Z corpus which contains 975, 836 tokens in version 6 leading to 15, 773, 656 nodes[12], requires only 1.6GB of memory.

There are still types of queries where graphANNIS is slower compared to the relational database implementation. E.g. the query which performs 28.59 times worse compared to the baseline (and thus is the worst query in the benchmark set) is `node & node & #1 ->dep[func="ART"] #2` : two very non-selective attribute definitions and an operator with a very selective edge annotation in between. Currently graphANNIS has to check all nodes if they have an outgoing edge that matches this edge annotation. This kind of query can be made much faster by adding an index which maps each edge annotation to its corresponding edge (with its source and target

[12] This version of TuebaD/Z contains several annotation layers like part of speech, constituent trees, and co-reference chains.

corpus	used memory (MB)	# node labels	# token
BeMaTaC_L1_2013-02.1	17.86	235563	11187
BeMaTaC_L2_2013-02.1	23.12	257490	12517
DDD-Tatian	43.96	849796	54677
falkoEssayL1v2.3	194.18	3639621	70615
falkoEssayL2v2.4	417.71	8103560	144619
FalkoWHIGL2v2.1	258.52	5593364	130949
Fuerstinnenkorrespondenz1.1	286.87	5043000	262465
HIPKON	40.15	727485	109045
KAJUK	65.66	907774	119420
kobaltL1v1.4	67.31	1186691	12984
kobaltL2v1.4	165.59	3033369	33368
Maerchenkorpus	55.90	1479400	295880
Parlamentsreden_Deutscher_Bundestag	588.53	15670960	3134192
pcc176	25.51	321544	33298
RIDGES_Herbology_Version4.1	223.94	3867351	154267
tiger2	427.01	6451776	888578
TueBa-DZ.6.0	1615.36	15773656	975836

Table 7: Main memory usage of the corpora in megabytes. The number of node labels and token is given as reference.

node ID) and using this index as an iterator for the LHS of the join. From the 25 slowest queries (compared to baseline) 19 queries match this pattern. Another typical problem are queries containing regular expressions that PostgreSQL can replace with index-optimized `LIKE` queries. For simple cases this is already supported in graphANNIS as well, but more complex regular expressions like `(N.|ART)` can result in a full search for all nodes with the annotation name even if a prefix based index lookup could be used. Figure 4 indicates that there is also some correlation between the number of attributes used in a query and the execution time. The number of attributes can be seen as an indicator for the complexity of a query since more attributes result in more joins. When the query statistics work well and the join order is optimized to produce as small intermediate results as possible the negative effect of the joins can be compensated. Additionally, queries with a larger number of attributes are less frequent in the workload. graphANNIS is able to handle the different query complexities and is faster than the baseline implementation even if queries having the same number of attributes are grouped into separate workloads (see Figure 5). Still there is room for improvement in join performance, as queries containing no join perform much better than the ones containing at least one join.

6 Related Work

There already exist several linguistic query systems which can be categorized via their data model and the expressiveness of their query language. The IMS Open Corpus Workbench (CWB) (Evert and Hardie, 2011) for example allows searching on annotations on tokens (positional annotations) and ranges of tokens (structural

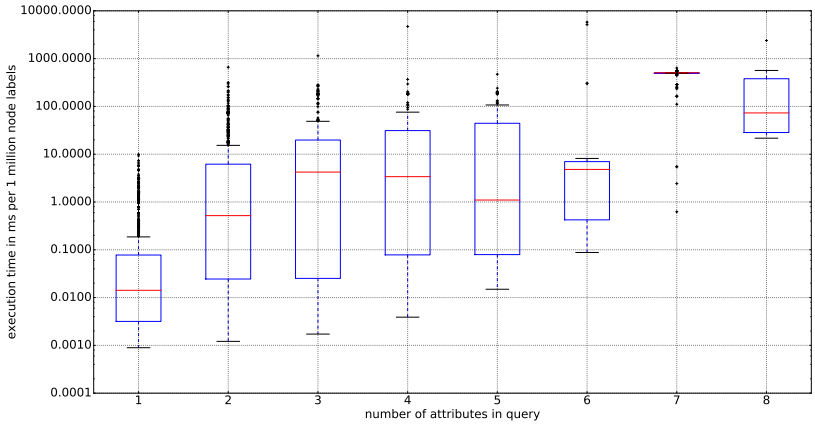


Figure 4: Box plot of the distribution of normalized execution times for each query grouped by the number of attributes a query has. To make the results comparable between different corpora, they have been normalized to the corpus size. The execution times are measured in milliseconds per 1 million node labels of the corresponding corpus.

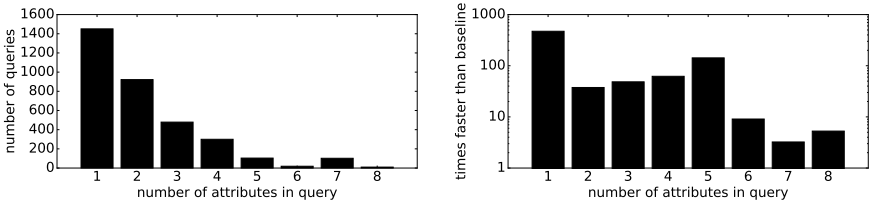


Figure 5: Distribution of number of attributes per query in the workload (on the left side) and speedup of graphANNIS compared to the baseline when the workload is grouped by the number of attributes per query (right side).

annotations). This is a rather "flat" annotation scheme which cannot properly encode general trees.[13] CWBs strength is the support for very large corpora: the authors of CWB claim they can support up to 2.1 billion words (see Evert and Hardie (2011) page 12f.). In contrast TIGERSearch (Lezius, 2002) was designed to handle a collection of sentences which are annotated with syntax trees according to the TIGER annotation scheme (Brants et al., 2004). TIGERSearch uses an index on the label values to restrict the search to single sentences, but it does not use an index on the tree structure itself. Since annotations are always limited to a single sentence TIGERSearch can still perform a very fast traversal search on each sentence and take advantage of the reduced search space. The approach to only use a single sentence as the search space is shared by other so called TreeBank systems like Tregex (Levy and Andrew, 2006) or tgrep2 (Rohde, 2005). The TüNDRA system (Losemann and Martens, 2012) uses the same query language as TIGERSearch and supports the same data model and input files. Queries are translated from TIGERSearch into XQuery and TüNDRA uses the XML database BaseX to perform the actual search. The KorAP system (Bański et al., 2014) also uses an external library to search the data. It can either use a Lucene based implementation (called Krill) or a Neo4j graph database. In both cases queries can be formulated in different query languages and are transformed into a unified representation which is executed by one of the back-ends. One of the query languages that is currently supported by KorAP is AQL.

In contrast to e.g. KorAP or CWB, graphANNIS is a main memory based system and thus the corpus size it can handle is limited by the amount of main memory available on the computer. [14] Another approach for a query system based on main memory was the ANNIS implementation from Rosenfeld (2012) on top of the MonetDB (`https://www.monetdb.org/`) column store. In this approach the original normalized relational database schema of ANNIS was used but SQL-queries were optimized to allow efficient joins in MonetDB. Since MonetDB also does not use any secondary indexes but sorts the tables by a column, overhead in representing the data was avoided. In the experiments it could be shown that using MonetDB allowed to execute the workload of selected queries on the Tiger2 corpus three times faster than PostgreSQL. At the same time the memory consumption for representing the Tiger2 corpus was about 400MB using MonetDB instead of the 8GB needed to store all tables and indexes in PostgreSQL.

7 Conclusions

This work presented graphANNIS, a main memory based search system for deeply annotated linguistic corpora implementing the popular AQL query language. We argued

[13] Evert and Hardie (2015) describe a not yet released new generation of the Corpus Workbench, where hierarchical and graph structures are supported.

[14] Estimating the maximal corpus size supported by graphANNIS is hard, since this not only depends on the number of tokens but also on the annotation layers of the corpus. E.g. for a Tiger2 like corpus graphANNIS could handle about 4 million tokens using 2 GB of main memory and more than 100 million tokens using 64GB main memory (e.g. on a powerful server).

for re-using existing data models and query languages, analyzed problems in the existing relational database implementation and implemented a more flexible approach, which allows to combine different optimization techniques for different kinds of annotations in one single data model and search system. In our mind, especially the concept of specialized Graph Storages for different kinds of annotation layers makes graphANNIS a truly multi-layer corpus search system. Using realistic queries in a benchmark, we could show that graphANNIS clearly outperforms a relational implementation of AQL by a factor of more than 40, and that it leads to speed-up in more than 95% of all tested queries. Being main memory-based, graphANNIS requires a machine proportional to the size of a corpus. All corpora we tested easily fit into 2GB memory, and even a corpus ten times larger than Tiger2 could be queried on a modern desktop computer with 16GB memory.

Besides being very fast, graphANNIS also has a number of other advantages compared to a relational implementation. First, graphANNIS does not require installing and maintaining a database server, a task which has proven very difficult for many linguistic research groups without any IT support. Second, the disk footprint of graphANNIS is much smaller, as none of its internal index structures are stored on disk. This also makes copying corpora much easier and faster; note that corpus import into relANNIS is a rather painful and slow process. Third, starting the AQL server is faster and easier with graphANNIS as no SQL server needs to boot. Finally, for all but the largest corpora the main memory requirements of graphANNIS are actually smaller than those of relANNIS, since the PostgreSQL server itself requires quite a lot of RAM.

Still, there is still room for improvement. For instance, query execution currently is single-threaded, which is a waste of resources on modern computers. Also cost estimates could be improved, and execution plans could be adapted to current CPU technologies (Willhalm et al., 2009). Also data compression has proven very effective in main memory database systems since it leads to much better cache line usage (Abadi et al., 2009). It would be also interesting to benchmark graphANNIS against other linguistic query languages or query systems. For instance, the KorAP system is able to execute AQL queries and if the corpora from our benchmark can be imported into KorAP the existing query set would allow for a good comparison.[15] Additionally to these specialized systems, AQL could be implemented on top of other general purpose main memory database implementations (either relational or graph based) and evaluated if mapping the data model negatively impacts performance.

References

Abadi, D. J., Boncz, P. A., and Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665.

Ágel, V. and Hennig, M. (2014). Kasseler Junktionskorpus (Version 1.1). Justus-Liebig-Universität Gießen. `http://hdl.handle.net/11022/0000-0000-2102-8`.

[15] A conversion should be possible by using the Pepper framework (Zipser et al., 2010).

Baumann, S. and Riester, A. (2013). Coreference, lexical givenness and prosody in German. *Lingua*, 136:16–37.

Bański, P., Bingel, J., Diewald, N., Frick, E., Hanl, M., Kupietz, M., Pęzik, P., Schnober, C., and Witt, A. (2014). KorAP: the new corpus analysis platform at IDS Mannheim. In Vetulani, Z. and Uszkoreit, H., editors, *Human Language Technology Challenges for Computer Science and Linguistics : 6th language & technology conference december 7-9, 2013, Poznań, Poland*, pages 586 – 587.

Brants, S., Dipper, S., Eisenberg, P., Hansen-Schirra, S., König, E., Lezius, W., Rohrer, C., Smith, G., and Uszkoreit, H. (2004). Tiger: Linguistic interpretation of a german corpus. *Research on Language and Computation*, 2(4):597–620.

Carletta, J., Evert, S., Heid, U., Kilgour, J., Robertson, J., and Voormann, H. (2003). The NITE XML toolkit: Flexible annotation for Multi-modal Language Data. *Behavior Research Methods, Instruments, and Computers*, 35(3):353–363.

Chiarcos, C., Dipper, S., Götze, M., Leser, U., Lüdeling, A., Ritz, J., and Stede, M. (2008). A flexible framework for integrating annotations from different tools and tag sets. *Traitment automatique des langues*, 49(2):271–293.

Coniglio, M., Donhauser, K., and Schlachter, E. (2014). HIPKON: Historisches Predigtenkorpus zum Nachfeld (Version 1.0). Humboldt-Universität zu Berlin. SFB 632 Teilprojekt B4. `http://hdl.handle.net/11022/0000-0000-2D18-4`.

Donhauser, K., Gippert, J., and Lühr, R. (2015). Deutsch Diachron Digital - Referenzkorpus Altdeutsch (Version 0.1). Humboldt-Universität zu Berlin. `http://hdl.handle.net/11022/0000-0000-7FC2-7`.

Evert, S. and Hardie, A. (2011). Twenty-first century corpus workbench: Updating a query architecture for the new millennium. In *Proceedings of the Corpus Linguistics 2011 conference*. University of Birmingham.

Evert, S. and Hardie, A. (2015). Ziggurat: A new data model and indexing format for large annotated text corpora. *Challenges in the Management of Large Corpora (CMLC-3)*, page 21.

Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., and Lehner, W. (2012). Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51.

Frick, E., Schnober, C., and Banski, P. (2012). Evaluating query languages for a corpus processing system. In *LREC*, pages 2286–2294.

Grust, T., Keulen, M. V., and Teubner, J. (2004). Accelerating XPath evaluation in any RDBMS. *ACM Transactions on Database Systems (TODS)*, 29(1):91–131.

Hilpert, M. (2008). *Germanic Future Constructions: A Usage-Based approach to Language Change*. John Benjamins, Amsterdam.

Hirschmann, H., Lüdeling, A., and Zeldes, A. (2008). What's hard? - Quantitative evidence for difficult constructions in German learner data. In *Proceedings of QITL 3. Helsinki*. `http://edoc.hu-berlin.de/docviews/abstract.php?lang=ger&id=37129`.

Krause, T. and Zeldes, A. (2016). ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, 31(1):118–139. http://dsh.oxfordjournals.org/content/31/1/118.

Lee, J., Yeung, C., Zeldes, A., Reznicek, M., Lüdeling, A., and Webster, J. (2015). Cityu corpus of essay drafts of english language learners: a corpus of textual revision in second language writing. *Language Resources and Evaluation*, 49(3):659–683.

Leech, G. N. (1997). Introducing corpus annotation. In Garside, R., Leech, G. N., and McEnery, T., editors, *Corpus Annotation: Linguistic Information from Computer Text Corpora*, pages 1–18. Longman, London.

Levy, R. and Andrew, G. (2006). Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the fifth international conference on Language Resources and Evaluation*, pages 2231–2234.

Lezius, W. (2002). TIGERSearch Ein Suchwerkzeug für Baumbanken. *Tagungsband zur Konvens*.

Losemann, K. and Martens, W. (2012). The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 101–112. ACM.

Lue, X. (2011). A corpus-based evaluation of syntactic complexity measures as indices of college-level esl writers' language development. *TESOL Quarterly*, 45(1):1–27.

Lüdeling, A. (2011). Corpora in linguistics: Sampling and annotation. In Grandin, K., editor, *Going Digital. Evolutionary and Revolutionary Aspects of Digitization.*, Nobel Symposium 147, pages 220–243. Science History Publications/USA, New York.

Lühr, R., Faßhauer, V., Prutscher, D., and Seidel, H. (2015). Fuerstinnenkorrespondenz 1.1. Universität Jena, DFG. http://hdl.handle.net/11022/0000-0000-82A0-7.

Odebrecht, C. (2011). Lexical Bundles. Eine korpuslinguistische Untersuchung. Master's thesis, Humboldt-Universität zu Berlin. http://edoc.hu-berlin.de/master/odebrecht-carolin-2012-03-12/PDF/odebrecht.pdf.

Odebrecht, C., Belz, M., Zeldes, A., Lüdeling, A., and Krause, T. (accepted 2016). RIDGES Herbology - Designing a Diachronic Multi-Layer Corpus.

Reznicek, M., Lüdeling, A., Krummes, C., Schwantuschke, F., Walter, M., Schmidt, K., Hirschmann, H., and Andreas, T. (2012). Das Falko-Handbuch. Korpusaufbau und Annotationen Version 2.01. Technical report, Technical report, Department of German Studies and Linguistics, Humboldt University, Berlin, Germany. https://www.linguistik.hu-berlin.de/de/institut/professuren/korpuslinguistik/forschung/falko/FalkoHandbuchV2/at_download/file.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media.

Rohde, D. L. (2005). Tgrep2 user manual. http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf.

Rosenfeld, V. (2010). An implementation of the Annis 2 query language. Technical report, Humboldt-Universität zu Berlin. https://www.informatik.hu-berlin.de/de/forschung/gebiete/ti/wbi/teaching/studienDiplomArbeiten/finished/2010/rosenfeld_studienarbeit.pdf.

Rosenfeld, V. (2012). A linguistic query language on top of a column-oriented main-memory database. Master's thesis, Humboldt-Universität zu Berlin. http://www.user.tu-berlin.de/viktor-rosenfeld/assets/publications/diplomarbeit.pdf.

Sauer, S. (2013). BeMaTaC. http://u.hu-berlin.de/bematac.

Schiller, A., Teufel, S., Stöckert, C., and Thielen, C. (1999). Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Universität Stuttgart, Institut fur maschinelle Sprachverarbeitung; Universität Tübingen, Seminar für Sprachwissenschaft.

Seufert, S., Anand, A., Bedathur, S., and Weikum, G. (2013). Ferrari: Flexible and efficient reachability range assignment for graph indexing. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1009–1020. IEEE.

Stede, M. and Neumann, A. (2014). Potsdam Commentary Corpus 2.0: Annotation for Discourse Research. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2014/pdf/579_Paper.pdf.

Telljohann, H., Hinrichs, E., Kübler, S., Zinsmeister, H., and Beck, K. (2009a). Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z). Technical report, Universität Tübingen Seminar für Sprachwissenschaft. http://www.sfs.uni-tuebingen.de/ascl/ressourcen/corpora/tueba-dz.html.

Telljohann, H., Hinrichs, E. W., Kübler, S., Zinsmeister, H., and Beck, K. (2009b). *Stylebook for the Tübingen Treebank of Written German (TüBa-D/Z)*. Universitüt Tübingen Seminar für Sprachwissenschaft, Wilhelmstr. 19 D-72074 Tübingen.

Walter, M. (2015). Märchenkorpus Version 1.0. Humboldt-Universität zu Berlin. http://www.textbewegung.de/. http://hdl.handle.net/11022/0000-0000-8211-9.

Willhalm, T., Popovici, N., Boshmaf, Y., Plattner, H., Zeier, A., and Schaffner, J. (2009). SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units. *Proceedings of the VLDB Endowment*, 2(1):385–394.

Wood, P. T. (2012). Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60.

Yildirim, H., Chaoji, V., and Zaki, M. J. (2010). Grail: Scalable reachability index for large graphs. *Proceedings of the VLDB Endowment*, 3(1-2):276–284.

Zeldes, A. (2016a). *ANNIS User Guide - Version 3.4.3*. http://corpus-tools.org/annis/resources/ANNIS_User_Guide_3.4.3.pdf.

Zeldes, A. (2016b). The GUM corpus: creating multilayer resources in the classroom. *Language Resources and Evaluation*, pages 1–32.

Zhang, H., Chen, G., Ooi, B. C., Tan, K.-L., and Zhang, M. (2015). In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1920–1948.

Zinsmeister, H., Reznicek, M., Brede, J. R., Rosén, C., and Skiba, D. (2012). Das Wissenschaftliche Netzwerk „Kobalt-DaF". *Zeitschrift für germanistische Linguistik*, 40(3):457–458. https://dx.doi.org/10.1515/zgl-2012-0030.

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

Paul Rayson, John Mariani, Bryce Anderson-Cooper, Alistair Baron, David Gullick, Andrew Moore, Stephen Wattam

Towards Interactive Multidimensional Visualisations for Corpus Linguistics

We propose the novel application of dynamic and interactive visualisation techniques to support the iterative and exploratory investigations typical of the corpus linguistics methodology. Very large scale text analysis is already carried out in corpus-based language analysis by employing methods such as frequency profiling, keywords, concordancing, collocations and n-grams. However, at present only basic visualisation methods are utilised. In this paper, we describe case studies of multiple types of key word clouds, explorer tools for collocation networks, and compare network and language distance visualisations for online social networks. These are shown to fit better with the iterative data-driven corpus methodology, and permit some level of scalability to cope with ever increasing corpus size and complexity. In addition, they will allow corpus linguistic methods to be used more widely in the digital humanities and social sciences since the learning curve with visualisations is shallower for non-experts.

1 Introduction

Corpus linguistics is a methodology for the study of language using large bodies (corpora, singular corpus) of naturally occurring written or spoken language (Leech, 1991). Corpus linguistics has collected together a number of computer-aided text analysis methods such as frequency profiling, concordancing, collocations, keywords and n-grams (also called clusters or lexical bundles) which have been utilised over the last forty years or so for language analysis in a number of areas in linguistics e.g. vocabulary, syntax, semantics, pragmatics, stylistics and discourse analysis. Corpus methods are inherently data driven, largely exploratory and allow the analyst to carry out empirical investigations, to discover patterns in the data that are otherwise difficult to see by other means e.g. by intuition about language (Sinclair, 2004).

The corpus linguistics methodology is based on comparing corpora or subsets of a corpus with each other in order to discover differences in the language represented by those corpora or sub-corpora. Many standard reference corpora have been collected to represent specific language varieties or genres. With the availability of more powerful computers and larger data storage facilities, these standard reference corpora have increased in size

over the years from the one million word LOB corpus (Johansson et al., 1978), 100 million word British National Corpus (BNC) (Leech, 1993), 385 million word COCA (Davies, 2009) and the two billion word Oxford English Corpus. However, corpus methods have largely remained the same over this time period. As a result, compromises have to be made with each type of corpus analysis e.g. higher cut-off values are used to filter key words and collocation results based on a need to reduce analysis time rather than for any specific level of significance. Concordance lines are thinned by large factors in order to fit with time scales of analysis rather than by variation and relevance factors. With the web-as-corpus paradigm (Kilgarriff and Grefenstette, 2003) gaining prominence, even larger collections of textual data sourced from websites are becoming available (Baroni et al., 2009) so the problem will continue to worsen. In addition, corpus linguistics methods are spreading to other research areas in linguistics, digital humanities and social sciences e.g. discourse analysis (Baker, 2006), sociolinguistics (Baker, 2010), conceptual history (Pumfrey et al., 2012), and psychology (Prentice et al., 2012). For these disciplines, it is imperative that the corpus tools and methods have a shallow learning curve (Rayson, 2006) and we hypothesise that interactive visualisation technologies will help with this expansion.

Basic visualisation techniques (e.g. bar charts for relative frequency plots) have been used in the past in corpus linguistics but these have focussed on one level (e.g. lexical, grammatical, semantic) or method of analysis at a time. Very few publications discuss specific requirements for extending corpus retrieval software (c.f. Smith et al. (2008)), and this paper goes some way to address this deficiency. The main contributions of this paper are the novel interactive and dynamic techniques that we have developed for extending advanced corpus linguistics methods. We also propose a framework to combine all these separate multiple dimensions together. We describe an interactive key word cloud for visualising keyness statistics, an interactive and dynamic method for visualising collocation statistics and a method for contrasting social network relations with language comparisons. These visualisation methods are an improvement on current state of the art in at least four ways. First, they are designed to support the data-driven multidimensional iterative exploration embodied in the corpus linguistics methodology. Second, they address the shortcomings of current static one dimensional corpus methods. Third, they are scalable in order to cope with increasing corpus size and complexity. Finally, they contribute to enabling the analysis methods of corpus linguistics to be accessible to a variety of audiences, for example, non-technical users in the wider social sciences and humanities.

2 Related Work

The corpus linguistics methodology for the study of language using large corpora consists of five core steps (adapted from Rayson (2008)):

1. Question: devise a research question
2. Build: corpus design and compilation
3. Annotate: manual or automatic analysis of the corpus
4. Retrieve: quantitative and qualitative analyses of the corpus
5. Interpret: manual interpretation of the results

The methodology is inherently data-driven and empirical, exploiting the collections of real language samples to drive the analysis and direct the results as opposed to the use of manually constructed language examples driven by intuition. Corpus retrieval software, our focus here, is intended to facilitate exploration of the annotated corpus data using a variety of quantitative techniques. These techniques include frequency profiling: listing all of the words (types) in the corpus and how frequently they occur, and concordancing: listing each occurrence of a word (token) in a corpus along with the surrounding context. The n-gram technique (also called clusters or lexical bundles) counts and lists repeated sequences of consecutive words in order to show fixed patterns within a corpus. A typical corpus investigation would proceed with a large number of retrieval operations conducted through the corpus retrieval software (e.g. to check the frequency of a particular word or linguistic feature, or to search for an item or pattern using the concordancing view), guided by the research question and the quantitative results obtained in earlier searches. Although this iterative process is often not reported in final publications, it is evident from the many textbook descriptions of corpus linguistics. Typically, the research question itself (step 1) is refined in the light of categorisation and analysis of concordance results and comparison operations between corpora, and then the stepwise process begins again. This refinement process specifically corresponds to the interactive exploratory approach that we propose here to be aided by improvements in visualisation methods. Although they are not necessarily viewed as such, some existing techniques in corpus linguistics can be considered as visualisations. In this and the next section we will consider three of the most prominent examples: concordances, collocations and key words.

First and foremost, the concordance view with one word of interest aligned vertically in the middle of the text and the left and right context justified in the middle, is a way of visualising the patterns of the context of a particular word, and is the main way that corpus linguists engage with corpora. By sorting the right and left context, we can more easily see the repeated

patterns. Concgrams (Cheng et al., 2006) takes this visualisation one step further by automatically highlighting repeated patterns in the surrounding context, as shown in figure 1.

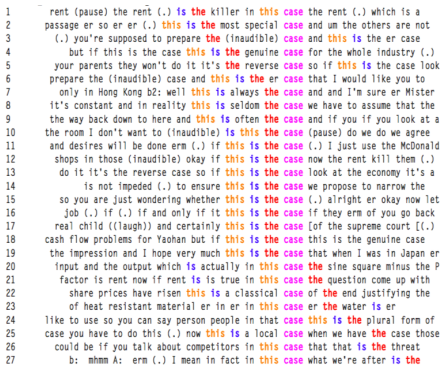


Figure 1: Concordance concgrams.

Another method in the corpus retrieval toolbox is collocation, for which Beavan (2008) has already explored visualisation techniques. Collocations are pairs or sequences of words that co-occur in a text more often than would be expected by chance, usually within a window of five words of each other. By taking the collocates of a word, ordering them alphabetically and altering the font size and brightness, the collocate cloud shown in figure 2 provides an intuitive view of a set of collocates. Here, font size is linked to frequency of the collocate and brightness shows the Mutual Information (MI) score (a statistical measure of the strength of association between the words). In this way, we can easily see the large and bright words that are frequent with strong collocation affinity. Also, in the area of collocations, McEnery (2006) employs a visualisation technique when manually drawing collocational networks (figure 3). These show key words that are linked by common collocates. McEnery's work is influenced by Phillips (1985) who uses similar (again, manually created) diagrams to study the structure of text.

Visualisation is finding application in many areas of the modern world; in science, arts, social media, and the news. The cognitive principles behind visualisation are well summarised by Meirelles (2011) when she writes "to record information; to convey meaning; to increase working memory; to facilitate search; to facilitate discovery; to support perceptual inference; to enhance detection and recognition; and to provide models of actual and theoretical worlds". Linguistics is no exception to the rule. An interesting

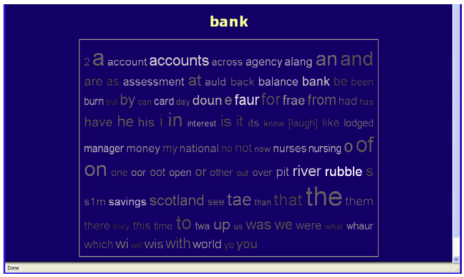


Figure 2: Collocate Cloud.

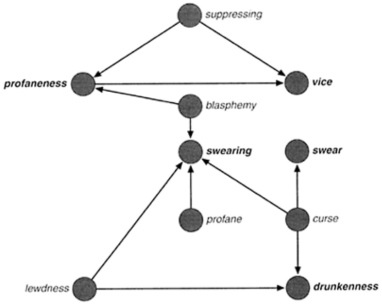


Figure 3: Collocational network created manually.

aspect of the work is a willingness to use existing tools, not only those specifically designed for corpora, but also more general visualisation toolsets.

Siirtola et al. (2010) foresaw some of this development when they discussed the use of the R statistical language and the Mondrian data visualisation tool. However, the R language is generally thought to have a steep learning curve and "the dreaded command line interface" (ibid). They argue for interactive tools, but of course, when using various tools from different sources, it can be difficult to link together the tools so that changes made in one are reflected in the other. Scrivner and Kubler (2015) describe a multi-dimensional parallel Old Occitan-English corpus. They use the ANNIS (Zeldes et al., 2009) search engine to provide graphical querying and displaying of multi-layered corpora. The user can specify their query graphically. In their example, they convert the retrieved data into the R data frame format and produce a motion chart, using GoogleViz.

The Text Variation Explorer (TVE) (Siirtola et al., 2014) harkens back to earlier work done in the visualisation field by Ben Shneiderman, and

the concepts of direct manipulation, continuous and immediate feedback and linked visualisations (see, for example, the Film Finder). This, as indicated above, can be difficult to achieve when using a tool chain. They refer back to the 2010 paper when they observe that while Mondrian can supply interactive graphs (for quickly formulating hypotheses about data and perhaps even managing to verify them in some cases) it lacks one essential: a connection to the text itself. The graphical aspect of TVE is a line graph; in the paper, they use James Joyce's "Ulysses" as an example. They split the text into windows (the size of which is specified by the user) and calculate three parameters, each one of which is represented as a line within the graph. So, going from left to right, we move from the first window of the novel to the last. The user can position themselves anywhere on the line graph, and the underlying text of the window they are selecting is also displayed (in context with the rest of the text). By accessing the text display, and selecting (a) word(s), their new position in the text is reflected in the line graph. This is known as "brushing", the ability to interact with one visualisation and have that interaction reflected in all other associated visualisations.

The WordWanderer (Dork and Knight, 2015) extends tag clouds into a navigational interface for text. Beginning with an alphabetically ordered tag cloud showing frequency. By moving the pointer over a word (in their "Hansel and Gretel" example), say "forest", common collocates are highlighted (this indicates that "children" is a common collocate). If the user now selects "forest", its collocates are organised according to their relative proximity in the text. Finally, if the user draws a line between two words, we get a comparison view, arranging collocates according to their relative strength of association to each of the two words. Hilpert (2011) proposed the use of motion charts (a series of time ordered scatter plots) to dynamically visualise language change in a diachronic corpus. This type of visualisation requires relatively large corpora.

A novel direction has emerged recently in two distinct areas: dialectology and spatial humanities. The common thread between these two approaches is map-based visualisations of language data. In order to understand regional linguistic variation in the US, Huang et al. (2015) collected a year of geo-tagged Twitter data. County-based results were plotted and hierarchically clustered dialect regions were derived from the analysis. In order to showcase the newly emerging area of spatial humanities which combines Geographic Information Systems with natural language processing and corpus linguistics, Murrieta-Flores et al. (2015) carried out an analysis of the UK Registrar General's Reports containing descriptions, census data and other information to examine how mentions of various diseases correlated with place names in the data (see figure 4 for an example of their results). Map-based visualisations are derived and were compared over decade spans. In general,

the spatial humanities method allows a researcher to ask three main types of questions of a dataset (a) where is the corpus talking about, (b) what is the corpus saying about these places, and (c) what is the corpus saying about specific themes e.g. health and disease, money and finance, in proximity to these places? In contrast, Knowles et al. (2015) have explored 'inductive visualisation' techniques that allow the exploration of time and space in holocaust testimonies which do not lend themselves to regular geographical and sequential time-based representations.

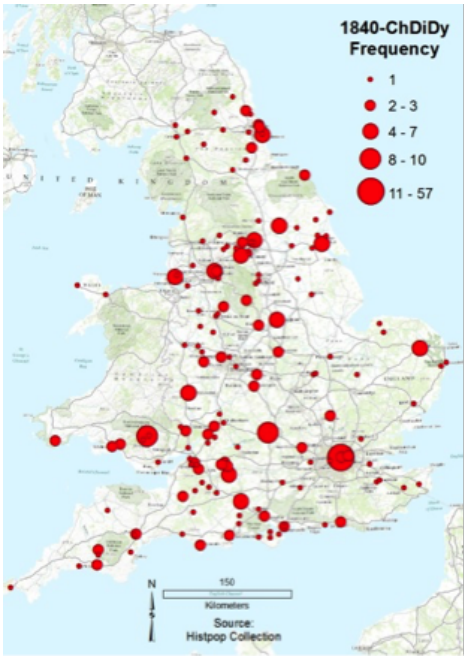


Figure 4: Frequency map of diseases in Registrar General data.

In other cognate areas, such as digital humanities and literary analysis, visualisation approaches are gaining ground. Keim and Oelke (2007) and Oelke et al. (2012) develop the idea of a literature fingerprint which is a pixel-based visualisation to view fine-grained detail of one particular value, where each pixel corresponds to one word. This value could represent the occurrence of a particular character name, function words, average sentence length or hapax legomena. Voyant Tools (Sinclair and Rockwell, 2016) provides a web-based text reading and analysis environment, complemented by a variety of visualisations including: bubbles and cirrus (similar to word

clouds), bubblelines (word repetitions), links (collocation relationships), and RezoViz (relationships between people, places and organisations). Wattenberg and Viégas (2008) present the Word Tree as an interactive version of the concordance view, first implemented in the IBM Many Eyes system. It provides a branching view of words and contexts occurring to the right of the word in the centre of the concordance largely preserving the linear view of the text. Culy and Lyding (2010) extended this (to be closer to the corpus linguistics approach) in their Double Tree implementation to include words on the left of the concordance line, frequency and part-of-speech information. Two further surveys of text visualisation techniques and related taxonomies were created by Kucher and Kerren (2015) and Jänicke et al. (2015).

The discussed examples allow us to highlight some further issues with the current technologies used to visualise large bodies of text. The first problem is the static nature of many of the technologies. This often presents users with far more information than necessary and offers no mechanism to limit the data to those aspects the user is interested in. This static nature can cause a significant amount of information overload, rather than reduce its impact and this is the second issue to be faced. This problem was partly tackled by TextArc amongst other tools which allow the interrogation of the data. However, this technology still displays the whole block of textual data at the same time, which will leave the graphic cluttered and possibly unclear. More generally, static and full text representations do not sit well with the iterative and data-driven nature of the corpus linguistics methodology. Very few of the existing techniques are tailored for the specific methods in corpus linguistics, and in addition, the existing corpus visualisations do not scale to large bodies of texts, a key requirement to tackle the growing size of corpora. All these reasons call for new visualisation techniques, or at least the adaptation of existing ones, in order to specifically address the particular needs of corpus linguistics in terms of scalability, and support for iterative exploration.

3 Case Studies

With the case studies presented in the following three subsections, we examine complementary aspects of visualising different dimensions of language corpora. Our case studies cover three of the five main methods in the corpus linguistic methodology: frequency lists, key words, and collocations. A fourth method, concordancing, is included in our multidimensional visualisation framework as proposed in section 4.

3.1 Case Study 1: key word and tag clouds

In this first case study, we propose a method that can be applied at multiple linguistic levels for the visualisation of key words results. The key words technique (Scott, 1997) is well known in corpus linguistics to users of WordSmith[1], Wmatrix[2], AntConc[3] and other tools. By comparing one corpus or text to a much larger reference corpus (or another comparable text), we can extract those words that occur with unusual frequency in our corpus relative to a general level of expectation. A keyness metric, usually chi-squared or log-likelihood, along with an effect size is calculated for each word to show how 'unexpected' its frequency is in the corpus relative to the reference corpus. By sorting on this keyness value we can order the words and see the most 'key' words at the top of a table. In the Wmatrix software (Rayson, 2008), we have included a visualisation of the key words results in a static but interactive 'key word cloud'. In contrast to tag clouds in Flickr and other social networking websites, where the frequency of a word is mapped to its font size, the key word cloud maps the keyness value onto font size. By doing so, we can quickly 'gist' a document by viewing the words in the key word cloud. In addition, we can apply the same comparison approach at other levels of linguistic analysis. Instead of comparing two word frequency lists, we can compare two part-of-speech frequency lists, or two semantic tag frequency lists. This extends the existing method and permits gisting by stylistic profile and key concepts. Previous work has used word clouds for visualising texts (Heimerl et al., 2014; Xu et al., 2016), but these have not exploited the keyness measures used in corpus linguistics. Vuillemot et al. (2009) does use the log-likelihood measure to compare sub-corpora but then relates word size to frequency rather than keyness. Our method also avoids the need for stop word removal of frequent closed class words which may well result in the loss of significant items of linguistic interest.

Here, we describe a case study using data drawn from the set of UK General Election 2015 Manifestos from the seven main political parties. Via this example, we show the key word and key concept cloud visualisation in practice. First, the seven manifestos for Conservatives, Labour, Liberal Democrats, Green Party, Plaid Cymru, Scottish National Party (SNP) and UKIP were downloaded from their websites in May 2015. Each file was converted from PDF by saving as text from Adobe Reader. Minor editing was required to format headers, footers and page numbers in XML tags, and converted n-dashes, pound signs, begin and end quotes to XML entities. Next, the resulting files were run through the Wmatrix tag wizard pipeline which assigns part-of-speech tags (Garside and Smith, 1997) and semantic

[1] http://www.lexically.net/wordsmith/
[2] http://ucrel.lancaster.ac.uk/wmatrix/
[3] http://www.laurenceanthony.net/software/antconc/

tags (Rayson et al., 2004a) and prepares word frequency and semantic tag frequency lists. Key word and semantic tag clouds are produced by comparing the frequency lists with the BNC Written Sampler corpus[4]. In these visualisations, the larger the font, the higher the log-likelihood score, so larger items are more significantly overused compared to the reference corpus.

The first two visualisations show the key words and key semantic categories for the Conservative party. Figure 5, at the word level, shows their focus on EU, tax, NHS and schools, amongst other items. Figure 6, at the semantic tag level, expands this and highlights their discourse on law and order, business, and employment, in particular.

Figure 5: Word cloud for Conservative manifesto.

Figure 6: Semantic tag cloud for Conservative manifesto.

These two clouds can be contrasted with all those from the other parties.[5] For reasons of space here, we include only one other party. For the Green

[4] http://ucrel.lancaster.ac.uk/bnc2sampler/sampler.htm

[5] See http://ucrel.lancaster.ac.uk/wmatrix/ukmanifestos2015/ for the full set.

Party, the six most key words in their manifesto are green_party, we, local, tax, energy and climate, as shown in figure 7. Alongside green issues, their key semantic cloud in figure 8 focusses on money and government.

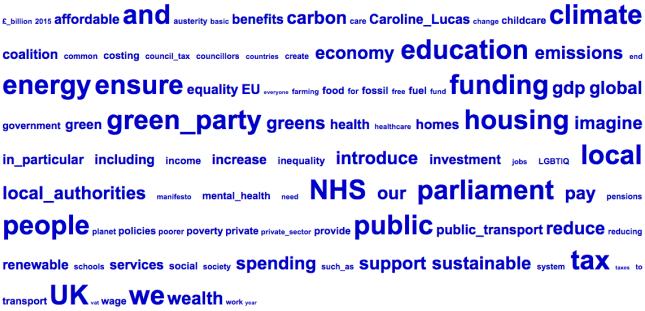


Figure 7: Word cloud for Green Party manifesto.

Figure 8: Semantic tag cloud for Green Party manifesto.

The Wmatrix software allows a user to click through the cloud in order to view concordance lines for a specific word or semantic tag, and by hovering over an item, the frequency and log likelihood statistic can be viewed. Thus the word and tag clouds do have interactive elements and represent multidimensional or multi-level visualisations.

3.2 Case Study 2: collocation networks

In the second case study, we propose to use interactive visualisation techniques to improve the interpretation and exploration of the collocation method in corpus linguistics. We have implemented these methods in both

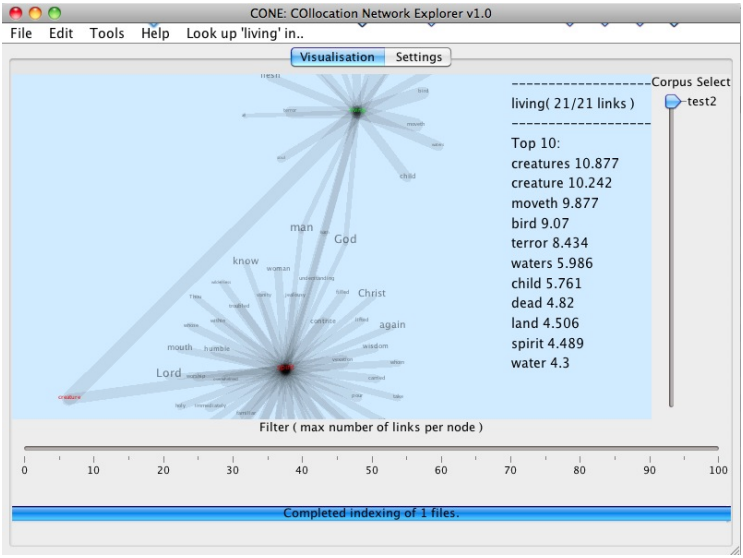


Figure 9: CONE

CONE (Gullick et al., 2010) and GraphColl (Brezina et al., 2015) (Figures 9 and 10 respectively) which provide visualisation of text collocation for all terms within a corpus simultaneously, presenting a graph that the user can manipulate and explore. The concept of collocational networks is a natural extension of collocation, and was first proposed before computing hardware was generally sufficient to provide an interactive visualisation (Phillips, 1985).

