The LiRI Corpus Platform

Johannes Graën, Jonathan Schaber, Daniel McDonald, Igor Mustač, Nikolina Rajović, Gerold Schneider, Teodora Vuković, Jeremy Zehr and Noah Bubenhofer
Linguistic Research Infrastructure
University of Zurich, Switzerland
first_name.last_name@linguistik.uzh.ch

Abstract

We present the LiRI Corpus Platform (LCP), a software system and infrastructure for querying a vast array of corpora of different kinds. It heavily relies on the PostgreSQL relational database management system, employing state-of-the-art data representation and indexing techniques, which lead to significant performance gains when querying, even for structurally complex queries involving nested logical operations and quantifiers. In this work, we describe the requirements that led to the development of this novel system, discuss methods from corpus linguistics and beyond that we considered key for such a system, and provide details on a number of technological features that we take advantage of. Our platform also comes with its own query language tailored both to the requirements in terms of information need and our philosophy of how to define corpora in an abstract way.

1 Introduction

Corpora are an important resource for empirical linguistic research, as well as text-based social studies such as digital humanities or media research (Meurers, 2005). There exist numerous tools and platforms for manipulating different corpus types (mono- vs. multilingual, text vs. multimodal, synchronic vs. diachronic, etc.), offering various means of access to corpora (web interfaces, command line interfaces, etc.), and providing different functionalities (editing, querying, annotating, etc.).[1] While, taken together, the set of tools and applications developed thus far covers a variety of corpora and corpus types, taken individually, most of them are actually designed for specific corpora or corpus types, and therefore lack the ability to generalize. Some tools provide a general interface for diverse corpora, handling a multitude of differently structured corpora and occasionally provide specific functionality needed for them (see Section 2 for a (small) overview of existing tools and applications).

Central to all corpus tools is the ability to query one or more datasets. We propose a distinction between two main querying strategies:

1. **Corpus exploration**, characterized by being an iterative, drilling-down process, where users run and refine their queries based on results, and by interfaces that prioritize speed (Hearst, 1999).

2. **Corpus analysis**, maximizing recall of results in a corpus, prioritizing accuracy and specificity, with results returned in a format suited for subsequent statistical analyses.

The latter strategy is of particular interest to approaches that require large amounts of data (“Big Data”), such as the training of large language models.

While those two strategies are more complementary than antagonistic, to our knowledge most tools tend to prioritize the latter strategy over the former (cf. Desagulier, 2019). We design our application to accommodate both approaches, supporting the construction of queries in a bottom-up fashion. When working with today’s often very large corpora, one does not always need full recall: as soon as the matches delivered are a reasonable approximation to all the results, the user may choose to stop instead of waiting for complete results. We discuss, besides other things, how we employ random sampling by

[1] https://corpus-analysis.com/, a page collecting tools for corpus linguistics, lists 266 entries (as of 2024-03-05).

means of logarithmic partitioning to achieve these goals (in Sections 3.2 and 4.2). The incorporation of both strategies constitutes a timely improvement over existing tools that presume their users to arrive with full-fledged, complex queries.

In this contribution, we present the LiRI Corpus Platform (LCP) a new tool that couples complex analysis of diverse corpora and interoperability with existing CLARIN resources. The software facilitates access to and re-use of research data, offering a flexible architecture, whereby a single backend can be connected to multiple frontends tailored to the requirements of specific research agendas. We will mostly focus on the data modeling and its implementation in an RDBMS, while only briefly explaining the different interfaces that we have implemented and plan to implement in the future. While interfaces are to some extent interchangeable, the dynamic data model and its implementation form a core feature and contribution of our tool.

After discussing existing corpus query systems in the next section, we will detail in Section 3 which methods from corpus linguistics were seminal to our platform, followed by an overview of technology and technological methods employed in Section 4. Section 5 lists important limitations of our platform's features, describes the current state of implementation and provides an outlook on future development.

2 Related Work

Myriad systems for storing and querying large corpora have been developed in past decades. Clematide (2015) gives an overview of corpus linguistic query language types, distinguishing between four families. Historically prominent were (I) sequence-based designs, such as CQP (Christ, 1994; Evert & Hardie, 2011) and other dialects of regular expressions, (II) structure-based designs, such as TGrep2 (Rohde, 2005) for querying syntactic trees. Many of them have organically developed within parameters set by technological restrictions of the time. For example, CQP is limited with regard to syntactic queries because its sequence-based conceptualization of text is ill-suited to express non-sequential structural information expect for containment expressed by the *within* keyword. More recently (III), the class of path-based languages which use the XPath query language have been implemented. Finally, (IV) the class of logic-based languages such as TigerSearch and ANNIS, which offers outstanding expressiveness, coupled with considerably longer retrieval times. For example, for performance reasons, the most recent major version of ANNIS (Krause & Zeldes, 2016) has abandoned the relational database PostgreSQL, and developed a custom implementation based on graphs.[2] In contrast, our proposed approach retains the expressiveness of logic-based languages while leveraging advanced data-representation and indexing techniques in order to offer faster retrieval times.

Several methods for speed-ups have been proposed. For instance, while older approaches rely on MapReduce techniques (Schneider, 2013), modern database management systems as PostgreSQL provide internal mechanisms for parallel computing and intelligent algorithms to use the most discriminate features first. Other methods are the use of sophisticated indexing and retrieval techniques, as in the the proposals of (Ghodke & Bird, 2012).

For reasons of space, our overview provides only a brief sketch of past and present resources. For example, we omit detailed description of several projects fine-tuned for particular corpora and particular tasks, for example Dependency Bank (Lehmann & Schneider, 2012), which allows fast syntactic queries on the British National Corpus (BNC).

While some published standards and guidelines have attempted to define a single digital format able to encompass all possible linguistic data and annotations (Gries & Berez, 2017; Ide & Romary, 2004, 2006), to our knowledge no actual working implementation of such proposals has been presented to date. While we do not suggest that our solution covers all possible needs, we offer an attractive solution that facilitates analysis of a very broad range of corpus types.

3 Methods

The methods described here were fundamental to the development of our corpus platform. Our target user groups come from different fields and are accustomed to different tools and methods of analyzing

[2] https://github.com/korpling/ANNIS/blob/main/CHANGELOG.md (2024-03-05).

language data. We aim at satisfying needs of most envisaged users, but we are well aware that a one-size-fits-all solution for corpus management and retrieval is unlikely to be developed in the near future.

3.1 Analysis and Collocations

In addition to showing matches in plain and keyword-in-contect (KWIC) format, LCP allows for several statistical analyses. We present two examples of statistical analyses: frequency analysis, which delivers frequency distributions, and collocations, in which a large range of standard collocation measures are offered. The query in Figure 1 illustrates those on a query for adjective premodification of a noun whose lemma is *preference*.

```
Segment s

sequence seq
    Token@s t1
        xpos2 = "ADJ"
    Token@s t2
        xpos2 = "SUBST"
        lemma = "preference"

KWIC => plain
    context
        s
    entities
        t1
        t2

AdjDist => analysis
    attributes
        t1.lemma
    functions
        frequency
    filter
        frequency > 1

LeftContext => collocation
    center
        t2
    window
        -2..-1
    attribute
        lemma
```

Figure 1: A query on the British National Corpus (BNC) searching for a sequence of adjectives and nouns with the lemma *preference*. Alongside a result set comprising both tokens within the sentence in which they appear ('KWIC'), a frequency analysis on the lemma of the adjectives ('AdjDist') and a collocational analysis ('LeftContext') are defined – for a window of the two tokens preceding the noun (positions -2 and -1).

The results pane shows three tabs, corresponding to the three result sets *KWIC*, *AdjDist* and *LeftContext* that we defined in the query above. Figure 2 reports the frequency table (left) and collocation table (right) generated by the query: the former lists absolute and relative frequencies, while the latter reports strongest associates, sorted by *simple log-likelihood* in this example.

3.2 Sampling

One strategy we deploy allowing us to deliver results very quickly is the use of random sampling combined with the incremental querying of increasingly large samples of documents, starting with the smallest sample, while constantly updating the information displayed, that is raw matches, frequencies and statistical measures.[3] Random sampling is typically the best sampling strategy if no assumptions can be made about the entire dataset (Cochran, 1977). The reliability of the partial results can be estimated by the standard error SE, which is defined as

$$\mathrm{SE} = \bar{\sigma} = \frac{\sigma}{\sqrt{n}} \tag{1}$$

where n is the number of matchses, and σ the standard deviation, i.e. the fluctuation in a sample.

While we do not know the exact standard deviation until we have found all the matches (for we cannot know the mean of the whole population) we can approximate it, as is e.g. done in the Welch's two-sample t-test, where the standard error is approximated via the samples.

[3] See also Section 4.2.

KWIC (521) | AdjDist (116) | LeftContext (426)

Filter by t1.lemma | Filter by frequency | Filter by relative freque

t1.lemma	frequency	relative frequency ↓
personal	74	8.159 %
first	62	6.836 %
strong	43	4.741 %
own	43	4.741 %
this	30	3.308 %
local	30	3.308 %
sexual	27	2.977 %
second	27	2.977 %
convertible	26	2.867 %
parental	26	2.867 %
individual	21	2.315 %
political	20	2.205 %
mating	19	2.095 %
clear	17	1.874 %

KWIC (521) | AdjDist (116) | LeftContext (426)

Filter by Text | Filter t | Filter by E | Filter by O/E | Filter by M | Filter by M | Filter by lo | Filter by | Filter by z-sc | Filter by sim

Text	O	E	O/E	MI	MI³	local-MI	t-score	z-score	simple-ll ↓
personal	75	0.109	688.971	9.428	21.886	707.123	8.648	226.987	830.498
convertible	31	0.003	10053.296	13.295	23.204	412.157	5.567	558.203	509.377
strong	48	0.125	383.64	8.584	19.754	412.013	6.91	135.347	475.422
first	64	0.759	84.343	6.398	18.398	409.484	7.905	72.6	441.183
own	50	0.462	108.337	6.759	18.047	337.969	7.006	72.92	369.448
parental	26	0.008	3132.988	11.613	21.014	301.946	5.097	285.317	366.603
redeemable	19	5.78e-4	32853.672	15.004	23.5	285.072	4.359	790.052	357.194
a	120	13.612	8.816	3.14	16.954	376.813	9.712	28.836	309.598
mating	19	0.002	7936.008	12.954	21.45	246.13	4.358	388.26	303.213
sexual	27	0.044	615.614	9.266	18.776	250.179	5.188	128.715	292.909
local	30	0.294	102.025	6.673	16.487	200.183	5.424	54.782	218.101
marked	16	0.013	1238.665	10.275	18.275	164.393	3.997	140.665	195.923
second	27	0.285	94.813	6.567	16.077	177.309	5.141	50.062	192.372
imperial	15	0.012	1268.548	10.309	18.123	154.634	3.87	137.834	184.393

Figure 2: Distribution of adjectives premodifying the noun *preference* (left) and collocations of the two preceding tokens of that noun (right).

$$t = \frac{\bar{x} - \mu}{\bar{\sigma}} = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}} = \frac{\bar{x} - \mu}{\sqrt{\frac{\sigma^2}{n}}} = \frac{(\bar{x_1} - \mu) + (\bar{x_2} - \mu)}{\sqrt{\frac{s_1^2}{n} + \frac{s_2^2}{n}}} = \frac{(\bar{x_1} - \bar{x_2})}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{2}$$

Accordingly, the frequencies found in a large set of samples give us a good estimate of the population frequency, to which it asymptotically converges. Our technique of logarithmic partitioning implements this random sampling strategy for faster callback while minimally compromising on accuracy (see Section 4.2 for more details).