Collocation networks are generated by computing a statistical measure of association between all terms within the corpus. Such terms form the nodes of the graph, with edges being drawn between those with a significant tendency to co-occur. The exact measure and policy for graph construction varies between implementations. Early implementations used the mutual information (MI) score (Williams, 2002). CONE implements the commonly-used log-likelihood score as a measure of significance (Rayson et al., 2004b), whereas GraphColl supports a number of measures, as well as implementation of bespoke approaches.

Graph exploration presents a number of design challenges. Firstly, the choice of statistical measure (and the significance or effect size threshold chosen) dramatically affects the resulting graph. This is compounded by the tendency of the constraint-based graph layout algorithms used in both CONE

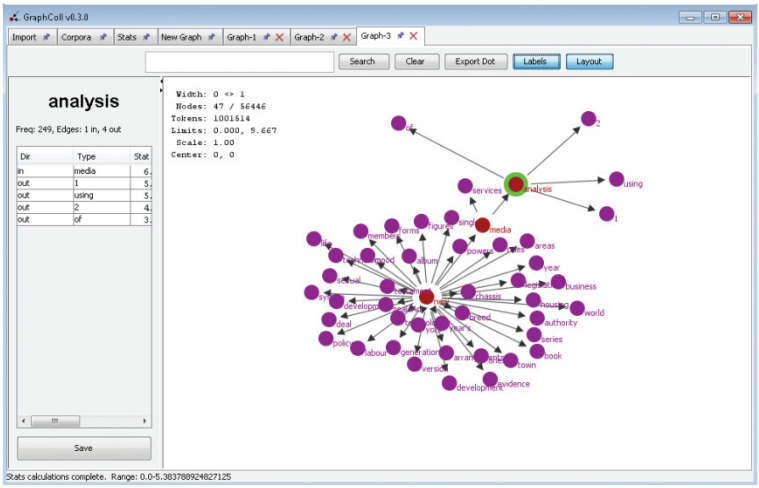


Figure 10: GraphColl

and GraphColl to produce non-deterministic layouts: even topologically similar graphs may appear different.

Graph layout is also a significant challenge to scalability. The Zipfian nature of linguistic data often yields graphs with high centralisation, leading to a dense mass of edges for diverse corpora, a problem also faced in a similar approach by Perkuhn (2007). This is mitigated in both tools by allowing the user to pan and zoom around the graph whilst rendering features at the same scale, essentially making them less dense at higher zoom levels.

Higher level visualisations such as those produced by CONE and GraphColl also present challenges to scientific replicability in that they present large amounts of data in a very dense manner, with the potential to embody many study designs. Both CONE and GraphColl permit partial exploration of graphs, accentuating this issue: a user chooses which nodes to expand (and thus compute collocates for), and this means it is possible to deliberately or unintentionally miss significant links to second-order collocates (or symmetric links back from a collocate to a node word). GraphColl's design attempts to minimise these issues by colouring links according to their "completed" state. This issue is also addressed in documentation, which presents a standardised method for reporting results from graph explorations which is intended to illustrate which design choices have been made during graph creation.

The ease-of-interpretability that visualisations offer presents scientific challenges: when the graph's generating function can be changed *during*

exploration, data dredging becomes as simple as moving a slider. To this end, GraphColl prohibits what many see as desirable features: wildcard searching, stoplists, and on-the-fly adjustment of statistical thresholds are all disallowed by design.

This issue is primed to affect any interactive visualisation. High-level visualisation tools such as CONE and GraphColl must walk a fine line between offering a useful perspective on data (which would not be possible otherwise) and providing such a strong lens as to render any observations largely dependent upon the tool itself. This tendency is evidenced in many areas of science already (such as genetics, which often relies on proprietary machinery), but is readily solvable through responsible reporting and efforts such as the open data movement (Kauppinen and de Espindola, 2011).

The high level from which data are seen also pose technical challenges for interchange formats, leading to a situation where data exported from such tools is either presented in a relatively arcane proprietary format, or stripped of much of the information from the data structures used for analysis. The solution to this lies in an approach of layered formats, which may yield further data where required: something that may take the form of an API to provide live interconnections between tools, or advancements in database representations.

Finally, it should be noted that GraphColl has a concordance feature built-in so that users can use the interface to more closely examine specific collocations in context. Either these things have to be built-in to support richer interaction, or there must be an interchange format to communicate with other corpus tools (a corpus data connector of some kind).

3.3 Case Study 3: social network relationships

This case study proposes the extension of an existing network visualisation based on 'follow relationships' in an online social network (Twitter) to instead be based on distances between language profiles. The overall aim of the study was to analyse potential political defections in the United Kingdom parliament. Using the Twitter REST API[6], the last 3,600 tweets (the maximum available) from verified UK Members of Parliament (MPs), and the list of verified MPs that each of these follow were collected. In total, 426 MPs of the 650 MPs in parliament were present in our dataset, the remaining MPs were not verified or not on Twitter.

The list of follow relationships were converted into a list of one-directional links between each MP who followed another MP, finding 29,345 links in total. If two MPs followed each other, two links were listed.

The words were collected from all tweets and a frequency list created for each MP. We removed URLs and user mentions from the list of words as

[6] `https://dev.twitter.com/rest`

URLs were very rarely repeated and were mostly auto-created short URLs for Twitter, and user mentions were removed to avoid overlap with follow relationships. All punctuation was removed and all words were converted to lowercase. A random sample of 2,000 words was taken for each MP, with MPs excluded who had used less than 2,000 words (thereby removing only 3 MPs). Each MP sample was compared against every other MP using two similarity measures: Jaccard and Log Likelihood. Jaccard looks at the similarity in the set of words used, whereas Log Likelihood looks at frequency differences. This process was repeated 10 times with a different 2,000 word random sample each time.

Both the follow relationships and the language relationships were visualised using force-directed graphs in D3.js[7]. In force-directed graphs, nodes are pushed away from each other while simultaneously pulled towards the centre of the graph. This allows any node's location to be based on their relative positions to one another, attempting to minimise crossing links and balance link distances. For follow relationships, all one-way links were of equal length but bi-directional links were set as half as long to represent a closer relationship. For language relationships, the link length was determined by the value that the similarity measure produced between the two nodes for that link. The more similar two nodes were, the lower the link lengths, bringing the nodes closer together. The closest nodes to any particular node are those that have the closest relationships. The resulting network graphs are shown for follow relationships in Figure 11, for Jaccard word similarity in Figure 12, and for log-likelihood word similarity in Figure 13. Note that the graphs are interactive, allowing particular parties to be highlighted. MP names and links between MPs can also be displayed. In all graphs, the positions of certain MPs and the orientation of the entire graph may vary as nodes are initially randomly placed, resulting in multiple possible stable arrangements. However, the overall pattern is consistent.

The follow relationship graph more visibly splits the MPs into distinct clusters related to political party. This may possibly be due to links not being present between all nodes, unlike in the word similarity graphs where a link is always present, but more or less distant depending on similarity. The word similarity graphs both do show the current three biggest UK political parties (Conservative: blue, Labour: red and Scottish Nationalist: yellow) generally clustered together, with outlier MPs (i.e. clustered closer to other parties) indicating possible interesting cases for further analysis. The interactive visualisation approach in this case study is a vital exploratory tool when developing the method (e.g. selecting appropriate distance measures) and analysing results (e.g. choosing subsets of MPs). Thus, our third case study shows that the existing visualisation technique previously used for exploring

[7] `https://d3js.org`

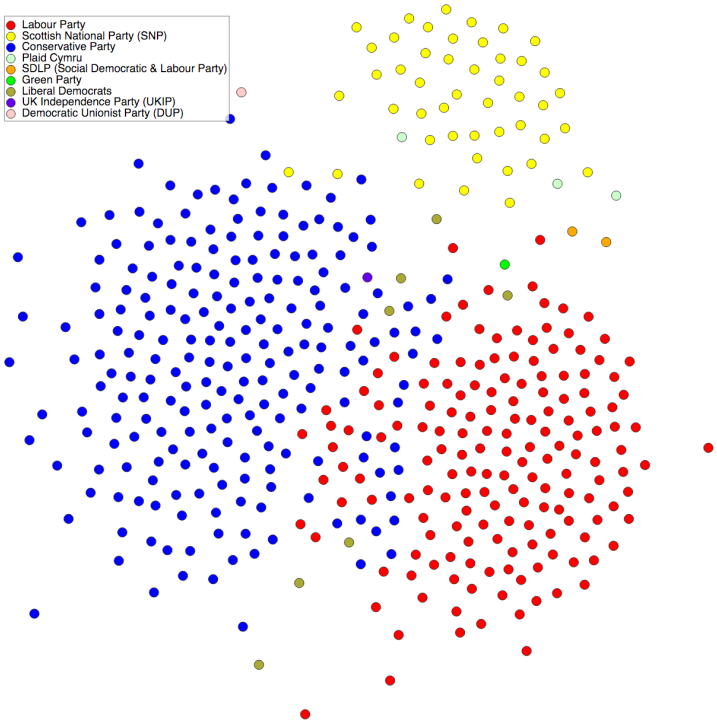


Figure 11: Follow relationships network of UK MPs on Twitter.

the network of relationships in an online social network can also be used to explore the linguistic similarity of specific subcorpora at the word level.

4 Proposal for Multidimensional Visualisation

Using the case studies demonstrated in the previous section, a proposal for putting these concepts into a multidimensional framework is described here. Our framework splits along three orthogonal dimensions: linguistic (lexical, grammar/syntax, semantics), structural (to permit sub-corpora) and temporal (for diachronic corpora). Our proposal for multidimensional visualisation explicitly supports key tenets of interactive visualisation such as navigating from a high level overview of the dataset, via filtering on specific dimensions to view slices or subcorpora (Heer and Shneiderman, 2012).

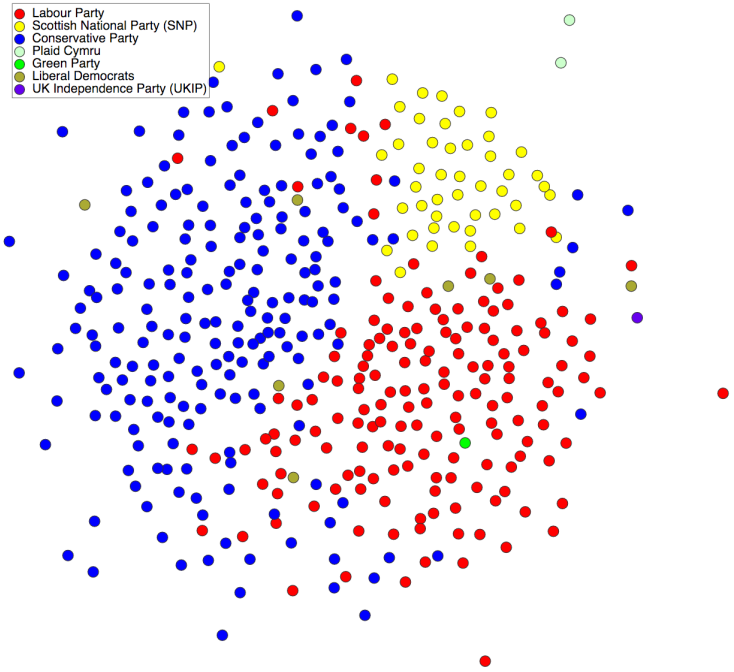


Figure 12: Jaccard word similarity network of UK MPs on Twitter.

Within the structural dimension there is a natural layering beginning with the whole corpus and subdividing into meaningful subsets dependent on the type of data (e.g. documents, chapters, tweets, person). At the top level, the whole corpus level can be analysed via the word clouds shown in section 3.1 where a user can find the most over/under-used words relative to a reference corpus. Incorporating the collocation network approach in section 3.2 the user would be able to click on a word in the key word cloud to explore the collocates for that word or tag, and from there to the concordance view. Furthermore a user should be able to select a group of words within the cloud and visualise collocates for those words to explore further similarities between the words. Second and subsequent layers would permit selection of subcorpora in order to exploit structure within the corpus, e.g. tweets as used in section 3.3. In addition, using the network visualisations shown in section 3.3 a user should be able to define subcorpora and visualise their similarities, differences alongside other relationships drawn from the dataset. A specific use case for our proposed framework can be extended from the

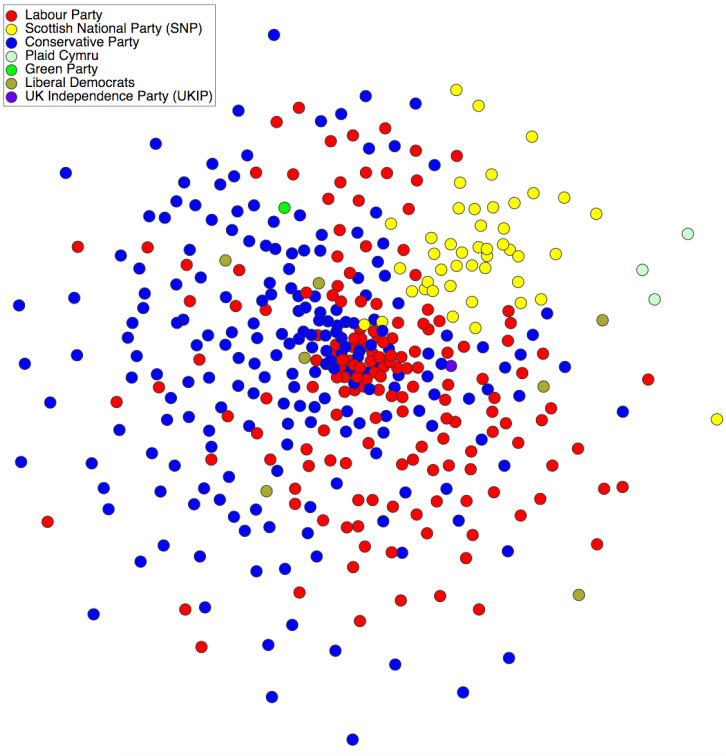


Figure 13: Log-likelihood word similarity network of UK MPs on Twitter.

case study described in section 3.3. A user would explore the MP network and explore the differences between two or more political parties or groups within those parties.

Within the linguistic dimension there are at least three prominent levels: lexical, grammatical and semantic levels, as exemplified in our case studies. The exploration could proceed as described for the structural use case above but would now be extended to cover other levels of linguistic annotation assuming that they were represented in the corpus.

The final dimension incorporated into our proposed framework is time which will assist with the exploration and visualisation of diachronic corpora. A prototypical example of this would be a Twitter corpus that has been collected over a number of months or years. The social network data would be visualised as points on a 2D time series graph. From this graph a user

can select groups of data to compare against within the key word clouds, collocation networks and social network relationships, and how each of these aspects varies over time.

This combination across three dimensions will therefore allow a user to explore the corpus on many different interconnected levels and visualisations. Employing multiple visualisations is of utmost importance to counter deficiencies in some methods (such as information loss and uncertainties in force-based methods) and to ensure the various model abstractions align with analysis tasks (Chuang et al., 2012). We envisage our framework would be developed by achieving interoperability between the existing tools rather than developing a new standalone system.

5 Conclusion and Future Work

In this paper, we have proposed the idea of using interactive information visualisation techniques for supporting the corpus linguistics methodology at multiple levels of analysis. We have highlighted tools and techniques that are already used in corpus linguistics that can be considered as visualisation: concordances, concgrams, collocate clouds, and described new methods of collocational networks and exploratory language analysis in social networks. In addition, we described the key word and semantic cloud approaches as implemented in the Wmatrix software.

With the CONE and GraphColl prototypes, we have proposed and illustrated a highly dynamic way of exploring collocation networks, as an example of our wish to add dynamic elements to both existing and novel visualisations. This would enhance their "data exploration" nature even further. To paraphrase Gene Roddenberry[8], we wish to allow linguists to explore their data in 'strange' new ways and to seek out new patterns and new visualisations. In this enterprise, we can assess the usefulness or otherwise of the new techniques. We have shown how the dynamic techniques align more closely to the iterative data-driven corpus linguistics methodology. With significantly larger corpora being compiled, we predict that the need for visualisation techniques will grow stronger in order to allow interesting patterns to be seen within the language data and avoid practical problems for the corpus linguist who currently needs to analyse very large sets of results by hand. In future work, we will explore techniques which are able to support longer explorations in order to avoid corruption or 'messiness' in the interface which still persists after a prolonged period of use. There is clearly a need for new static analysis techniques as well; to extract the data required as well as novel methods for displaying and exploring the data.

[8] See http://en.wikipedia.org/wiki/Gene_Roddenberry

Acknowledgements

Francois Taiani was involved in supervision of the original CONE project. This work was partly funded by the EPSRC vacation bursary grant awarded to David Gullick at Lancaster University. GraphColl software development was supported by the ESRC Centre for Corpus Approaches to Social Science, ESRC grant reference ES/K002155/1. The UCREL research centre supported the development of the integrated visualisation framework.

References

Baker, P. (2006). *Using Corpora in Discourse Analysis.* Continuum.

Baker, P. (2010). *Sociolinguistics and Corpus Linguistics.* Edinburgh University Press.

Baroni, M., Bernardini, S., Ferraresi, A., and Zanchetta, E. (2009). The wacky wide web: A collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–231.

Beavan, D. (2008). Glimpses through the clouds: collocates in a new light. In *Proceedings of Digital Humanities 2008, University of Oulu, 25-29 June 2008.*

Brezina, V., McEnery, T., and Wattam, S. (2015). Collocations in context: A new perspective on collocation networks. *International Journal of Corpus Linguistics*, 20(2):139–173.

Cheng, W., Greaves, C., and Warren, M. (2006). From n-gram to skipgram to concgram. *International Journal of Corpus Linguistics*, 11(4):411–433.

Chuang, J., Ramage, D., Manning, C. D., and Heer, J. (2012). Interpretation and trust: Designing model-driven visualizations for text analysis. In *ACM Human Factors in Computing Systems (CHI).*

Culy, C. and Lyding, V. (2010). Double Tree: An Advanced KWIC Visualization for Expert Users. In *14th International Conference Information Visualisation*, pages 98–103.

Davies, M. (2009). The 385+ million word corpus of contemporary american english (1990–2008+): Design, architecture, and linguistic insights. *International Journal of Corpus Linguistics*, 14(2):159–190.

Dork, M. and Knight, D. (2015). WordWanderer: A navigational approach to text visualisation. *Corpora*, 10(1):83–94.

Garside, R. and Smith, N. (1997). A hybrid grammatical tagger: CLAWS4. In Garside, R., Leech, G., and McEnery, T., editors, *Corpus Annotation: Linguistic Information from Computer Text Corpora.*, pages 102–121. Longman.

Gullick, D., Rayson, P., Mariani, J., Piao, S., and Taiani, F. (2010). CONE: COllocational Network Explorer [Computer Software]. `http://ucrel.lancaster.ac.uk/cone/`.

Heer, J. and Shneiderman, B. (2012). Interactive dynamics for visual analysis. *Queue*, 10(2):30:30–30:55.

Heimerl, F., Lohmann, S., Lange, S., and Ertl, T. (2014). Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii International Conference on System Sciences*, pages 1833–1842.

Hilpert, M. (2011). Dynamic visualizations of language change: Motion charts on the basis of bivariate and multivariate data from diachronic corpora. *International Journal of Corpus Linguistics*, 16(4):435–461.

Huang, Y., Guo, D., Kasakoff, A., and Grieve, J. (2015). Understanding U.S. regional linguistic variation with twitter data analysis. *Computers, Environment and Urban Systems*, 59:244–255.

Johansson, S., Leech, G., and Goodluck, H. (1978). *Manual of information to accompany the Lancaster-Oslo/Bergen corpus of British English, for use with digital computers.* Department of English, University of Oslo.

Jänicke, S., Franzini, G., Cheema, M. F., and Scheuermann, G. (2015). On Close and Distant Reading in Digital Humanities: A Survey and Future Challenges. In Borgo, R., Ganovelli, F., and Viola, I., editors, *Eurographics Conference on Visualization (EuroVis) - STARs.* The Eurographics Association.

Kauppinen, T. and de Espindola, G. M. (2011). Linked open science-communicating, sharing and evaluating data, methods and results for executable papers. *Procedia Computer Science*, 4:726 – 731. Proceedings of the International Conference on Computational Science, ICCS 2011.

Keim, D. A. and Oelke, D. (2007). Literature fingerprinting: A new method for visual literary analysis. In *IEEE Symposium on Visual Analytics Science and Technology, 2007. VAST 2007.*, pages 115–122.

Kilgarriff, A. and Grefenstette, G. (2003). Introduction to the special issue on the web as corpus. *Computational Linguistics*, 29(3):333–347.

Knowles, A. K., Westerveld, L., and Strom, L. (2015). Inductive Visualization: A Humanistic Alternative to GIS. *GeoHumanities*, 1(2):233–265.

Kucher, K. and Kerren, A. (2015). Text visualization techniques: Taxonomy, visual survey, and community insights. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 117–121.

Leech, G. (1991). *The state of the art in corpus linguistics.*, pages 8–29. Longman.

Leech, G. (1993). 100 million words of English: a description of the background, nature and prospects of the British National Corpus project. *English Today*, 33(9).

McEnery, T. (2006). *Swearing in English.* Routledge, London.

Meirelles, I. (2011). Visualizing data: new pedagogical challenges. In *Selected Readings of the 4th Information Design International Conference*, pages 73–83.

Murrieta-Flores, P., Baron, A., Gregory, I., Hardie, A., and Rayson, P. (2015). Automatically analysing large texts in a GIS environment: the Registrar General's reports and cholera in the nineteenth century. *Transactions in GIS*, 19(2):296–320.

Oelke, D., Kokkinakis, D., and Malm, M. (2012). Advanced visual analytics methods for literature analysis. In *Proceedings of the 6th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 35–44. Association for Computational Linguistics.

Perkuhn, R. (2007). Systematic exploration of collocation profiles. In *Proceedings of Corpus Linguistics 2007, Birmingham, UK.*

Phillips, M. (1985). *Aspects of Text Structure: An investigation of the lexical Organisation of Text*, volume 52. North Holland.

Prentice, S., Taylor, P., Rayson, P., and Giebels, E. (2012). Differentiating act from ideology: evidence from messages for and against violent extremism. *Negotiation and Conflict Management Research*, 5(3):289–306.

Pumfrey, S., Rayson, P., and Mariani, J. (2012). Experiments in 17th century english: manual versus automatic conceptual history. *Literary and Linguistic Computing*, 27(4):395–408.

Rayson, P. (2006). *AHRC e-Science Scoping Study Final report: Findings of the Expert Seminar for Linguistics.* AHRC e-Science Scoping Study (eSSS) project report.

Rayson, P. (2008). From key words to key semantic domains. *International Journal of Corpus Linguistics*, 13(4):519–549.

Rayson, P., Archer, D., Piao, S., and McEnery, T. (2004a). The UCREL semantic analysis system. In *Proceedings of the workshop on Beyond Named Entity Recognition Semantic labelling for NLP tasks in association with 4th International Conference on Language Resources and Evaluation (LREC 2004), 25th May 2004, Lisbon, Portugal.*, pages 7–12.

Rayson, P., Berridge, D., and Francis, B. (2004b). Extending the cochran rule for the comparison of word frequencies between corpora. In *7th International Conference on Statistical analysis of textual data (JADT 2004)*, pages 926–936.

Scott, M. (1997). Pc analysis of key words – and key key words. *System*, 25(2):233–245.

Scrivner, O. and Kubler, S. (2015). Tools for digital humanities: Enabling access to the old occitan romance of flamenca. In *Proceedings of NAACL-HLT Fourth Workshop on Computational Linguistics for Literature*, pages 1–11, Denver, Colorado.

Siirtola, H., Räihä, K.-J., Säily, T., and Nevalainen, T. (2010). Information visualisation for corpus linguistics: Towards interactive tools. In *IVITA'10*, pages 33–36, Hong Kong. ACM.

Siirtola, H., Säily, T., Nevalainen, T., and Räihä, K.-J. (2014). Text variation explorer: Towards interactive visualization tools for corpus linguistics. *International Journal of Corpus Linguistics*, 19:3:418–429.

Sinclair, J. (2004). *Trust the text: language, corpus and discourse.* Routledge.

Sinclair, S. and Rockwell, G. (2016). Voyant tools. `http://voyant-tools.org/`.

Smith, N., Hoffmann, S., and Rayson, P. (2008). Corpus tools and methods, today and tomorrow: Incorporating linguists' manual annotations. *Literary and Linguistic Computing*, 23(2):163–180.

Vuillemot, R., Clement, T., Plaisant, C., and Kumar, A. (2009). What's being said near "Martha"? Exploring name entities in literary text collections. In *IEEE Symposium on Visual Analytics Science and Technology, 2009. VAST 2009.*, pages 107–114.

Wattenberg, M. and Viégas, F. B. (2008). The word tree, an interactive visual concordance. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1221–1228.

Williams, G. (2002). In search of representativity in specialised corpora: Categorisation through collocation. *International Journal of Corpus Linguistics*, 7(1):43–64.

Xu, J., Tao, Y., and Lin, H. (2016). Semantic word cloud generation based on word embeddings. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 239–243.

Zeldes, A., Ritz, J., Lüdeling, A., and Chiarcos, C. (2009). Annis: A search tool for multi-layer annotated corpora. In *Proceedings of Corpus Linguistics*, Liverpool, UK.

Christian Pölitz

Data Mining Software for Corpus Linguistics with Application in Diachronic Linguistics

Abstract

Large digital copora have become a valuable resource for linguistic research. We introduce a software tool to efficiently perform Data Mining tasks for diachronic linguistics to investigate linguistic phenomena with respect to time. As a running example, we show a topic model that extracts different meanings from large digital copora over time.

1 Introduction

From the 1960s on, modern digital text corpora offer large text collections like newspaper articles, social media content, but also language reference corpora for linguistic analysis. With the Internet, even more textual information has become available for everybody. To use such large amounts of digital texts, non-manual methods to extract information for linguistic research must be used. Data Mining methods, see Manning and Schütze (1999) for example, can help to automatically analyze such large document collections and corpora. Data Mining methods try to discover knowledge from data sources and perform automatic analysis tasks based on identification of patterns in the data. The goal is to find information in the data when manual analyses are not possible, too expensive or too time-consuming.

To demonstrate the need for Data Mining in corpus linguistics, we investigate large digital corpora for temporal dynamics: We show the development of meanings over time. The results will show how useful Data Mining methods can be for such linguistic tasks. In this paper, we propose a software tool for meaning extraction that systematically extends standard approaches to explicitly adopt to corpora from heterogeneous language resources with information about time.

For example, from the Dictionary of the German Language, see Geyken (2007), we can retrieve KWIC[1]-lists of snippets containing the German word *Platte*. The snippets are drawn from documents from different genres over a time period from 1900 to 1999. For these documents, we extract different meaning of the word *Platte* by Latent Dirichlet Allocation (Blei et al. (2003)). In Figure 1, we illustrate two extracted meanings. At the top, we show the words that are most important for each meaning by a Word Cloud[2]. We see that we identify two different meanings of the word *Platte*. First, *Platte* in the meaning of a hard drive is found. The most important words are highly computer related. The second meaning identifies the word *Platte* as photographic plate. The important words are all connected to photography.

[1] Key-word-in-context: Examples of word (or expression) usage in texts.
[2] A Word Cloud visualizes frequent words and their importance by font size.

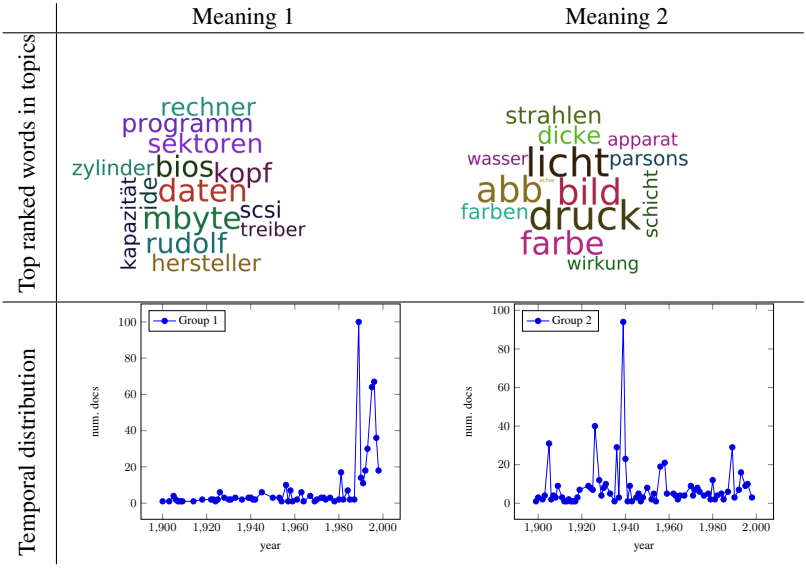


Figure 1: Two possible meanings for the word *Platte* (plate) extracted from KWIC-lists of snippets from the Core Corpus of the Dictionary of the German Language. Top: The most important words in the topics. Bottom: The temporal distribution of the meanings.

At the bottom, we plot the amount of the usage of the different meanings over the time: We count how many documents are assigned to the meanings in each year. We see that *Platte* in the context of a hard drive is mostly used at the end of the 20th century, while *Platte* in the context of photography is mainly used in the 1950s.

Based on this first study. we developed a software tool to implement different versions of topic models and topic models with temporal information that can be used for diachronic linguistics. Besides, algorithms for topic modeling, our software tool offers methods to evaluate and visualize the results on large digital corpora.

2 Topic Models

Topic models are statistical models that extract semantics in document collections based on co-occurrence statistics. For these models, we assume the Multinomial Model (MM): The words in the documents are drawn from Multinomial distributions. The most prominent latent topic models are Probabilistic Latent Semantic Analysis (Deerwester et al. (1990)) and Latent Dirichlet Allocation (Blei et al. (2003)). Both models are mixture models (McLachlan and Basford (1988)) that model the joint probability of words and documents as linear combination of conditional distribution of the latent topics.

2.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) as proposed by Blei et al. (2003) is a generative probabilistic topic model that estimates document-topic distributions θ and topic-word distributions β with Dirichlet priors $\mathrm{Dir}(.)$. Given a corpus C of m documents, each represented by a sequence of words $\mathbf{d} = (w_1, \cdots, w_n)$, LDA models the generative process of generating documents as random draws over random mixtures of latent topics t. We briefly summaries the generative process of documents as the following:

1. For each topic t:
   a) Draw $\beta_t \sim \mathrm{Dir}(\eta)$
2. For each document $d \in C$:
   a) Draw $\theta_d \sim \mathrm{Dir}(\alpha)$
   b) For each word i:
      i. Draw $t_i \sim \mathrm{Mult}(\theta_d)$
      ii. Draw $w_i \sim \mathrm{Mult}(\beta_{t_i})$

First, we draw for each topic t the word probabilities β_t for each word in the corpus. Next, for each document d we draw a T-dimensional Dirichlet distributed random vector θ_d. Then, for each word in the document d we draw a topic t_i from a Multinomial distribution parametrized by θ_d and a word w_i from a Multinomial distribution parametrized by β_{t_i}. In the original approach by Blei et al., β_t does not have a Dirichlet prior $Dir(\eta)$. This is important for sampling based approaches for LDA and for possible extensions with different (more complicated) priors.

In the literature there are two major approaches to estimate an LDA topic model. First, variational inference can be used to approximate the posterior distribution of the latent variables by a simpler variational distribution (Blei et al. (2003)). Second, Gibbs sampling defines a sequence of random draws that converges to a sequence of topic assignments that follows the joint distribution of the topic model (Griffiths and Steyvers (2004)). In the software tool, we implemented both approaches.

3 Temporal Topic Models

While the standard topic models group only words and documents in semantically related topics, we are further interested in the distribution of the topics over time. Certain meanings of words, for example, might be used only in certain time periods: The word *cloud* for instance has recently become a new meaning of a "data cloud". Further, there can be certain trends or attentions to topics. Topics about US presidents for example will very likely be highly present around a year of elections.

In order to extract the distribution of topics over time, we use topic models that consider temporal information about the documents. Each document has a time stamp τ. We assume that each word in the documents is associated with this time stamp. The time stamps follow the

distribution $p(\tau|m)$ for meta parameters m and are assumed to be conditionally independent given a topic. Hence, we model the time stamps as additional observed random variables that depend on the topics.

A specific instance of a probability distributions of the time stamps is the Beta distribution. Wang and McCallum (2006) introduced this model to investigate topics over time. They call this method Topics over Time (TOT). The generative process of standard LDA is extended such that for each word w_i in each document, we also draw a time stamp $\tau_i \sim \text{Beta}(m_{t_i})$ with $m_{t_i} = (a, b)$ the shape parameters of the Beta distribution for topic t_i.

In the software tool, we provide implementations for several distributions. The parameters of the distributions are additionally estimated by Maximum Likelihood Estimation using Newton-like gradient descent with a standard BFGS optimization solver, see Liu and Nocedal (1989).

4 Evaluation

We implemented several standard evaluation methods for topic models. Beside coherence measures that estimate the quality of a topic based on external knowledge of word correlations, we also provide methods to estimate likelihoods of test document collections. To qualitatively evaluate topics, we provide statistics to visualize the results of a topic model.

4.1 Coherence Measures

Frequently used quantitative evaluation methods are based on the relations of the highest ranked words in each topic. The coherence measures estimate how well the model fits an as coherent expected outcome. The definition of this expected coherent outcome is usually based on user studies and experience with topic modeling in practice. A fundamental assumption for topics or factors to be coherent is based on the top ranked words. Each topic is associated with a value how present this topic is for given words. This value can be directly read of from the multinomial distribution β_t. Ranking the words for each topic results in a compact representation of the each topic.

For T latent variables with corresponding top-k words in ranking lists $V_t = \{w_{1t}, \cdots, w_{kt}\}$ with respect to each latent variable that is extracted by a latent variable model, the overall coherence measure is the mean over individual coherence values $U(V_t)$:

$$U(V) = \frac{1}{T} \sum_{t=1}^{T} U(V_t).$$

To estimate the individual values for a given latent variable model, we use several coherence measures that have been proposed in the literature. All measures use statistics of co-occurring words from an additionally given reference document collections like Wikipedia articles. For a detailed description of the quality measures see Röder et al. (2015). In the next subsections, we describe the coherence measures mostly used in literature for topic models.

4.1.1 UMass

In Mimno et al. (2011), the authors propose a topic coherence measure that depends on co-occurrences of words. Based on user studies, they show that this measure corresponds well with the top ranked topics by the users. In the literature the measure is called the U_{Mass} measure and is defined as

$$U_{Mass}(V_t) = \sum_{m=2}^{k} \sum_{l=1}^{m} \log \frac{D(w_{mt}, w_{lt}) + 1}{D(w_{lt})}. \tag{1}$$

The measure is the sum of the log-ratios of the by 1-smoothed co-occurrence frequency of any two ordered words in the top ranked list, $D(w_{mt}, w_{lt})$, and the document frequency of the lower ranked word, $D(w_{lt})$.

4.1.2 Pointwise Mutual Information

The authors in Newman et al. (2010) introduce Pointwise Mututal Information (PMI) as measure for topic coherence. The PMI is the log-ratio of the joint probability of two random variables and the product of their marginal probabilities. It measures how likely two random variables are jointly distributed and not independently distributed. The PMI of two words w_1 and w_2 is defined as the following:

$$PMI(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}.$$

The PMI can be interpreted as how much likely the two words w_1 and w_2 appear together in contrast to how likely they appear alone.

For a latent topic, respectively factor t and the top-k ranked words V_t, the PMI is defined as:

$$PMI(V_t) = \frac{1}{(k-1)k/2} \sum_{m<n}^{k} \log \frac{p(w_{mt}, w_{nt})}{p(w_{mt})p(w_{nt})}. \tag{2}$$

In Aletras and Stevenson (2013) propose to use the Normalized Pointwise Mututal Information (NPMI) to estimate the coherence of the topics. The NPMI is the PMI divided by the negative log probability of the two words appearing together. The reason to use NPMI is twofold. First, NPMI is normalized between -1 and 1. Second, low frequencies of the words are less critical. Especially the second reason is important, since small outliers can result in very small joint probabilities that overtake the whole coherence measure. Formally the NPMI is defined as:

$$NPMI(V_t) = \frac{1}{(k-1)k/2} \sum_{m<n} \frac{\log \frac{p(w_{mt}, w_{nt})}{p(w_{mt})p(w_{nt})}}{-\log p(w_{mt}, w_{nt})}. \tag{3}$$

In our software tool, we provide these coherence measures via an interface to the library Palmetto (Röder et al. (2015)). The estimations of the frequencies and probabilities are all based on word indices from Wikipedia corpora (German and English). To generate these indices, the

Wikipedia corpora must be retrieved (for example from the Institute of the German Language[3]) and a Lucene-based index must created as explained here: `https://github.com/AKSW/Palmetto/wiki/How-to-create-a-new-index`.

4.1.3 Temporal Coherence for Temporal Topic Models

Similar to the coherence of the top ranked words, we estimate the temporal coherence as distance of the distribution of the time stamps associated with a latent topic, with the distribution of the time stamps for the top words over all documents in the corpus. We assume that the documents containing the top words from latent variables approximate the content of the underlying concept. The temporal difference of the time stamps of these documents indicates how well this latent information captures the true temporal dynamics in the corpus. A topic is temporal coherent if the estimated distribution of the time stamps in this topic is similar to the temporal distribution of the time stamps for the top words in the whole corpus. The documents that contain the top two words approximate the semantics behind the topic. Hence, documents containing the top two words in the corpus can be used as coherence reference.

The empirical distributions of the time stamps of the topics and the top words in the corpus are estimated by histograms h_t and h_{w_1,w_2}. For a topic t, the empirical probability of the time between two time stamps τ_1 and τ_2 can be approximated by

$$P([\tau_1, \tau_2]|t) = p(\tau_2|t) - p(\tau_1|t) \propto \sum_{\tau} I_{\tau_1 \leq \tau < \tau_2}(\tau) n_{\tau,t}.$$

For $n_{\tau,t}$ the number of tokens assigned to topic t from a document with time stamp τ and the indicator function

$$I_{\tau_1 \leq \tau < \tau_2}(\tau) = \begin{cases} 1, & \tau \in [\tau_1, \tau_2] \\ 0, & else \end{cases} .$$

Now, we define the histogram of the temporal distribution of topic t as function $h_t : \mathbb{N} \to \mathbb{N}$ such that

$$h_t(\tau_1, \tau_2) = \sum_{\tau} I_{\tau_i \leq \tau < \tau_{i+1}}(\tau) n_{\tau,t},$$

for a given number of intervals $[\tau_1, \tau_2], \cdots, [\tau_{e-1}, \tau_e]$.

For two words w_1 and w_2 for topic t, the empirical probability can be approximated by

$$P([\tau_1, \tau_2]|w_1, w_2) = p(\tau_2|w_1, w_2) - p(\tau_1|w_1, w_2) \propto \sum_{\tau} I_{\tau_1 \leq \tau < \tau_2}(\tau) n_{w_1,w_2,\tau}$$

for $n_{w_1,w_2,\tau}$ the number of tokens in the documents that contain both words w_1 and w_2 in the corpus with time stamp τ. The histogram of the temporal distribution of the words w_1 and w_2 in

[3] `http://www1.ids-mannheim.de/kl/projekte/korpora/verfuegbarkeit.html`

the corpus is the function $h_{w_1,w_2} : \mathbb{N} \to \mathbb{N}$ such that

$$h_{w_1,w_2}(\tau_1, \tau_2) = \sum_{\tau} I_{\tau_1 \leq \tau < \tau_2}(\tau) n_{w_1,w_2,\tau}.$$

There are several distance measures possible. We propose to use the Minkowski distance to estimate how much the distributions over the time stamps differ based on histograms. The Minkowski distance of two histograms for topic t and the corresponding top two words w_{1t}, w_{2t} is defined as

$$D(h_t, h_{w_{1_t},w_{2_t}}, p) = \sqrt[p]{\sum_i |h_t(\tau_i, \tau_{i+1}) - h_{w_{1_t},w_{2_t}}(\tau_i, \tau_{i+1})|^p}.$$

Using $p = 2$ is the Euclidean distances and $p = 1$ is the l_1 distance.

4.2 Likelihood

The coherence measures estimate the quality of latent topics based on statistics from different document collections and user information. To estimate how good a factor or topic model fits the corpus we estimate the likelihood of the data under this model. Depending on the specific model, we can directly estimate the likelihood or we need special assumptions. For topic models, the likelihood of a set of test documents in corpus C_{te} given a topic model by its parameters is

$$p(C_{te}|\alpha, \beta) = \prod_{d \in C_{te}} p(\mathbf{d}|\alpha, \beta).$$

4.2.1 Sequential Monte Carlo

As proposed by Wallach in Wallach et al. (2009), Sequential Monte Carlo Method can be used to estimate the likelihood of a topic model. For a sequence of words from a hold-out test data set, the probability of the test words w is

$$p(\mathbf{d}|\alpha, \beta) = \prod_m p(w_m|\mathbf{d}_{<m}, \alpha, \beta).$$

A Sequential Monte Carlo algorithms to estimate the likelihood of a held-out data set for a given topic model can be defined in the following way: Given a new document d as sequence of tokens, $\mathbf{d} = (w_1, \cdots, w_N)$, we re-sample topic proportions for each token w_m in $\mathbf{d}$, given all tokens before, $\mathbf{d}_{<m} = (w_1, \cdots, w_{m-1})$, using the point estimate of the topic-word distributions. To compensate the uncertainty in these estimates for a single document, we keep M independent samples. These samples are called particles. For the m_{th} word in the sequence, the probability is

$$p(w_m|\mathbf{d}_{<m}, \alpha, \beta) = \sum_{i=1}^{T} p(t_i|\theta) p(w_m|\mathbf{d}_{<m}, t_i, \beta) \tag{4}$$

$$= \sum_{i=1}^{T} \frac{n_{d,i,<m} + \alpha_k}{n_i + \sum_{k'} \alpha_{k'}} \beta_{t_i, w_m}. \tag{5}$$

This is a mixture of multinomial Distribution with Dirichlet prior $\mathrm{Dir}(\eta)$, with mixing weights $p(t_i|\theta)$ for Dirichlet ($\mathrm{Dir}(\alpha)$) distributed $p(t|\theta)$.