3.3 Syntactic Queries

Corpora that model dependency relations can be queried for syntactic relations. For example, the corpus in Figure 3 defines an entity named *DepRel* which models syntactic dependency relations. The illustrated query delivers all pairs of tokens that belong to the same sentence *Segment*, where the head is a form of the verb 'take' and the dependent is its object (the labelset uses 'dobj' for direct objects).

See also Section 4.4 for representation and implementation details regarding dependency structures.

3.4 Query Language

The CQP query language (Christ, 1994) ingeniously translated the concept of regular expressions into corpus queries, supporting any pattern of adjacent tokens with attributes that can be modeled as automata. Its widespread use attests to its versatility. Other query languages have extended CQP's potential by providing for syntactic query operators. The ANNIS query language AQL (Zeldes et al., 2009), in particular, provides a multitude of operators for both constituency and dependency syntax.[4]

Clematide (2015) classifies several query languages with regard to their purpose and functionality. In general, a trade-off between efficiency and complexity (of both the data and the query language) can be observed. However, to determine this trade-off in a sound way, or even comparing different query languages regarding those effects is very hard as pointed out by Lai and Bird (2010, p. 59):

> The problem of comparison [of query languages] is exacerbated by the fact that most implementations are tailored to the flat file representation used by a specific corpus, and cannot

[4]Fundamental syntactic queries had already been established by its predecessor TIGERSearch (König & Lezius, 2000).

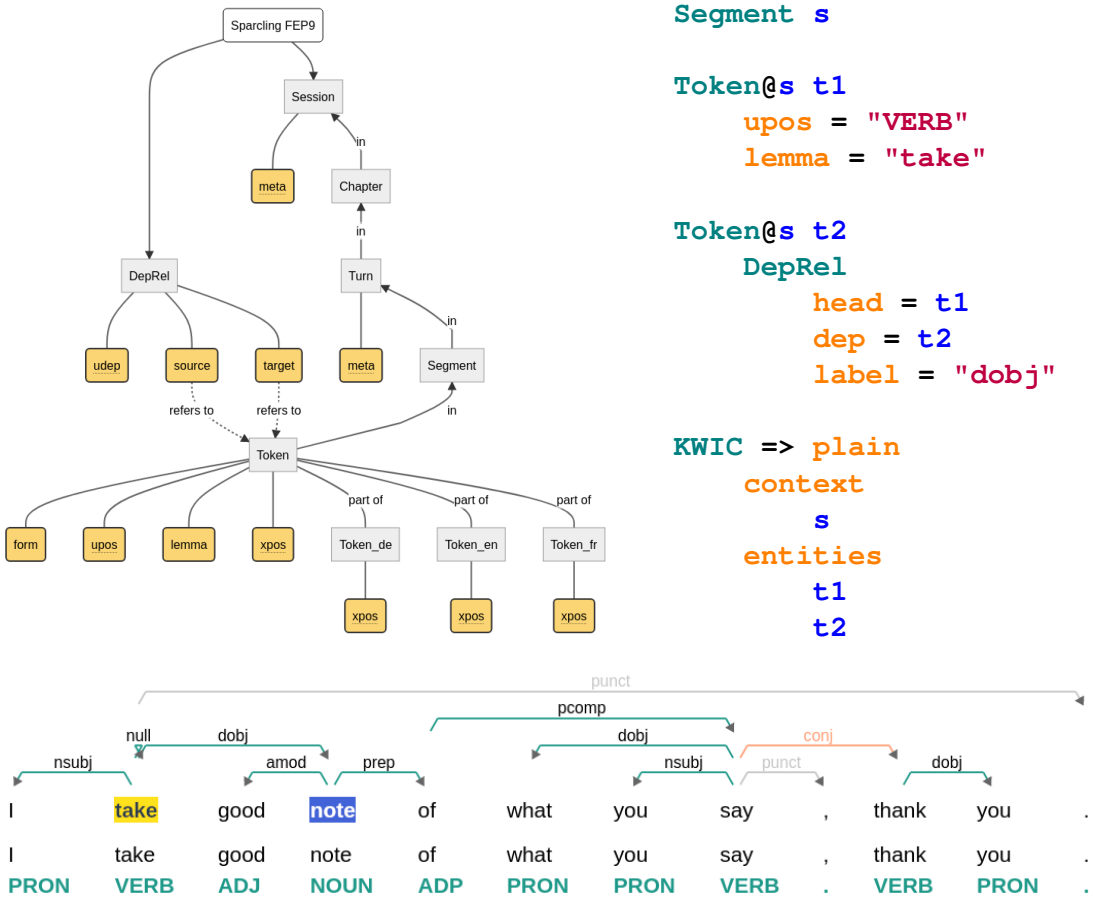

Figure 3: The automatically generated visualization of the corpus structure (left), a syntactic query for a simple dependency relation to be run against the said corpus (right) and an example sentence from the 'KWIC' result set (bottom).

> be compared directly. Comparing code is hampered by the fact that the code is not always available, and because query processing is intertwined with idiosyncratic indexing and storage. In general, these languages are not compiled into an existing general purpose language (such as SQL), which means that their relationship to such languages is not known. Moreover, standard indexing and optimisation techniques cannot be applied, and the implementations we have experimented with do not scale [...]. For these reasons, it is difficult to establish the formal expressiveness of existing linguistic tree query languages, or establish the asymptotic efficiency of their implementations.

For our query language DQD (Descriptive Query Definition), a key concept was the Entity–relationship (ER) model used for modeling data structures in software development, especially for database schemata. For corpora hosted by our infrastructure, the structure needs to be defined in terms of entities (including a list of their attributes and attribute types), and relations between them; we refer to this structure as *corpus template*. In contrast to free modeling in ER diagrams, we limit data and relation types to a predefined set, and require corpus developers to define three fundamental units:

1. the main unit of interest (e.g. tokens),
2. meaningful ordered collections of those units (e.g. sentences), and
3. the hierarchically highest unit that contains ordered elements (e.g. documents)

The query language DQD then refers to a particular constellation of entities and their relations (including non-existing parts). We can put constraints on entities (existing and non-existing ones) in form of logical formulae represented as trees. DQD uses indention to express subordination.[5]

In absence of root-level logical operators, we assume conjunction (as used for conditions on *Token 't2'* in Figure 1). Similarly, we assume existential quantification if existence is not explicitly negated (for more details on quantification, see the following section). [6] A DQD query introducing entities e_1 to e_n with their subordinated propositions p_1 to p_n is thus matching everything where $\bigwedge_i \exists e_i \wedge p_i(e_i)$ holds.

DQD uses three set-defining operators:

1. **sequence** for patterns on units with optional repetition ranges similarly to the ones introduced by the CQP query language,
2. **group** for conflating elements or sets of elements of the same type into a single set, and
3. **set** for a – potentially empty – set of units of the same type defined by a logical proposition (see *tverbs* in Figure 4)

```
Segment s

Token@s tverb
    upos = "VERB"

set tdeps
    Token@s tx
        DepRel
            head = tverb
            dep = tx
            label != "punct"
```

```
KWIC => plain
    context
        s
    entities
        tverb
        tdeps
```

Figure 4: A query searching for verbs and all their dependencies with a label different to "punct". The results set returns verbs and dependencies within the context of sentence *Segments*.

Besides the definition of such a constellation, DQD also defines one or many datasets to be returned by the query. We currently support three types, namely:

1. a **plain** view on the data within a defined **context** (e.g. sentence or document) and the entities in this context to be marked (e.g. tokens or sentences),
2. a statistical **analysis** of the results on one or more **attributes**, using one or more **functions** (e.g. frequency), applying an optional **filter**,[7] and
3. a **collocation** analysis on a defined **attribute**, with either a **center** unit (token) and a **window**, or a **space**, which is a set of tokens[8]

A list of operators are defined to operate on numbers and number ranges (including times and time ranges), strings and entities. Alignment operators can be employed for defining correspondence on parallel corpora.[9]

[5] DQD comes in a textual and a corresponding JSON format; for reasons of simplicity, we will only refer to the textual representation when discussing DQD features.

[6] By convention, we use lowercase for attributes (lemma, upos, label, ...) and camel case for entities (Token, DepRel, ...). Logical operators and quantifiers are required to be written with uppercase letters (NOT, AND, EXISTS, ...).

[7] The filtering happens after the aggregates have been calculated

[8] The latter is useful for collocation analysis in dependency relations.

[9] Along the lines of "hierarchical alignment" described in (Graën, 2018, p. 76ff).

3.5 Existential Constraints

An important aspect of query languages targeted at linguistic corpora concerns the possibility to express constraints on the *absence* of data. Consider, for example, the pseudo-query "find all sentences that have no verb", which requires the query language to have some means to express both universal quantification and negation on the level of quantification. This query can be expressed in relational calculus as $\{\langle s\rangle|\neg\exists t \in s \wedge \text{verb}(t)\}$ or, universally quantified, as $\{\langle s\rangle|\forall t \in s \rightarrow \neg\text{verb}(t)\}$.

```
Segment s

Token@s t1
    upos = "NOUN"

¬EXISTS
    Token@s t2
        upos = "VERB"
```

```
KWIC => plain
    context
        s
    entities
        t1
```

Figure 5: A query searching for sentence *Segments* with at least one noun but no verb.

While existentially quantified entities have a global scope in DQD and can be referred to from anywhere after their declaration in DQD, universally quantified entities cannot as their existence is bound to the scope of their quantification. The *Token* 't2' in Figure 5, could thus only be referred to within the scope of the '¬EXISTS' quantification. For the result set 'KWIC', 't1' is thus the only visible entity.

As universal and existential quantification can be translated to one another by negating both the quantification and the logical formula under the quantification's scope, we only support negated existential quantification in DQD – in addition to regular existential quantification.[10] Given that a query needs to declare at least one result set for it to return anything at all, and that result sets need to make use of at least one entity, all DQD queries consequently have at least one entity that is existentially quantified.[11]

Lai and Bird, 2010 demonstrate that while TGrep2 only enforces an "(implicit) outermost quantification [that] is always existential" and allows other forms of quantification on lower scoped variables while inhibiting "negation outside an entire expression", in TIGERSearch "all variables are existentially quantified" and the language further inhibits "negation to scope over this implicit existential".

4 Technology

We offer a flexible solution that we have designed for several types of corpora, ranging from Digital Editions (where a reference to an area in the original manuscript is essential), multimodal corpora (where movies and temporal annotation are required), and very large corpora (where short retrieval times are a priority, even for complex corpus templates).[12]

The corpora that we make available through the system differ in various aspects such as size, annotation layers, and complexity. While structurally complex corpora pose a challenge for the construction and processing of a dynamic query language, very large corpora, on the other hand, demand the efficient retrieval of previously unseen queries. Because of the latter, we cannot precompute partial results, but we can still precompute efficient data representations and index structures.