We apply Sequential Monte Carlo Methods using particle learning (PL) methods as proposed by Scott and Baldridge (2013) and by Naesseth et al. (2014). To get an estimate for the topic weights, we use aggregated counts of topic assignments for topics i: n_i, respectively for the document d: $n_{d,i}$. For $m = 1, \cdots, M$, we use aggregated counts $n_{d,i,<m}$, with count assignments for all tokens up to the m_{th}, sampled iteratively from

$$p(t = i|w_m, t_{<m}) \propto \frac{\alpha_k}{\sum_{k'} \alpha_{k'}} \beta_{m,i}$$

and collected as particles. We re-sample for topic proportions for the documents, but use the point estimate for the word distribution in each topic. Then, we sample for each particle and its corresponding aggregated counts, topic assignments and add them to these counts. This means, we have Z estimates of the aggregated counts and consequently can estimate Z times $p(w_m)$. This models the uncertainty about the assignment by Z particles.

We define particles $T_{m,z} \sim p(t|w_1, \cdots, w_m, \tau_d, \beta)$ for $z = 1, \cdots, Z$. The $T_{m,z}$ are iteratively sampled such that $T_{N,z} \sim p(t|\mathbf{d}, \tau_d, \beta)$.

For temporal topic models by additional random variables that depend on the latent topics, we can easily extend to Sequential Monte Carlo method from above to estimate the likelihood of hold-out documents with additional time stamps:

$$p(w_m, \tau_d|\mathbf{d}_{<m}, \alpha, \beta) = \sum_{i=1}^{T} p(t_i|\theta) p(w_m, \tau_d|\mathbf{d}_{<m}, t_i, \beta) \tag{6}$$

$$= \sum_{i=1}^{T} \frac{n_{d,i,<m} + \alpha_k}{n_i + \sum_{k'} \alpha_{k'}} \beta_{t_i, w_m} p(\tau_d|t_i). \tag{7}$$

This is a mixture of multinomial distributions with Dirichlet prior $\mathrm{Dir}(\eta)$, with mixing weights $p(t_i|\theta)p(\tau_d|t_i)$ for Dirichlet ($\mathrm{Dir}(\alpha)$) distributed $p(t|\theta)$ and Shifted-Gompertz distributed $p(\tau|t)$. This density is analogue to Equation 4 using additional time stamps. The difference lies in the integration of the temporal distributions. This is easy due to the independence assumption in temporal topic modeling.

Besides the joint likelihood of the words and the time stamps, we are also interested in the conditional likelihood. The conditional likelihood is the likelihood of the sequence of words in a test document given the time stamp: $p(\mathbf{d}|\tau_d)$. This conditional likelihood estimates the likelihood of words from the documents at the time of the document. This measure focuses on the quality of the estimated word distribution. Due to the independence assumption in the topic models, we have the following conditional probability of a sequence of words in a document given the corresponding time stamp:

$$p(\mathbf{d}|\tau_d) = \prod_n p(w_n|\tau_d).$$

The partial conditional probabilities can be calculated via

$$p(w_n|\tau_d) = \frac{p(w_n, \tau_d)}{p(\tau_d)}.$$

The joint probability $p(w_n, \tau_d)$ is estimated as in Equations 5 and the probability of the time stamp τ_d is

$$p(\tau_d) = \sum_t p(\tau_d|t).$$

4.3 Qualitative Evaluation

In practice, the topic models are used qualitatively. Experts interpret results of the models by exploring the topics. For linguistic tasks for example, the results of latent topic modeling are mostly manually investigated. For example, if we are interested in usage patterns of expression and words in context and over time, we need methods to manually evaluate the results of a latent topic model in terms of the words and documents. Rather, than abstract numbers that describe the results, we are interested in how explanatory the topics are. A good format and a visualisation of the results is needed to help evaluating the models by linguists. There are several possible ways to visualize the results of topic models. In the literature there are usually the following aspects considered: First, how can we show the tendency of words and documents to certain topics. Second, how can we show the distribution of the topics over the words and documents. Finally, how can we show the distribution of topics, words and documents over time. The latter is important for diachronic linguistics.

4.3.1 Word Clouds

A concrete visualization of the words with respect to their importance is a World Cloud. Each of the top-k words from the ranking list is written in a figure with size proportional to $\beta_{t,w}$. On the left in Figure 2, we see a Word Cloud from a topic about US president Obama. Besides Word Clouds, we can list the top-k words in decreasing order of importances (in the concrete values of $\beta_{t,w}$) and additionally plot these values as histogram. This can be seen on the right in Figure 2. In the software tool, we provide the top ranked words from a topic model as CSV[4] result file.

[4] A CSV file contains table like data as comma separated values.

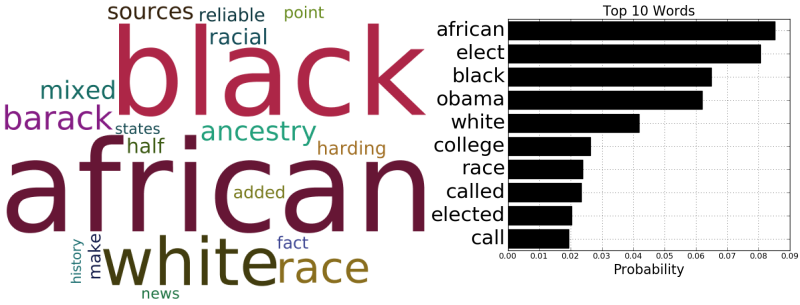


Figure 2: Visualization of the most important words for a given topic. Left: a Word Cloud from the highest ranked words from a topic model on Wikipedia talk pages containing the word “president”; right: a sorted list of the highest ranked words.

4.3.2 Temporal Distribution

Ranking lists and Word Clouds can visualize the word distributions for different topics. For the temporal distributions of the documents with respect to the topics, we need to display the course of the importance of the latent topic over time. The amount of a certain topic in a given time can be estimated by grouping documents by time and averaging the document-topic proportions $n_{d,i} \propto \theta$. The document-topic proportion tells how much present a certain topic is in a document.

Each document d has its time stamp τ_d. Grouping these values into n intervals:

$$[0, \tau_1], [\tau_1, \tau_2], \cdots, [\tau_{n-1}, \tau_n],$$

we assign the documents to the corresponding intervals, hence $d \rightarrow [\tau_i, \tau_{i+1}]$ with $\tau_i \leq \tau_d \leq \tau_{i+1}$. Now, we can average $n_{d,i}$ in each interval to get a histogram of topic i over time. Additionally, we can plot the estimated temporal distribution. In Figure 3, we show the density of an estimated distribution of time stamps together with the corresponding histogram of time stamps for a topic from a topic model trained on Wikipedia talk pages about presidents. Similar to the top words, we additionally protocol the time stamps associated with the topics from a temporal topic model and the estimated parameters for the temporal distributions.

5 Software

This section describes the plugin “Corpus Linguistic Plugin” as extension for the Data Mining tool RapidMiner. We explain in detail the software and how it can be used in diachronic linguistics. The software is already used in corpus linguistic research and teaching at the TU Dortmund University and the Mannheim University. We start the software description with an introduction to Data Mining tool RapidMiner. RapidMiner is used and extended for corpus linguistic tasks in our software tool. The plugin can be downloaded from: `http://sfb876.tu-dortmund.`

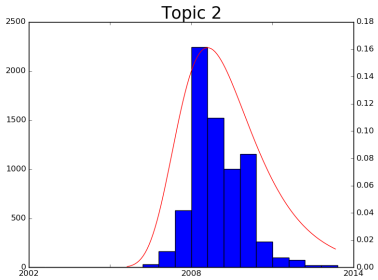


Figure 3: Visualization of the distribution of topics over time as plot of a histogram of topic proportions for a given topic over time from a topic model about presidents. Additionally, we plot the density of the time stamps fitting the corresponding time stamps for this topic.

`de/auto?self=Software` under the link: "Corpus Linguistics Plugin" or directly at `http://sfb876.tu-dortmund.de/auto?self=%24es4hb8h0cg`.

5.1 RapidMiner

RapidMiner, as for instance described by Hofmann and Klinkenberg (2013), is a Data Mining toolbox used to perform data analysis on different data sources. RapidMiner offers the classical analysis and Data Mining steps from data retrieval to data transformation and pre-processing, performance of analysis and Data Mining methods to evaluation methods, post-processing and visualization. Individual processing steps are performed by so called **Operators**. The standard operators are separated into several categories and are organized in an ontology represented as folder structure in the operator explorer view on the left of the main screen as seen in Figure 4. The main categories of operators are:

- import/export operators: reading and writing of data
- data transformation operators: pre- and post-processing of data
- modeling: analytic and Data Mining methods on data
- evaluation operators: quality estimation of the modeling results.

The operators are compiled to a sequence of steps summarized in a so called **Process**. This process defines a flow of input data to processing operators that output result data. In the middle Figure 4, an example process is shown with the execution order of the individual operators. Starting with reading data as CSV-file, the data is pre-processed by transforming nominal to numeric data. The modeling operator **SVM** builds a classification model that is applied on test data which is additional read in. Finally, the **Performance** operator is used to evaluate the model by standard measures. The operators have a number of parameters to be specified. On the right Figure 4, the **Parameters** panel is shown as input mask for all parameters. Clicking on an operator, this

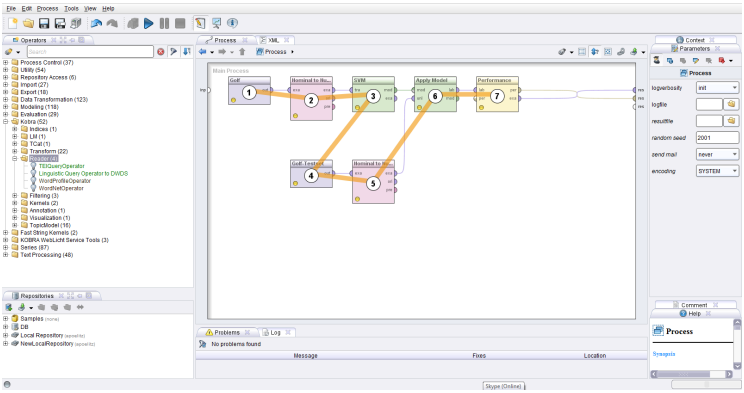


Figure 4: RapidMiner user interface. Left: Operators for data loading, pre-processing, Data Mining methods, post-processing and data export. In the middle: Data Mining process. Right: Properties and parameters of the process and the operators.

panel shows the parameters that need to be set for this operator. Additional, a description of each operator can be found on the **Help** panel. A general introduction for Data Mining with RapidMiner can be found in the book by North (2012).

5.2 Corpus Linguistics Plugin

RapidMiner offers a convenient interface and a plethora of available analyses methods. Compared to low level interfaces and libraries for different programming languages, RapidMiner offers a more user friendly tool box. This makes the introduction of our methods more easy for linguistic researchers - even with little knowledge in computer science. We implemented different versions of topic models and evaluation methods as a plugin for the RapidMiner. For the different topic models, different operators are available. Besides standard LDA with Gibbs sampling and Variational Inference, supervised versions with Gaussian, Beta, Uniform and Gompertz distributed document labels can be used for diachronic linguistics. Additionally, an implementation of topic models with word features and word groups via special Laplace and Group-Sparsity inducing priors is available to integrate word informations.

To access different corpora, operators to execute linguistic queries on corpora at the Berlin Brandenburger Academia of Science are available. Besides the standard corpora, we also provide access to dictionaries and GermaNet (the German version of WordNet). To access Wikipedia corpora, a TEI-reader is implemented that extends a standard XML-stream reader to process Text Encoding Initiative (TEI) tags, see Beißwenger et al. (2012). Finally, pre-processing and post-processing operators provide methods for text transformations and text visualization. In the next subsections, concrete examples for the use of the plugin are described.

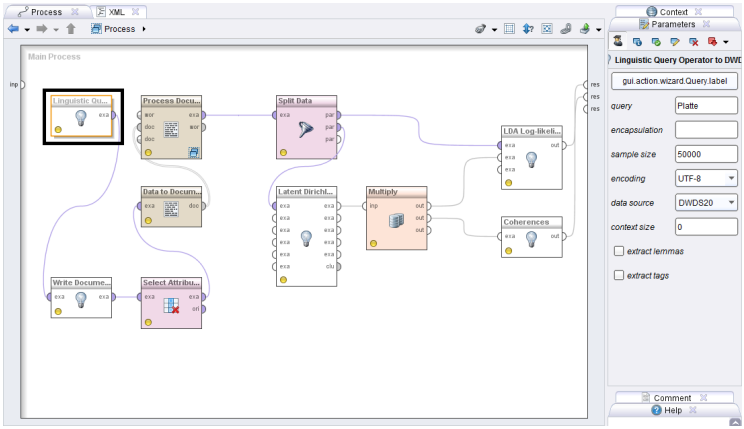


Figure 5: Linguistic Query Operator as first step in a process to perform a linguistic task by LDA. For a given query, we retrieve KWIC-lists from a corpus.

5.2.1 Interface to Linguistic Resources

The first step to perform linguistic tasks with the Corpus Linguistic Plugin is the retrieval of the data. KWIC-lists or documents are extracted and internally represented as strings. Standard text documents can be opened by the **Read CSV** operator from RapidMiner. For linguistic corpora, we implemented the **Linguistic Query Operator** as shown in Figure 5. Via the parameter **data source**, the Linguistic Query Operator provides access to the DWDS Core Corpus of the German text archive and the Die Zeit corpus of news articles from 1947 to 2014. For a given linguistic query, the operator retrieves a number of snippets and generates an example set that contains the texts, time stamps and additional information about authors and sources. The query is sent to a server at the Berlin Brandenburger Academia of Science and a Perl script runs the query against a data base containing the corpora, see Sokirko (2003). The KWIC-lists are returned as JSON[5] files and the operator parses the information and generates a result set for further processing. Additionally, the position of the query match in each retrieved snippet is given to efficiently identify the match. In Figure 7, we show the resulting example set from the Linguistic Query Operator.

For corpora and documents in TEI format, the **TEI Query Operator** provides a stream reader to process large files. Since these files are not indexed as the corpora from the Dictionary of the German Language, we cannot pose linguistic queries. Instead, standard regular expressions can be queried. For the TEI formatted corpora, as the Wikipedia articles and talk pages from the Institute of the German Language, the operator retrieves matches of the regular expressions on sentence, paragraph or postings level. These levels are semi-automatic annotated (see Margaretha and Lüngen (2014)). The operator itself implements an XML based stream reader to iterate over

[5] http://www.json.org/

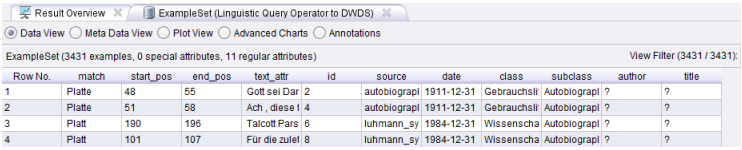


Figure 6: Result example set from Linguistic Query Operator for a linguistic query. For each match of the query, we have an example with information about the match.

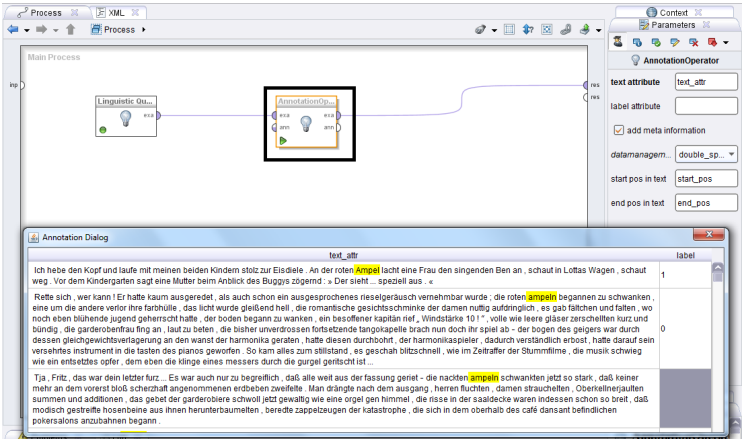


Figure 7: Annotation Operator and annotation environment. Given an example set from the TEI or Linguistic Query Operator, the snippets are visualized and the match is highlighted to inspect the results. Additional annotation like labels can be added.

the elements from the TEI file. The resulting example set has the same schema as the example set from the Linguistic Query Operator.

To efficiently inspect the retrieved KWIC-lists, the **Annotation Operator** visualizes the text snippets and highlights the matches in the texts. We can also add additional labels or attributes to the texts to further annotated them. The operator generates a result as example set containing the texts of the snippets and the additional annotations.

For the retrieval of information from the additional language resources like dictionaries and WordNets, we implemented operators that can extract these information from local files (WordNet for instance) and retrieve them from the Dictionary of the German Language. The **WordNet Operator** takes a word as parameter and extracts similar words from an existing WordNet instance, given the path to the index. The **GermaNet Operator** works the same, but uses the GermaNet source provided by the Seminar für Sprachwissenschaften at University Tübingen. The retrieval of the information is done by a web service at the Berlin Brandenburg Academia of Science via JSON files. The resulting example set contains for a word of interest given as parameter, the

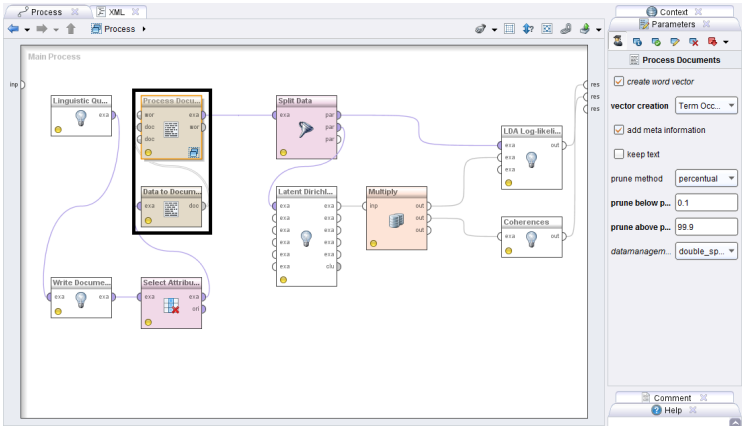


Figure 8: Generation of Word-Vectors from example sets with a text attribute.

hyponyms and hyperonyms with additional examples and descriptions. Further, the **WordProfiles Operator** retrieves the word profiles provided by the Dictionary of the German Language. These profiles contain words that co-occur with a given word of interest and gives information about the relation between them, see Didakowski and Geyken (2013).

5.2.2 Text Processing

Before we can use the KWIC-lists for latent topic models, we need to generate Word-Vectors[6]. With the **Text Processing Plugin** as provided by RapidMiner, text (general strings) can be transformed into a Bag-of-Words and Word-Vectors. Given text in an example set, an internal data structure to represent the text is a generated. This data structure is called a **Document**. These Documents are further transformed into Word-Vectors containing word occurrences and possible weights as TF-IDF. The text processing plugin offers additional methods to tokenize texts. Filtering operators can be used to filter out stop words, large tokens or tokens with no characters. Additional pruning mechanisms can be used to filter out words that appear in too many or too few documents. In Figure 8, we show how the KWIC-lists are transformed into Documents by the **Data to Documents Operator** and how we further generate Word-Vectors by the **Process Documents Operator**.

5.2.3 Latent Topic Models

Using the Bag-of-Words representation with word occurrences for each document from corpus, we can extraction of topics by different topic models. The **Latent Dirichlet Allocation** operator, for instance, takes the texts as Word-Vector with pure word occurrences and extracts latent topics by Gibbs sampling. In addition, an operator that performs Variational Inference for LDA is also

[6] A Word-Vector represents a text as vector.

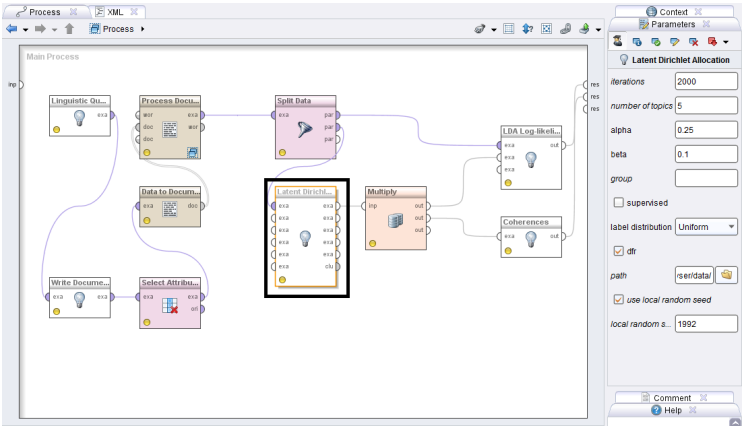


Figure 9: Latent Dirichlet Allocation operator to extract latent topics from a document collection given as Bag-of-Words.

available. For the Gibbs sampler, we need to specify how many iterations we want to process. Further, the number of topics to be extracted and the meta parameters of the Dirichlet priors need to by specified. For standard LDA, we un-check the supervised check box. For temporal topic models, we check the supervised check box and specify the label distribution (Uniform, Beta, Gompertz). If we use the supervised version of LDA, the input example set must contain a label attribute additional to the Word-Vectors - the time stamps. For visualization of the results we check the **dfr** check box and specify the path to the data folder for the DFR-Browser. Figure 9 shows the Latent Dirichlet Allocation Operator in a process. The resulting example sets of this operator consist of the word-topic-distributions, the document-topic distributions and possible the estimated parameters for a label distribution.

5.2.4 Evaluation

Different methods to evaluate latent topic models can be used via the **Coherence** operator and the **LDA Log-likelihood** operator. The Coherence operator takes as input an example set containing word probabilities for topics as a result from the Latent Dirichlet Allocation operator. We leverage the Palmetto Toolbox, see Röder et al. (2015), to estimate the different coherence measures based on the top words. From the word probabilities the most likeliest words (the number is given by a parameter) are used for the coherences. To use this operator we need a Lucene-based index from a large text collection that is used as a reference corpus. We use the Wikipedia articles to generate such an index that contains coherence values using co-occurrences and relative frequencies. We calculated such indices from German and English using the Palmetto library and the Wikipedia corpora from the Institute of the German Language. The LDA Log-likelihood Operator needs no additional resources. We calculate the likelihoods of a test set of documents by Sequential

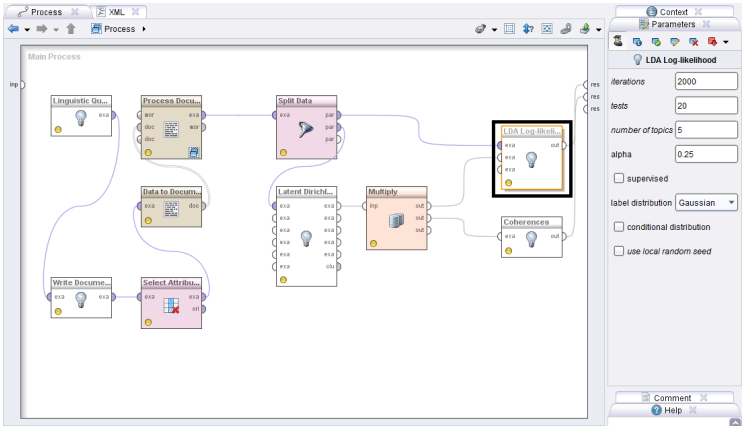


Figure 10: Log-likelihood operator to calculate the likelihood of a test document collection based on Sequential Monte Carlo sampling.

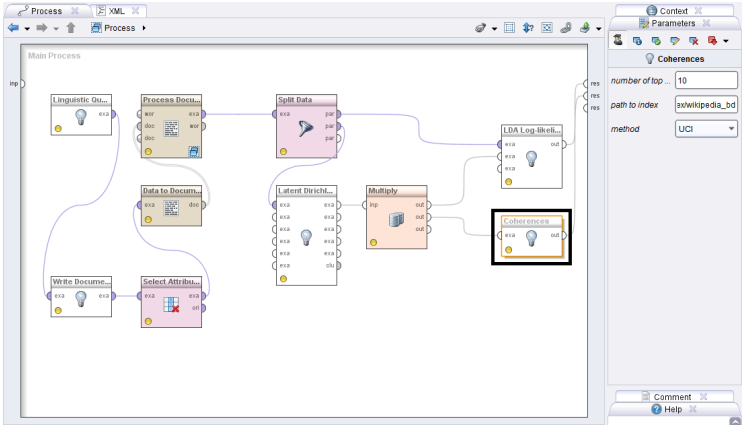
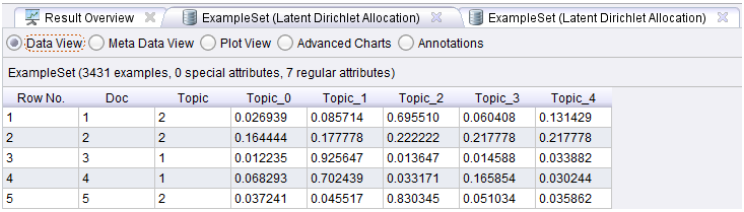


Figure 11: Coherence operator to estimate standard coherence measures using the Palmetto library.

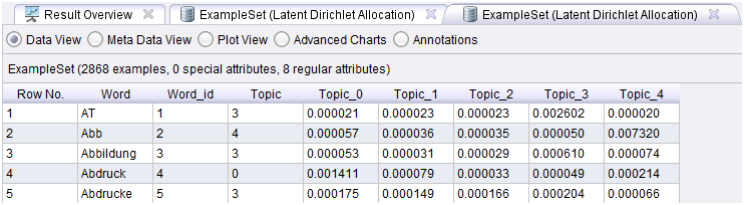

Result Overview | ExampleSet (Latent Dirichlet Allocation) | ExampleSet (Latent Dirichlet Allocation)

Data View | Meta Data View | Plot View | Advanced Charts | Annotations

ExampleSet (3431 examples, 0 special attributes, 7 regular attributes)

Row No.	Doc	Topic	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4
1	1	2	0.026939	0.085714	0.695510	0.060408	0.131429
2	2	2	0.164444	0.177778	0.222222	0.217778	0.217778
3	3	1	0.012235	0.925647	0.013647	0.014588	0.033882
4	4	1	0.068293	0.702439	0.033171	0.165854	0.030244
5	5	2	0.037241	0.045517	0.830345	0.051034	0.035862

Figure 12: Document-topic distribution: For each document, one example contains the document number, the distribution over the topics and the most likeliest topic (Topic).


Result Overview | ExampleSet (Latent Dirichlet Allocation) | ExampleSet (Latent Dirichlet Allocation)

Data View | Meta Data View | Plot View | Advanced Charts | Annotations

ExampleSet (2868 examples, 0 special attributes, 8 regular attributes)

Row No.	Word	Word_id	Topic	Topic_0	Topic_1	Topic_2	Topic_3	Topic_4
1	AT	1	3	0.000021	0.000023	0.000023	0.002602	0.000020
2	Abb	2	4	0.000057	0.000036	0.000035	0.000050	0.007320
3	Abbildung	3	3	0.000053	0.000031	0.000029	0.000610	0.000074
4	Abdruck	4	0	0.001411	0.000079	0.000033	0.000049	0.000214
5	Abdrucke	5	3	0.000175	0.000149	0.000166	0.000204	0.000066

Figure 13: Topic-word distribution: For each word, one example contains the word, the word id, the distribution over the topics and the most likeliest topic (Topic).

Monte Carlo methods. As input, the LDA Log-likelihood Operator takes a set of test documents as Word-Vectors with occurrence data in an example set and the word-topic distributions resulting from the LDA operator. The number of iterations specifies the number of Monte Carlo Samples for the estimation of the likelihood. To reduce variance in the likelihood estimation, a number of independent tests are performed. The result is an example set that contains for each test the log-likelihood.

5.2.5 Results

The results from the latent topic models can be used either in tabular form, in example set form or in special formats for visualization. Using the results as it is given from any topic modeling operator, we get two example sets containing the topic-word distributions and document-topic distribution. As additional attribute we report the most likeliest topic for each word and each document in the example sets. In the Figures 12 and 13, the example sets from the results of the LDA operator are shown as they are internally represented in RapidMiner.

Additionally, we implemented export methods for the results from the topic models and the corpora for visualization by the DFR-Browser from Andrew Goldstone[7]. To process the documents from the corpus for information extraction needed for the visualization, we implemented the **Write Document Reference** operator. As shown in Figure 14, from an example set containing texts with

[7] `https://github.com/agoldst/dfr-browser`

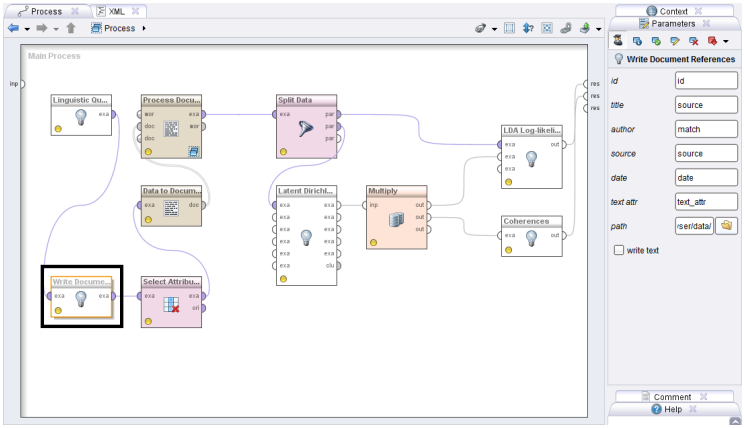


Figure 14: Write Document Reference operator: Writes formatted references from the document collection for visualization by the DFR-Browser.

information about a title, author, publication date and source as attribute information are saved locally where the visualization tool DFR-Browser finds them. The DFR-Browser can be started as a web server and the visualization can be seen in web browser like Firefox.

5.3 Diachronic Linguistics Process

To perform diachronic linguistic tasks, we use the Latent Dirichlet Allocation operator. From the linguistic corpora, we take the information about publication date to extract labels for each document. The attribute **date** from the resulting example set from the TEI or Linguistic Query Operator contains the time information as string. First, we need to convert this into a numerical value by the operators **Nominal to Date** and **Date to Numeric**. Here, the concrete date format (for example "yyyy-MM-dd") must be given and the we need to specify the time unit into which we transform the date to numeric (for example years since 1900). This is illustrated on the left in Figure 15 (most important operators are framed). After the extraction of the information for visualization by the **Write Document References** operator, we select the text attribute **text_attr** and the date attribute by the **Select Attribute** operator and process the text to Word-Vectors by the text processing operator. Before the date attribute can be used for temporal topic modeling, we need to assign it to the **label** role via the **Set Role** operator. Now, the documents can be used together with the date attribute to extract topics and to estimate label distributions. For temporal topic modeling, we need to check the **supervised** check box in the Latent Dirichlet Allocation operator. We also need to specify by which distribution the labels shall be modeled. For time stamps we can use the Beta, the Uniform and the Gompertz distribution. As result, we get besides the topic-word distributions and the document-topic distributions also the parameters

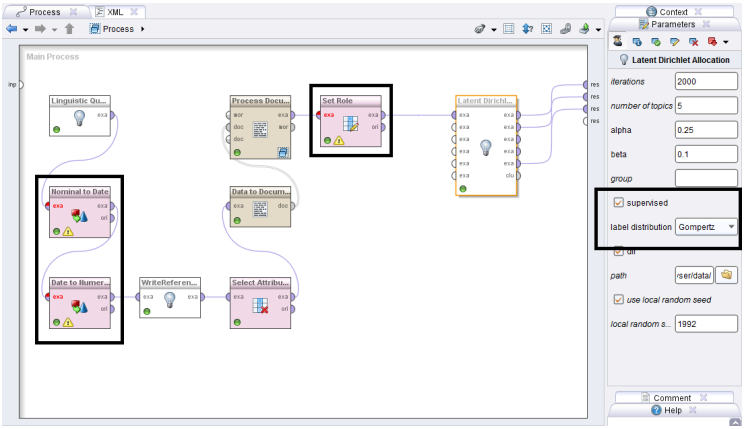


Figure 15: Process for Diachronic Linguistics: From a linguistic data source, we retrieve a KWIC-list with time information. The date is used as numeric label in temporal topic modeling.

that are estimated by Maximum Likelihood Estimation (MLE) during the topic modeling for the corresponding label distribution.

References

Aletras, N. and Stevenson, M. (2013). Evaluating topic coherence using distributional semantics. In *Proceedings of the 10th International Conference on Computational Semantics*, volume 10 of *IWCS '13*, pages 13–22, New York, NY, USA. ACM.

Beißwenger, M., Ermakova, M., Geyken, A., Lemnitzer, L., and Storrer, A. (2012). A TEI schema for the representation of computer-mediated communication. *Journal of the Text Encoding Initiative [Online]*, (3).

Blei, D., Ng, A., and Jordan, M. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Didakowski, J. and Geyken, A. (2013). From dwds corpora to a german word profile – methodological problems and solutions. In *Network Strategies, Access Structures and Automatic Extraction of Lexicographical Information*, pages 43–52, Mannheim, GE. Mannheim: Institut für Deutsche Sprache. (OPAL - Online publizierte Arbeiten zur Linguistik X/2012).

Geyken, A. (2007). The dwds corpus: A reference corpus for the german language of the 20th century. In *Fellbaum, Christiane (Hg.): Collocations and Idioms: Linguistic, lexicographic, and computational aspects*, pages 23–41, London, UK. Continuum International Publishing Group.

Griffiths, T. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl. 1):5228–5235.

Hofmann, M. and Klinkenberg, R. (2013). *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman & Hall/CRC, London, UK.

Liu, D. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(3):503–528.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Margaretha, E. and Lüngen, H. (2014). Building linguistic corpora from wikipedia articles and discussions. *Journal of Language Technology and Computational Linguistics*, 29(2):59–82.

McLachlan, G. and Basford, K. (1988). *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker, New York.

Mimno, D., Wallach, H., Talley, E., Leenders, M., and McCallum, A. (2011). Optimizing semantic coherence in topic models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, volume 11 of *EMNLP '11*, pages 262–272, Stroudsburg, PA, USA. ACL.

Naesseth, C. A., Lindsten, F., and Schön, T. (2014). Sequential monte carlo for graphical models. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26 of *NIPS '13*, pages 1862–1870, Red Hook, NY, USA. Curran Associates, Inc.

Newman, D., Lau, J. H., Grieser, K., and Baldwin, T. (2010). Automatic evaluation of topic coherence. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 100–108, Stroudsburg, PA, USA. Association for Computational Linguistics.

North, M. (2012). *Data mining for the masses*. Global Text Project.

Röder, M., Both, A., and Hinneburg, A. (2015). Exploring the space of topic coherence measures. In *Proceedings of the eight International Conference on Web Search and Data Mining, Shanghai, February 2-6*.

Scott, J. and Baldridge, J. (2013). A recursive estimate for the predictive likelihood in a topic model. In *Artificial Intelligence and Statistics Conference*, volume 31 of *JMLR Proceedings '13*, pages 527–535, Cambridge, MA, USA. MIT Press.

Sokirko, A. (2003). Ddc - a search engine for linguistically annotated corpora. In *Proceedings of Dialogue (published in Russian)*, pages 25–64, Moskow, RU. Nauka.

Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. (2009). Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, volume 24 of *ICML '09*, pages 1105–1112, New York, NY, USA. ACM.

Wang, X. and McCallum, A. (2006). Topics over time: A non-markov continuous-time model of topical trends. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 12 of *KDD '06*, pages 424–433, New York, NY, USA. ACM.

Nils Diewald, Eliza Margaretha

Krill: KorAP search and analysis engine

1 Introduction

KorAP[1] (*Korpusanalyseplattform*) is a corpus search and analysis platform for handling very large corpora with multiple annotation layers, multiple query languages, and complex licensing models (Bański et al., 2013a). It is intended to succeed the COSMAS II system (Bodmer, 1996) in providing DeReKo, the German reference corpus (Kupietz and Lüngen, 2014), hosted by the Institute for the German Language (IDS).[2] The corpus consists of a wide range of texts such as fiction, newspaper articles and scripted speech, annotated on multiple linguistic levels, for instance part-of-speech and syntactic dependency structures. It was reported to contain approximately 30 billion words in September 2016 and still grows continually.

Krill[3] (*Corpus-data Retrieval Index using Lucene for Look-ups*) is a corpus search engine that serves as a search component in KorAP. It is based on Apache Lucene,[4] a popular and well-established information retrieval engine. Lucene's lightweight memory requirements and scalable indexing are suitable for handling large corpora whose size increases rapidly. It supports full-text search for many query types including phrase and wildcard queries, and allows custom implementations to cope with complex linguistic queries.

In this paper, we describe Krill and how its index is designed to handle full-text and complex annotation search combining different annotation layers and sources of very large corpora. The paper is structured as follows. Section 2 describes how a search works in KorAP (starting from receiving a search request until returning the search results). Section 3 explains how corpus data are represented and indexed in Krill. Section 4 describes various kinds of queries handled by Krill and how they are processed for the actual search on the index. The Krill response format containing search results is described in Section 5. We present related and further work in Section 6 and 7 respectively. The paper ends with a summary.

2 Search Flow

The KorAP architecture's design is based on a microservice architecture comprising small independent components that are easy to extend, to replace and to maintain (Diewald et al., 2016). Figure 1 illustrates the KorAP architecture and the interactions

[1] https://korap.ids-mannheim.de/
[2] http://www.ids-mannheim.de/
[3] https://github.com/KorAP/Krill
[4] https://lucene.apache.org/

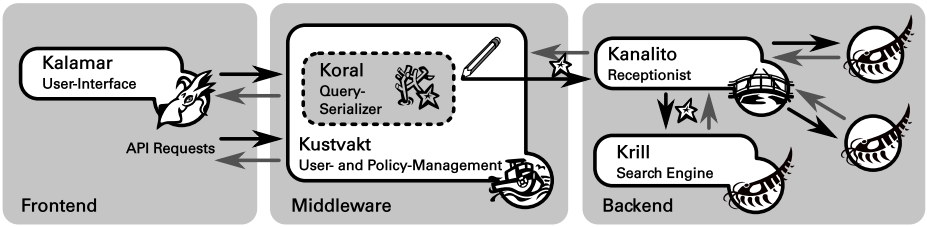


Figure 1: The KorAP Architecture

among its components. In addition to the search engine Krill, the components include a user interface (Kalamar), a query serializer (Koral), a user and resource policy management system (Kustvakt) and a *receptionist service* (cf. Büttcher et al., 2010, ch. 14.1.1) for the search engine to provide parallel search (Kanalito).

A KorAP search process starts by sending a query in a particular query language, either using the Kalamar frontend or a direct API request to Kustvakt. Supported query languages include COSMAS II (Bodmer, 1996), ANNIS (Rosenfeld, 2010), and Poliqarp (a CQP variant; Przepiórkowski et al., 2004).

The query is then serialized by Koral resulting in a generic representation as KoralQuery (Bingel and Diewald, 2015). Kustvakt may relay the KoralQuery to a single Krill instance to conduct a search or to Kanalito for a parallel search of distributed Krill instances. The actual search is performed in Krill, and query results are eventually returned to the API endpoint and may be displayed in Kalamar.

Kustvakt is the API provider of KorAP managing the interaction of all components. One of its primary tasks is to monitor user access to resources, and thus to guarantee the retrieval of resources with respect to their intellectual property rights before commencing a search (Bański et al., 2014). When a user request involves unauthorized resources, Kustvakt may rewrite the corresponding KoralQuery in order to limit the query to only those resources available to that user. Besides, it may inject values from user preferences such as default annotation sources for each annotation layer. Because the central user and license mechanism is done in Kustvakt and not in Krill, redundancy in distributed search and re-implementation in different search component systems can be avoided.

3 Index Representation

Krill, as it is based on Lucene, uses an inverted index[5] to provide full-text search capabilities in a corpus. In contrast to tools like *grep*, this approach does not treat textual data as a sequence of characters, but as a sequence of tokens (most often "words"). Thus, searching for these tokens provides direct access to the occurrence of these tokens in a collection of documents. An inverted index consists of a term dictionary (based on the tokens) with direct access to the associated information on the

occurrence of a token in the corpus, so-called postings lists (see Fig. 4). While most of the time the dictionary is kept in memory for fast access, postings lists are only loaded (partially) in memory, everytime a term is retrieved as part of the query process.