We accommodate structurally complex corpora, and in particular parallel corpora, by representing alignments as structural elements, similar to dependency relations.[13] We employ PostgreSQL, a modern database management system,[14] to deal with the latter challenge, unlike the developers of ANNIS

[10] Recall from Section 3.4 that all entities are existentially quantified if not declared otherwise.

[11] Even if a result set refers to a set instead of a singleton and all instances of said set are empty, it still exists.

[12] See Section 3.4.

[13] There are three main differences between these two relations: 1) alignments represent correspondence and are thus undirected, 2) alignment links are typically unlabeled, and 3) the data type of alignments are (nested) sets rather than trees (see Graën, 2018).

[14] https://www.postgresql.org/

(Krause & Zeldes, 2016), who replaced PostgreSQL as a storage and query backend with their own solution.[15]

The description of the methods we employ focuses on the central task of querying. Our methods have been designed to scale to very large corpora with complex annotation schemes. To this end, we have designed a query language named DQD (Descriptive Query Definition), which has two interconvertible representations, one in JSON and one in a textual format (see also Section 3.4). We can translate queries from other query languages like the CQP query language to DQD, but not the other way around, since there are queries which can be represented in DQD, but not in other languages (e.g. queries using syntactic relations or alignment structures).

Due to the complexity of its architecture (see the following section) and the intended use as a platform for making linguistic data available e.g. through the CLARIN-network, LCP is implemented as a service that can be accessed over the Internet; either through one of several dedicated web applications or directly via an API.[16] Both access methods require matching credentials.

In this section, we limit ourselves to central techniques that we leverage as part of our platform's results retrieval process. We describe further improvements not explained here in Section 5.

4.1 Platform Architecture

LCP is designed in a modular way; individual components can be replaced and layers be extended, allowing us to scale horizontally, address future needs when they arise, and offer tailored solutions to individual user groups. In particular, new frontend applications can easily be integrated and an increasing number of concurrent users can be dealt with by scaling up the number of workers and replicas in the database cluster.

Figure 6 shows the architecture of LCP. The entry point for natural users and automated access is the reverse proxy, which connects to Shibboleth for authentication if authentication is needed.[17] Requests are then forwarded to the individual applications.[18] Media files (audio, video and images) are sent directly to the client, provided approval by LAMa, the LiRI Account Manager, which handles users, groups and permissions.

The backend for all applications consists of an application that manages sessions and routes different types of queries to either regular workers or background workers, based on priorities. Regular workers perform incremental queries as described in Section 3.2 using Redis[19] for aggregating data retrieved from the database. Background workers, on the other hand, perform more time-consuming tasks such as exporting large datasets and converting them into different file formats.

Connections to the database run through a pgpool II instance that acts as both, connection pooler and load balancer, in front of the actual database cluster, which consists of a main server and several replicas, for reasons of performance and availability.

4.2 Logarithmic Partitioning

LCP places no inherent restrictions on corpus size.[20] Running a complex query on a multi-billion word corpus can potentially require hours if data is to be processed sequentially, prohibiting a seamless iterative querying approach, despite this being a typical workflow in corpus-driven linguistic research (Rayson et al., 2017). To quickly retrieve initial results, and thus enable researchers to refine their query and estimate

[15] "Instead of using the relational database PostgreSQL, a custom AQL implementation based on graphs called graphANNIS is used." (https://github.com/korpling/ANNIS/blob/main/CHANGELOG.md (2024-03-05).

[16] We provide a general overview of corpora and tools available at https://lcp.linguistik.uzh.ch.

[17] Automated access uses project-based access keys instead.

[18] We currently provide an application for querying text corpora named *catchphrase*. Another application for speech corpora is scheduled to be released in March 2024, and another one for multimodal corpora in May 2024.

[19] https://redis.io/

[20] This is one of the advantages of LCP over other systems like e.g. the Corpus Workbench, whose initial architecture restricts it to a maximum corpus size: "[...] the internal format of the indexed and compressed corpora imposes [a] 2.1 billion token limit" (Evert & Hardie, 2011, p. 13). This shortcoming is to be addressed by a new data representation format named Ziggurat (Evert & Hardie, 2015), which has not been released at this point. The authors have also witnessed the performance of ANNIS (version 3.x) dropping considerably for medium-size datasets (less than 10m tokens).