As a search component of KorAP, Krill indexes and provides search on primary data (i.e. corpus text), various layers of annotation data (e.g. lemma, part-of-speech, constituent annotations) and metadata of the resources. The KorAP data model (see Bański et al., 2013b) separates primary data and annotations (i.e. a *standoff* architecture) to allow for multiple, potentially concurring layers of annotations (see Fig. 2).[6] This model is the foundation of the KorAP input format *KorapXML* as well as the foundation of the Krill index design.

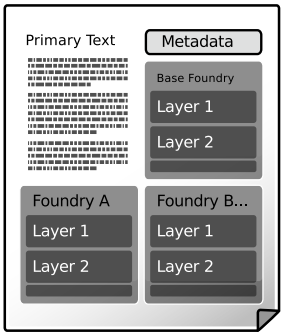


Figure 2: KorAP data model

Metadata provides information on the *document-level* (cf. Zobel and Moffat, 2006), such as author information of a text or the publication date. The data structure of the term dictionary for metadata depends on the stored data type and the expected matching operations. For example, publication dates may need to be stored as numerical data to provide support for range queries (e.g. "Search in all documents published since 1786"), while titles may be stored as full-text searchable data (e.g. "Search in all documents containing the word 'Reise' in the title"), and organizing the term dictionary for string data using hashes may be efficient, but may also prevent access using regular expressions.

Metadata fields in Krill can have the following types:[7]

date	Date field (e.g. publication date)
int	Numerical integer field (e.g. number of tokens)
keyword	Multiple strings (e.g. keyword tags)
store	Retrieve-only value (e.g. associated file reference)
string	Fixed string (e.g. text identifier)
text	Tokenized full-text field (e.g. text title)

Annotation data provides information on the *word-level* of the text and is subdivided into *foundries* denoting the resource of a certain annotation (e.g. the annotation tool used to generate the annotation) that may contain multiple annotation *layers*. In KorAP, a user can search for an annotation in a specific foundry and layer, by adding a prefix to the query term, specifying the requested annotation, for example `[mate/l=sun]` to

[5] For a brief introduction to inverted indices for full-text search, see Zobel and Moffat (2006).

[6] Although Krill supports overlapping annotations, at the moment it is limited to one stream of tokens (i.e. a single tokenization for all layers).

[7] Currently, metadata fields are defined by DEREKO. In the future, Krill will support arbitrary metadata fields of the given types.

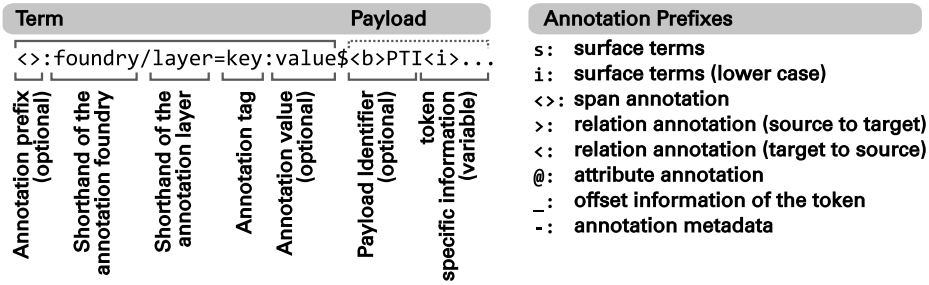


Figure 3: Term structures for annotations in the Krill index

search for the term "sun" in the lemma layer of the *Mate*-foundry using the Poliqarp+ query notation (KorAP's extension to Poliqarp).

To distinguish between annotations in the index, all annotation terms in the term dictionary have foundry and layer prefixes (see Fig. 3, left). Annotations in Krill can have the following types, specified as optional additional type prefixes to the dictionary terms (see Fig. 3, right):

Token Token-level information terms
(e.g. the surface form of a term or the lemma)

Span Span-level information terms
(e.g. sentences or nominal phrases)

Relation Relational information spanning between terms and/or spans
(e.g. dependency relations)

Attribute Annotations to tokens, spans, or relations
(e.g. the `href`-attribute of an HTML anchor element)

Further annotation types are being considered, for example to describe punctuations.[8]

The document identifier and the starting position of the annotation are stored in the postings list, along with additional retrievable information encoded in byte streams (so-called *payloads*),[9] for example the length of a span or the annotation a relation refers to.

Additional information supported by all annotation types includes a leading byte for term type identification, a unique token identifier, and a byte value indicating the level of confidence of the annotation source.

Character offsets of tokens (relevant for matching highlights, see Sec. 5) are stored once per token position. Span terms may store additional character offset information in payloads, to include surrounding non-token characters, for example punctuation and quotation marks.

[8] Krill does not support punctuation search yet. In the future, punctuations will be indexed as attachements to surrounding tokens in order to keep searching for sequences of words simple.

[9] The documentation of payload byte streams is available at `https://github.com/KorAP/Krill/blob/master/misc/payloads.md`.

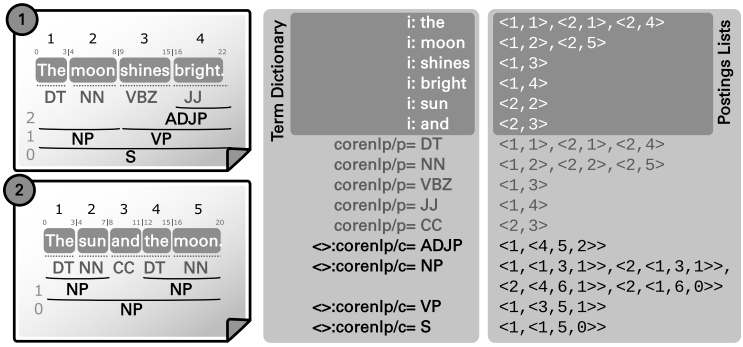


Figure 4: Example index representation of two documents, showing the term dictionary and the postings lists

Figure 4 exemplifies the structure of the Krill index for two documents. Both documents have a sequence of tokens (*token stream*), a token-level annotation and a span-level annotation. The term dictionary contains all annotations including prefixes. The associated postings lists are illustrated as lists of tuples starting with the identifier of the text in which the annotation occurs, followed by occurrence-specific information, that vary according to the annotation type. The surface terms (e.g. `i:moon`) and the token-level annotations (e.g. the part-of-speech annotation `corenlp/p=DT`) store their token position in the token stream. The span-level annotations (e.g. the constituency annotation `<>:corenlp/c=S`) store their start and end positions, and the level in the tree hierarchy, partially encoded in payloads.

An annotation as in Figure 4 would be encoded in the following structure:

```
<>:corenlp/c=NP$<b>64<i>0<i>8<i>2<b>1
```

This structure is expected by Krill when processing token streams. The prefix `<>` denotes the span type, `corenlp/c` denotes the foundry and the layer, and `NP` denotes the annotation value. `$` splits the term dictionary part and the postings list part whose structure may differ depending on the information needed to be stored. Everything that follows `$` is stored per token as a byte stream. The information in the angular brackets defines the data type to decode the following information (`<b>` for byte, `<i>` for integer etc.). The first byte (`<b>64`) introduces the payload identifier for denoting a span-level annotation, the following two integers encode the character offsets of the annotation (`<i>0` to `<i>8`), the next integer contains the token position after the end of the span (`<i>2`), and the final byte represents the depth of the annotation in a hierarchy (`<b>1` means the annotation is a direct child to the root).

In KorAP, a separate internal preprocessing pipeline is used to enrich corpus data (based on the I5 format of DeReKo; see Lüngen and Sperberg-McQueen, 2012) by adding standoff annotations from multiple foundries (including CoreNLP, Mate or Xerox

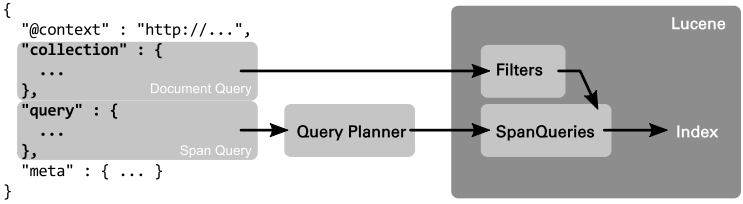


Figure 5: KoralQuery translation into Lucene applicable queries

parsers) resulting in KorapXML (Bański et al., 2012). A `base` foundry provides the minimum annotation necessary for Krill, including sentences, paragraphs, and text boundaries. The KorapXML data can be transformed into a JSON document consisting of a single token stream of the Krill term structure described above by using a conversion tool.[10] The resulting JSON document can be indexed by Krill.

A Krill index is not static and is updated every time a new document is added or deleted. Deletion is done by maintaining a list of deleted documents that are excluded from every virtual corpus requested (see Sec. 4.1). The documents will be purged from the index regularly as a defragmentation of the index. Moreover, Krill uses unique text identifiers in the metadata for updates: documents are updated by deleting the old version of a document and adding a new one to the index. As a consequence, every modification of the document, such as an addition of annotations, will result in a new version of the whole document.

4 Query Processing

KorAP is designed to handle various kinds of queries from existing corpus query languages. These queries of different syntaxes are represented in a common format by the KoralQuery protocol (see Sec. 2 and Bingel and Diewald, 2015) serialized in JSON-LD (Sporny et al., 2014). KoralQuery describes complex linguistic queries as nested objects with various types and operations. Krill is the reference implementation for KoralQuery consumption, aiming for completely supporting it.

On the root level, KoralQuery distinguishes between *document queries* and *span queries*. Document queries allow to specify a virtual corpus (i.e. a defined subcollection of documents) by means of document-level metadata constraints (see Bański et al., 2013b), while span queries define a textual span to search in the virtual corpus on the word-level.

Figure 6 shows a KoralQuery document. The `query` section is a serialization of the query `[orth=sun][][orth=moon]?` (Poliqarp notation). The `collection` section defines a virtual corpus to limit the search to all documents where the `author` metadata field contains the term “Goethe”, the `pubDate` field contains a date since 1786, and

[10] `https://github.com/KorAP/Korap-XML-Krill`

```
{
  "@context" : "http://korap.ids-mannheim.de/ns/koral/0.3/context.jsonld",
  "collection" : {
    "@type": "koral:docGroup",
    "operation": "operation:and",
    "operands": [{
      "@type": "koral:doc",
      "key": "author",
      "match": "match:contains",
      "value": "Goethe"
    },{
      "@type": "koral:docGroup",
      "operation": "operation:and"
      "operands": [{
        "@type": "koral:doc",
        "key": "pubDate",
        "match": "match:geq",
        "type": "type:date",
        "value": "1786"
      },{
        "@type": "koral:docGroup",
        "operation": "operation:or"
        "operands": [{
          "@type": "koral:doc",
          "key": "title",
          "match": "match:contains",
          "value": "Reise"
        },{
          "@type": "koral:doc",
          "key": "title",
          "match": "match:contains",
          "value": "Wahlverwandtschaften"
        }],
      }]
    }]
  },
  "query": {
    "@type": "koral:group",
    "inOrder": true,
    "operation": "operation:sequence",
    "operands": [{
      "@type": "koral:token",
      "wrap": {
        "@type": "koral:term",
        "key": "sun",
        "layer": "orth",
        "match": "match:eq"
      }
    },{
      "@type": "koral:token"
    },{
      "@type": "koral:group",
      "operation": "operation:repetition",
      "boundary": {
        "@type": "koral:boundary",
        "max": 1,
        "min": 0
      },
      "operands": [{
        "@type": "koral:token",
        "wrap": {
          "@type": "koral:term",
          "key": "moon",
          "layer": "orth",
          "match": "match:eq"
        }
      }]
    }]
  }
}
```

Figure 6: A KoralQuery document

Figure 7: Bit vector calculation of the virtual corpus as defined in Figure 6 based on five documents

a `title` field that contains either the term "Reise" or "Wahlverwandschaften" (see Diewald and Bingel, 2015, for the KoralQuery specification).

To process a query, Krill parses and validates the KoralQuery and transforms the virtual corpus into applicable Lucene Filters and span queries into applicable Lucene SpanQueries (see Fig. 5). The span query may need to be rewritten into an intermediate query tree according to a query plan to be applicable. The following sections describe these steps.

4.1 Document Queries

In Krill, every search is limited to a virtual corpus, therefore the virtual corpus is prepared first based on the nested constraints in KoralQuery. Each constraint is described by a key (e.g. "author" for the name of the author of a document), potentially a value (e.g. "Goethe" as the author's name), a matching operator, expressing how the requested value has to match with the document's value (e.g. as a prefix or a postfix) and the type of the constraint (e.g. if the constraint represents a string match, a date range, a regular expression; cf. Figure 6, line 7–10). These constraints can be nested in groups of boolean operations. If no constraint is defined, the virtual corpus contains all available documents.

For each metadata term in the term dictionary, a postings list of the documents is stored in which the term occurs (see Sec. 3). As metadata information is stored on the document level, postings lists for metadata information can simply be treated as bit vectors with one bit per document in the corpus. In case a metadata term exists for one document, the associated bit is set. Using bitwise boolean operations, arbitrary complex virtual corpora can be constructed. Figure 7 shows the calculation of the virtual corpus for the nested document query in Figure 6 based on five documents, using bitwise *or* (`|`) and bitwise *and* (`&`). The resulting virtual corpus can be represented as a bit vector itself, allowing for space-efficient caching.

The actual implementation of document level postings lists varies depending on several factors (e.g. the density of documents or if a postings list is stored on disk or cached in memory). For sparse cached postings lists (e.g. only a fraction of the corpus' documents contain the word "Wahlverwandtschaften" in the title field) Lucene, starting with version 5, implements bit sets as so-called *Roaring bitmaps* focusing on efficient compression and fast bit operations (Chambi et al., 2016). Range queries as mentioned

in the example above are precompiled as a disjunction of the postings lists of all terms in the given range optimized using indexed postings lists of sub ranges (see Schindler and Diepenbroek, 2008, pp. 1957f).

Operations on the virtual corpus include counting the number of texts in the virtual corpus (equivalent to the cardinality of the bit vector), aggregating stored numerical data (e.g. number of tokens, sentences, paragraphs in the virtual corpus), and grouping of statistical data by certain metadata fields (e.g. the distribution of sentences per genre in the virtual corpus). Document queries restrict span queries to documents that are elements of the virtual corpus (see next section).

4.2 Span Queries

In addition to the *document-level* term index for metadata information, the *word-level* term index for tokenized textual data requires more information than the existence in a document, for example, the position of a certain token in a text (cf. Sec. 3). To find, for example, a sequence of two words like "the sun" in a corpus, both terms are searched in the term dictionary and their associated postings lists are analyzed in parallel, to find matching documents and consecutive token positions that indicate a sequence of these two tokens.

Krill supports various query constructs as specified in KoralQuery. Analogously to document queries, these query constructs can be nested and become arbitrarily complex. The resulting query tree consists of leaf nodes with term look-ups, returning the span information of a term, and inner nodes of operations that can be applied on the nested spans.

The result of a span query is always a span, meaning that it always contains information on the start position, the end position, and additional optional information in collected payloads. Operations may use this information to compare nested spans, for example, if two tokens are consecutive in an operation requiring a sequence of two tokens. Payloads may also be passed further to nested operations. Moreover, operations are capable of adding new information to the span's payloads.

If some spans satisfy the requirements of a span query, new spans are created and returned, for example in the case of the consecutive tokens, a span starting at the first token and ending at the last one is created as the resulting span.

Lucene implements such operations as so-called *SpanQueries*. While document queries may use fast bit operations for matching, SpanQueries always iterate sequentially over the postings lists of every term that is part of the query as well as every temporary postings list, that is the result of a span operation.[11]

[11] Lucene uses deterministic *SkipLists* (Pugh, 1990; Munro et al., 1992) to improve performance of moving forward to specific documents in postings lists, for example to skip documents that are not part of the virtual corpus.

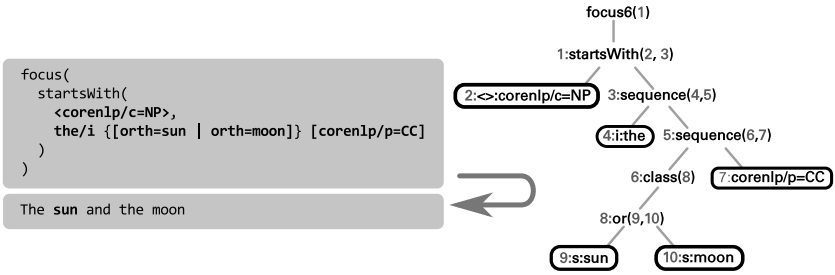


Figure 8: A nested span query with its query tree visualization

To be able to fully support KoralQuery, the set of Lucene SpanQueries was largely extended and new concepts (like *classes*) were introduced.[12] Supported query constructs in Krill can be categorized as follows:

Term queries index look-up of tokens, spans, relations, and attributes (see p. 4), including regular expressions

Comparison queries include logical operations (*and*, *or*, *not*), distance (matching two spans with a defined number of tokens or spans in between), position (matching one span in a positional relation to another, e.g. embedding or overlapping), repetition (matching one span with a defined sequential repetition), etc.

Span modification queries include extensions to left or right, etc.

Class queries include class setting and focus

Figure 8 shows a query in Poliqarp+ notation, searching for a sequence of the word "the" (case-insensitive), followed either by the word "sun" or "moon", and ending with a coordinating conjunction (`CC`) annotated by the `corenlp` foundry. As a wrapping constraint, the sequence needs to be at the beginning of a nominal phrase (`NP`) as annotated by the `corenlp` foundry. In addition, a fragment of the query – the *or*-relation of "sun" and "moon" – is marked as a class (using curly brackets) and at the root of the query tree, this class is focused, meaning a match would return the span of "sun" or "moon" without their contexts.

After the query is serialized as KoralQuery and transformed into Lucene SpanQueries, the resulting query tree has the structure as illustrated on the right side of Figure 8. Term look-ups are pictured as leaf nodes in white boxes.

[12] Recent versions of Lucene support a wider variety of query constructs, that were already introduced to Krill, such as *SpanWithinQuery*.

4.3 Query Planning

Because the constructs of the KoralQuery protocol have their origins in the various corpus query languages covered by Koral that were developed for various corpus search technologies (including concepts coming from regular expressions, relational databases, XML processing and others), not all of these constructs can be directly translated into a term based search. Therefore Krill has a planning phase to rewrite queries before they get translated into actual Lucene SpanQueries.

```
[orth=sun] [] [orth=moon]?
```

The example query above[13] requests a sequence of tokens, with the first having the surface form `sun`, the second matching any token, and the last optionally matching the surface form `moon`. The *any* token of CQP-based corpus query languages matches all tokens in a token sequence (see `[]` in Evert and the OCWB Development Team, 2010, p. 11). In a linear search, this query can recognize the context of the token sequence and would accept any token as a match. An inverted index, however, has no knowledge of linear contexts of token sequences, meaning this token is not directly retrievable from the index or it would need to aggregate the postings lists of all surface terms in the term dictionary.

The query planner translates *any* tokens in sequences into either distance operations between the surrounding parts or, in case an *any* token is at the end or at the beginning of a sequence, into extension operations, that simply extend the resulting span by a certain number of tokens. For the example above, a distance of one token between `[orth=sun]` and `[orth=moon]?` is added.

```
distance(+1, [orth=sun], [orth=moon]?)
```
[14]

This query is however insufficient. Although the optional token `[orth=moon]?` at the end of the example query is retrievable in case it occurs, it needs to be reformulated to an *or*-relation operation to be processable in case it does *not* occur. For the case `[orth=moon]` occurs, `[][orth=moon]?` could be redefined as an extension of one token to the left of `[orth=moon]` and for the case it doesn't occur, as an *any* token.

```
[orth=sun] (extend(-1, [orth=moon]) | [])
```

As described above, an independent *any* token is unretrievable. Thus, the *or*-relation scope needs to be expanded to the whole query. The first case is reformulated as a distance query of one token between `[orth=sun]` and `[orth=moon]`, and the second as an extension of one token to the right of `[orth=sun]`.

```
distance(+1, [orth=sun], [orth=moon]) | extend(+1, [orth=sun])
```

[13] See the `query` section of Figure 6, line 37–69, for a KoralQuery serialization of the query.
[14] The illustration of the query plan is written as CQP based *pseudo* queries.

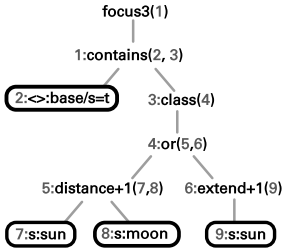


Figure 9: Rewritten span query tree for the Poliqarp query `[orth=sun][][orth=moon]?`

The right extension (`extend(+1, [orth=sun])`) may exceed the length of a matching text (imagine a text ending with the word "sun"). To prevent mismatches regarding right extensions, the query planner introduces a new constraint requiring the result being inside the text boundary as annotated by the `base` foundry. The final query tree transformed into a Lucene SpanQuery is shown in Figure 9.

Not all valid KoralQuery requests can be successfully rewritten in the way described above. In other words, not all valid queries can be answered by Krill. For example, Krill will respond with an error to a query containing just a single *any* token (`[]`), and it will respond with a warning to a query with a single optional token (e.g. `[orth=moon]?`), saying the process will ignore the optionality of the query.

The query planning phase is also used for query optimization, whenever the reorganization of the query tree can be beneficial to performance. In the example above, retrieving the postings list for `[orth=sun]` twice may result in slower performance and therefore could be rewritten to a different query plan.

5 Search Results

Although the search response format of Krill is based on JSON-LD,[15] the text snippets of each match are embedded as HTML fragments. In case of *KWIC* (Keyword in Context) results, these fragments only contain the textual content and the match marker. When retrieving further information about a match reconstructed from the inverted index, the snippet can be enriched with multiple layers of annotations inline.[16]

Figure 10 shows a match of "die Sonne" containing information about the constituency (`c`) layer of the `corenlp` foundry. API clients (like Kalamar) can easily parse this information and process it in various ways, for example to show table views for token-level information, or tree views for hierarchical spans and relations (see Fig. 11).

Classes (as introduced in the previous section) may also be used as partial highlights of a match, to map sections of the query directly to parts of the match. Figure 11 shows

[15] The search response format of KorAP is not fully specified as part of KoralQuery yet.

[16] Krill makes a distinction between token-, span- and relation-based annotations. Only concurrent retrieval of multiple token-based annotations is supported. Span- and relation-based annotations can only be retrieved separately to avoid overlapping.

```
<span class:"context-left"></span>
<mark>
  <span title:"corenlp/c:CS">
    <span title:"corenlp/c:ROOT">
      <span title:"corenlp/c:S">
        <span title:"corenlp/c:NP">die Sonne</span>
        war
        <span title:"corenlp/c:CAP">hoch und hei ß </span>
      </span>,
      <span title:"corenlp/c:S">
        ich mu ß te
        <span title:"corenlp/c:S">
          <span title:"corenlp/c:NP">meine Kleidung</span>
          erleichtern,
          <span title:"corenlp/c:S">
            die ich
            <span title:"corenlp/c:PP">
              bei der veränderlichen Atmosphäre
              <span title:"corenlp/c:NP">des Tages</span>
            </span>
            oft wechsele
          </span>
        </span>
      </span>
    </span>
  </span>
</mark>
<span class:"context-right"></span>
```

Figure 10: Match snippet with enriched annotation from `corenlp/c`

a search result in the frontend component Kalamar for the query `[corenlp/p=ART] ({1:Sonne} | {2:Mond}) []` (Poliqarp+ notation), which has two numbered classes (in curly brackets). Depending on the matching term, "Sonne" and "Mond" are underlined using different colors.

6 Related Work

Corpus search differs from typical information retrieval (e.g. web search) in terms of relevance and ranking. Whilst information retrieval aims to obtain relevant resources and ranks its results by the degree of relevance, there is typically no variation in the degree of relevance of corpus search results. Corpus search focuses on accuracy and only correct matches with 100% accuracy are regarded as results.

Many corpus search engines such as the IMS Corpus Workbench (CWB) and Poliqarp are based on a tabular data model whose rows represent token sequences and columns various annotations. CWB4 or Ziggurat attempts to overcome the limitations of CWB3 that can only handle up to 2.1 billion tokens and to meet all the CQLF metamodel levels by extending the CWB3 data model extensively (Evert and Hardie, 2015). However, to simplify the data model representation, access, and implementation, it would be limited to work only with static corpora where the tokenization and annotation values cannot be modified and the documents cannot be added to or deleted from the indexed corpora. Moreover, corpora cannot be combined into a single virtual corpus.

Relational databases are deemed to have problems to scale up to large corpora, particularly with billions of words (Evert and Hardie, 2015). Ghodke and Bird (2008)

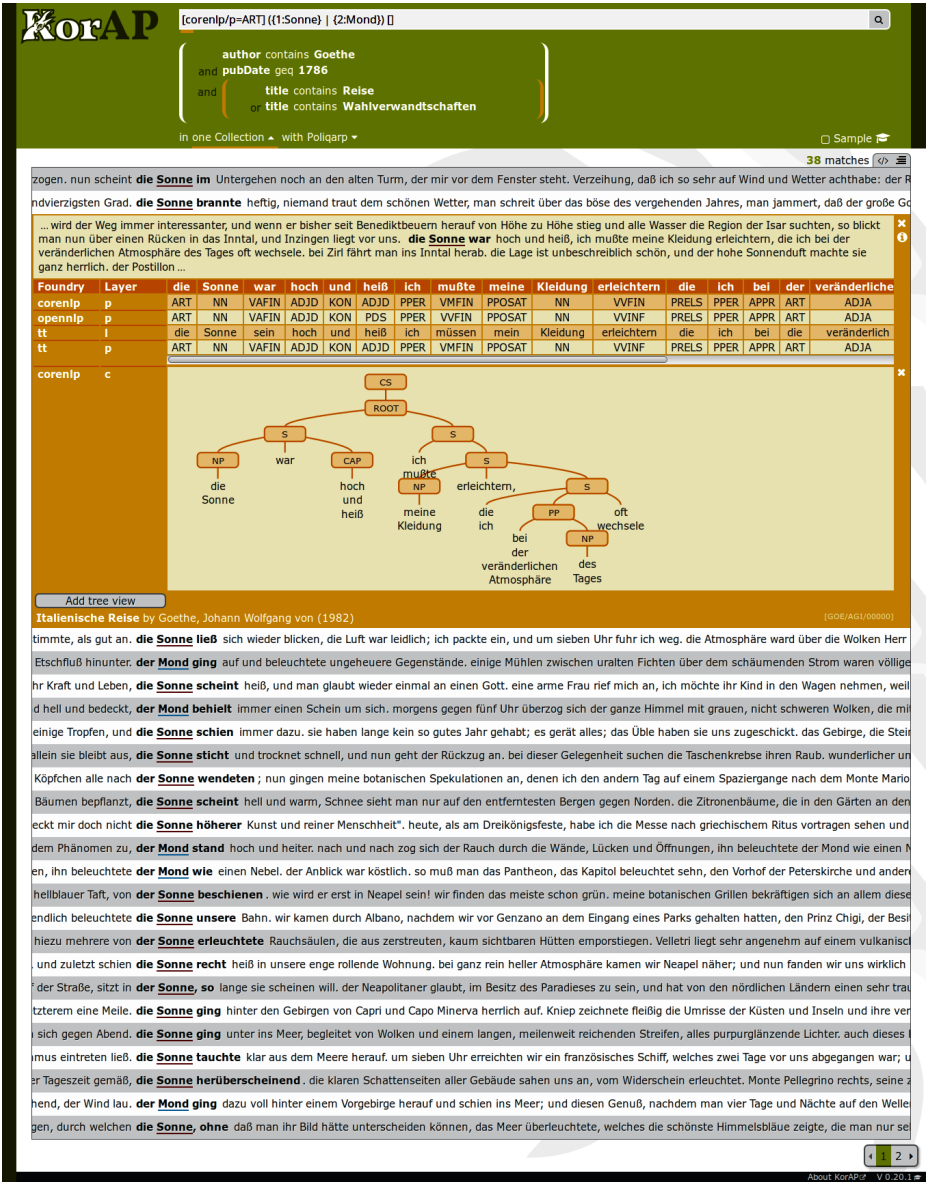


Figure 11: Screenshot of the Kalamar Frontend

illustrate this problem by comparing query execution time between a native XML DB (eXist) and a relational database (Oracle). Their experiments show that the query execution times of both approaches increase greatly as the dataset size increases. Recent evaluations have shown however, that parallelization can be a reasonable approach in dealing with these issues (Schneider, 2012). Besides, Davies (2005) suggests that his n-gram table approach has no limitations on the corpus size and the number of annotation tables and works very fast on pattern matching and synonym queries. However, Evert and Hardie (2015) argue about its redundancy issue and capability to handle more complex linguistic data structures. ANNIS (Zeldes et al., 2009) supports complex linguistic annotations, but it has only been tested with relatively small annotated corpora.[17]

XML database technologies such as BaseX and eXist rely heavily on XQuery and XPath to navigate and extract data from XML documents. Using XPath to query data with multiple stand-off annotation files is rather ineffective due to the need to resolve pointers between the files repeatedly (Mayo et al., 2006). The re-implementation of NXT's query language using XQuery (NQL, Mayo et al., 2006) is shown to be able to load about four times more data than the former NXT Search with Java implementation (Carletta et al., 2005) while utilizing the same memory size. However, using XQuery with stand-off data format decreases the speed performance in executing queries (Mayo et al., 2006).

Krill is based on prior research on the applicability of Lucene for complex linguistic corpus search tasks (Schnober, 2012). During the development of Krill, similar Lucene-based search engines emerged, like BlackLab[18] and MTAS.[19] BlackLab also supports multiple query languages (e.g. CQL, Lucene QL), but only basic features have been supported yet.[20] MTAS is a new corpus search engine in its early stages with support for distributed search using Apache Solr.[21]

7 Further Work

To support horizontal scalability in KorAP, multiple Krill instances can be utilized in a document-partitioned cluster for distributed search managed by Kanalito, that is still under development (see Sec. 2). With the introduction of this receptionist service, more features will be available, like facet search and methods for sorting, that have been already supported by COSMAS II. In addition, statistical analysis of query results through frequency and co-occurrence are in preparation.

We plan on evaluating Krill performance, for instance in terms of searching and indexing with regard to corpus size and in comparison to other corpus search systems,

[17] https://corpling.uis.georgetown.edu/annis-corpora/
[18] http://inl.github.io/BlackLab/
[19] https://meertensinstituut.github.io/mtas/
[20] https://github.com/INL/BlackLab/wiki/Blacklab-query-tool
[21] http://lucene.apache.org/solr/

such as COSMAS II. We also plan an evaluation of the scope of supported query constructs, especially with respect to the work on *CQLF* (Bański et al., 2016).

8 Summary

We have described Krill, a search component in KorAP, and its interaction with other components in the KorAP architecture. Krill is based on Lucene and uses an inverted index to perform full-text search. We have extended Lucene to support a wide range of corpus query operations on multiple annotations, and the flexible creation of virtual corpora. Krill is open source, available on GitHub[22] and published under the BSD-2 License. Despite being developed in the context of KorAP, Krill can be used as a stand-alone application.

9 Acknowledgements

We would like to thank our colleagues for their contributions, especially Carsten Schnober for his preliminary work on the KorAP search backend and Piotr Pęzik for his recommendations on the term structure. Regarding this manuscript we would like to thank all reviewers for their helpful feedback.

Krill is developed as part of the KorAP Corpus Analysis Platform at the Institute for the German Language, funded by the Leibniz-Gemeinschaft,[23] supported by the KobRA project,[24] funded by the Federal Ministry of Education and Research (BMBF), and the CLARIN-D project,[25] funded by the BMBF and the Ministry of Science, Research and the Arts (Baden-Württemberg).

References

Bański, P., Bingel, J., Diewald, N., Frick, E., Hanl, M., Kupietz, M., Pęzik, P., Schnober, C., and Witt, A. (2013a). KorAP: the new corpus analysis platform at IDS Mannheim. In Zygmunt Vetulani et al., editor, *Human Language Technologies as a Challenge for Computer Science and Linguistics. Proceedings of the 6th Language and Technology Conference*, pages 586–587, Fundacja Uniwersytetu im. A. Mickiewicza, Poznań, Poland.

Bański, P., Diewald, N., Hanl, M., Kupietz, M., and Witt, A. (2014). Access control by query rewriting: the case of KorAP. In Calzolari, N., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*, pages 3817–3822, Reykjavik, Iceland. European Language Resources Association (ELRA).

Bański, P., Fischer, P. M., Frick, E., Ketzan, E., Kupietz, M., Schnober, C., Schonefeld, O., and Witt, A. (2012). The new IDS corpus analysis platform: Challenges and prospects. In

[22] https://github.com/KorAP/Krill

[23] http://www.leibniz-gemeinschaft.de/en/about-us/leibniz-competition/projekte-2011/2011-funding-line-2/

[24] http://www.kobra.tu-dortmund.de

[25] http://de.clarin.eu/

Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 2905–2911, Istanbul, Turkey. European Language Resources Association (ELRA).

Bański, P., Frick, E., Hanl, M., Kupietz, M., Schnober, C., and Witt, A. (2013b). Robust corpus architecture: a new look at virtual collections and data access. In Hardie, A. and Love, R., editors, *Corpus Linguistics 2013 Abstract Book*, pages 23–25, Lancaster. UCREL. `http://ucrel.lancs.ac.uk/cl2013/doc/CL2013-ABSTRACT-BOOK.pdf`.

Bański, P., Frick, E., and Witt, A. (2016). Corpus Query Lingua Franca (CQLF). In Calzolari, N., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 2804–2809, Portorož, Slovenia. European Language Resources Association (ELRA).

Bingel, J. and Diewald, N. (2015). KoralQuery – a general corpus query protocol. In *Proceedings of the Workshop on Innovative Corpus Query and Visualization Tools at NODALIDA 2015*, Vilnius, Lithuania.

Bodmer, F. (1996). Aspekte der Abfragekomponononente von COSMAS-II. *LDV-INFO. Informationsschrift der Arbeitsstelle Linguistische Datenverarbeitung*, 8:112–122.

Büttcher, S., Clarke, C. L. A., and Cormack, G. V. (2010). *Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press.

Carletta, J., Evert, S., Heid, U., and Kilgour, J. (2005). The NITE XML Toolkit: data model and query. *Language Resources and Evaluation Journal*, 39(4):313–334.

Chambi, S., Lemire, D., Kaser, O., and Godin, R. (2016). Better bitmap performance with Roaring bitmaps. *Software: Practice and Experience*, 46(5):709–719. `http://arxiv.org/abs/1402.6407`.

Davies, M. (2005). The advantage of using relational databases for large corpora: Speed, advanced queries and unlimited annotation. *International Journal of Corpus Linguistics*, 10(3):307–334.

Diewald, N. and Bingel, J. (2015). KoralQuery 0.3. Technical report, IDS Mannheim, Germany. Working draft, `https://KorAP.github.io/Koral`, last accessed 2/2/2017.

Diewald, N., Hanl, M., Margaretha, E., Bingel, J., Kupietz, M., Bański, P., and Witt, A. (2016). KorAP architecture – diving in the deep sea of corpus data. In Calzolari, N., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 3586–3591, Portorož, Slovenia. European Language Resources Association (ELRA).

Evert, S. and Hardie, A. (2015). Ziggurat: A new data model and indexing format for large annotated text corpora. In *Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora (CMLC-3)*, pages 21–27.

Evert, S. and the OCWB Development Team (2010). CQP query language tutorial. Technical report, The OCWB Development Team. `http://cwb.sourceforge.net/files/CQP_Tutorial.pdf`.

Ghodke, S. and Bird, S. (2008). Querying linguistic annotations. In *Proceedings of the 13th Australasian Document Computing Symposium*, pages 69–72, Hobart, Australia.

Kupietz, M. and Lüngen, H. (2014). Recent developments in DeReKo. In Calzolari, N., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*, pages 2378–2385, Reykjavik, Iceland. European Language Resources Association (ELRA).

Lüngen, H. and Sperberg-McQueen, C. M. (2012). A TEI P5 document grammar for the IDS text model. *Journal of the Text Encoding Initiative*, 3. `http://jtei.revues.org/508`.

Mayo, N., Kilgour, J., and Carletta, J. (2006). Towards an alternative implementation of NXT's query language via XQuery. In *Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006)*, pages 27–34, Trento, Italy.

Munro, J. I., Papadakis, T., and Sedgewick, R. (1992). Deterministic skip lists. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 367–375, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Przepiórkowski, A., Krynicki, Z., Dębowski, Ł., Woliński, M., Janus, D., and Bański, P. (2004). A search tool for corpora with positional tagsets and ambiguities. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004*, pages 1235–1238.

Pugh, W. (1990). Skip Lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676.

Rosenfeld, V. (2010). An implementation of the Annis 2 query language. Technical report, Humboldt-Universität zu Berlin.

Schindler, U. and Diepenbroek, M. (2008). Generic XML-based framework for metadata portals. *Computers & Geosciences*, 34(12):1947–1955. `http://epic.awi.de/17813/1/Sch2007br.pdf`.

Schneider, R. (2012). Evaluating DBMS-based access strategies to very large multi-layer corpora. In *Proceedings of the LREC 2012 Workshop: Challenges in the management of large corpora*. European Language Resources Association (ELRA).

Schnober, C. (2012). Using information retrieval technology for a corpus analysis platform. In *Proceedings of Konvens 2012*, pages 199–207, Vienna, Austria.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., and Lindström, N. (2014). JSON-LD 1.0. a JSON-based serialization for linked data. Technical report, W3C. W3C Recommendation; `http://www.w3.org/TR/json-ld/`.

Zeldes, A., Ritz, J., Lüdeling, A., and Chiarcos, C. (2009). ANNIS: A search tool for multi-layer annotated corpora. In Mahlberg, M., González-Díaz, V., and Smith, C., editors, *Proceedings of the Corpus Linguistics 2009 Conference*, Liverpool, UK.

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2).

Maria Skeppstedt, Carita Paradis, Andreas Kerren

PAL, a tool for Pre-annotation and Active Learning

Abstract

Many natural language processing systems rely on machine learning models that are trained on large amounts of manually annotated text data. The lack of sufficient amounts of annotated data is, however, a common obstacle for such systems, since manual annotation of text is often expensive and time-consuming.

The aim of "PAL, a tool for Pre-annotation and Active Learning" is to provide a ready-made package that can be used to simplify annotation and to reduce the amount of annotated data required to train a machine learning classifier. The package provides support for two techniques that have been shown to be successful in previous studies, namely active learning and pre-annotation.

The output of the pre-annotation is provided in the annotation format of the annotation tool BRAT, but PAL is a stand-alone package that can be adapted to other formats.

1 Introduction

Some *artificial* intelligence systems rely heavily on *human* intelligence in the form of training data that is collected by manual annotation. Within the field of natural language processing and computational linguistics, this data collection is typically performed through different kinds of manual annotations of text. Manual text annotation is, however, often an expensive and time-consuming task. The lack of sufficient amounts of manually annotated data that can be used for training machine learning classifiers is therefore a common obstacle.

There are a number of techniques that can be used to reduce the amount of annotated data required for training a machine learning classifier and to simplify the annotation process. Two examples of such techniques are i) the selection of training samples that are informative to the classifier by the use of active learning, and ii) pre-annotation of the data. These techniques are not included as a standard procedure in text annotation projects, presumably because annotation tools typically do not include this functionality. The aim of the development of the PAL package presented here is therefore to take the first step towards changing this standard. The package uses active learning and pre-annotation to facilitate annotation of text chunks. The package is tailored towards the annotation format of text chunks in the annotation tool BRAT (Stenetorp et al., 2012), but it is written as a stand-alone package that can be adapted to other formats.

2 Background

There is a large amount of previous studies on active learning, as well as a number of studies in which pre-annotation has been used, but to the best of our knowledge there are no freely available code packages that can be directly used to apply both of these techniques.

The objective of this paper is to present "PAL, a tool for Pre-annotation and Active Learning", which provides this functionality. Since the tool is built on established techniques, an evaluation of the techniques incorporated in the tool is not within the scope of this paper. Instead, references to previous studies are given.

The main inspiration for PAL was provided by Olsson (2008), who shows the usefulness of active learning for annotation of text chunks and who recommends the use of pre-annotation. The usefulness of active learning was however shown purely by simulation, and no tools for performing active learning or pre-annotation were provided.

2.1 Other approaches for obtaining manually annotated data

Apart from the two techniques mentioned, there are also other possible approaches for obtaining large resources of manually annotated data in a low-resource project. Examples of such techniques are crowd sourcing, community annotation and gamification. However, we argue that the use of these approaches is neither possible nor desirable in all cases.

Manually annotated training data is often collected by the use of crowd sourcing platforms such as Amazon Mechanical Turk. There are several problems associated with this approach (Fort et al., 2011; Archambault et al., 2016). Most important is the ethical issue, i.e., criticism against annotation projects that are carried out by very low-paid annotators. Another concern is the skills required, as it might be difficult to find crowd sourcing annotators with specialised skills, for instance in linguistics, the domain of the text, or annotators that speak the specific language required. This means that the use of crowd sourcing is not a universal solution to the problem of creating enough annotated textual data in a low-resource project.