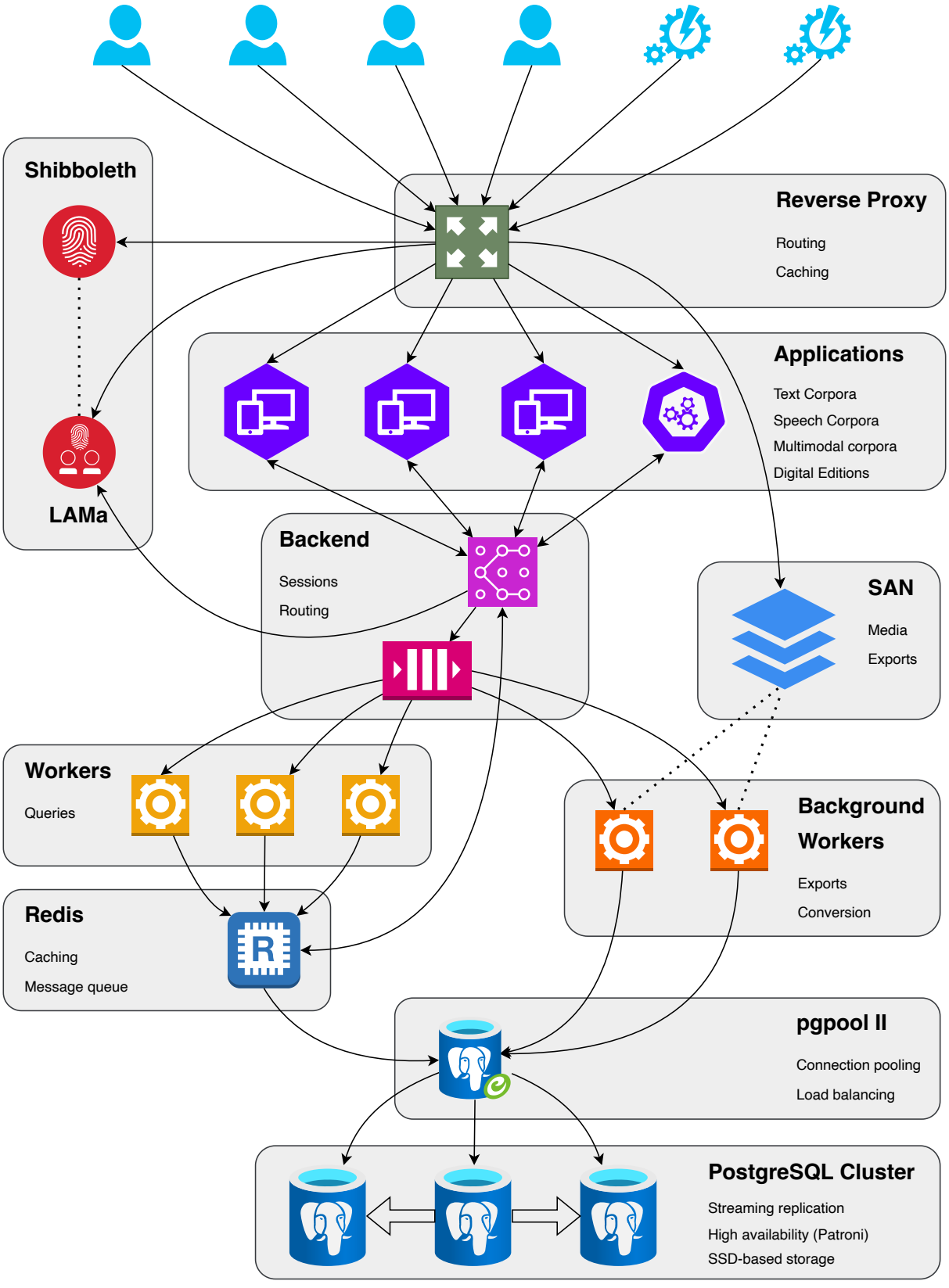

Figure 6: Architecture of LCP showing its modular design. Any number of frontend applications (purple) use a common backend, which, in turn, makes use of a variable number of workers that handle short-term or long-term requests. The database cluster can easily be extended both vertically (more disk space) and horizontally (more nodes).

the distribution of the phenomenon in question on the whole dataset, we employ a structure that allows incremental querying randomly sampled subcorpora of increasing size.

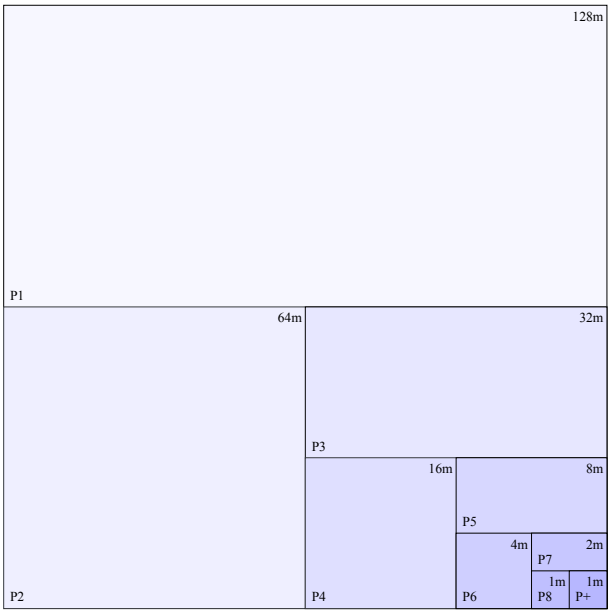

Figure 7: A corpus with 256 million segments is split into nine partitions. The last one, P+, is of approximately equal size to the second-to-last one, P8, as it contains all segments that remain after the eight-fold bisection.

To this end, we generate Universally Unique Identifiers (UUIDs) (Leach et al., 2005) for the linguistically salient units in each corpus – typically, sentences, sentence segments or utterances. We then subdivide the total space of potential UUIDs into partitions of decreasing size, always splitting one part into two halves (see Figure 7). We use Version 4 UUIDs, as they are (for the largest part) generated by a truly random algorithm,[21] so we can expect each partition to comprise a known share of the total number of units (granted a small distributional deviation that depends on the corpus size), that is, in relation to the total size, a half, a fourth, an eighth etc.

In the case of a one billion word corpus with an estimated average of ten words per sentence and a smallest partition of at least one million tokens, we would thus create eleven partitions.[22]

This logarithmic partitioning allows us to run a query on the smallest partition and extrapolating from the first result set retrieved both to the expected amount of results on the whole corpus and to the optimal partition to be queried next in order to satisfy the request for a particular number of results.

For the investigation of frequent phenomena, a small random sample is often enough. If more data is needed, our approach seamlessly scales from a quick 'pilot study' on a subset to a complete analysis of the entire corpus. Such pilot studies also allow researchers to develop and test their queries quickly, and to assess where interesting differences by the available metadata can be found (period, variety, genre, gender, etc.) in an exploratory fashion.

4.3 Indexable Vector Representations

Corpora added into our database are passed through a pipeline that computes a vector representation of each sentence, preserving the positional information of tokens; in PostgreSQL this data structure is implemented under the name of "tsvector" (Bartunov & Sigaev, 2001). This is related to the classic information retrieval task of phrase searches (Manning, 2009). In most text corpora, tokens show various layers of annotation (e.g. lemmas, part-of-speech tags, morphological features etc.). With the vector representation implemented with the help of that data structure, we can define multiple layers of infor-

[21] We actually only require the first N bits to be random, where N depends on the overall size of the corpus.

[22] 1b words / 1m words = $1000 \approx 2^{10}$ (ten regular partitions plus the remainder).

mation per position (in a sentence). In order to take them apart when querying, we need to distinguish word forms from lemmas, tagsets and so on. To this end, we prepend each string with one character per layer to tell them apart.

The advantage of such a vector structure is that it can be efficiently indexed[23] and thus allows performant retrieval of sentences that match specific criteria. Typical search patterns like sequences of tokens with additional constraints on annotations, like a CQP-style `[pos="DET"] [pos="ADJ"] [lemma="linguist"]` can be converted to a vector query that makes use of this index and quickly finds matching sentences. This often allows us to drastically reduce the number of sentences to which the original query will be applied, hence it is used to prefilter the corpus.

4.4 Nested Sets

Syntactic analyses in the form of dependencies form a tree-structure – in the mathematical sense of the word – over sentences (Crystal & Alan, 2023, p. 137). Relational Databases do not provide a straightforward way how to represent and, especially, query such data. However, several solutions for this problem have been proposed, one of them being a representation of tree-structures in the form of nested sets, which we also chose to employ for LCP. Nested sets can conceptually be conceived as sets that are completely contained within each other. The nested set representation of a tree can be computed by traversing the tree depth-first and attaching a number when a node is passed the first and when it is passed the second time (called the left and right anchor, respectively). Figure 4.4 shows the result of this process for an example sentence.

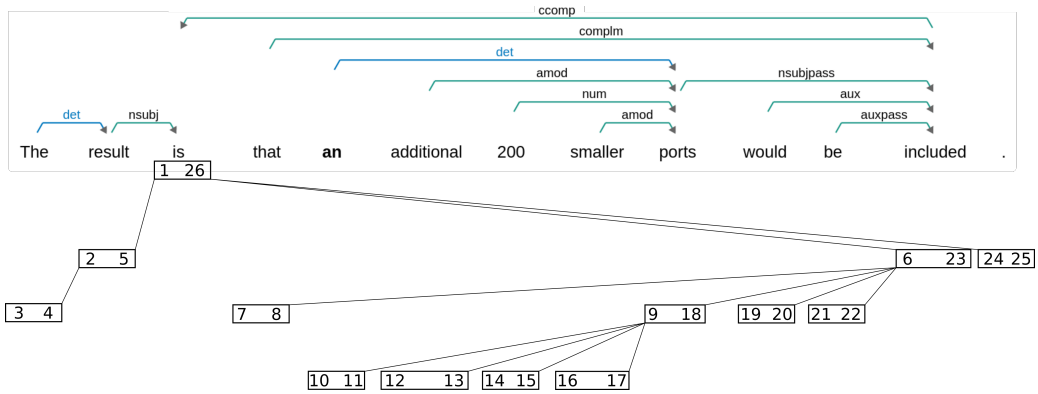

Figure 8: The dependency tree and its corresponding pairs of anchors in the nested set representation. The third word “is” is the root of the tree, and the traversal start and ends there.

As demonstrated by e.g. Celko, 2012, p. 49ff. nested sets have some useful properties: They allow, among others, e.g. (a) simple selection of subtrees, (b) the efficient computation of the height of a (sub-)tree, (c) finding paths, by restricting selections on the anchors. In other, possibly more intuitive representations (such as adjacency list, where each node points to its parent), this could only be achieved by writing more complex and computationally more expensive queries.

4.5 Linear Projection of Hierarchical Structure

Different corpora tend to be quite heterogeneous regarding the super-segmental structures that the use. They can potentially range from flat structures (e.g. a corpus consisting of recorded interviews, where no further subdivisions exists between the document level, i.e. the interview, and the utterance) to deeply nested ones (e.g. a corpus of legal texts, with a hierarchy of *Act* → *Article* → *Paragraph* → *Subparagraph* → *Point* → *Sentence*.[24] The straightforward way of representing such structures in a relational way would entail defining entities for each of the levels, with foreign-primary key relations between each pair of adjacent levels. This leads to a potentially large number of tables that need to be joined when querying

[23] https://github.com/postgrespro/rum

[24] Cf. the official style guide of the European Union for the formulation of acts: https://publications.europa.eu/code/en/en-120700.htm (2024-03-05).

the database to reach the required hierarchy level to restrict for query needs ("search for phenomenon X in sentences from only acts U, V, and W"). Our approach consists in positioning all tokens of a corpus in one virtual stream of characters[25] and compute start and end positions for all tokens, sentences, and all other hierarchy levels in this stream. This allows us to make use of PostgreSQL's native range type and operators[26], which allow for "skipping" hierarchy levels: To restrict matching sentences to only appear in certain acts, the character ranges of the respective acts can be directly compared to the indices of the sentences, without the need to first join the hierarchy chain down via the intermediate levels. In other words, this technique makes use of the fact that hierarchical containment is a transitive relation.