Another approach is to use community annotation (Uzuner et al., 2010), i.e., to share the annotation task among many annotators by distributing it to a number of researchers in a community. There are disadvantages associated with this approach as well, for instance that it entails a large portion of administrative work and that it is not efficient for difficult annotation tasks, for which initial training phases are required. Another possibility is to gamify the annotation, i.e., to apply game design principles to the task. By making the annotation more fun, there is a potential to gain voluntary annotators (Hanbury et al., 2015). This approach has, for instance, been applied to word sense labelling (Venhuizen et al., 2013) and to identify important expressions in medical case reports (Dumitrache et al., 2013). Yet not all annotation tasks are possible to gamify, at least not effortlessly.

2.2 The type of annotations targeted

The PAL package is meant to be used for annotations of named entities or other types of shorter chunks of text.

The aim could be to create annotated corpora, which in turn can be used for training a classifier to detect the types of text chunks annotated. That is, the machine learning classifier should i) automatically detect interesting tokens (or chunks of interesting tokens) in a text, and ii) automatically categorise these tokens into pre-defined classes. Annotation and detection of named entities such as names of people and places (Nadeau and Sekine, 2007), or of tokens that signal specific functions in a language such as marker words for negation and speculation (Konstantinova et al., 2012) are examples from previous studies.

Another aim could be to create annotated corpora, on which to perform corpus linguistic studies of language phenomena that are expressed through shorter text chunks. Examples of such annotation tasks from previous corpus linguistic studies are annotations of spans of tokens that express attitude (Taboada and Carretero, 2012), evaluation (Fuoli and Hommerberg, 2015), or sensory perceptions (Paradis and Eeg-Olofsson, 2013).

It should be noted that the use cases for annotating these two types of corpora are different. When the aim is to create a corpus for training a machine learning classifier, it is enough to annotate an actively selected subset of the corpus, whereas the entire corpus must be annotated in a corpus study. Otherwise it will not be possible to draw statistically valid conclusions from the annotations. As suggested by Olsson (2008), the methodology when annotating the entire corpus is i) to annotate an actively selected subset of the corpus to achieve a model that can perform pre-annotation with a high accuracy, and ii) to use pre-annotation to annotate the remaining part of the corpus.

To create a corpus for training a classifier, only the first step has to be carried out.

2.3 Pre-annotation

Pre-annotation, or pre-tagging, refers to the procedure to automatically annotate a text corpus by using an existing automatic system and to present these annotations to the human annotator. The human annotator then typically corrects mistakes or omissions made by the automatic system (Chou et al., 2006; Henriksson et al., 2015), or alternatively makes a choice between different options given by the automatic system (Brants and Plaehn, 2000).

The pre-annotation could be built on a rule-based system, such as a system that performs rule-based matching against an existing lexicon (Albright et al., 2013). This approach does not require annotated data. It could also use an existing machine learning system, which may be trained on data from another text domain (Henriksson et al., 2015). A third option is to use pre-annotation based on a machine learning model and to iteratively improve the pre-annotation by retraining it on the new data that is created in the annotation process (Tomanek et al., 2012).

For the task of named entity annotations within the medical domain, lexicon-based pre-annotation has led to an increase in annotation speed, ranging from 14% to 22% per entity for different experiments (Lingren et al., 2014). The same study also included an investigation of whether the pre-annotation functionality biased the annotators, but no bias that stemmed from the pre-annotations could be detected.

2.4 Active learning

Active learning is a technique that is used to reduce the number of training samples that are required to successfully train a machine learning model. The standard method used for manual text annotation is to randomly select which data samples to annotate. For active learning, the training data samples that are estimated to be most useful for the machine learning classifier are instead actively selected from a pool of unlabelled data (Tomanek, 2010).

The estimation of which data samples in the pool that are most useful can, for instance, be based on the level of disagreement among a number of different classifiers (query by committee). That is, the more different classifiers disagree, the more informative is the sample likely to be (Olsson, 2008, pp. 25–29). The estimation can also be based on properties specific to the type of model that is used. When using support vector machines, the unlabelled sample closest to the separating hyperplane of the classifier can be selected, which is the sample that is expected to result in the largest model change when added to the training set (Tomanek, 2010; Tong and Koller, 2002).

Another frequently used method for active selection is *uncertainty sampling*. This technique is built on active selection of the unlabelled training samples that the machine learning model is least certain of how to classify. The approach for a binary classification task is then to select samples for which the classifier has no clear classification preference.

One option for a multi-class classification task is to use the confidence for the most probable class as the measure of uncertainty. However, this option only uses the certainty level of the most probable classification. Thereby, some of the information available is discarded, i.e., the information regarding the certainty levels of the less probable classifications (Settles, 2009). An alternative approach is to base the sample selection for a multi-class classifier on the difference in certainty level between the two most probable classifications. Given c_{p1} as the most probable classification and c_{p2} as the second most probable classification for the observation $\boldsymbol{x}_n$, the margin for measuring the uncertainty of that sample would then be:

$$M_n = P(c_{p1}|\boldsymbol{x}_n) - P(c_{p2}|\boldsymbol{x}_n) \tag{1}$$

Samples with a large margin (M_n) are easy to classify since the classifier is much more certain of the most probable classification than of the second most probable. Samples with a small margin, on the other hand, are difficult to classify. Therefore, in the process of uncertainty-based active selection of training samples, samples with a small margin are preferred (Schein and Ungar, 2007).

This kind of uncertainty selection, based on the confidence difference between the two most probable classifications, is the sampling method that is implemented in the current version of the PAL package.

2.5 Functionality of previous annotation tools

Among 13 annotation tools included in an annotation tool survey from 2012 (Neves and Leser, 2012), only two (JANE and WordFreak) provide functionality for active learning. JANE does not provide a functionality for pre-annotations (Tomanek et al., 2007), and WordFreak only provides pre-annotation performed by pre-defined NLP tools that are independent of the annotation task, e.g., part-of-speech tagging (Morton and LaCivita, 2003).

Among the tools included in the survey, there are others that provide a pre-annotation functionality, typically based on lexicon-matching. There are also more recent annotation tools that either provide the functionality of lexicon-based pre-annotation (Campos et al., 2013) or the functionality of active learning (Kucher et al., 2016).

A disadvantage with pre-annotation based on lexicon-matching is that this approach relies on the existence of a lexicon in the domain of the annotation task. The approach is therefore limited to domains in which such a lexicon exists. There are also tools that pre-annotate without a lexicon, e.g., WebAnno (Yimam et al., 2014). For this tool, the pre-annotations are performed by a machine learning model that is trained on the labelled data provided by the annotator, and the model is iteratively re-trained with more data as more text is manually annotated.

PAL combines the functionality of i) a tool such as WebAnno, in which the pre-annotations are provided by models trained on data that is annotated when using the tool, with ii) the functionality of tools such as JANE and WordFreak, in which data selection through active learning is carried out. We are not aware of any previously constructed, freely available, chunk annotation tools in which these two functionalities are combined.

3 Method

The general functionality of the PAL package consists of training a classifier to select suitable data to annotate from a pool of unlabelled data and to carry out pre-annotation of this data. The pre-annotated data can then be loaded into the BRAT tool, to be revised by the human annotator.

3.1 Active learning and pre-annotation

The PAL package is structured around three different data folders:

1. The *labelled* folder, which contains the data that has been manually labelled.
2. The *unlabelled* folder, which contains the pool of unlabelled data.

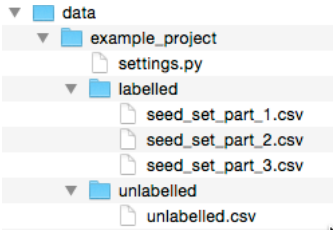


Figure 1: Files and folders that are to be available when a project starts.

3. The *tolabel* folder, to which the pre-annotated, actively selected data is written.

When a project starts, it must contain the data and the folders that are shown in Figure 1. In the *labelled* folder, there must be at least one file containing the seed set of annotated data, which is required to start the active learning process. In the example in Figure 1, the seed set is split into three different files of annotated data. The file that contains the pool of unlabelled data must be called *unlabelled.csv* and must be positioned in the *unlabelled* folder.

When the process of active learning and pre-annotation is run, all .csv-files located in the *labelled* folder are used to train a machine learning model. This model is then employed to perform the active selection of training samples and the pre-annotation. Data samples to be labelled are selected from the file *unlabelled.csv*, pre-annotated and written to the three files that are created in the *tolabel* folder, as shown in Figure 2. These three new files receive a name containing their creation timestamp. The two files named *brat_tolabel_20160928_174414.ann* and *brat_tolabel_201609_28174414.txt* contain the pre-annotated data in BRAT format. That is, those two files are the ones that can be directly imported into BRAT for manual annotation. The file *unlabelled.csv* is also updated and does not any longer contain the data samples that have been selected for annotation. That is, the selected data samples have been removed from the pool of unlabelled data. A copy of the original version of the unlabelled data is also created, the file *unlabelled_20160928_174414.csv* in the *unlabelled* folder.

The data in the .csv-files must be available in tab-separated format in which the data has been tokenised with each token on a separate line. The data must also be segmented into sentences with an empty line signalling a sentence break. See Figure 5 for an example of the data format. The tokens are expected to be labelled according to the BIO-format (Jurafsky and Martin, 2008, pp. 763–764), i.e., a token could be the *B*eginning of, *I*nside or *O*utside of a named entity (or of another type of text chunk). The data samples in the active selection process consist of the sentences. This means that text units in the form of a number of tokens that are separated by an empty line are the units which are selected in the active sampling process.

In the current version of PAL, there are two types of machine learning classifiers to be chosen from, one structured and one non-structured. The following is a step-by-step

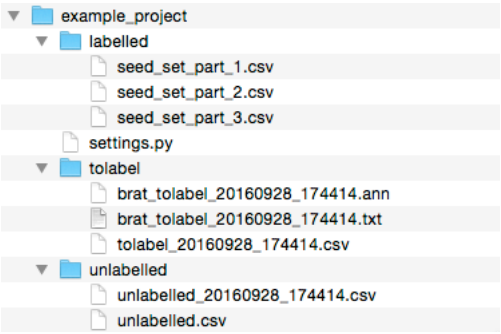


Figure 2: Files and folders created by running active learning and pre-annotation.

description of how the active learning and pre-annotation process is carried out for the non-structured classifier:

1. A machine learning model is trained using the data in the .csv-files that are located in the *labelled* folder.

2. The data located in the *unlabelled* folder is classified with this model. For each of the N tokens in the unlabelled data $(t_n)_{n=1}^N$, there will be an observation in the form of the features for this token, i.e., $(\boldsymbol{x}_n)_{n=1}^N$. For each of these observations, the model provides a probability score for every category in the data.

3. The sentences in the unlabelled data are then ranked according to the uncertainty sampling criterion described in Equation 1, i.e., $M_n = P(c_{p1}|\boldsymbol{x_n}) - P(c_{p2}|\boldsymbol{x_n})$. This is carried out by measuring the difference in probability score between the most likely classification and the second most likely classification for a token. A sentence is represented by the lowest M-value among the tokens it includes, and the sentences are ranked according to this M-value.

4. The k sentences with the k lowest M-values are selected. The value of k (the number of sentences to select in each iteration) is determined by the user in a settings file (see Section 3.2).

5. To achieve a variation among the k samples selected, the same word predicted as belonging to another class than the Outside class is only allowed to occur once among the sentences selected. If such a re-occurring word is present among the sentences selected, the lowest ranked sentence containing this word is removed from the selected set. The sentence that is removed is replaced by the sentence next in rank, i.e., the sentence ranked at position $k + 1$.

6. The k selected sentences are pre-annotated according to the classification made by the machine learning model.

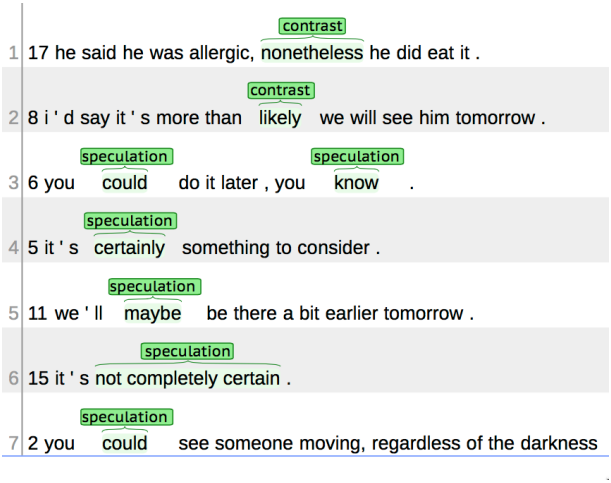


Figure 3: An example of pre-annotated text for the two entity categories consisting of marker words signalling *speculation* and *contrast*, which is imported and displayed in BRAT.

7. The pre-annotated sentences are written to the *tolabel* folder, in a tab separated format and in the BRAT-format.

The same selection procedure is carried out for the structured model, except that the classifications are not made on token level, but on sentence level. M is thereby not computed for the difference in probability score between different token classifications, but between different sentence level classifications. In addition, to reduce the processing-time of the system, only a subset of the alternative sentence classifications are taken into account when computing M.

The pre-annotated sentences can then be opened in BRAT, as shown in Figure 3. This allows the human annotator to carry out the annotation, which then consists of correcting and supplementing the pre-annotations. Figure 4 shows how an incorrect pre-annotation is deleted.[1]

3.2 Configuring and running PAL

To run the package, a directory for the project needs to be created. This directory needs to contain the directories with *labelled* and *unlabelled* data (as shown above), as well

[1] In the BRAT options menu, different annotation modes can be selected. The recommendation is to not use the *Careful* mode, since this mode prompts the user for a confirmation every time an annotation is removed.

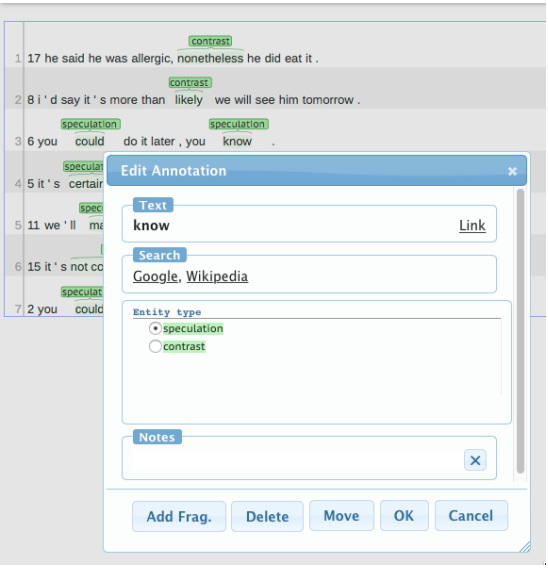


Figure 4: The annotator uses BRAT to delete the incorrect pre-annotations.

as a *settings.py* file with configuration information. The most important configuration parameters are described here.[2]

```
# Classes to include with their prefix (B or I).
# Classes, apart from "O" the outside-class, not in this list, will be ignored
minority_classes = ["B-speculation", "I-speculation", "B-contrast", "I-contrast"]

# Number of sentences to be actively selected and pre-annotated in each round
# (referred to as k above)
nr_of_samples = 20

# Type of model to use
model_type = NonStructuredLogisticRegression

# The context around the current word to
# include when training the classifiers
number_of_previous_words = 2
number_of_following_words = 1

# A cut-off for the number of occurrences of a token in
# the data for it to be included as a feature (current and context-token)
min_df_current = 2
min_df_context = 2
```

[2]See the readme-file of the package, for a full description of additional configuration parameters.

The *settings.py* file needs to include at least 1) the classes that represent named entities or other types of relevant text chunks that are to be included in the active learning and pre-annotation process, 2) the number of samples to select in each iteration of the active learning process (referred to as k above), 3) the type of model (structured or non-structured), 4) the number of context tokens to include when constructing the feature vector for a token, and 5) the minimum number of occurrences of a token (and a context token) for it to be included in the vector model constructed for the tokens occurring in the data (see Section 3.3).

The functionality for selecting and pre-annotating samples can then be run with the following command, in which the location of the data and the settings file is specified:

```
python active_learning_preannotation.py --project=data.example_project
```

The data files that are created by this command (the .txt and the .ann files) can then be moved to the BRAT *data* folder to make it available for the annotator to select for annotation.[3]

When the annotator has finished the annotation/correction of the pre-annotated files, they can be transformed back to a tab separated format and positioned in the folder with annotated data with the following command (on one line, without line breaks):

```
python transform_from_brat_format.py
 --project=data.example_project
 --annotated=my_brat_path/data/example_project/brat_tolabel_20160928_174414
```

The name *brat_tolabel_20160928_174414* is the path to the .txt-file and to the human-annotated version of the .ann-file, but without the suffices. The two files must be positioned in the same folder.

3.3 Models and feature representations

The PAL package is written in Python 3. It is developed to be used in a Unix environment, and it has been tested on Ubuntu 12.04.5.

There are currently two types of machine learning models that are supported by PAL, a logistic regression model (Bishop, 2006) and a conditional random fields model (Sutton and McCallum, 2006). The logistic regression model is built on the LogisticRegression class of the Scikit-learn library. This library includes, among many other components, Python classes for performing non-structured prediction (Pedregosa et al., 2011). The conditional random fields model is built on the ChainCRF class that is available in the PyStruct library, which is a library that contains a number of Python classes for performing structured prediction. A structured prediction model takes the structure of the output labels into account for training the classifiers and performing the predictions (Müller, 2013). The ChainCRF model is an implementation of a linear-chain conditional random fields model, in which an output variable is only directly dependent on its immediate neighbours (Sutton and McCallum, 2006, p. 9).

[3] See the BRAT user documentation for more information about this.

n (Token number)	Word	Label	
..	..	..	
21	it	O	
22	’	O	
23	**s**	O	← previous context token
24	**not**	B-speculation	← previous context token
25	**completely**	I-speculation	← current token
26	**sure**	I-speculation	← following context token
27	.	O	
28	Another	O	
29	sentence	O	
..	..	..	

$\boldsymbol{x}_{25} =$ [Current token: 0 1 0 0 0 0 | Context -2: 0 0 1 0 0 | Context -1: 0 1 0 0 0 | Context +1: 0 0 0 1]

Figure 5: An example that shows how the feature vector for token number 25 is constructed, i.e., $\boldsymbol{x}_{25}$. In this example, the user has configured PAL to include a context in the form of two preceding tokens and one following token. That is, *number_of_previous_words*=2 and *number_of_following_words*=1. The words used for constructing the feature vector are marked with bold and colours. The vector representations of these words are concatenated to form the feature vector $\boldsymbol{x}_{25}$. See Figure 6 for an example of how vector representations for words are constructed.

What features to use for training and predicting the label of a token, i.e., the observation vector $\boldsymbol{x}_n$, is decided by the parameters *number_of_previous_words*, *number_of_following_words*, *min_df_current*, and *min_df_context* in the settings-file. Two different matrices that represent the words in the labelled data are created, using the CountVectorizer class of the Scikit-learn library. One of the matrices contains the vector representation of the tokens for which a label is to be predicted and the other matrix contains the vector representation of the context tokens (Figure 5). Each word that occurs at least *min_df_current* times in the labelled corpus receives its unique vector representation in the matrix for the current tokens and each word that occurs at least *min_df_context* times in the labelled data receives a unique vector representation in the matrix for context tokens. Such a vector representation is exemplified in Figure 6.

The feature vector to use for training and prediction (the observation vector $\boldsymbol{x}_n$) is constructed by concatenating the vector representation of the current token with the vector representations of the tokens surrounding it. How many surrounding tokens to include is decided by the setting parameters *number_of_previous_words* and *number_of_following_words* (See Figure 5).

Word	From labelled corpus
be	[1 0 0 0 0]
completely	[0 1 0 0 0]
not	[0 0 1 0 0]
s	[0 0 0 1 0]
sure	[0 0 0 0 1]

Figure 6: Matrix of vector representations of words in a very small, hypothetical corpus. In this corpus, only five words occur frequent enough to be included in the matrix. The value of "frequent enough" is decided by the user configuration, i.e., by the variables *min_df_current* (if the matrix is a representation of words when they occur as the current token) and *min_df_context* (if the matrix is a representation of words when they occur as the context tokens).

3.4 Incorporation of unsupervised approaches

Information harvested in an unsupervised fashion from unlabelled corpora has for a long time (Miller et al., 2004; Freitag, 2004) been used as classifier features for improving the performance of named entity recognition and other types of chunk classification tasks. For instance, information from Brown clustering has been used for biomedical entity recognition (de Bruijn et al., 2011), predictive class bigram model clustering for general named entity recognition in several languages (Täckström et al., 2012), Random Indexing for named entity recognition in clinical text (Henriksson, 2015), and different kinds of word embeddings have been used for recognising opinion targets (Liu et al., 2015).

The PAL package has support for adding unsupervised features through the Gensim library (Řehůřek and Sojka, 2010). PAL can be configured to append external vector representations from Gensim to the feature vectors. This functionality has so far only been tested with vectors from an out-of-the-box word2vec model for English, which has been trained on Google news (Mikolov, 2013; Mikolov et al., 2013).

3.5 Availability

The tool is freely available to clone or download from GitHub at:

`https://github.com/mariask2/PAL-A-tool-for-Pre-annotation-and-Active-Learning`

PAL makes use of three external libraries, which all are freely available:[4]

1. For training the machine learning models for performing structured prediction, the PyStruct library is used (Müller and Behnke, 2014).

2. For performing non-structured prediction and for vectorising the data, the LogisticRegression and the CountVectorizer classes of the Scikit-learn library are used (Pedregosa et al., 2011).

[4] See the readme-file of PAL for a description of how to install them.

3. For incorporating unsupervised features, the Gensim library is used (Řehůřek and Sojka, 2010).

These libraries are in turn dependent on cvxopt, numpy and scipy.

4 Discussion and future work

Although PAL incorporates a number of functions to facilitate annotation, the tool can still be improved and extended. There are, for instance, possible limitations of the design decisions applied and potential problems with the general approach of pre-annotation.

4.1 Design decisions

One potential issue is the level of technical knowledge that is required to use PAL. No programming skills are required to configure and run PAL, but knowledge of how to use a Unix command line and how to modify a settings file is required. These two skills are, however, also required for configuring the BRAT annotation tool. Therefore, since the target users of PAL are those that typically would configure BRAT, or a similar annotation tool, the technical skills required for using PAL should not be a problem for its target users. There are other annotation tools that incorporate active learning, which provide a graphical user interface by which settings can be altered and by which actively selected annotation batches can be generated (Kucher et al., 2016). To add such a functionality to PAL would extend its group of target users.

Another design decision is that PAL is constructed as a stand-alone tool for pre-annotation and active learning and that it is not fully integrated with an annotation tool. This design decision has the advantage of making the package more flexible to adapt to other annotation tools. The only modification that has to be carried out is the transformation of the output files that are generated by PAL to the format of the other annotation tool.

4.2 Potential problems with the use of pre-annotation

As stated above, there are studies in which pre-annotation has been shown to increase the speed of annotation without introducing a bias (Lingren et al., 2014) and annotation studies in which pre-annotation has been applied successfully (Albright et al., 2013). In contrast, other studies have shown that pre-annotation has had no effect on the time taken (South et al., 2014) or that pre-annotation has slowed down the annotators due to the low quality of the pre-annotations provided (Ogren et al., 2008). There are also cases where high-quality pre-annotations have been shown to be successful in general, with an increase in annotation speed and general annotation quality, but where the pre-annotations have reduced the annotator attention and resulted in a bias (Fort and Sagot, 2010).

The first of these two potential problems, i.e., that low quality pre-annotations are slowing down the annotator, is not a limitation for the functionality provided by PAL. If

the annotator does not consider the pre-annotations to be beneficial, the pre-annotation functionality can be switched off by removing the content of the .ann-file created by PAL. The quality of the pre-annotations might, for instance, be too low in the early stages of the annotation process when there is only a limited amount of annotated data available for training the machine learning model. The methodology proposed by Olsson (2008) is to start the process of annotating for active learning without providing pre-annotation and to introduce the functionality of pre-annotation when the model has reached a certain level of performance. The risk of providing pre-annotations that are not useful would then be minimised.

The usefulness of the pre-annotation functionality provided by PAL has so far been tested in an initial annotation project. A corpus of 6,000 sentences was annotated for text chunks signalling topic-independent expressions of agreement and disagreement, according to a previously defined annotation scheme (Skeppstedt et al., 2016). A seed set that contained 1,500 sentences was first annotated with BRAT without the use of the pre-annotation functionality. The remaining sentences were then annotated with the help of pre-annotation. The quality of the pre-annotation was in this case assessed by the annotator as high enough to be useful.

The issue that pre-annotation might introduce a bias is a more serious problem. This problem is, however, not unique to PAL, but is a problem shared by all tools that provide pre-annotations. We have previously published an outline for an annotation interface for text chunk annotation that would alleviate this problem (Skeppstedt, 2013). The idea is to always provide two alternative pre-annotations to the annotator, i.e., the two pre-annotations that are assessed as most probable by the pre-annotation functionality. No information will however be given to the annotator regarding which of these two possible pre-annotation choices is considered the most probable one. The annotator will thereby always be forced to make an active choice, and will not be biased towards an annotation decision made by the pre-annotation functionality.

The next step for the PAL tool will be to implement and evaluate this functionality.

4.3 Additional future work

Future work also includes the addition of functionality for creating the initial seed set that is required for starting the active learning process. In the current version of PAL, it is assumed that random selection of training data is applied to create this initial seed set. While this procedure is the standard technique used when applying active learning, it should be mentioned that there is research on other methods for creating a seed set. Knowledge of existing vocabularies can be leveraged, for example, by selecting seed set samples that contain instances of these vocabulary terms (Tomanek et al., 2007). There are also techniques that use unsupervised machine learning approaches to create the seed set. For instance, by clustering the unlabelled data and constructing the seed set by selecting data samples from different clusters, a more diverse seed set can be achieved (Debarr and Wechsler, 2009). These types of techniques can also be used in

the process of active selection of data samples (Symons et al., 2006; Settles and Craven, 2008)

Another branch of active learning research is concerned with techniques for estimating when the addition of more training data samples no longer contributes to an increase in classifier performance. There are a number of techniques for performing such an estimation (Olsson, 2008), but none of them are included in the functionality currently provided by PAL.

Apart from uncertainty sampling, there are also many other methods for performing active learning, as mentioned in the background. Although these methods are not included in the current version of PAL, its code structure makes it easy to add other PyStruct and Scikit-learn classifiers, as well as other methods for performing active learning. We hope that this will inspire others to further develop the package with classifiers and active learning methods that suit their annotation and classification tasks.

Yet another topic for possible future work includes a practical evaluation of the benefit for the user of the functionality provided by the PAL package. As previously mentioned, the package has been used for annotating text chunks signalling agreement and disagreement in a corpus of 6,000 sentences. The functionality provided by PAL was found useful in this context, but the annotation was performed by the developer of PAL, which makes the assessment of PAL's usefulness likely to be biased (Munzner, 2008). Instead, usability studies of PAL should be performed with independent annotators and for different types of annotation tasks.

5 Conclusion

Although there are a number of techniques for simplifying text annotation and for reducing the amount of data required for training a machine learning classifier, these techniques are not included as a standard procedure in text annotation projects. The reason may be that they are typically not included by default in annotation tools.

The aim of "PAL, a tool for Pre-annotation and Active Learning" is to take the first step towards changing this standard. The package provides a functionality that includes pre-annotation and active selection of training data.

The output of the pre-annotation is provided in the annotation format of the annotation tool BRAT, but it is a stand-alone package that can be adapted to other formats. The tool is freely available to clone or download from GitHub at:
https://github.com/mariask2/PAL-A-tool-for-Pre-annotation-and-Active-Learning.

Acknowledgements

This work was funded by the StaViCTA project, framework grant "the Digitized Society – Past, Present, and Future" with No. 2012-5659 from the Swedish Research Council (Vetenskapsrådet).

References

Albright, D., Lanfranchi, A., Fredriksen, A., Styler, 4th, W. F., Warner, C., Hwang, J. D., Choi, J. D., Dligach, D., Nielsen, R. D., Martin, J., Ward, W., Palmer, M., and Savova, G. K. (2013). Towards comprehensive syntactic and semantic annotations of the clinical narrative. *J Am Med Inform Assoc*.

Archambault, D., Hoßfeld, T., and Purchase, H. C. (2016). Crowdsourcing and Human-Centred Experiments (Dagstuhl Seminar 15481). *Dagstuhl Reports*, 5(11):103–126.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer, New York, NY.

Brants, T. and Plaehn, O. (2000). Interactive corpus annotation. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, Paris, France. European Language Resources Association (ELRA).

Campos, D., Lourencço, J., Nunes, T., Vitorino, R., Domingues, P., Matos, S., and Oliveira, J. L. (2013). Egas – collaborative biomedical annotation as a service. In *Proceedings of the Fourth BioCreative Challenge Evaluation Workshop*, volume 1, pages 254–259. http://www.biocreative.org/media/store/files/2013/ProceedingsBioCreativeIV_vol1_.pdf.

Chou, W.-c., Tzong-han Tsai, R., and Su, Y.-s. (2006). A semi-automatic method for annotating a biomedical proposition bank. In *ACL Workshop on Frontiers in Linguistically Annotated Corpora*, pages 5–12, Stroudsburg, PA, USA. Association for Computational Linguistics.

de Bruijn, B., Cherry, C., Kiritchenko, S., Martin, J. D., and Zhu, X. (2011). Machine-learned solutions for three stages of clinical information extraction: the state of the art at i2b2 2010. *J Am Med Inform Assoc*, 18(5):557–562.

Debarr, D. and Wechsler, H. (2009). Spam detection using clustering, random forests and active learning. In *CEAS 2009 – Sixth Conference on Email and Anti-Spam, Mountain View*, pages 16–17, Mountain View, California USA.

Dumitrache, A., Aroyo, L., Welty, C., Sips, R., and Levas, A. (2013). "Dr. Detective": combining gamification techniques and crowdsourcing to create a gold standard in medical text. In *Proceedings of the 1st International Workshop on Crowdsourcing the Semantic Web, Sydney, Australia, October 19, 2013*, pages 16–31, Aachen, Germany. CEUR-WS.org.

Fort, K., Adda, G., Sagot, B., and Mariani, J. (2011). Crowdsourcing for language resource development: Critical analysis of Amazon Mechanical Turk overpowering use. In *LTC, 5th Language and Technology Conference*, Poznan, Poland. https://hal.archives-ouvertes.fr/hal-00648187.

Fort, K. and Sagot, B. (2010). Influence of pre-annotation on pos-tagged corpus development. In *Proceedings of the Fourth Linguistic Annotation Workshop*, LAW IV '10, pages 56–63, Stroudsburg, PA, USA. Association for Computational Linguistics.

Freitag, D. (2004). Trained named entity recognition using distributional clusters. In *Conference on Empirical Methods in Natural Language Processing*, pages 262–269, Association for Computational Linguistics. Stroudsburg, PA, USA.

Fuoli, M. and Hommerberg, C. (2015). Optimising transparency, reliability and replicability: annotation principles and inter-coder agreement in the quantification of evaluative expressions. *Corpora*, 10(3):315–349.

Hanbury, A., Kazai, G., Rauber, A., and Fuhr, N., editors (2015). *Second International Workshop on Gamification for Information Retrieval*, Berlin/Heidelberg, Germany. Springer Verlag. Lecture Notes in Computer Science vol. 9022.

Henriksson, A. (2015). Learning multiple distributed prototypes of semantic categories for named entity recognition. *Int. J. Data Min. Bioinformatics*, 13(4):395–411.

Henriksson, A., Kvist, M., Dalianis, H., and Duneld, M. (2015). Identifying adverse drug event information in clinical notes with distributional semantic representations of context. *J Biomed Inform*, 57:333–49.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Prentice Hall, Saddle River, NJ, USA, second edition.

Konstantinova, N., de Sousa, S. C., Cruz, N. P., Maña, M. J., Taboada, M., and Mitkov, R. (2012). A review corpus annotated for negation, speculation and their scope. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, pages 3190–3195, Paris, France. European Language Resources Association (ELRA).

Kucher, K., Kerren, A., Paradis, C., and Sahlgren, M. (2016). Visual Analysis of Text Annotations for Stance Classification with ALVA. In Isenberg, T. and Sadlo, F., editors, *EuroVis 2016 - Posters*. The Eurographics Association.

Lingren, T., Deleger, L., Molnar, K., Zhai, H., Meinzen-Derr, J., Kaiser, M., Stoutenborough, L., Li, Q., and Solti, I. (2014). Evaluating the impact of pre-annotation on annotation speed and potential bias: natural language processing gold standard development for clinical named entity recognition in clinical trial announcements. *J Am Med Inform Assoc*, 21(3):406–13.

Liu, P., Joty, S., and Meng, H. (2015). Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1433–1443, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mikolov, T. (2013). https://code.google.com/archive/p/word2vec/.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Miller, S., Guinness, J., and Zamanian, A. (2004). Name tagging with word clusters and discriminative training. In *Proceedings of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL HLT)*, pages 337–342, Stroudsburg, PA, USA. Association for Computational Linguistics.

Morton, T. and LaCivita, J. (2003). Wordfreak: An open tool for linguistic annotation. In *Proceedings of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Demonstrations - Volume 4 (NAACL HLT)*, pages 17–18, Stroudsburg, PA, USA. Association for Computational Linguistics.

Müller, A. C. (2013). What is structured learning? https://pystruct.github.io/intro.html (Accessed 2016-09-29).

Müller, A. C. and Behnke, S. (2014). PyStruct - learning structured prediction in Python. *Journal of Machine Learning Research*, 15:2055–2060.

Munzner, T. (2008). Process and pitfalls in writing information visualization research papers. In Kerren, A., Stasko, J. T., Fekete, J.-D., and North, C., editors, *Information Visualization: Human-Centered Issues and Perspectives*, pages 134–153. Springer, Berlin/Heidelberg, Germany.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Neves, M. and Leser, U. (2012). A survey on annotation tools for the biomedical literature. *Briefings in Bioinformatics*, 15(2):327–340.

Ogren, P., Savova, G., and Chute, C. (2008). Constructing evaluation corpora for automated clinical named entity recognition. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, pages 3143–3149, Paris, France. European Language Resources Association (ELRA).

Olsson, F. (2008). *Bootstrapping Named Entity Annotation by Means of Active Machine Learning*. PhD thesis, University of Gothenburg. Faculty of Arts.

Paradis, C. and Eeg-Olofsson, M. (2013). Describing sensory experience: The genre of wine reviews. *Metaphor and Symbol*, 28(1):22–40.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Paris, France. European Language Resources Association (ELRA).

Schein, A. I. and Ungar, L. H. (2007). Active learning for logistic regression: an evaluation. *Mach. Learn.*, 68(3):235–265.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report #1648, University of Wisconsin–Madison, http://research.cs.wisc.edu/techreports/2009/TR1648.pdf.

Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (EMNLP), pages 1070–1079, Stroudsburg, PA, USA. Association for Computational Linguistics.

Skeppstedt, M. (2013). Annotating named entities in clinical text by combining pre-annotation and active learning. In *Proceedings of the Student Research Workshop at ACL*, pages 74–80, Stroudsburg, PA, USA. Association for Computational Linguistics.

Skeppstedt, M., Sahlgren, M., Paradis, C., and Kerren, A. (2016). Unshared task: (Dis)agreement in online debates. In *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*, Stroudsburg, PA, USA. Association for Computational Linguistics.

South, B. R., Mowery, D., Suo, Y., Leng, J., Ferrández, Ó., Meystre, S. M., and Chapman, W. W. (2014). Evaluating the effects of machine pre-annotation and an interactive annotation interface on manual de-identification of clinical text. *J Biomed Inform*, 50:162–72.

Stenetorp, P., Pyysalo, S., Topic, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). BRAT: a web-based tool for NLP-assisted text annotation. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 102–107, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sutton, C. and McCallum, A. (2006). An introduction to conditional random fields for relational learning. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, USA.

Symons, C. T., Samatova, N. F., Krishnamurthy, R., Park, B. H., Umar, T., Buttler, D., Critchlow, T., and Hysom, D. (2006). Multi-criterion active learning in conditional random fields. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '06, pages 323–331, Washington, DC, USA. IEEE Computer Society.

Taboada, M. and Carretero, M. (2012). Contrastive analyses of evaluation in text: Key issues in the design of an annotation system for attitude applicable to consumer reviews in English and Spanish. *Linguistics and the Human Sciences*, 6(1-3):275–295.

Täckström, O., McDonald, R., and Uszkoreit, J. (2012). Cross-lingual Word Clusters for Direct Transfer of Linguistic Structure. In *Proceedings of the North American Chapter of the Associ- ation for Computational Linguistics on Human Language Technology (NAACL HLT)*, pages 477–487, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tomanek, K. (2010). *Resource-Aware Annotation through Active Learning*. PhD thesis, Technical University of Dortmund.

Tomanek, K., Daumke, P., Enders, F., Huber, J., Theres, K., and Müller, M. (2012). An interactive de-identification-system. In *Proceedings of SMBM 2012 - The 5th International Symposium on Semantic Mining in Biomedicine*, pages 82–86, Zürich, Switzerland. Institute of Computational Linguistics, University of Zurich.

Tomanek, K., Wermter, J., and Hahn, U. (2007). Efficient annotation with the Jena ANnotation Environment (JANE). In *Proceedings of the Linguistic Annotation Workshop*, pages 9–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tong, S. and Koller, D. (2002). Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66.

Uzuner, Ö., Solti, I., Xia, F., and Cadag, E. (2010). Community annotation experiment for ground truth generation for the i2b2 medication challenge. *J Am Med Inform Assoc*, 17(5):519–523.

Venhuizen, N., Basile, V., Evang, K., and Bos, J. (2013). Gamification for word sense labeling. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, pages 397–403, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yimam, M. S., Biemann, C., Eckart de Castilho, R., and Gurevych, I. (2014). Automatic annotation suggestions and custom annotation layers in WebAnno. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 91–96, Stroudsburg, PA, USA. Association for Computational Linguistics.

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
$\lambda_V : V \to \Sigma$	a mapping from a node to a node label
$\lambda_E : E \to \Sigma$	a mapping from an edge to an edge label
$\mu : (V \cup E) \times R \to S$	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. `DiscourseDocumentGraph`s 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.

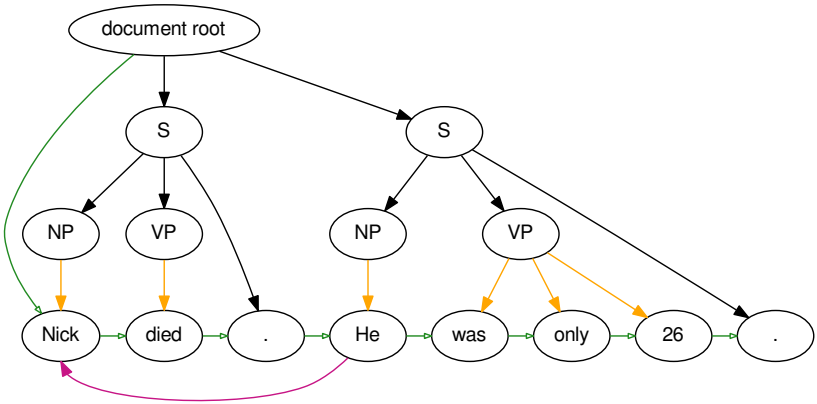


Figure 1: Example document annotated with phrase structure and coreference using all types of relations (directed edges) available in `discoursegraphs`. **Dominance relations** (black) hold between the document root node and the root nodes of the sentences it contains, as well as between constituents 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`

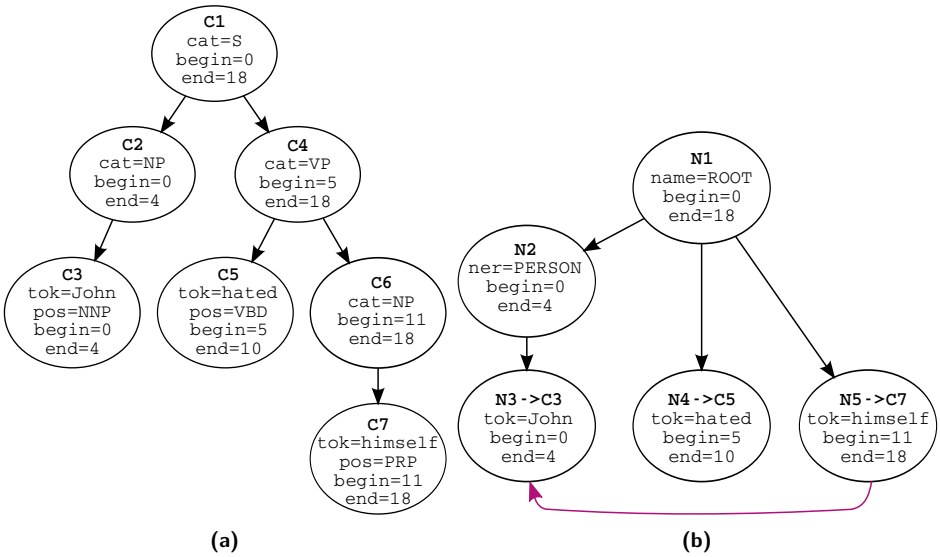


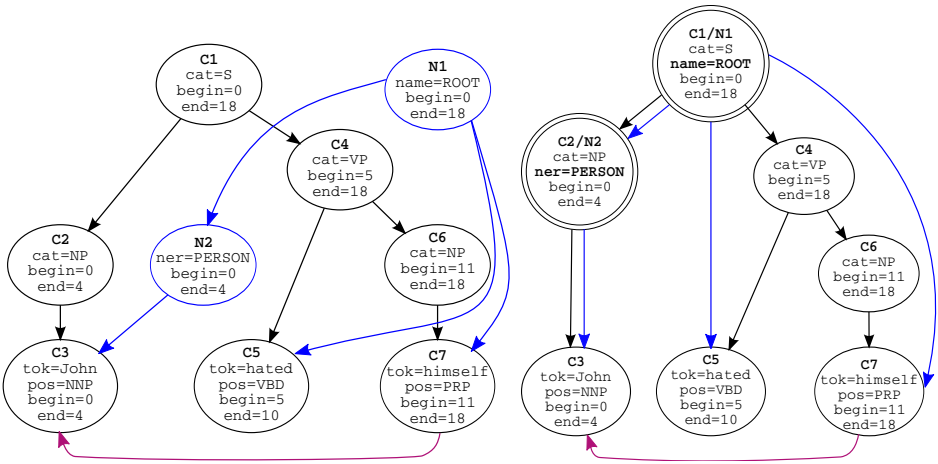
Figure 2: Two simplified document graphs representing the constituent parse tree (a), as well as the named entity and coreference annotation (b) of the sentence "*John hates himself*". The coreference is signaled by a pointing relation (purple).

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.

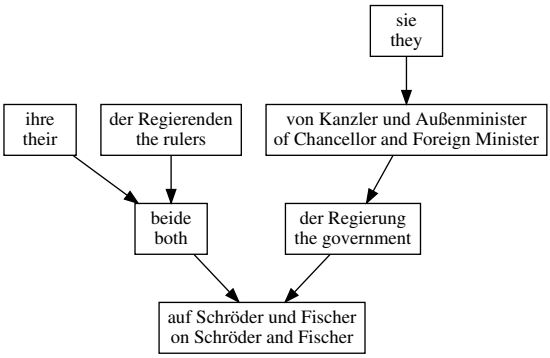


Figure 4: A graph representing three (wrongly) annotated coreference chains which share some nodes. If we accept that "Chancellor and Foreign Minister" and "the government" are coreferent, all the entities in this graph should be merged into one coreference chain.

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$_t$ool/*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*.

Thomas Schmidt

Construction and Dissemination of a Corpus of Spoken Interaction - Tools and Workflows in the FOLK project

Abstract

This paper is about the workflow for construction and dissemination of FOLK (Forschungs- und Lehrkorpus Gesprochenes Deutsch – Research and Teaching Corpus of Spoken German), a large corpus of authentic spoken interaction data, recorded on audio and video. Section 2 describes in detail the tools used in the individual steps of transcription, anonymization, orthographic normalization, lemmatization and POS tagging of the data, as well as some utilities used for corpus management. Section 3 deals with the DGD (Datenbank für Gesprochenes Deutsch - Database of Spoken German) as a tool for distributing completed data sets and making them available for qualitative and quantitative analysis. In section 4, some plans for further development are sketched.

1 Introduction

FOLK, the Forschungs- und Lehrkorpus Gesprochenes Deutsch (Research and Teaching Corpus of Spoken German) is being constructed in the program area "Oral Corpora" of the Institute for the German Language (IDS). Recognizing the lack of a larger, publicly available digital resource for studying spoken German in interaction (Deppermann/Hartung 2010), FOLK was started in 2009 as a long-term project to compile a diverse and systematic collection of audio and video recordings of spontaneous, authentic interactions across the whole spectrum of verbal interaction in German society.

FOLK is growing steadily, both in terms of quantity and variety of transcribed interactions. In its latest version (April 2016), the corpus comprises 219 interactions corresponding to 169 hours of audio and video recordings and 1.6 million transcribed word tokens. As testified by close to 6000 registrations (January 2017) for the Database of Spoken German (DGD, Schmidt 2014) through which FOLK is distributed and in which it is by far the most used corpus (Fandrych et al. 2016), the research community shows great interest in this resource.

To maximize (re)usability of the data, FOLK follows (and partly helps to define) current best practices in the handling and processing of data with respect to technological, methodological and legal issues (see also Schmidt 2016). In this paper, I am going to concentrate on the technological instruments, more specifically the software tools, which are used in the different steps of the corpus construction and dissemination workflow. Since FOLK is a spoken language corpus, the individual tools that make up this workflow differ fundamentally from the tools used in the compilation of written language corpora. Most importantly, the "primary" data of FOLK cannot be acquired automatically: recording authentic interactions requires an appropriate field access which must be negotiated for each individual case, and after data from the field have been obtained, project members have to screen and assess, and

finally transcribe, them "manually"[1]. As described in Kupietz/Schmidt (2015), these two "bottlenecks" – field access and transcription – still prevent oral corpora from growing to the same dimensions as written corpora, and the transcription bottleneck makes up a great part of the technological challenges which FOLK faces.

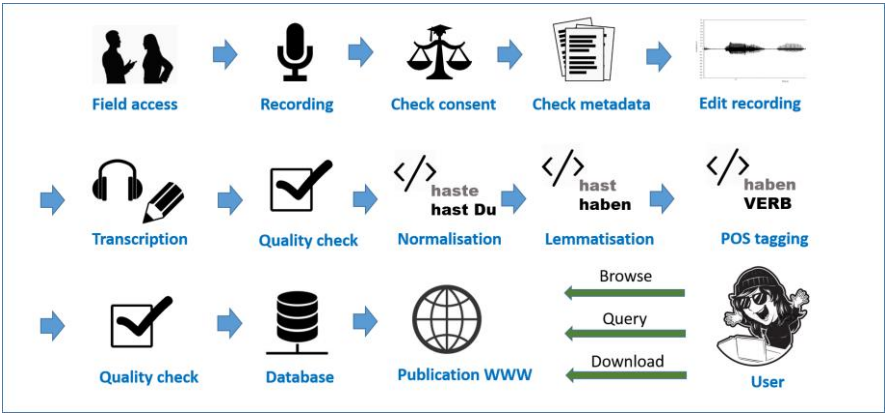


Figure 1: FOLK workflow overview

Figure 1 depicts (a simplified version of) the corpus compilation workflow in FOLK from the moment of field access to the final step of data publication. As a matter of principle, we do not start work on the data until the project coordinator has bindingly confirmed that all consent and metadata forms belonging to the recorded interaction are complete and usable. Once the data have passed this "gatekeeper" stage, recordings are prepared in the project's media studio for transcription. Depending on the recording, this step involves an appropriate conversion, cutting, denoising and/or normalization of the audio file as well as a synchronisation of different media files in cases where an interaction has been recorded in more than one file. Standard professional ("commercial") audio and video editing software (such as Samplitude and Adobe Premiere) is used for these tasks, which shall not be described in further detail here. Specialized linguistic tools come into play once an edited recording has been distributed to a student assistant for transcription.

Section 2 of this contribution starts at this stage, describing in detail the tools used in the individual steps of transcription, anonymization, orthographic normalization, lemmatization and POS tagging of the data, as well as some utilities used for corpus management. Section 3 then deals with the Database of Spoken German as a tool for distributing completed data

[1] It has been noted (p.c.) that "intellectually" may be the more appropriate term in this context, since it is the researcher's informed involvement with the material (judging the authenticity and quality of a recording, taking interpretative decisions in the transcription process, etc.), rather than pure manual labour, that is decisive. I am using the word "manual" here because it is the most commonly used antonym to "automatic" when speaking about processing methods for language resources.

sets and making them available for qualitative and quantitative analysis. In section 4, some plans for further development are sketched.

2 Tools for Corpus Compilation

2.1 Transcription and Anonymization: FOLKER

FOLKER – the FOLK EditoR (see also Schmidt/Schütte 2010) – is based on, and to a great part reuses data models and code of, the EXMARaLDA system (Schmidt/Wörner 2014). The crucial difference to its "mother system" is that FOLKER is optimized for the particular task of transcription in the FOLK project. This means, first, that functionality not required in FOLK (such as manual alignment of existing legacy transcripts or free annotation in an arbitrary number of dependent tiers) is removed, thus reducing the complexity of the user interface, making it quicker and easier to learn for student transcribers and decreasing the number of opportunities for making errors in the transcription process. Second, in contrast to the EXMARaLDA Partitur-Editor (and similar tools like ELAN or Praat) which always presents data in a musical score view, FOLKER offers three different forms of data visualisation, each of which is particularly suitable for a specific step in the workflow. Third, FOLKER has direct built-in support for the transcription guideline of the FOLK project, the cGAT system. The following sections describe the functionality of FOLKER in more detail.

2.1.1 Transcription

Transcription in FOLK is done according to the guidelines for the cGAT minimal transcript (Schmidt et al. 2015). cGAT is based on the GAT2 system (Selting et al. 2009) which can be considered one of the most widely established transcription conventions in (German) conversation analysis and related fields. A cGAT minimal transcript requires careful transcription of individual words in modified orthography ("literarische Umschrift" – literary transcription, sometimes also referred to as "eye dialect"), a precise measurement of silences above 0.2s duration and a description of audible non-verbal interaction phenomena (like breathing, coughing or laughing). Aiming at a minimization of interpretative decisions in transcription, the cGAT minimal transcript does not require the identification of linguistic units above the word level (such as intonation phrases), the annotation of prosodic details (like primary accent or lengthening of syllables, voice quality, speed and modulation of speech) or comments interpreting individual parts of utterances (such as "ironic").

The initial transcription according to these guidelines is done in FOLKER's segment view (see figure 2). FOLK transcribers select suitable segments, typically of 2 to 5 seconds duration, in the wave form visualisation of the audio recording, and create a time-aligned segment of the recording (start and end times in the first and second column) which is assigned to a speaker (third column) and for which the actual transcription text can be entered (column 4). During transcription, this text is checked for formal compliance with the cGAT conventions. If an error is detected (such as the missing closing bracket for the pause in segment 26), this is indicated by a red cross (column 5), otherwise a green check mark

confirms to the transcriber that the text entered can be parsed according to cGAT. Likewise, each segment is checked for "self-overlaps" with other segments, i.e. a red cross would indicate (in column 6) whenever the time intervals corresponding to two segments assigned to the same speaker overlap. All properties of a segment are freely editable at any time in the transcription process: time alignment can be adjusted, speaker assignment corrected, and transcription text modified whenever necessary.

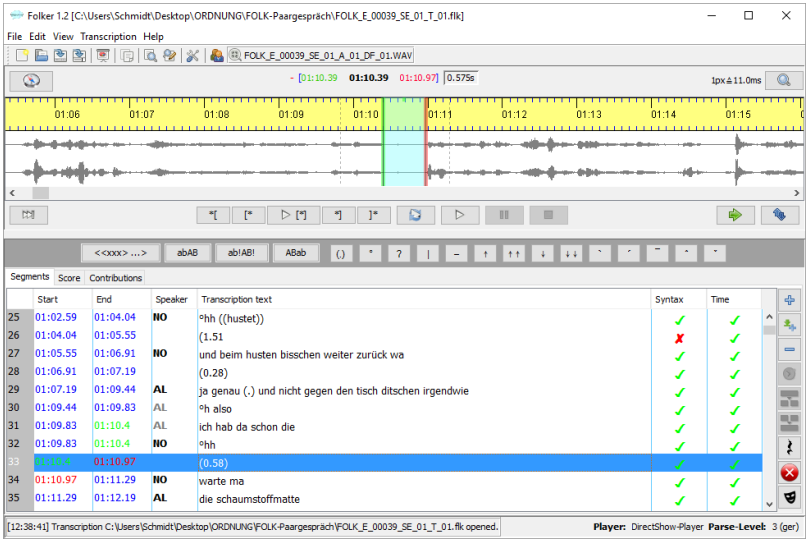


Figure 2: FOLKER's segment view

Although FOLKER is not meant for video transcription proper – meaning the systematic annotation of (non-verbal) behaviour visible in video images – a video file can be loaded into the editor in addition to the audio file to facilitate speaker identification and the understanding of passages with no or little verbal output (figure 3).

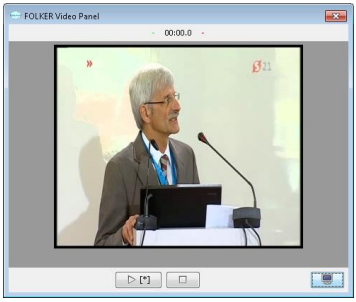


Figure 3: FOLKER's video panel

In cases of simultaneous or overlapping speech (a ubiquitous property of the types of interaction included in FOLK), transcribers first create independent segments for each speaker. The precise specification of the start and end of an overlap of one speaker's segment in relation to another speaker's segment can then be carried out by switching to FOLKER's musical score ("Partitur") view whose two-dimensional layout presents temporal relations in a more intuitive way than the segment view (see figure 4).

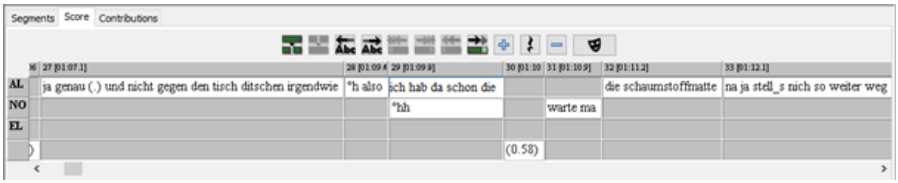


Figure 4: FOLKER's musical score ("Partitur") view

While transcription itself is thus done in the segment and musical score view with transcribers freely switching between the two as appropriate, a third view is used for intermediate or concluding quality checks on the data. In the contribution view (figure 5), consecutive segments assigned to the same speaker are summarized into the larger unit of a speaker contribution. This visualisation makes it easier to read the transcription as a whole and thus makes the correction process more efficient.

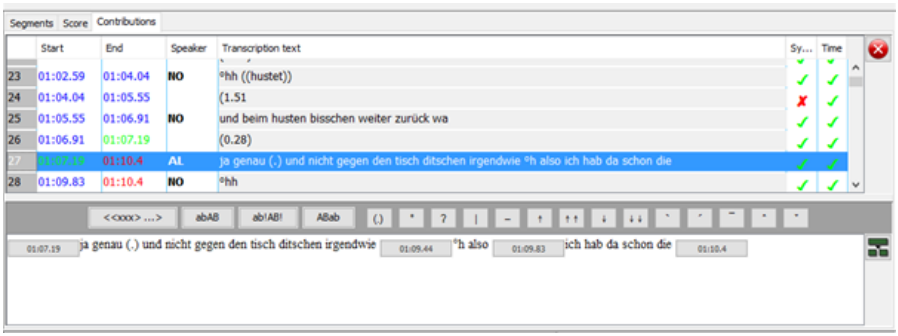


Figure 5: FOLKER's contribution view

2.1.2 Data format

FOLKER reads and writes an XML data format in which all relevant entities of the transcription – recordings, speakers, timepoints, speaker contributions – are represented as elements, and their relationships – speaker assignment, temporal alignment – encoded via IDREF/ID pointers. For a transcript which follows in its entirety the conventions for the cGAT minimal transcript (i.e. for which a transcriber sees nothing but green check marks in

the respective columns of the editor), FOLKER can parse the transcription text, resulting in additional markup of word, pause, breathing etc. elements underneath the speaker contributions. Figure 6 shows the XML corresponding to the transcript excerpt used in the previous section.

```
<speakers>
    <speaker speaker-id="NO"><name>Norbert</name></speaker>
    <speaker speaker-id="EL"><name>Elena</name></speaker>
</speakers>

<recording path="FOLK_E_00039_SE_01_A_01_DF_01.WAV"/>

<timeline>
    <timepoint timepoint-id="TLI_0" absolute-time="0.0"/>
    <!-- [...] -->
    <timepoint timepoint-id="TLI_25" absolute-time="65.555"/>
    <timepoint timepoint-id="TLI_26" absolute-time="66.915"/>
    <timepoint timepoint-id="TLI_27" absolute-time="67.195"/>
    <timepoint timepoint-id="TLI_28" absolute-time="69.44"/>
    <timepoint timepoint-id="TLI_29" absolute-time="69.83"/>
    <timepoint timepoint-id="TLI_30" absolute-time="70.4"/>
    <!-- [...] -->
</timeline>

<contribution speaker-reference="NO" start-reference="TLI_25" end-reference="TLI_26">
    <w>und</w><w>beim</w><w>husten</w><w>bisschen</w>
    <w>weiter</w><w>zurück</w><w>wa</w>
</contribution>
<contribution start-reference="TLI_26" end-reference="TLI_27" parse-level="2">
    <pause duration="0.28"/>
</contribution>
<contribution speaker-reference="AL" start-reference="TLI_27" end-reference="TLI_30">
    <w>ja</w><w>genau</w>
    <pause duration="micro"/>
    <w>und</w><w>nicht</w><w>gegen</w><w>den</w>
    <w>tisch</w><w>ditschen</w><w>irgendwie</w>
    <time timepoint-reference="TLI_28"/>
    <breathe type="in" length="1"/>
    <w>also</w>
    <time timepoint-reference="TLI_29"/>
    <w>ich</w><w>hab</w><w>da</w><w>schon</w><w>die</w>
</contribution>
```

Figure 6: FOLKER's XML format

The FOLKER XML format, in its unparsed as well as in its parsed version, and also with additional lemma and POS information for the tokens (see sections 2.2 and 2.3), is isomorphic and can be easily transformed to the TEI-based ISO standard "Transcriptions of Spoken Language" (ISO/ TC 37/SC 4/WG 6, cf. Schmidt/Hedeland/Jettka 2017), published in August 2016. FOLKER as well as OrthoNormal offer export filters, and the DGD enables users to download FOLK excerpts in this format (see section 3.3). In future developments, we will make sure to maintain interoperability with the ISO standard, and, eventually, rebase the whole FOLK workflow on it.

2.1.3 Anonymization / Pseudonymization / Masking

Recordings in FOLK are done with informed consent of the speakers wherever this is legally required (i.e. almost always). The standard consent form (for audio recordings[2]) guarantees that all information that could lead to direct identification of an individual, in particular the mention of individuals' names, addresses or other specific biographic details, are replaced in metadata and transcriptions with suitable pseudonyms and in the recordings with a silence or noise. Identifying the places to be masked in the recordings and maintaining a consistent set of pseudonyms for use in the transcription is another laborious task requiring adequate tool support. In most cases, the best moment to decide on anonymization issues is during the transcription process itself, when transcribers carefully listen to the recordings anyway and can thus be sure to notice mentions of proper names etc. FOLKER therefore includes a set of functions which support transcribers in this task. Whenever an anonymization issue is identified, the corresponding part of the recording can be selected and a masking segment created which is stored separately from the transcription. In order to ensure consistency in the choice of pseudonyms (i.e. to make sure that one and the same name is always replaced by one and the same pseudonym), transcribers can create and maintain a table of mappings from real names to pseudonyms (see figure 7).

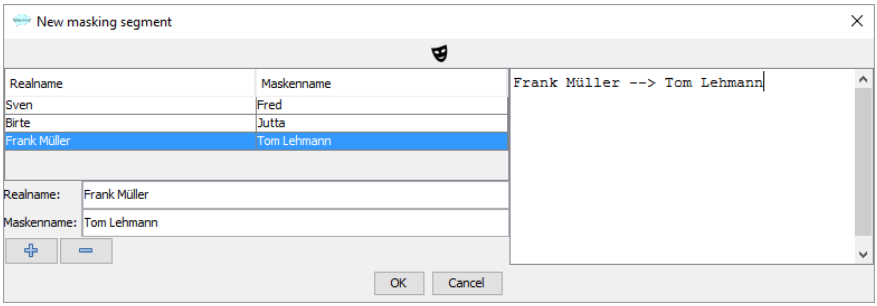


Figure 7: Creating a mask segment and managing pseudonyms in FOLKER

Since anonymization is, ultimately, not a decision for which student transcribers can or should take full responsibility, completed transcripts and the anonymizations proposed by the transcribers are checked by a project coordinator. For this task, FOLKER offers a summary of all existing masking segments as illustrated in figure 8.

[2] The same anonymization principles apply to the sound track of video recordings. We do not, however, attempt to anonymize the video image, for instance by blurring faces, since this would render the video useless for many analysis purposes. Instead, we obtain the speakers' consent to use the unmasked video image.

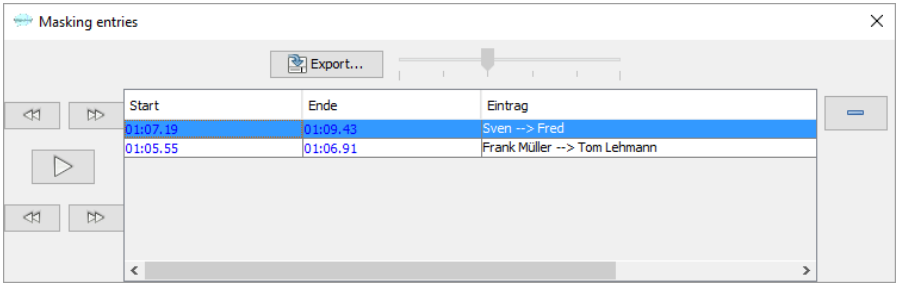


Figure 8: Overview of masking segments in FOLKER

When all masking segments for a given transcript have been identified and checked in this way, FOLKER can insert the required noises into the audio file automatically (figure 9). We use a Brownian noise, because, in contrast to a simple silence, this makes clear to the listener that he is dealing with an artefact in the recording, and because, compared to a white noise, it is less disagreeable to the ear. Masking information is stored in a separate section of the document so that it can be easily removed before publication of the data. We archive this information internally in order to be able to quickly identify masked passages later.

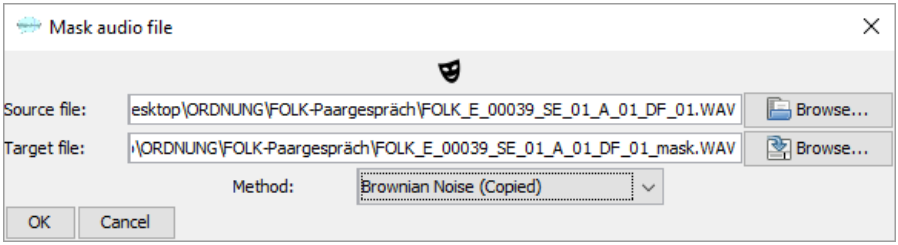


Figure 9: Automatic masking of an audio file via FOLKER

2.2 Orthographic normalisation: OrthoNormal

As described above, primary transcription in FOLK is done according to cGAT, meaning that all word tokens are written in lower case, and that deviations from standard pronunciation are modelled by using a modified orthography (e.g. “zwo” as a colloquial pronunciation of the number 2 or “haste” as a contracted form of “hast Du”, have you – “dunno” for “don’t know” would be an analogous case for English). While this has the advantage of following conversation analytic tradition and making spoken language phenomena more readily visible in the transcription text, it also has the disadvantage of rendering queries and further automatic processing on this data unreliable. In order to optimize FOLK data for the application of corpus linguistic and computational linguistic methods, we therefore add a second annotation layer in which tokens in modified orthography are mapped to a standard orthographic

equivalent.[3] This is done on a token-by-token basis with the help of the OrthoNormal annotation tool using a set of normalisation guidelines (Schmidt/Winterscheid 2015).

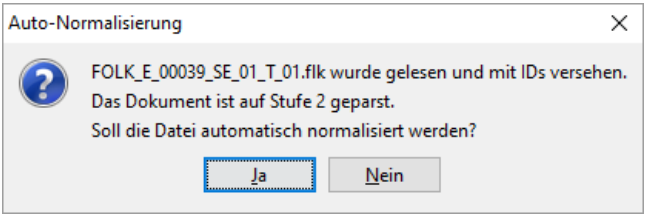


Figure 10: Automatic normalisation in OrthoNormal

When a completed FOLKER transcript is loaded into OrthoNormal, an automatic normalisation step can be applied to all word tokens (see figure 10). This method proceeds as follows:

1 It looks up each word in a normalisation lexicon in which (manually verified) transcription/normalisation pairs of previous normalisations are stored with their frequencies.[4] Whenever a form is encountered that has an entry in this lexicon, the most frequent corresponding normalised form is automatically inserted. As an example, see the form "hab" in figure 11 which has been correctly normalised to "habe" (*have*, first person singular), but also the form "wa" which has been incorrectly normalised to "wir" (*we*, where "was" – *what* would have been correct).

2 It looks up each word in a list of word forms that only occur in upper case in German, extracted from the DeReWo list of inflected forms (Institut für Deutsch Sprache 2014) which itself is based on the billion words DeReKo corpus of written German (Kupietz/Schmidt 2015). If no lexicon entry for a given form has been found in step (1) and the form with an upper-case initial is found in the word list, this upper case form is inserted as the normalised form. As an example, see the form "tisch" in figure 11 which has been correctly normalised to "Tisch" – *table*.

[3] As a reviewer has duly pointed out, other projects such as Verbmobil have proceeded in the reverse manner, i.e. standard orthography was used in the primary transcription and pronunciation deviations added as annotations to the orthographic words. Our choice to transcribe in modified orthography is mainly motivated by the fact that this is the standard procedure in conversation analysis and therefore more easily reconcilable with existing transcription conventions. Furthermore, FOLK data abound with dialectal and other features of spontaneous speech so that the rate of forms deviating from standard orthography is rather high (more than 50% of all tokens in some cases). A partial automation of the mapping between modified and standard forms is therefore important for reasons of efficiency, and it is obviously much easier to automatically map modified onto standard forms than the other way around.

[4] This lexicon is updated with each release of FOLK, i.e. it grows by the manually verified normalization entries for roughly 300.000 transcribed tokens each year.

3 It checks all word forms against a full list of inflected German word forms (again, based on DeReWo). If a form is not found in the list, it will be marked as a likely normalisation candidate for the manual normalisation process. As an example, see the forms “aufnahmejerät” (=”Aufnahmegerät” – *recording device*) and “ooch” (=”auch” – *too* – in Berlin dialect), both highlighted in red in the table on the right hand side of figure 11.

This simple process leads to recall and precision rates both roughly around 80%, meaning that 80% of forms that need to be normalised are detected in that process and that 80% of all automatically inserted normalised forms are correct. Since the normalisation layer is absolutely crucial for all further processing steps, the automatically normalised transcripts with this error rate are manually checked and corrected by student annotators. The OrthoNormal tool makes this step efficient by offering an ergonomic interface optimised for the task. As figure 11 illustrates, the interface is divided into three parts: the upper left part displays the transcript as a list of contributions with normalised forms added in red. When a contribution is selected in this list, the lower left part displays this contribution and makes it available for editing. Clicking on any word token in this view will bring up a normalisation dialog in which a normalised form can either be freely entered in a text field, or selected from a list of candidates extracted from the normalisation lexicon. The right part of the screen, finally, displays pairs of transcribed and normalised forms in a table. When ordered alphabetically, several transcribed forms can be normalised in one go in this table, and a regular expression filter can be used to select specific patterns of transcribed forms (such as: all forms starting with a certain prefix).

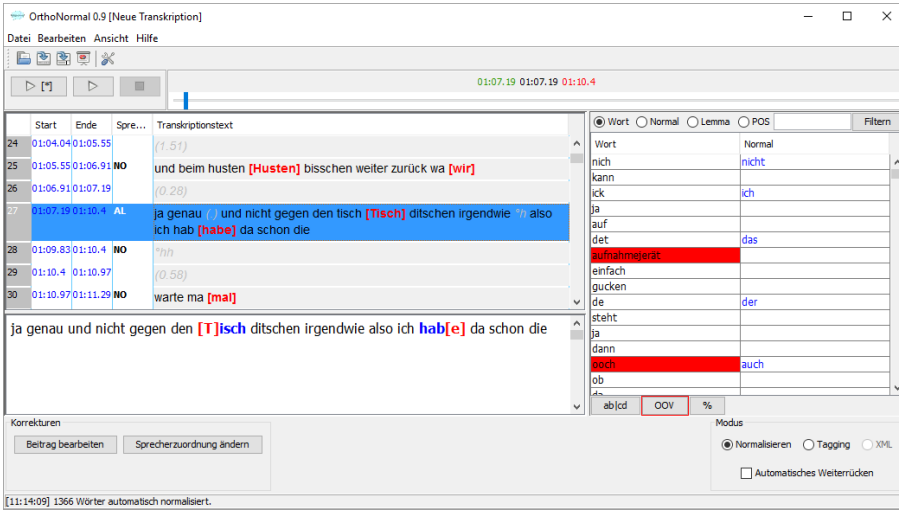


Figure 11: OrthoNormal user interface

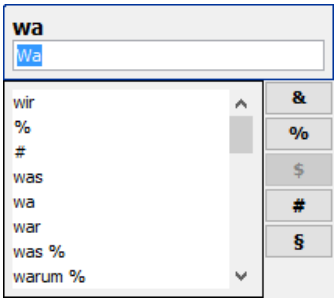


Figure 12: Normalisation dialog

Although manual normalisation means that a student annotator has to go through the whole transcript once more (and the result checked again by a supervisor), this step is by far less time-consuming than transcription itself. While we have to calculate with as much as 100 hours of manual work for the transcription of 1 hour of recording, the same amount of data can usually be orthographically normalised in less than 5 hours. The normalised forms are stored as @n attributes on the <w> elements of the original transcription (see figure 13).

```
<contribution speaker-reference="NO" start-reference="TLI_25" end-reference="TLI_26">
    <w id="w37">und</w>
    <w id="w38">beim</w>
    <w id="w39" n="Husten">husten</w>
    <w id="w40">bisschen</w>
    <w id="w41">weiter</w>
    <w id="w42">zurück</w>
    <w id="w43" n="was">wa</w>
</contribution>
```

Figure 13: XML of normalised transcript

2.3 Lemmatisation and POS-Tagging

Once the manually checked normalisation layer is available for a transcript, an automatic lemmatization and POS tagging can be carried out on the normalised data. We use TreeTagger (Schmid 1994) via TT4J (Eckart de Castilho, no data) to perform this task.

Up until the current version of FOLK, the default TreeTagger parameter file for German, trained on newspaper text with the Stuttgart-Tübingen-Tagset (STTS), was used to do the tagging. The results were acceptable only as a first approximation, because they had an error rate of over 10% for POS tags (less than 2% for lemmas), and because the tagset itself was underspecified especially for those word classes that are specific to spoken language (such as particles and interjections). Westpfahl (2015) therefore developed an extension of STTS optimized for the kind of spoken language data found in FOLK. In the FOLK project, a 100.000 tokens gold standard (Westpfahl/Schmidt 2016) was tagged manually according to this STTS extension, again by means of the OrthoNormal tool (in the "tagging" rather than the "normalisation" mode, see figure 14). Using this gold standard, a new TreeTagger pa-

rameter file was trained which can be used for lemmatization and POS tagging of future versions of FOLK. Evaluations have shown that an error rate as low as 5% can be attained with this improved procedure.

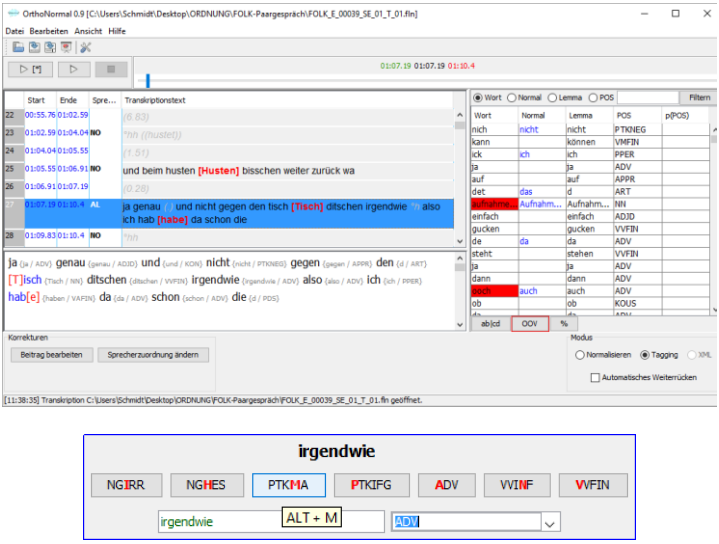


Figure 14: Using OrthoNormal do carry out manual correction on POS tags

Lemmas and POS tags are, again, stored as @lemma and @pos attributes, respectively, on the <w> elements of the original transcription (see figure 15).

```
<contribution speaker-reference="NO" start-reference="TLI_25" end-reference="TLI_26">
    <w id="w39" pos="KON" lemma="und">und</w>
    <w id="w40" pos="APPRART" lemma="beim">beim</w>
    <w id="w41" n="Husten" pos="NN" lemma="Husten">husten</w>
    <w id="w42" pos="ADV" lemma="bißchen">bisschen</w>
    <w id="w43" pos="ADV" lemma="weiter">weiter</w>
    <w id="w44" pos="PTKVZ" lemma="zurück">zurück</w>
    <w id="w45" pos="SEQU" lemma="Wa" n="was">wa</w>
</contribution>
```

Figure 15: XML of lemmatized and POS tagged transcript

2.4 Metadata

Metadata capturing salient characteristics of the interactions and speakers involved are collected alongside the recordings in the field by means of a project specific paper form. Once a recording is approved for inclusion in FOLK and the project coordinator has checked that consent and metadata forms for this recording are complete and consistent, metadata are transferred to a digital form. This is done with the help of an online interface based on the XMLSpy Editor (see figure 16). The interface is aware of the underlying XML schema (Gasch et al. 2008) and can thus support the entry process, for instance by providing closed vocabulary lists for values of appropriate fields.

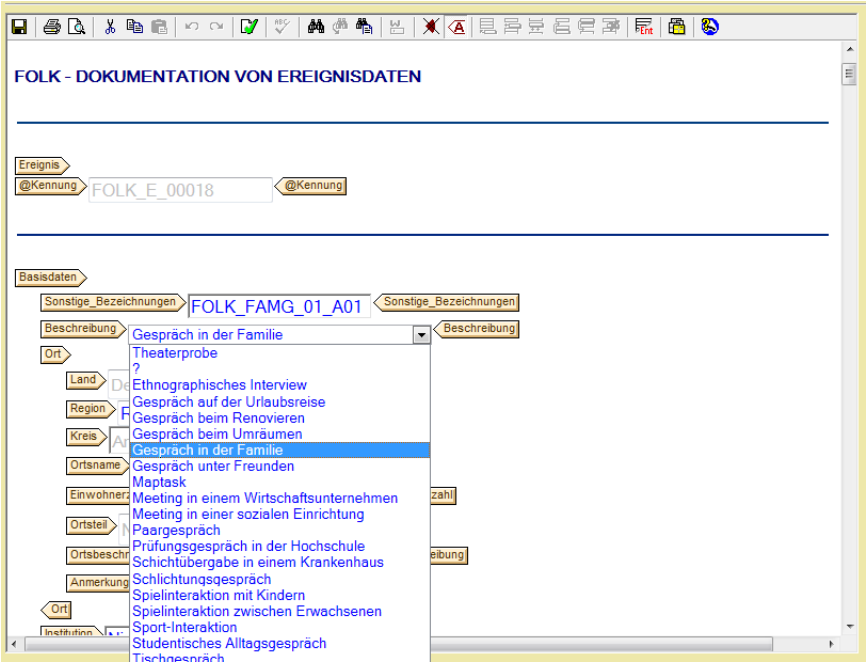


Figure 16: Web interface for entering metadata

2.5 Data Management

Because of the large amount of manual work necessary for transcribing and annotating the data, the FOLK project continuously employs a team of 10 to 15 student assistants. This, and the fact that the acquisition of new recordings cannot be planned centrally, but has to be managed individually for each new type of interaction with the respective cooperation partners, lead to a considerable administrational overhead. As the project progresses, we are attempting to develop tools not only for transcription and annotation itself, but also for supporting the project management in monitoring the workflows.

1 In order to monitor progress on individual transcription files, FOLKER offers transcribers the possibility to keep a transcription log, a simple list of logging entries with information about the time in which a transcript was edited, the name of the person who did the editing, and a free text field describing the editing steps carried out (see figure 17). When aggregated over a larger number of files, this information can also be used to measure transcription ratios.

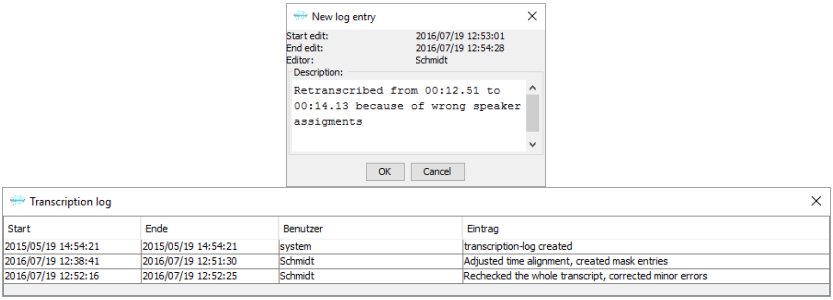


Figure 17: Transcription logs in FOLKER

2 In order to monitor progress of transcription and normalisation on the corpus as a whole, a set of batch scripts has been implemented to produce so-called snapshots of the current state of corpus development. A single click will start a process which runs through all folders in the project's working directory, calculates measurements for transcription progress (e.g. amount of audio available, amount of audio transcribed, number of files normalized, number of metadata entries completed) and generates HTML visualizations for transcript logs and transcription files. For instance, the snapshot depicted in figure 18 informs the project coordinator that, out of a total of 38.5 hours of recordings, roughly 31.5 hours have been transcribed at least in a first pass, and a little more than 13 hours have already entered the normalisation stage.

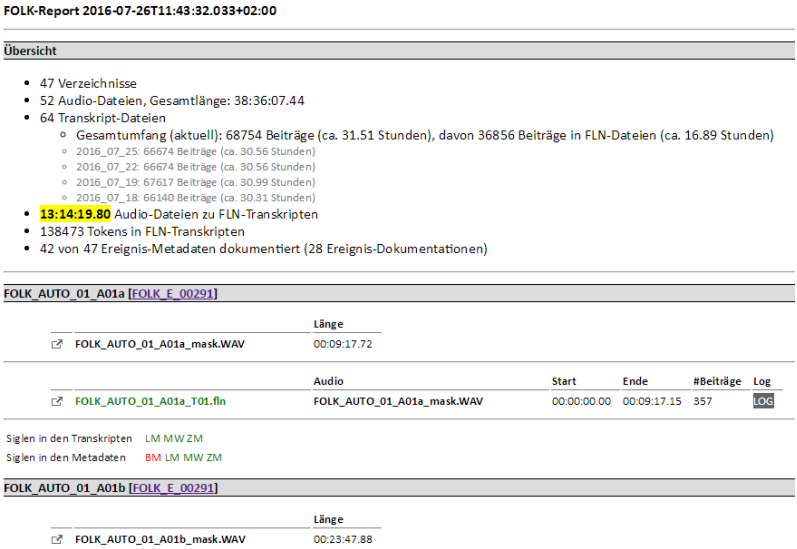


Figure 18: Report on progress in the transcription and annotation process

3 The workflow schema depicted in figure 1 is oversimplified in one important detail: it fails to capture cycles in the workflow that arise from the fact that transcriptions and

annotations of oral language data will always contain a portion of genuine errors. Some of these errors are discovered (by project members or users of the corpus) only after a piece of data has been declared complete and disseminated via the database. Although, owing to the different quality control steps in the workflow, this is rare, it is not rare enough to be simply ignored. An additional important part of the workflow is therefore the management of correction cycles. In order to control these cycles, we manage the transcription data (as well as the metadata, for which similar problems can occur) via Subversion (SVN) as a version control system and coordinate the yearly extension of FOLK with corrections that have in the meantime been applied to the already published part.

3. Tools for Corpus Analysis: Database for Spoken German (DGD)

The observation that "[...] a corpus by itself can do nothing at all, being nothing more than a store of used language" is no less true for oral corpora than for the written corpora Hunston (2002: 20) refers to. Corpus linguists need adequate tools not only for constructing but also for analysing corpora. In the case of FOLK, the Database for Spoken German (Datenbank für Gesprochenes Deutsch, DGD, http://dgd.ids-mannheim.de) is the principal means of making the corpus data available for analysis. The DGD in its present form (versions 2.x – following up on the predecessor system described in Fiehler/Wagner 2005), first released in 2012, acts as a platform not only for disseminating FOLK, but also various other oral corpora stored at the Archive for Spoken German (AGD).

The DGD allows for two principal approaches to oral corpus data:

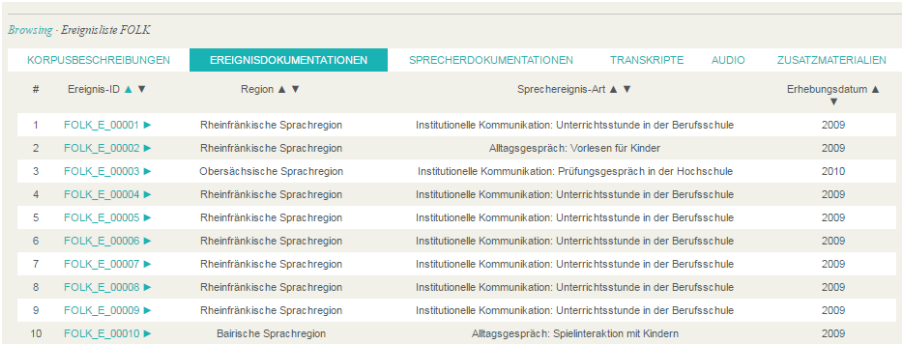
1 **Browsing** a corpus, i.e. reading corpus metadata and transcripts and listening to the corresponding (aligned) audio is a means of getting acquainted with a corpus, of exploring individual data sets in a holistic manner and of identifying and analysing in depth selected excerpts of transcripts. Related to this is the possibility of **downloading** selected excerpts and further processing them with suitable tools (e.g. for acoustic analysis, for additional annotations) on a local machine. The browsing approach is particularly suited for qualitative paradigms, such as conversation analysis.

2 **Querying** a corpus, i.e. searching through the entire data for all instances of a given annotation pattern and further manipulating and analysing the results of such searches. This is the functionality typically expected from a corpus interface to written language data, and it is equally central to the work with oral data. Corpus queries are essential for quantitative research paradigms, certain corpus linguistic methods being the most obvious case in point.

Of course, the real potential of a corpus like FOLK lies in an innovative mixture of these two approaches, and, as will be shown in the following sections, the DGD takes great care to enable users to effectively combine "semi-automatic" query methods with "manual" ways of inspecting the data.

3.1 Browsing / Collections

The browsing mode of the DGD gives access to individual data sets in FOLK. Starting from a tabular overview (see figure 19), users can navigate and view metadata on speech events and speakers, listen to audio files and read transcripts.



#	Ereignis-ID ▲ ▼	Region ▲ ▼	Sprechereignis-Art ▲ ▼	Erhebungsdatum ▲ ▼
1	FOLK_E_00001 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
2	FOLK_E_00002 ►	Rheinfränkische Sprachregion	Alltagsgespräch: Vorlesen für Kinder	2009
3	FOLK_E_00003 ►	Obersächsische Sprachregion	Institutionelle Kommunikation: Prüfungsgespräch in der Hochschule	2010
4	FOLK_E_00004 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
5	FOLK_E_00005 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
6	FOLK_E_00006 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
7	FOLK_E_00007 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
8	FOLK_E_00008 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
9	FOLK_E_00009 ►	Rheinfränkische Sprachregion	Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	2009
10	FOLK_E_00010 ►	Bairische Sprachregion	Alltagsgespräch: Spielinteraktion mit Kindern	2009

Figure 19: Tabular overview of FOLK speech events

Hyperlinks between the representations of the different data types allow for an exploration of the relationships between them. For instance, starting from the metadata for a given speaker, the speech event(s) this speaker participates in can be displayed, and from there, a link to the corresponding audio file(s) and transcript(s) is available. The transcript, in turn, contains links to the speech event and speaker metadata, and clicking on any word in the transcript will start playback of the corresponding part of the aligned audio (see figure 20).

Figure 20: Display of a transcript with aligned audio (current playback position indicated by the dashed line)

The default display shows the transcript text in modified orthography together with non-speech tokens (pauses etc.) in a line-for-line notation (one contribution per line). Alternatively, FOLK transcripts can be displayed as musical scores (see figure 21), which makes it easier to understand the temporal flow of events (especially simultaneous and overlapping speech), or in a "normalised" version which displays the text in standard orthography and omits all non-speech tokens, making it easier to read for users who are not familiar with the specialised transcription forms.

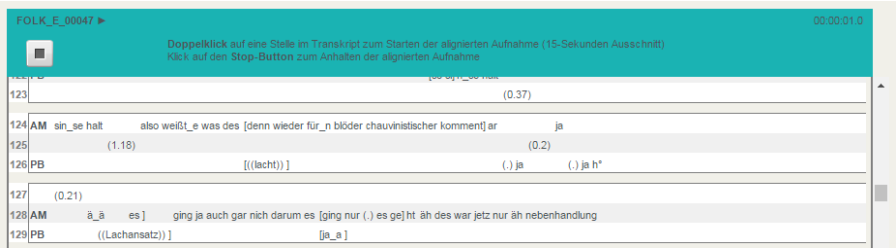


Figure 21: Transcript in musical score display

In order to retain relevant and interesting excerpts that are identified in the browsing process, users can add them to a collection and store them inside the database (figure 22).

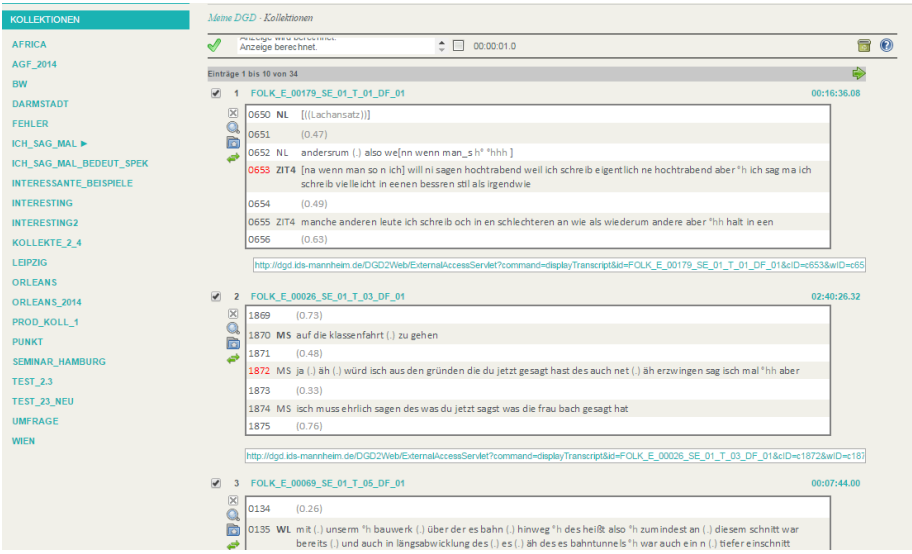


Figure 22: A user-generated collection of transcript excerpts

3.2 Query

Queries on DGD data can be carried out in three different manners.

First, a **full text search** is a simple way of searching through metadata and transcripts to obtain a global impression of the occurrences of a given term. Being realised via Oracle's full text functionality, the full text search is both relatively quick and flexible in so far as it does not make advanced assumptions about document structures. It can thus be used on different types of XML documents (such as metadata XML and FOLKER's transcript XML as illustrated above) and on plain text or PDF files (these are both file types in which the transcripts of some legacy corpora in the DGD are stored).

Since full text search, however, by definition, gets rid of most of the structure represented in XML elements and attributes, it is not suitable to exploit the FOLK data in its full complexity, including the different annotation levels and the links between transcripts and metadata. The DGD therefore also offers two types of 'structured' searches.

The **structured metadata query** is a means of finding speech events or speakers with certain properties. For instance, a structured metadata search could be used on FOLK to find all instances of interactions with male participants older than 30 years from Northern Germany (as in figure 23). The result of such a query is a list of matching speech events, possibly combined with a list of matching speakers. The list can be saved as a virtual corpus and used as a basis for structured token searches.

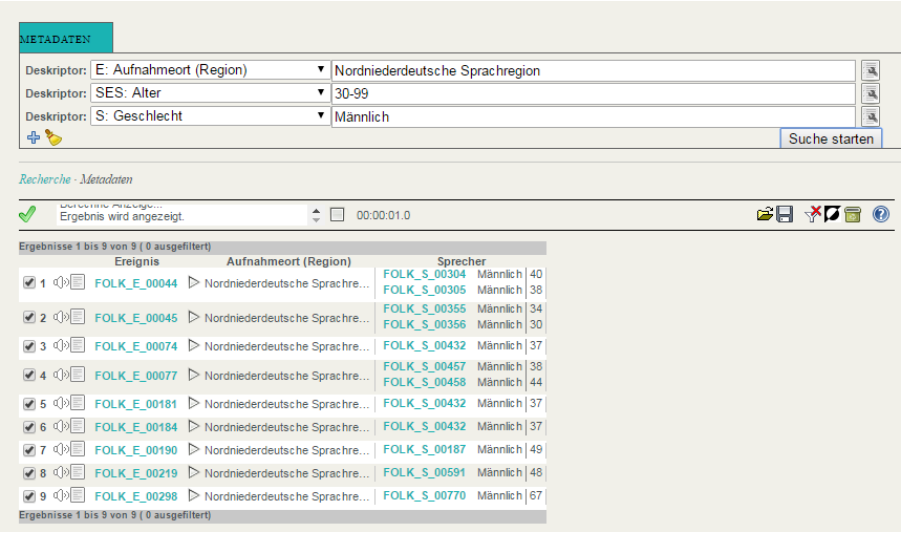


Figure 23: Metadata query on FOLK resulting in a virtual corpus

The **structured token query**, finally, is the core component of the DGD. Its base functionality is to allow the user to specify one or more properties of a token, such as its transcribed or orthographic form, its lemma and/or its part of speech, and to display as a KWIC con-

cordance all the matching tokens in the selected corpus or corpora (see figure 24). Properties can be specified as plain strings or as string patterns in the form of regular expressions.

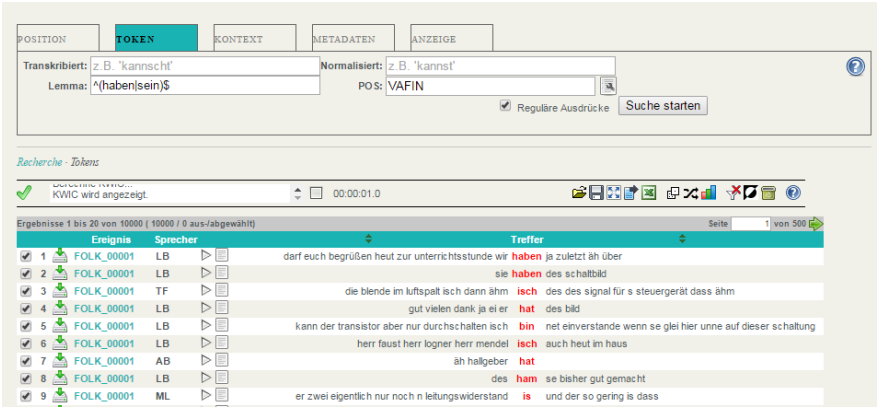


Figure 24: Token Query for lemmas ‚haben' or 'sein' as finite auxiliaries (POS=VAFIN)

A query will thus always start with a concordance for a single class of tokens. Additional functionality allows the user to further explore and refine this result.

The refinement can be done manually by inspecting individual search results. Audio playback for the corresponding part of the recordings is available directly from the KWIC concordance. For each line of the concordance, metadata about the corresponding speech event and speaker can be displayed. To explore the interaction context of the matching token, the corresponding transcript excerpt can be folded out underneath the KWIC line (see figure 25). In that way, automated query can be combined with detailed qualitative analyses. By deselecting individual lines of the concordance, users have the possibility to clean the result from false positives identified in that process.

Figure 25: KWIC with deselected lines (1,2 and 4) and transcript folded out (line 5)

It is also possible to use additional filters on a search result. Via the ‘Context’ tab, the properties of further tokens in the context of the matching token can be specified. ‘Context’ here can be limited to single contributions (the default case), to all contributions of the respective speaker or to the entire transcript. The context window can be specified as left, right or both-sided and in terms of token distance (see figure 26).

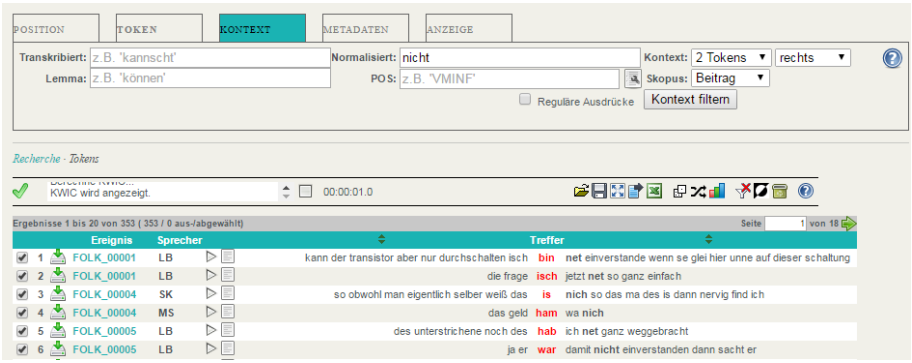


Figure 26: A search result filtered for the normalized form ‘nicht’ in a distance of two tokens in the right context

Using a context filter (also repeatedly, i.e. filtering first for one, then for another item in the context, which corresponds to a Boolean *and*) can thus serve to identify co-occurrences of two or more tokens. Items which do not match the filter will not be deleted immediately, but only deselected in the concordance. In that way, the effect of a filter can be evaluated (and, if necessary, reversed) in a transparent manner. Similarly, the ‘Metadata’ tab can be used to filter search results according to properties of the respective speech events or speakers. In an analogous manner to the structured metadata query, users can, for instance, specify a metadata filter for conversations including male speakers from a certain region.

A filter type specific to interaction data is implemented in the “Position” tab where the user can (before carrying out the actual token query) restrict searches to specific positions in the interaction, such as “within *n* tokens of the beginning/end of a contribution” / “within *n* tokens of a change of speaker” / “inside or immediately before/after an overlap” / “in the vicinity of a pause” (see figure 27). Making queries sensitive to the interactive structure of the FOLK data is especially useful for investigating phenomena which conversation analysis and related fields are interested in. In particular, it enables the study of functional aspects of certain items (such as discourse markers) in speakers’ organisation of turn-taking. An example would be corpus-guided studies of turn-initiations as described in Heritage (2013).

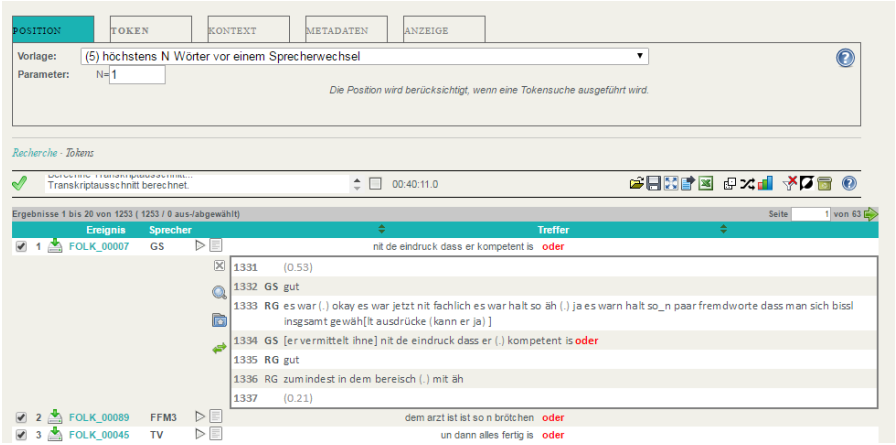


Figure 27: A query for the form ‚oder' restricted to the position immediately before a speaker change

Further operations can be carried out on the KWIC:

1 A KWIC can be stored in the database for later reuse, i.e. so that the underlying query and subsequent manual refinements do not have to be repeated.

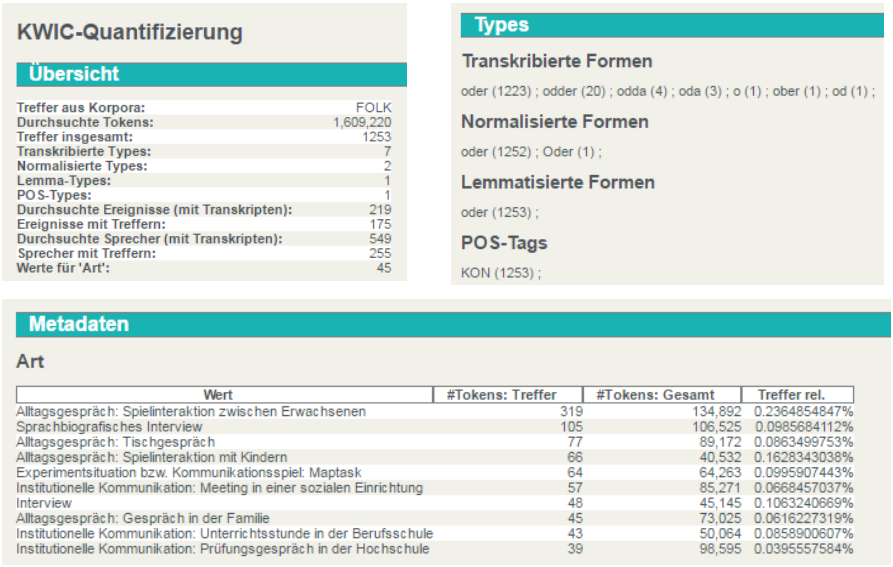
2 KWICs can be printed or exported to text or XML files. We notice that users especially value the possibility to export the KWIC in a file that can be further processed by spreadsheet applications such as MS Excel.

3 A random sample of an arbitrary size can be extracted from a KWIC. This is useful for obtaining a non-biased excerpt of a large result in order to keep manual inspection manageable.

4 KWICs can be scrambled randomly.

5 KWICs can be sorted according to any of the available columns. When sorting is applied to the left or right context column, it can serve to visually identify prominent co-occurrence patterns.

6 KWICs can be quantified, giving a concise summary of the number of tokens and types, of their combination with selected metadata parameters and of frequencies normalised with respect to the amount of data available (see figure 28).



KWIC-Quantifizierung

Übersicht

Treffer aus Korpora:	FOLK
Durchsuchte Tokens:	1,609,220
Treffer insgesamt:	1253
Transkribierte Types:	7
Normalisierte Types:	2
Lemma-Types:	1
POS-Types:	1
Durchsuchte Ereignisse (mit Transkripten):	219
Ereignisse mit Treffern:	175
Durchsuchte Sprecher (mit Transkripten):	549
Sprecher mit Treffern:	255
Werte für 'Art':	45

Types

Transkribierte Formen

oder (1223) ; odder (20) ; odda (4) ; oda (3) ; o (1) ; ober (1) ; od (1) ;

Normalisierte Formen

oder (1252) ; Oder (1) ;

Lemmatisierte Formen

oder (1253) ;

POS-Tags

KON (1253) ;

Metadaten

Art

Wert	#Tokens: Treffer	#Tokens: Gesamt	Treffer rel.
Alltagsgespräch: Spielinteraktion zwischen Erwachsenen	319	134,892	0.2364854847%
Sprachbiografisches Interview	105	106,525	0.0985684112%
Alltagsgespräch: Tischgespräch	77	89,172	0.0863499753%
Alltagsgespräch: Spielinteraktion mit Kindern	66	40,532	0.1628343038%
Experimentsituation bzw. Kommunikationsspiel: Maptask	64	64,263	0.0995907443%
Institutionelle Kommunikation: Meeting in einer sozialen Einrichtung	57	85,271	0.0668457037%
Interview	48	45,145	0.1063240669%
Alltagsgespräch: Gespräch in der Familie	45	73,025	0.0616227319%
Institutionelle Kommunikation: Unterrichtsstunde in der Berufsschule	43	50,064	0.0858900607%
Institutionelle Kommunikation: Prüfungsgespräch in der Hochschule	39	98,595	0.0395557584%

Figure 28: Various types of quantification for a search result: general figures (top left), types and tokens (top right), result counts relative to the metadata attribute 'interaction type' (bottom)

3.3 Download and Citation

The DGD's browsing and query functionality as described in the two previous section addresses most requirements concerning the discovery of data relevant for a given analysis purpose – if it is in the data, the DGD user has a good chance of finding it there. However, FOLK (and, in fact, oral corpora in general) captures in its transcriptions and annotations only a selected part of the phenomena audible in the recordings, and quite a few phenomena that can play a role for linguistic analysis are thus not available for complete analysis inside the DGD. Prominent cases in point are prosodic features of speech like intonation and stress which are not taken into account in FOLK's minimal transcriptions. An essential feature of the DGD is therefore that it enables the user to go beyond the information captured in the existing transcriptions and annotations and beyond the analysis functions offered by the web platform by downloading data for relevant excerpts onto a local computer and further processing them there.

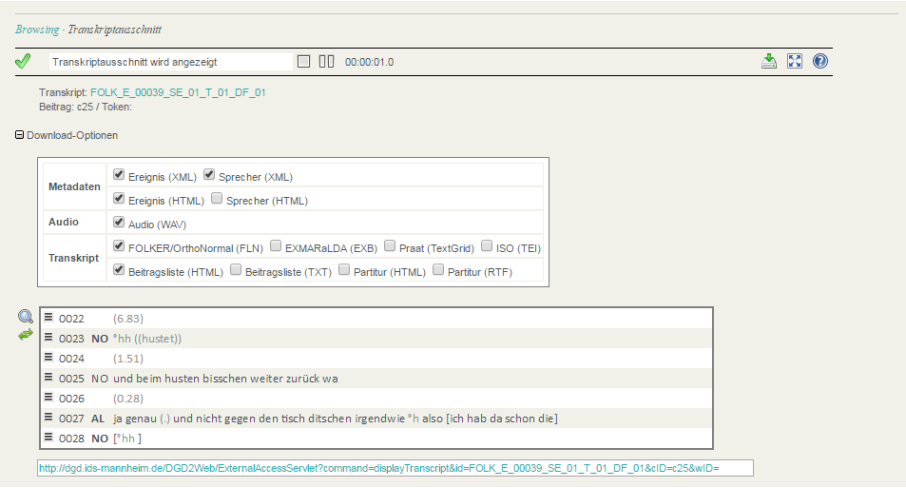


Figure 29: Download of different data types and formats for a transcript excerpt

Different formats are offered for all data types. Metadata can be downloaded as XML or HTML, audio as PCM-WAV. Visualisations of transcripts are available, for instance, as RTF files for integration into text processing software, and, most importantly, the transcripts themselves can be downloaded:

1 as FOLKER XML files to be further processed (e.g. segmented, retranscribed) with the FOLKER or OrthoNormal tools described above, or

2 as EXMARaLDA XML files to be further processed (e.g. annotated on additional tiers) with the EXMARaLDA system, or

3 as Praat TextGrid to be further processed (e.g. subjected to instrumental phonetic analysis) with the Praat software (see figure 30), or

4 as TEI XML conforming to the guidelines of the Text Encoding Initiative and the ISO standard "ISO 24624:2016: Language resource management -- Transcription of spoken language" to open further possibilities of interoperating with other tools.

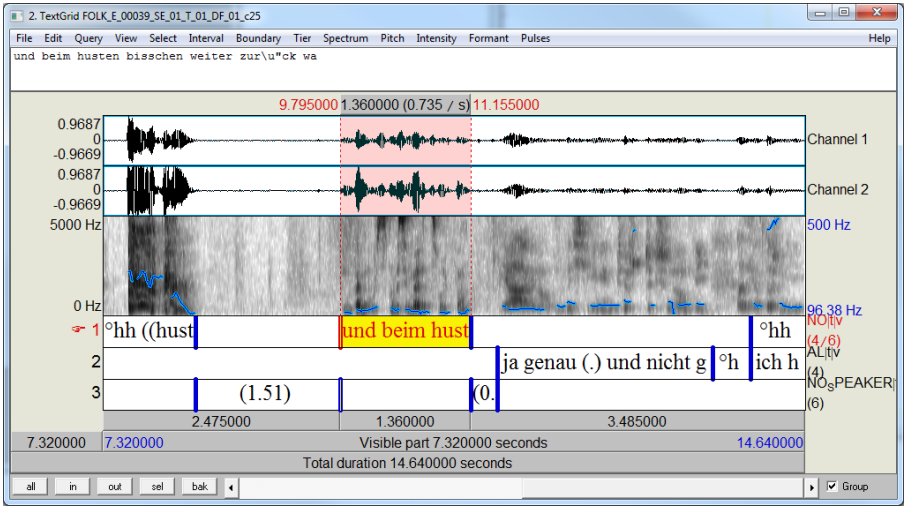


Figure 30: Audio and transcript excerpt with different visualisations of the audio signal in Praat

Users of the data are thus enabled to re-enter the workflow depicted in figure 1 at the stages of transcription or annotation, and the decisions the FOLK project makes in order to reduce the annotation effort do not have to result in a principle obstacle for certain types of exploitation of the audio data.

Finally, the DGD also makes it possible to directly address transcript excerpts via a single URL. This is meant to support citations of data, for instance in publications, where readers may want to not just read the transcript, but also get back to the audio. As an example, consider the following link which takes a registered user to the transcript excerpt depicted in figure 29:

http://dgd.ids-mannheim.de/DGD2Web/ExternalAccessServlet?command=displayTranscript&id=FOLK_E_00039_SE_01_T_01_DF_01&cID=c25&wID=w41

4. Outlooks

The FOLK corpus itself as well as the tools and platform described here are under active development. Seven years after the start of the FOLK project, six years after the first full versions of the transcription and annotation tools and four years after the launch of the beta version of the DGD, we are confident that our workflow for corpus construction and dissemination is effective for our purposes and need not change in any fundamental way in the near future.

Regarding the corpus construction tools, this is corroborated also by the fact that other projects have made productive use of the transcription and annotation tools. This includes projects independent of FOLK, such as the Hamburg Corpus of Bilingual Language Acquisition (HABLA, Kupisch et al. 2012) or the Last Minute Corpus (Rösner et al. 2012), as well as projects with which we have been or are in a close collaboration, such as different ongo-

ing research and dissertation projects. Typically, in such collaborations, researchers receive technical advice from us in return for a part of their transcribed data which we integrate into FOLK. By making the tools themselves freely available, adequately documenting their use (through manuals and guidelines) and offering training and support also for external researchers, we hope, in the long run, to be able to motivate more and more colleagues for this kind of joint effort. Ideally, the practice surrounding the tools for corpus compilation will thus have a measurable effect on the development of the corpus itself, and future extensions of FOLK will profit more and more from external contributions.

The corpus construction workflow has developed gradually, combining existing tools and methods (like EXMARaLDA and GAT) wherever possible, extending and adapting them (to FOLKER and cGAT, respectively, for instance) wherever necessary. In the long run, an obvious option for improvement lies in a tighter integration of the individual components, for instance a more direct interfacing of the annotation tools with the instruments for managing metadata and corpus organisation. Ideally, and following a general trend in this area (see, for example, tools like WebAnno in CLARIN, de Castilho et al. 2014), the workflow could be remodelled in an integrated web-based environment, meaning that "manual" standalone tools for annotation like FOLKER and OrthoNormal would become browser applications in a client-server architecture, and that automatic processes like orthographic normalisation and POS tagging would be realised as web-services. Such an environment would also make it easier to distribute tasks to external partners and, ultimately, make a modest form of "crowd sourcing" a realistic option for FOLK. First steps towards implementing components of the workflow as web-services have already been taken (see Schmidt et al. 2017). The crowd-sourcing aspect for transcriptions is currently explored in a pilot project at the AGD.

The DGD as the corpus dissemination tool is used not only for FOLK, but also for most other spoken language corpora at the Archive for Spoken German (such as large dialect corpora, see Stift/Schmidt 2014). The transcriptions of these corpora can be accommodated by FOLKER's data model, and typically have a somewhat simpler structure: most of them are transcribed orthographically, so there is no need for a second normalisation layer, non-speech tokens like pauses and non-verbal descriptions play a less prominent role, and the interaction structure is often not represented in as much detail (especially regarding overlaps). The browsing and query mechanisms suitable for FOLK are therefore usually more than sufficient also for this other type of data.

Besides the obligatory maintenance requirements and future extensions of FOLK and other corpora in the DGD, we see three areas as prioritized for the development of new functionality in the platform:

1 Since it is becoming more and more common to study spoken language on the basis of video data – making it possible to take into account also the embodied dimension of interaction –, the DGD, as a minimum requirement, will have to provide means of accessing videos in its browsing and query modes. Roughly a third of the FOLK data are already available as digital video files, and we plan to integrate these data alongside suitable visualisation methods into the DGD in the near future.

2 An obvious user need when working with the DGD is to save and retrieve virtual corpora, collections and search results not just for individual use, but also for collaborative

work involving other users. We are therefore working on mechanisms of sharing such data in personal and group workspaces inside the platform.

3 So far, the platform is ready to exploit annotations only on the token level, which is the only type of annotation so far included in FOLK and in all other corpora of the Archive for Spoken Language. There are many cases, however, where spoken language transcriptions are annotated on larger segments, for example for pragmatic functions of chunks or utterances or for conversation topics of larger stretches of a transcript. As we can see already on the occasion of the planned integration of a new resource in the DGD – the GeWiss corpus of Academic Speech (Fandrych et al. 2012), which has been partly annotated for discourse comments as well as for quotes and references – such annotations will have to be accommodated by the data model as well as made accessible through the browsing and query interfaces.

In principle, we think that the DGD interface could also be usable and useful for spoken language corpora constructed or archived in other contexts. Several such resources have become available in the last years, such as the ESLO corpus (Eshkol-Taravella 2012) and corpora in the CLAPI database (Groupe ICOR, in press) for French, the oral parts of the Czech National corpus (Kren 2015) or the Slovene GOS corpus (Verdonik et al. 2013), to name just a few. However, in contrast to the situation for annotation tools, where long-standing development efforts combined with interoperability improvements (see for example Schmidt et al. 2008) have led to a fair degree of conversion, we find that Anthony's (2009) observation that "[Tools widely used by corpus linguists] all offer a different user-experience, because each tool is created in isolation and thus offers a different user interface, control flow, and functionality" is still largely true for tools providing access to spoken language corpora. Technically, the DGD is not ready to be transferred to other contexts, but we hope that, in the mid-term, its design can serve as one source of inspiration for an effort to develop tools for accessing spoken language corpora that are less bound to a specific institutional context.

References

ANTHONY, L. (2009). „Issues in the design and development of software tools for corpus studies: The case for collaboration.“ In: Baker, P. (ed.), Contemporary corpus linguistics. London: Continuum Press, pp. 87-104.

ECKART DE CASTILHO, R. (no date). „TreeTagger for Java – TT4J”. [https://reckart.github.io/tt4j/]

ECKART DE CASTILHO, R. / BIEMANN, C. / GUREVYCH, I. / YIMAM, S.M. (2014). „WebAnno: a flexible, web-based annotation tool for CLARIN”. In: Proceedings of the CLARIN Annual Conference (CAC) 2014, Soesterberg, Netherlands. Linköping University Electronic Conference Proceedings

ESHKOL-TARAVELLA, I. / BAUDE, O. / MAUREL, D. / HRIBA, L. / DUGUA, C. / TELLIER, I., (2012). „Un grand corpus oral ‚disponible’ : le corpus d’Orléans 1968-2012.“ In: Ressources linguistiques libres, TAL. 52,3/2011, pp. 17-46.

FANDRYCH, C. / MEIßNER, C. / SLAVCHEVA, A. (2012). „The GeWiss Corpus: Comparing Spoken Academic German, English and Polish.” In: Schmidt, T., Wörner, K. (eds.): Multilingual Corpora and Multilingual Corpus Analysis. Hamburg Studies in Multilingualism (14). Amsterdam: Benjamins, pp. 319-337.

FANDRYCH, C. / FRICK, E. / HEDELAND, H. / ILIASH, A. / JETTKA, D. / MEIßNER, C. / SCHMIDT, T. / WALLNER, F. / WEIGERT, K. / WESTPFAHL, S. (2016). „User, who art thou? User Profiling for Oral Corpus Platforms. ” In: Proceedings of the 10th Conference on International Language Resources and Evaluation (LREC 2016), Portorož, Slovenia. Paris: European Language Resources Association (ELRA), pp. 280-287. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-50774]

FIEHLER, R. / WAGENER, P. (2005). „Die Datenbank Gesprochenes Deutsch (DGD) – Sammlung, Archivierung und Untersuchung gesprochener Sprache als Aufgaben der Sprachwissenschaft.” In: Gesprächsforschung – Online-Zeitschrift zur verbalen Interaktion. 6/2005, pp. 136-147. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-6869]

GASCH, J. / BRINCKMANN, C. / DICKGIEßER, S. (2008). „memasysco: XML schema based metadata management system for speech corpora.” In: Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008), Marrakesch, Marokko. Paris: European Language Resources Association (ELRA), pp. 2865-2870 [http://www.lrec-conf.org/proceedings/lrec2008/pdf/729_paper.pdf]

GROUPE ICOR (H. BALDAUF-QUILLIATRE, I. COLON DE CARVAJAL, C. ETIENNE, E. JOUIN-CHARDON, S. TESTON-BONNARD, V. TRAVERSO) (IN PRESS). „CLAPI, une base de données multimodale pour la parole en interaction : apports et dilemmes.” In Avanzi, M., Béguelin, M.-J. & Diémoz, F. (eds), Corpus de français parlés et français parlés des corpus, Cahiers Corpus.

HERITAGE, J. (2013). „Turn-initial position and some of its occupants.” Journal of Pragmatics (2013), [http://dx.doi.org/10.1016/j.pragma.2013.08.025]

HUNSTON, S. (2002). „Corpora in applied linguistics.” Cambridge: Cambridge University Press.

INSTITUT FÜR DEUTSCHE SPRACHE (2014). „Korpusbasierte Wortformenliste DeReWo.”, Institut für Deutsche Sprache, Programmbereich Korpuslinguistik, Mannheim, Deutschland. [http://www.ids-mannheim.de/derewo]

KŘEN, M. (2015). „Recent Developments in the Czech National Corpus.” In: Bański, P., Biber, H., Breiteneder, E., Kupietz, M., Lüngen, H., Witt, A. (eds.) (2015): Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora (CMLC-3). Mannheim: Institut für Deutsche Sprache, pp. 1-4.

KUPIETZ, M. / SCHMIDT, T. (2015). „Schriftliche und mündliche Korpora am IDS als Grundlage für die empirische Forschung.” In: Eichinger, L. M. (ed.): Sprachwissenschaft im Fokus. Positionsbestimmungen und Perspektiven. Berlin/Boston: de Gruyter. (Jahrbuch des Instituts für Deutsche Sprache 2014), pp. 297-322. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-34824]

KUPISCH, T. / BARTON, D. / BIANCHI, G. / STANGEN, I. (2012). „The HABLA-corpus (German-French and German-Italian).” In: Schmidt, T. & Wörner, K. (eds.): Multilingual Corpora and Multilingual Corpus Analysis. Amsterdam: Benjamins, pp. 63-179.

RÖSNER, D. / FROMMER, J. / FRIESEN, R. / HAASE, M. / LANGE, J. / OTTO, M. (2012). „LAST MINUTE: A Multimodal Corpus of Speech-based User-Companion Interactions.” In: : Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012), Istanbul, Turkey. Paris: European Language Resources Association (ELRA), pp. 2559-2566 [http://www.lrec-conf.org/proceedings/lrec2012/pdf/550_Paper.pdf]

SCHMID, H. (1994). „Probabilistic Part-of-Speech Tagging Using Decision Trees.” Proceedings of International Conference on New Methods in Language Processing, Manchester, UK.

SCHMIDT, T. (2016). „Good practices in the compilation of FOLK, the Research and Teaching Corpus of Spoken German”. In: Kirk, J. M. and Andersen, G. (eds.): Compilation, transcription, markup

and annotation of spoken corpora, Special Issue of the International Journal of Corpus Linguistics [IJCL 21:3], pp. 396-418. [http://dx.doi.org/10.1075/ijcl.21.3.05sch]

SCHMIDT, T. (2014). „The Database for Spoken German – DGD2." In: Proceedings of the 9th Conference on International Language Resources and Evaluation (LREC 2014), Reykjavik, Iceland. Paris: European Language Resources Association (ELRA), pp. 1451-1457. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-24425]

SCHMIDT, T./DUNCAN, S. / EHMER, O. / HOYT, J. / KIPP, M. / LOEHR, D. / MAGNUSSON, M. / ROSE, T. / SLOETJES, H. (2009). „An exchange format for multimodal annotations." In: Kipp, M., Martin, J.-C., Paggio, P., Heylen, D. (eds.): Multimodal corpora: from models of natural interaction to systems and applications. Berlin/Heidelberg: Springer, 2009, pp. 207-221.

SCHMIDT, T. / SCHÜTTE, W. (2010). „FOLKER: An Annotation Tool for Efficient Transcription of Natural, Multi-party Interaction." In: Proceedings of the 7th Conference on International Language Resources and Evaluation (LREC 2010), Valletta, Malta. Paris: European Language Resources Association (ELRA), pp. 2091-2096. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-22323]

SCHMIDT, T. / WÖRNER, K. (2014). „EXMARaLDA." In: Durand, J., Gut, U., Kristoffersen, G. (eds.): The Oxford Handbook of Corpus Phonology. Oxford: OUP 2014, pp. 402-419.

SCHMIDT, T. / HEDELAND, H. / JETTKA, D. (2017). „Conversion and Annotation Web Services for Spoken Language Data in CLARIN." To appear in: Proceedings of the CLARIN Annual Conference (CAC) 2016, Aix en Provence, France. Linköping University Electronic Conference Proceedings

STIFT, U.-M. / SCHMIDT, T. (2014). „Mündliche Korpora am IDS: Vom Deutschen Spracharchiv zur Datenbank für Gesprochenes Deutsch." In: Institut für Deutsche Sprache (Hrsg.): Ansichten und Einsichten. 50 Jahre Institut für Deutsche Sprache. Redaktion: Melanie Steine, Franz Josef Berens. S. 360-375 - Mannheim: Institut für Deutsche Sprache, 2014. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-24779]

WESTPFAHL, S. / SCHMIDT, T. (2016). „FOLK-Gold – A GOLD standard for Part-of-Speech-Tagging of Spoken German." In: Proceedings of the 10th Conference on International Language Resources and Evaluation (LREC 2016), Portorož, Slovenia. Paris: European Language Resources Association (ELRA), pp. 1493-1499. [http://nbn-resolving.de/urn:nbn:de:bsz:mh39-50786]

VERDONIK, D. / KOSEM, I. / ZWITTER-VITEZ, A. / KREK, S. / STABEJ, M. (2013). „Compilation, transcription and usage of a reference speech corpus: The case of the Slovene corpus GOS." Language resources and evaluation, Dec. 2013, vol. 47, iss. 4, pp. 1031-1048. [http://dx.doi.org/10.1007/s10579-013-9216-5].

Ruprecht von Waldenfels, Michał Woźniak

SpoCo - a simple and adaptable web interface for dialect corpora

We present SpoCo, a simple, yet effective system for the web-based query of dialect corpora encoded in ELAN that provides users with advanced concordancing functions, as well as the the possibility to edit and correct transcriptions if needed. SpoCo is easy to use and maintain, and can be adapted to different spoken corpora in a straightforward way. Simplicity is emphasized to facilitate use by a wide range of users and research groups, including those with limited technical and financial resources, and encourage collaboration and data exchange across such groups. Relying on existing technology and pursuing a modular architecture, SpoCo is developed bottom-up: it was initially devised for a specific dialect project and is being continually adapted for use in other projects in a network of Slavic dialect projects that cooperate in tool development and data sharing. SpoCo thus takes a middle position between systems that are developed for the purposes of a specific dialect corpus, on the one hand, and general-use systems designed for a wide range of data and usage cases, on the other.

1 Overview

While the last years have seen the development of a number of corpus query systems that support spoken data, we observe a a lack of powerful, yet simple and effective corpus tools for dialect corpora with aligned audio that are accessible and manageable for linguists with limited computational expertise. Consequently, many dialect projects still do not realize the potential that modern corpus methods provide for their work.

We present SpoCo, a system that provides a workable, stable, and adaptable environment for the presentation of audio-aligned **spo**ken **co**rpora (the acronym alludes to Polish *SpoCo*, 'it's all right, don't worry'). It offers concordancing, statistical functions and user-provided correction of spoken data using standard corpus and web technologies and relying on the de facto standard ELAN format for its input files. SpoCo relies on the possibilities that are provided by the well-established corpus manager OpenCWB (Evert and Hardie, 2011) and adds only a single function - transcription correction by user feedback - to the existing set of functions.

A main feature of SpoCo is its *simplicity* which we see as key in an effort to provide a tool that is easily accessible for researchers that are not particularly versed with computational tools. Despite being user-friendly in its simplicity and intuitiveness, SpoCo does not forgo the possibilities of a modern corpus system, and in fact is one of a handful of systems available that deal with audio primary data.

SpoCo was first developed as a tool for dialectologists in the Ustja River Basin Corpus Project (von Waldenfels et al., 2014) on Russian and has been subsequently adapted

for two other projects working on Slavic dialects, namely the Corpus of Spoken Rusyn (Rabus and Šymon, 2015) and the Corpus of the Spisz Dialect (Grochola-Szczepanek, ta). These and other projects work together on tool development and data sharing in the research network *SlaSpoCo*[1]. SpoCo is a system that is developed bottom-up, meeting the needs of specific projects. At the same time, care is taken to develop an adaptable system so that development work can benefit the whole ecosystem.

The present paper is organized as follows. In the following Section 2, we list the requirements and aims of SpoCo. In Section 3, we describe the architecture in some detail. Section 4 covers procedures of data import and automatic annotation; Section 5 describes the user interface in more detail. In Section 6 we illustrate the use of the interface with a typical usage case. We conclude with a perspective on future developments.

2 Requirements and aims of SpoCo

SpoCo was developed to meet four key requirements.

A first requirement was to create an interface that is simple and intuitive without restricting the complex possibilities that a modern corpus manager offers. Simplicity and intuitive accessibility were crucial requirements in the design of SpoCo because a large part of the intended audience of our corpora consists of dialectologists who work in a traditional, rather than variationist or corpus-based, paradigm, and are easily dissuaded from using corpus tools if they present a learning curve that is too steep. This issue is exacerbated by the fact that dialectologists working on Russian who do have corpus experience are typically used to the Russian National Corpus (RNC, www.ruscorpora.ru), which has had an exceptionally simple interface from its very beginnings[2].

In general, we find simplicity to be an undervalued, but key issue in spreading corpus use in and beyond the research community; one of the few cases where this issue was explicitly raised and evaluated was during the construction of the GigaFida corpus of Slovenian, the user base of which was considerably broadened by an effective redesign of its query interface (Arhar Holdt et al., 2012, 19). This issue is similarly relevant to our corpora, which we make accessible to interested lay people and scholars from other fields such as anthropology or history. Overall, we think that simplicity is key in the enhanced relevance of such projects such as ours in the context of the *digital humanities*.

[1] For the Ustya River Basin Corpus, see `http://parasolcorpus.org/Pushkino`; for the Corpus of Spoken Rusyn, see `www.russinisch.uni-freiburg.de/corpus`; for the Corpus of the Spisz Dialect, see `https://spisz.ijp-pan.krakow.pl`. The projects collaborate as part of the network *Corpus-based Research into Sociolinguistic and Dialectal Variation in Slavic Languages* (with the Acronym *SlaSpoCo*, which stands for *Slavic spoken corpora*; see `parasolcorpus.org/Spoken_Slavic`), as well as in other ways.

[2] This is due to the fact that the RNC was initially developed by non-computational linguists in collaboration with the search engine company Yandex and modeled on other interfaces aimed at a general audience. Most other corpora, in contrast, were first developed by computational linguists and more directed at a computer-savvy audience.

A second requirement of SpoCo was to enable researchers to make the actual audio recordings available for listening and download, so that detailed analyses can then be made in specialized tools such as PRAAT. As opposed to corpora of written language, the primary data of a dialect corpus is actual speech in its audible form; any representation of this data in written form constitutes an interpretation that to some extent reflects the primary research question. Making this data directly available is thus crucial. Conversely, ready access to the audio data alleviates the demands on the written representation, as it can be viewed to merely represent an access point to the primary data – therefore, rather pragmatic solutions such as transcription in a standard orthography can be pursued. Standard orthography additionally has the desirable effect that it makes the use of standard tools for annotation, such as taggers and lemmatizers, much more straightforward.

The third requirement has to do with flexibility: as the interface is in continuous development and used for multiple corpus projects, the interface must be easily adaptable. To achieve this, we use AngularJS as a programming tool. In the current version of SpoCo new search fields representing different tiers of annotation and transcription, as well as metadata categories, can easily be specified and semi-automatically integrated into the interface. We feel this is crucial in addressing the inherent contradiction between avoiding a cluttered interface and ensuring simplicity of interaction with the GUI (graphical user interface) on the one hand, and using the interface for a wide range of dialect corpora, on the other. The general workflow used to adopt SpoCo is described in section 5.2.

A fourth, basic requirement is the adherence to best practices in data formats and handling. Most importantly, this means using standard formats wherever possible. While adherence to a standard has obvious advantages such as making it easier to use already existing tools, this requirement also has to do with our view of the status of our tool, which we see as principally provisional. We assume that SpoCo will be superseded by more advanced tools in the coming years, and that the data will be migrated to a new system. Since the data have a much longer life expectancy, potentially being archived for decades or longer, it is imperative that we work with formats that are as standard as possible and will be not be problematic from a middle or long-term perspective. For this reason, we store transcriptions in the XML-encoded ELAN[3] file format, WAV-encoded files for the audio data, and transparently encoded XML files for speaker metadata. We choose ELAN since it has become a de facto standard for spoken corpora with a wealth of available corpora and the capacity to represent complex data in a stand-alone format time-aligned to media files themselves[4].

This specialization and these requirements distinguish SpoCo from other corpus tools that also make time-aligned audio data available, but cater to a wider range of tasks.

[3] ELAN is developed at the Max Planck Institute for Psycholinguistics in Nijmegen and available at `http://tla.mpi.nl/tools/tla-tools/elan/`; see Sloetjes and Wittenburg (2008)

[4] ELAN as a tool is used in some, but not all the projects using SPOCO; in the URB project, most transcribers prefer PRAAT, which is more stable and arguably affords a quicker work flow. Here ELAN is used only to convert the PRAAT files to ELAN format before inclusion.

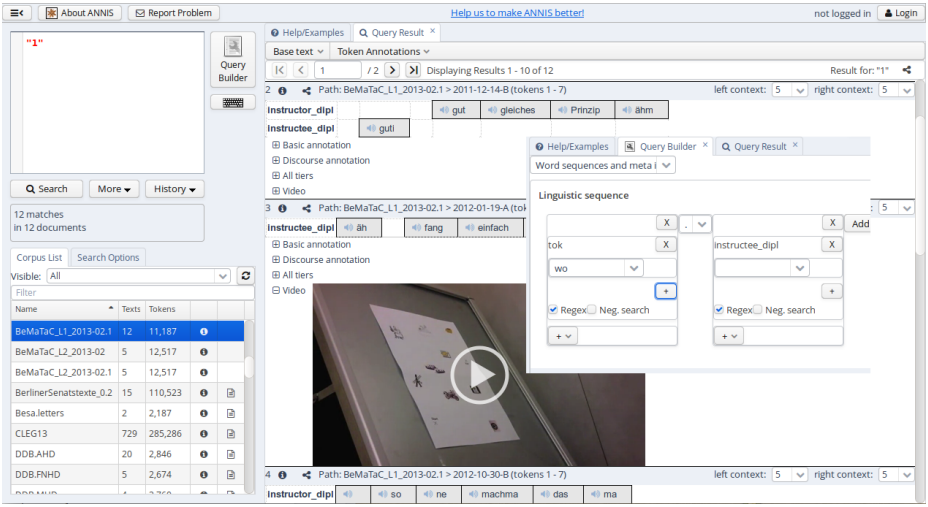


Figure 1: An example query result in ANNIS3, with query builder pasted into the image. ANNIS is very powerful, but also rather complex in use.

Two such systems seem to be particularly relevant for the kind of task that we are faced with. First, ANNIS3 (Krause and Zeldes, 2016) is a corpus system that is geared towards handling annotations of great complexity with multiple corpora of many types. However, ANNIS3 does not fulfill at least two of our requirements since it has a rather complex interface (see figure 1) and does not allow users to download chunks of the aligned primary audio data for further analysis. A second such tool is GLOSSA (Kosek et al., 2015), developed at the University of Oslo for the inclusion of a great variety of data, including the Nordic dialect corpus; GLOSSA has a number of functions that directly cater to dialect corpora building. However, in our experience, it proved difficult to install and it is unclear how to maintain and adapt it to our specific needs without offering the GUI simplicity that we are looking for. Moreover, it does not address the archiving problem since the corpus is essentially kept in CWB vertical format, rather than a standard XML format of some sorts.

Other, more specialized tools are likewise too complex to handle for a small dialectological group; these include the tools developed by the Czech national corpus project for the DIALEKT and ORTOFON corpora (Kopřivová et al., 2014), or the Edisyn interface (Barbiers, 2015).

In the design of SpoCo, we aim to cover the ground between complex, one-size-fits-all interfaces, such as GLOSSA, ANNIS, or CQPWeb (Hardie, 2012) on the one hand, and specialized corpus interfaces on the other hand, such as Edisyn Barbiers (2015) and many other in-house solutions that are never released to the general public as software.

Our approach is to straddle these two worlds by developing a system that is constructed bottom-up, driven by concrete tasks, but at the same time stays flexible and adaptable to new projects, all of which share development costs in a network of related projects.

3 SpoCo architecture and set-up

3.1 Overview of the architecture

SpoCo consists of three main components: 1) the actual linguistic data, 2) the corpus management back end, which supplies concordancing and statistical functions, and 3) the web interface. Each part is largely independent from the rest, which makes changing or replacing individual parts straightforward. SpoCo is designed to be deployed on a standard machine running Ubuntu Linux with Apache (LAMP server) and CWB; no further components are required. Currently, a number of different technologies are involved in the corpus preparation and management process, most notably XML, XSLT, PHP and Apache Server, perl, python and AngularJS. Below we describe the three components in more detail.

3.2 Linguistic data: types and import procedures

The linguistic resources SpoCo uses consist of three types: audio recordings, transcriptions of these, and speaker metadata (i.e., gender, age, place of residence and recording, mobility, etc.). Speaker metadata are technically optional, but they play a potentially crucial role in analysis and are thus useful when searching and presenting results. During corpus encoding, the transcription is split into text segments that were delimited as utterances during transcription in ELAN (or a different transcription tool which the data is converted from). Sound files (supplied in lossless wav format) are split into the corresponding audio segments.

Audio and transcription data are kept in two separate directories and are implicitly linked by identical files names. Adding new files is as simple as copying them into the appropriate directory and issuing a command to re-encode the corpus. Note that while new sound files are typically only added after field work trips, ELAN files are added and updated continuously as transcriptions become completed.

Metadata concerning speakers and recordings are managed separately in a dedicated database (for that purpose we use a DJANGO-based system not described here[5]). For archiving and inclusion purposes into the query system, they are exported and saved as an XML-file; specific metadata fields which should be available in the corpus can be specified during installation.

3.3 Corpus management back end

For storing and querying corpus data, SpoCo uses Corpus Workbench (CWB), a stable and powerful corpus management software which provides sophisticated query and

[5] Metadata management was implemented as part of the TriMCo project by Ilya Khait, Leipzig.

statistical functions. CWB is widely supported, so that, e.g., integration into R is easily accomplished, and it is actively under development (Hardie and Evert 2014). CWB is not resource intensive, in our experience very stable and easy to install, and thus ideal for our purposes. We anticipate that the current version of CWB will eventually be replaced, quite possibly by its successor CWB4 which promises better handling of XML files as well as a new, improved data structure (Evert and Hardie, 2015)

3.4 Web interface

The transcribed data are available for advanced querying through a corpus interface that prepares and sends queries to the corpus manager; this interface is based on previous interfaces for parallel and diachronic data (von Waldenfels, 2011; von Waldenfels and Rabus, 2015). CWB is configured to return the results in XML, which are then displayed using XSLT sheets. This approach affords the advantage of clear separation of corpus manager and output display, as well as simple adaptation of the result page to different needs. Thus, using a different corpus manager is greatly simplified and adding new export formats (e.g., csv) is as simple as specifying a different XSLT sheet in the output. Altogether, this makes the inclusion of new data types straightforward.

Currently, this interface exists in two versions, both of which are geared towards maximal simplicity to make it accessible for a wide range of users. Both versions share most of their functionalities: user management, corpus querying, a correction module, full-text browsing. The main difference between them is the technology they are built on: the initial version (developed for the URB) uses mostly simple HTML and some JavaScript and PHP, while the second version is built with the modern JavaScript framework AngularJS (version 2.1). We chose this framework because it is interface oriented, flexible and scalable; web-page content is easily updated without the need to refresh, and therefore features such as the construction of the CQL query on-the-fly or switching interface languages are easily accessible for both the user and the developer. Both interfaces produce identical output: CQP queries that are channeled through the back end. A more detailed comparison is provided in the next section. Both interfaces are completely interchangeable, which is a good example of the flexibility that the SpoCo modular architecture allows.

4 Using SpoCo I: the back end

4.1 SpoCo integration

For the installation of SpoCo for use with a specific corpus, the interface is copied into a directory that Apache can access, and the settings files are set to contain paths to CWB and data directories, as well as (in the second version) information about the metadata fields in use and languages available in the interface. Depending on specifics of the corpus data (e.g. the number of transcription levels and automatic annotation procedures), the web interface and the inclusion script require adaptation.

4.2 Corpus preprocessing and conversion

After the ELAN-encoded transcriptions are added to the corpus (i.e., copied into the appropriate directories), these are enhanced using automatic tools and converted to the corpus manager CWB for easier querying and simple html files for reading. This involves the following steps triggered by a shell script calling a heterogeneous set of utilities:

- the ELAN files and the XML file containing the metadata are converted into a single, CWB-compatible file in *vertical format*: one token or xml tag on each line
- further annotation such as lemma and morphological tags are then added to this file using standard tools. This is, in general, corpus-specific: in the case of the URB, the Treetagger (Schmid, 1999) is employed using a model trained on the Russian National Corpus. In the case of Rusyn corpus, a custom-made approach to tagging with Levenshtein distances is being developed to take into account variation due to diverse transcription standards (ongoing work by Achim Rabus, Freiburg, and Yves Scherrer, Geneva.); this approach is expected to be relevant for other corpora in the network, as well.
- based on the segmentation in the ELAN file, the audio files are cut into small chunks for downloading and broadcasting
- html versions of each transcribed text are prepared for full in-context reading

This script is invoked each time the corpus data is changed and can be triggered by users via the web interface.

5 Using SpoCo II: interface features

5.1 Corpus query

In the following, we focus on describing the initial version that was developed for the URB but also highlight differences in the newer version used in the two other projects.

Initially, users are asked to log in to reach the query page. There are three user categories: *guests* can only search the corpus; *registered users* can also correct transcriptions; *administrators* can validate corrections and re-encode the corpus.

Figure 2 shows the main query page (a second pane with help, sample queries and corpus statistics is not shown here for reasons of space). Queries are available on three levels of complexity: *simple search* allows users to search for contiguous phrases just like in the search field of a word processor; *advanced search* allows users to specify search words in terms of wordform, lemma and morphological tag with variable distance between them. This option is very similar to the RNC's interface and therefore familiar to scholars working on Russian. Finally, the *complex (CQP) search* allows users to enter any valid CQP query. This is used for more complex queries, sorting and metadata

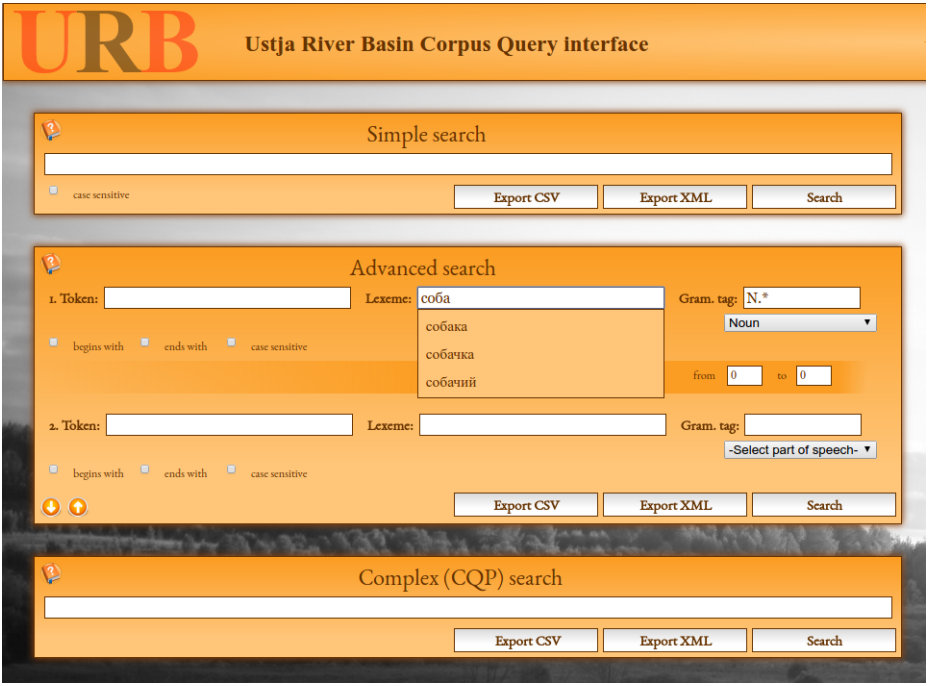


Figure 2: Search interface for the URB, with simple, advanced and expert search options, offering drop-down lists of word forms and lemmata and an interactive panel for morphological tag construction.

filtering; for example, negative conditions or restrictions on specific categories of speakers can be formulated here.

For the *Corpus of Spoken Rusyn* (Rabus and Šymon, 2015) and subsequently for the *Spisz Dialect Corpus*, the query page was taken to the next step. While the rest of the system remains essentially identical, the query page was reprogrammed in AngularJS, allowing for multiple interface languages to be included and for search fields to be more easily adapted, and thus for customization for different corpora. It introduces three new features (see Figure 3): First, metadata can now be searched for in the GUI; this is customized in a settings file where these fields, their display names and default values are specified during installation. Second, the interface now integrates a Google maps application for geographic visualization and filtering. The interface is described in detail in Rabus and Šymon (2015). Third, the basic and CQP searches are now linked dynamically: filling in fields in the *basic search* automatically constructs the *CQP search*, which can then be manually adapted as needed for more advanced queries.

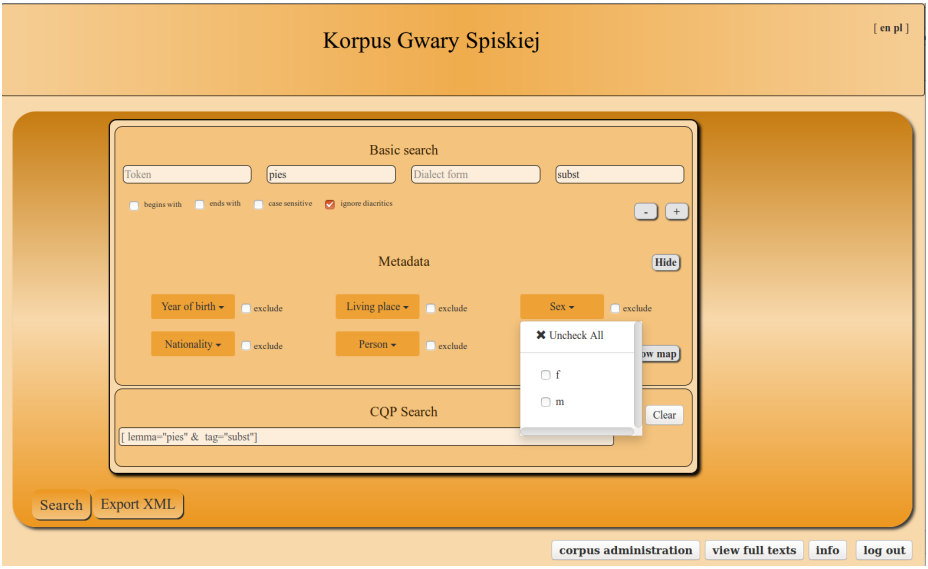


Figure 3: The search interface for the *Spisz Dialect Corpus*, with additional metadata search options, interactive maps for places of recording (not shown), and a dynamic link between the *basic search* and *CQP search*.

This last feature not only has the effect that complex CQP queries can be constructed more quickly, it also makes learning CQP much easier for novice users who can now observe the CQP queries as they are being constructed in response to their filling in the *basic search* fields. Then, users can learn to use CQP by changing these queries, rather than having to construct them from scratch. In our experience, this greatly lowers the threshold for beginning to use complex queries in CQP.

5.1.1 Corpus query results

Figure 4 displays the beginning of the concordance for the lexeme *sobaka* 'dog'; here, each example includes the respective audio segment. Each corpus hit is provided with links to a tab-delimited csv view, with directly downloadable audio segments for analysis in PRAAT or other speech analysis software, and with a file containing basic metadata for each speaker. Users can also examine each example with more context in a separate window (see Figure 5); the URLs for these separate windows serve as a unique identifiers based on the location in the audio timeline. To the very right, registered users can click the paper-and-pen icon in order to be able to edit the transcription.

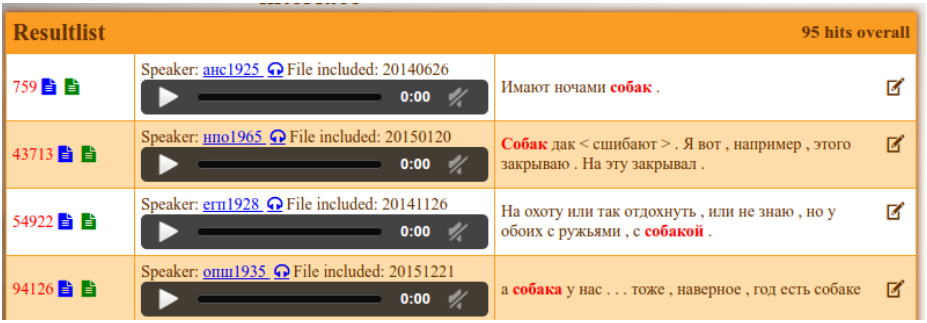


Figure 4: Query results for *sobaka* 'dog'. Each row in the result list provides access to (from left to right): context view with persistent URL (blue icon), csv-export view (green icon), link to *speaker metadata* found in a google spreadsheet, audio fragment in wav format (headphones icon), the date the transcription was added, the text itself with search item highlighted in red, a link for registered users to provide corrections (paper-and-pen icon).

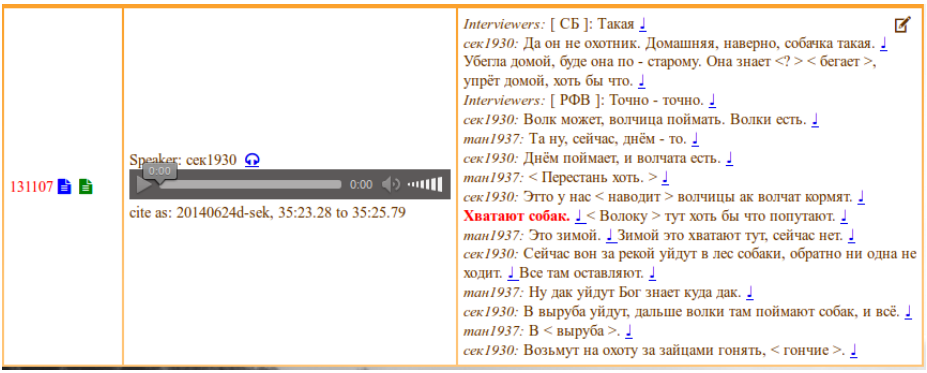


Figure 5: Context view, with citation instructions and multiple speakers.

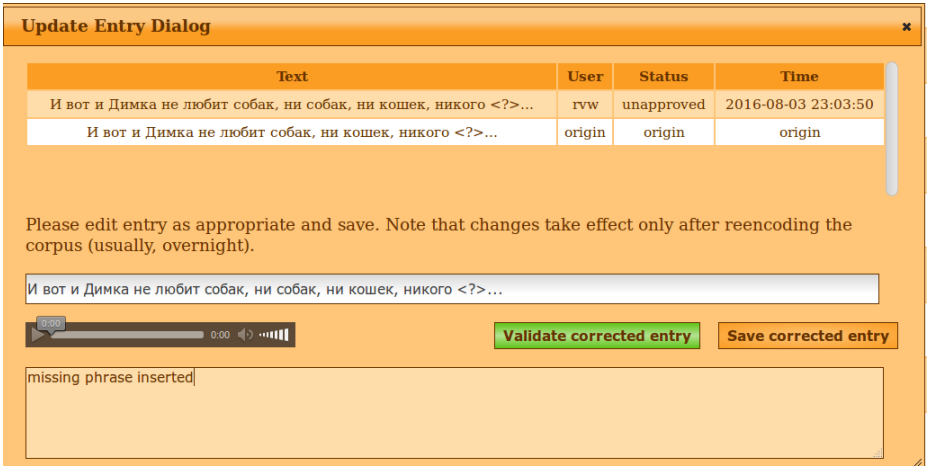


Figure 6: Window for transcription correction.

5.1.2 Transcription edit function (error correction)

The interface for editing transcriptions is implemented using XSLT and JavaScript and shown in figure 6. It is only available to registered users. The user is provided with an option to enter a new transcription and leave a comment; the segmentation itself cannot be changed online. This new transcription is written directly to the ELAN files as an additional transcription together with the user name, time stamp, and a status field stating that this is an unconfirmed correction. This new and all previous versions are kept in the ELAN file in parallel; unconfirmed changes need to be reviewed and validated by an administrator before they are marked as accepted (or reverted if necessary); see Figure 7. In this way, each correction is double-checked.

Changes appear in corpus results only after the corpus has been re-encoded in CWB; this is done every night automatically or on demand after the validation process. Corrections are flagged in the query results, together with their status as confirmed or unconfirmed, and the edit function gives access to all previous versions of the transcription segment. This function is used quite a lot; in the URB corpus today, which has roughly 750 000 tokens (excluding interviewers), around 3000 lines have been corrected.

5.1.3 Full text views

In many cases, users want to read interview transcripts in their entirety as full text. In addition to using the query page, registered users may listen to and read complete transcriptions, as shown in figure 8. These full texts are derived from the ELAN files

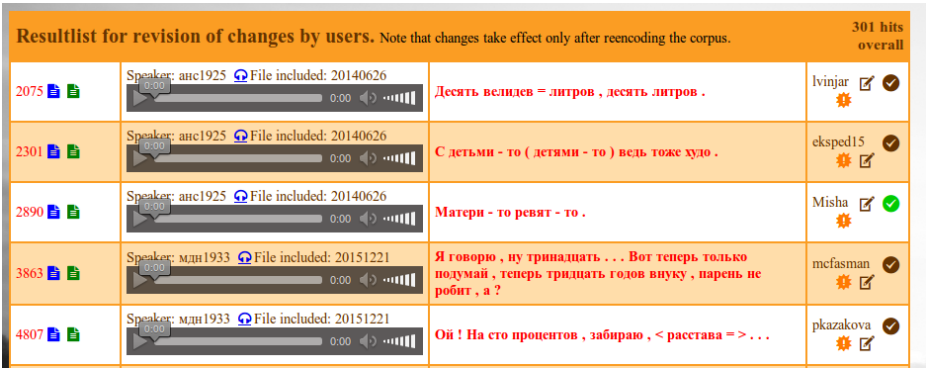
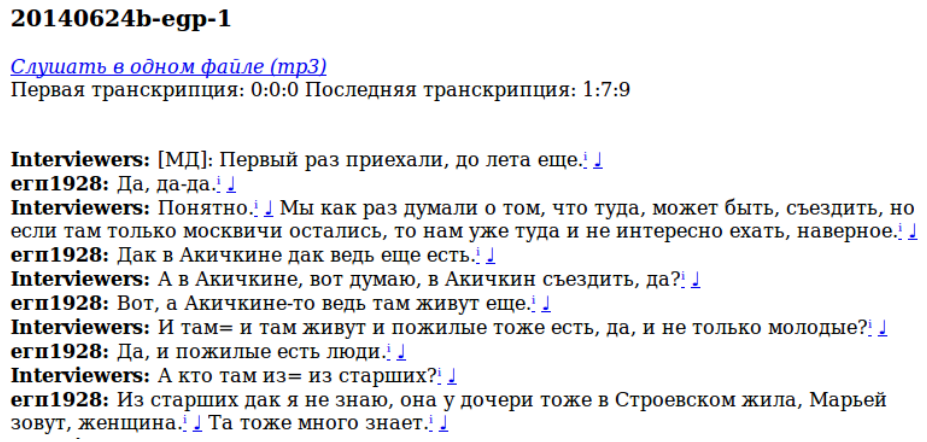


Figure 7: List of unconfirmed edits for administrators, with basic information on who made the change, its status, and the possibility of one-click confirmations.



20140624b-egp-1

Слушать в одном файле (mp3)
Первая транскрипция: 0:0:0 Последняя транскрипция: 1:7:9

Interviewers: [МД]: Первый раз приехали, до лета еще.
егп1928: Да, да-да.
Interviewers: Понятно. Мы как раз думали о том, что туда, может быть, съездить, но если там только москвичи остались, то нам уже туда и не интересно ехать, наверное.
егп1928: Дак в Акичкине дак ведь еще есть.
Interviewers: А в Акичкине, вот думаю, в Акичкин съездить, да?
егп1928: Вот, а Акичкине-то ведь там живут еще.
Interviewers: И там= и там живут и пожилые тоже есть, да, и не только молодые?
егп1928: Да, и пожилые есть люди.
Interviewers: А кто там из= из старших?
егп1928: Из старших дак я не знаю, она у дочери тоже в Строевском жила, Марьей зовут, женщина. Та тоже много знает.

Figure 8: Full text view. The note symbols after each sentence enable the researcher to listen to audio segments and jump to the corpus view for each utterance.

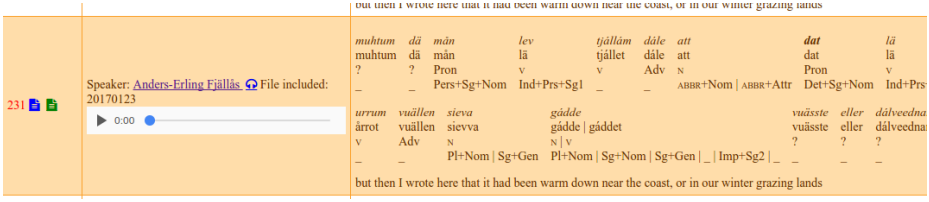


Figure 9: A result page from the glossed Pite Saami corpus.

using an XSLT transformation, and provide links to the full audio file, audio segments as well as to the corpus view for each utterance.

5.2 Adapting SpoCO: a sample case case

In late 2016, SpoCo was adopted for two non-Slavic corpora; first, a corpus of Pite Saami (Wilbur, 2017), and second, a corpus of dialectal Lithuanian (as part of the planned TrimCo[6] corpus). As many projects aimed at documenting endangered languages, these corpora supply morphological glosses; support for this feature was added to SpoCo on this occastion. The adaptation of SpoCo consisted in three steps:

- adaptation of the scripts converting ELAN to CWB; this entailed decoding the hierarchical relationship in the ELAN-file to obtain token-based glosses and encoding free translations as xml attributes at utterance level
- adaptation of settings in SpoCo to query and return glosses and free translations
- adaptation of the XSLT sheet that displays the resulting XML using an open source library that displays glossed text

Figure 9 shows the results of a query in the Pite Saami corpus.

6 A sample workflow and desirable features

To exemplify the role of SpoCo, below we describe the workflow of a typical investigation of a dialect variable in the URB project. Specifically, our example concerns the dialectal shift of /a/ to /e/ between palatalized consonants (a-raising, see Požarickaja 2005, 42f.) that is characteristic of the speech of older speakers of the Ustja dialect. In the URB, as well as in most of the projects using SpoCo, transcriptions are written in standard orthography with only very limited representation of dialectal features; see von Waldenfels et al. (2014); Gerstenberger et al. (2017) for discussions of the advantages of such an approach.

[6] Triangulation Approach for Modelling Convergence with a High Zoom-In Factor; see `http://www.trimco.uni-mainz.de/`. PI: Björn Wiemer, Mainz; Lithuanian Subcorpus: Kirill Kozhanov, Moscow.

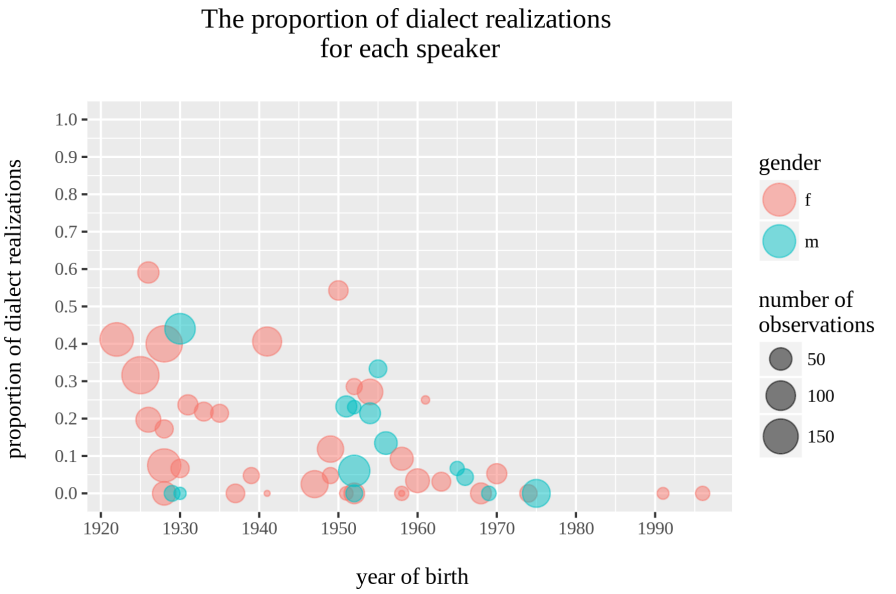


Figure 10: An example variable (Kazakova, 2016). The plot gives the relative instances of historical /a/raising to [e] between soft consonants for speakers born between 1922 and 1995; red circles represent females, green circles males. The size of the circles represent the number of instances in the study.

In a first step, the envelope of variation is defined and searched for in terms of the standard language – in this case, all word forms that contain /a/ between soft consonants in the standard orthography are queried, downloaded as csv, copied, and pasted in OpenOffice Calc or some other offline tool. The CSV contains links to the audio segment files, so that each example is categorized with respect to the actual audio data (rather than a transcription). Since the download also includes basic speaker metadata, the resulting categorization thus affords a simple plot of speakers, ordered by date of birth, and with respect to the relative proportion of dialectal as opposed to standard pronunciation. Figure 10 gives an example of such a plot, which nicely shows the dialectal feature's tendency to recede in an apparent time perspective.

Transcription into standard orthography as opposed to a phonetic alphabet (IPA or similar) effectively allows the phonetic analysis to be postponed until it is needed for specific research question, thereby streamlining and focusing it and limiting the overall work load involved. In the future, it would be highly desirable to allow users to upload the result of such annotation tasks so that they can be viewed and used by future users.

7 Summary and future developments

We have presented SpoCo, a system to query and analyze spoken corpora with aligned audio data. The system is *pragmatic* in that it aims to provide facilities that are needed in a number of concrete Slavic dialect projects. At the same time, it follows an overarching agenda to enable collaborative tool development across different projects; with this in mind, care is taken that the system is modular and expandable for use with other, related projects.

An important aim of SpoCo is to create a system with a low threshold of use for a wide range of projects, including those with limited computational expertise and resources. Specifically, we aim for a stable, hassle-free, easy-to-use system that is simple yet effective. We see this as as an important methodological contribution to the field since using such a system and its collaborative development goes hand in hand with data sharing and the adoption of innovative research methods. An important aim for the future is thus to make the deployment of SpoCo easier for new projects and to work on making the customization of SpoCo simpler yet more flexible as well as to work on further integration of the workflow of corpus file management (including distributed archiving), metadata acquisition, and subsequent data annotation.

References

Arhar Holdt, Š., Kosem, I., and Logar Berginc, N. (2012). Izdelava korpusa Gigafida in njegovega spletnega vmesnika. In Erjavec, T. and Žganec Gros, J., editors, *Zbornik Osme konference Jezikovne tehnologije*, pages 12–17. Institut Jožef Stefan, Ljubljana.

Barbiers, S. (2015). European dialect syntax: Towards an infrastructure for documentation and research of endangered dialects. In Jones, M., editor, *Endangered Languages and New Technologies. Cambridge: CUP*. CUP, Cambridge.

Evert, S. and Hardie, A. (2011). Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium. In *Proceedings of the Corpus Linguistics 2011 Conference, Birmingham, UK*. University of Birmingham.

Evert, S. and Hardie, A. (2015). Ziggurat: A new data model and indexing format for large annotated text corpora. In *Proceedings of the 3rd Workshop on the Challenges in the Management of Large Corpora (CMLC-3)*, page 21–27, Lancaster, UK.

Gerstenberger, C., Partanen, N., Rießler, M., and Wilbur, J. (2017). Utilizing language technology in the documentation of endangered Uralic languages. In Pirinen, T. A., Trosterud, T., and Tyers, F. M., editors, *Northern European Journal of Language Technology: Special Issue on Uralic Language Technology*.

Grochola-Szczepanek, H. (t.a.). Korpusowe badania języka mieszkańców spisza w polsce – cele i zadania. *Jezikoslovni zapiski*, 22(2).

Hardie, A. (2012). CQPweb - combining power, flexibility and usability in a corpus analysis tool. *International Journal of Corpus Linguistics*, 17(3):380–409.

Kazakova, P. (2016). Alternation of [a]/[e] between palatalized consonants under stress in the dialect of the village mikhalevskaya. Unp. manuscript.

Kopřivová, M., Klimešová, P., Goláňová, H., and Lukeš, D. (2014). Mapping diatopic and diachronic variation in spoken Czech: the ORTOFON and DIALEKT corpora. In Chair), N. C. C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 376–382, Reykjavik, Iceland. European Language Resources Association (ELRA).

Kosek, M., Nøklestad, A., Priestley, J., Hagen, K., and Johannessen, J. B. (2015). Visualisation in speech corpora: maps and waves in the Glossa system. In Grigonytė, G., Clematide, S., Utka, A., and Volk, M., editors, *Proceedings of the Workshop on Innovative Corpus Query and Visualization Tools at NODALIDA 2015, May 11-13, 2015, Vilnius, Lithuania*, pages 23–31. Linköping University Electronic Press.

Krause, T. and Zeldes, A. (2016). ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, 31(1):118–139.

Požarickaja, S. K. (2005). *Russkaja dialektologija*. Gaudeamus, Moscow.

Rabus, A. and Šymon, A. (2015). Na novŷx putjax isslidovanja rusyns'kŷx dialektu: korpus rozhovornoho rusyns'koho jazŷka. In Koporova, K., editor, *Rusyn'skŷj literaturnŷj jazŷk na Slovakiji. 20 rokiv kodifikaciji / The Rusyn literary language in Slovakia. 20th anniversary of its codification. IV. International Congress of the Rusyn Language. Prjašiv, 23. - 25. 09. 2015*, pages 40–54.

Schmid, H. (1999). Improvements in part-of-speech tagging with an application to German. In Armstrong, S., Church, K., Isabelle, P., Manzi, S., Tzoukermann, E., and Yarowsky, D., editors, *Natural Language Processing Using Very Large Corpora*, volume 11 of *Text, Speech and Language Processing*, pages 13–26. Kluwer Academic Publishers, Dordrecht.

Sloetjes, H. and Wittenburg, P. (2008). Annotation by category – ELAN and ISO DCR. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*.

von Waldenfels, R. (2011). Recent developments in ParaSol: Breadth for depth and XSLT based web concordancing with CWB. In Majchráková, D. and Garabík, R., editors, *Natural Language Processing, Multilinguality. Proceedings of Slovko 2011, Modra, Slovakia, 20–21 October 2011*, pages 156–162, Bratislava. Tribun EU.

von Waldenfels, R., Daniel, M., and Dobrushina, N. (2014). Why standard orthography? Building the Ustya River Basic Corpus, an online corpus of a Russian dialect. In *Komp'juternaja lingvistika i intellektual'nye technologii: Po materialam ežegodnoj Meždunarodnoj konferencii «Dialog» (Bekasovo, 4 — 8 ijunja 2014 g.) Vyp. 13 (20)*, Moskva. Izd-vo RGGU.

von Waldenfels, R. and Rabus, A. (2015). Recycling the metropolitan: building an electronic corpus on the basis of the edition of the *Velikie Minei Čet'i. Scripta & e-Scripta*, 14/15:27–38.

Wilbur, J. (2008–2017). Pite saami. In *Endangered Languages Archive (ELAR)*. SOAS, University of London.

Author Index

Bryce Anderson-Cooper
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
b.anderson1@lancaster.ac.uk

Alistair Baron
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
a.baron@lancaster.ac.uk

Nils Diewald
Institut für Deutsche Sprache
R5 6-13, 68161 Mannheim, Germany
diewald@ids-mannheim.de

David Gullick
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
david.s.gullick@gmail.com

Ulf Leser
Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
leser@informatik.hu-berlin.de

Andreas Kerren
Department of Computer Science
Linnaeus University
SE-351 95 Växjö, Sweden
andreas.kerren@lnu.se

Thomas Krause
Institut für deutsche Sprache und Linguistik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
krauseto@hu-berlin.de

Anke Lüdeling
Institut für deutsche Sprache und Linguistik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
anke.luedeling@hu-berlin.de

Eliza Margaretha
Institut für Deutsche Sprache
R5 6-13, 68161 Mannheim, Germany
margaretha@ids-mannheim.de

John Mariani
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
j.mariani@lancaster.ac.uk

Andrew Moore
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
a.moore@lancaster.ac.uk

Arne Neumann
Applied Computational Linguistics
Department Linguistik, Haus 14
Karl-Liebknecht-Straße 24-25
14476 Potsdam, Germany
arne.neumann@uni-potsdam.de

Carita Paradis
Centre for Languages and Literature
Lund University
Box 201
SE-221 00 Lund, Sweden
carita.paradis@englund.lu.se

Christian Pölitz
Lehrstuhl für künstliche Intelligenz
Institut fuer Informatik
OH12 R4.012, Otto-Hahn-Straße 12, 44227 Dortmund, Germany
christian.poelitz@tu-dortmund.de

Paul Rayson
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
p.rayson@lancaster.ac.uk

Thomas Schmidt
Oral Corpora PA
Institut für Deutsche Sprache
R5 6-13, 68161 Mannheim, Germany
thomas.schmidt@ids-mannheim.de

Maria Skeppstedt
Gavagai
Slussplan 9, SE-111 30 Stockholm, Sweden
Department of Computer Science
Linnaeus University
SE-351 95 Växjö, Sweden
maria@gavagai.se

Ruprecht von Waldenfels
Slavisches Seminar
Universität Zürich
CH-8032 Zürich
ruprecht.waldenfels@gmail.com

Stephen Wattam
School of Computing and Communications
Lancaster University
Lancaster, LA1 4WA, UK
s.wattam@lancaster.ac.uk

Michał Woźniak
Institute of Polish
Polish Academy of Sciences
Al. Mickiewicza 31, 31-120 Cracow, Poland
michal.wozniak@ijp-pan.krakow.pl