5 Discussion and Conclusions

We describe a new system that uses a modern database and successfully applies innovative structural and indexing solutions to the modeling and complex querying of large corpora, yielding performant retrieval of query results. Our strategy of logarithmic partitioning means that the first trends emerging from the first partitions are available quickly, which allows for prototyping on corpora of nearly unlimited size. The data model we employ in the database is designed to be as flexible as possible, allowing the representation of various corpora with very different, possibly deeply nested hierarchical structures (books, chapters, verses, parliamentary discussions, etc.).

We provide a tool for users to import corpora into LCP. It currently supports corpora that can be represented in the CoNLL-U format; additional formats will be accepted in the future to support the import of more complex corpora and of those with media files attached. Other types of corpora have been and will be imported directly by us.

In its current state, LCP comes with some limitations, of which we give both descriptions and suggestions for improvement below.

- Designing a whole new query language allowed us to make it both powerful and flexible, but its novelty also means that new users need to study it. To support them, our interfaces provides a query editor with auto-completion feature and a visual representation of the corpus structure. While we plan to provide options to translate CQP statements to DQD queries to ease the transition for users familiar with that language, DQD aims to be expressive and declarative and can unleash the full potential of our platform. The examples we provide for each corpus can easily be edited, and we are actively working on extending our documentation, as well as planning to teach workshops on LCP.

- Although CoNLL-U provides a reliable standard for corpus imports, it also comes with limitations on what information it encodes, and the importing process remains non-trivial. We will first extend support to the CoNLL-U Plus format, which allows for arbitrarily many fields, and will document common use cases such as named entities of multi-media corpora as we automatize their importing process.

- The fast and optimized architecture entails that hosting new server instances takes considerable knowledge and needs a relatively expensive infrastructure. Parties interested in deploying their own setup will find all the code of the individual modules accessible as open source software.[27]

- Our basic unit of analysis is the sentence. A consequence of this is that delivering other levels, for instance entire documents, requires extra retrieval steps. The platform compensates some of the extra cost by caching the results of recent queries, so that future queries can partially or completely reuse previous data in an optimized way.

- Negated existential queries are complex, as the entire corpus needs to be searched. Fortunately, the use of vectors allows us to query efficiently also in this situation.

[25] Similarly, we use a time axis for speech and multimodal corpora.
[26] https://www.postgresql.org/docs/current/functions-range.html
[27] The source code can be obtained from https://github.com/liri-uzh/lcp.

- We only support dependency syntax. This is a limitation compared to some other systems. However, LCP allows to define entities spanning arbitrary sequences, which are used, for instance, for the annotation of Named Entities, and could also be employed to mark phrases.

In the future, we will integrate further and more varied corpora. We have built frontends for text, speech and multimodal corpora, as well as a tool for visualizing content analysis with a purely graphical interface, allowing keyword extraction, topic modeling and time series analysis, using, among others, a large collection Swiss newspaper articles from the past decades (Graën et al., 2023).

References

Bartunov, O., & Sigaev, T. (2001). *Full-Text Search in PostgreSQL* (tech. rep.).

Celko, J. (2012). *Joe Celko's Trees and Hierarchies in SQL for Smarties*. Elsevier.

Christ, O. (1994). A Modular and Flexible Architecture for an IntegratedCorpus Query System.

Clematide, S. (2015). Reflections and a Proposal for a Query and Reporting Language for Richly Annotated Multiparallel Corpora. In G. Grigonyte, S. Clematide, A. Utka, & M. Volk (Eds.), *Proceedings of the Workshop on Innovative Corpus Query and Visualization Tools at NODALIDA* (pp. 6–16). Linköping University Electronic Press.

Cochran, W. G. (1977). *Sampling Techniques* (3rd). Wiley.

Crystal, D., & Alan, C. (2023). *A dictionary of linguistics and phonetics*. John Wiley & Sons.

Desagulier, G. (2019). Can word vectors help corpus linguists? *Studia Neophilologica*.

Evert, S., & Hardie, A. (2011). Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium.

Evert, S., & 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)*, 21–27.

Ghodke, S., & Bird, S. (2012). Fangorn: A system for querying very large treebanks. *Proceedings of COLING 2012: Demonstration Papers*, 175–182.

Graën, J. (2018). *Exploiting Alignment in Multiparallel Corpora for Applications in Linguistics and Language Learning* [Doctoral dissertation, University of Zurich].

Graën, J., Mustac, I., Rajovic, N., Schaber, J., Schneider, G., & Bubenhofer, N. (2023). Swissdox@ LiRI – a large database of media articles made accessible to researchers. *CLARIN Annual Conference Proceedings*, 111–115.

Gries, S. T., & Berez, A. L. (2017). Linguistic Annotation in/for Corpus Linguistics. *Handbook of Linguistic Annotation*, 379–409.

Hearst, M. A. (1999). Untangling Text Data Mining. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 3–10.

Ide, N., & Romary, L. (2004). International Standard for a Linguistic Annotation Framework. *Natural Language Engineering*, *10*(3–4), 211–225.

Ide, N., & Romary, L. (2006). Representing Linguistic Corpora and Their Annotations. *LREC*, 225–228.

König, E., & Lezius, W. (2000). A Description Language for Syntactically Annotated Corpora. *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*.

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

Lai, C., & Bird, S. (2010). Querying Linguistic Trees. *Journal of Logic, Language and Information*, *19*, 53–73. https://doi.org/10.1007/s10849-009-9086-9

Leach, P., Mealling, M., & Salz, R. (2005). *A Universally Unique IDentifier (UUID) URN Namespace* (tech. rep.).

Lehmann, H. M., & Schneider, G. (2012). BNC Dependency Bank 1.0. In S. O. Ebeling, J. Ebeling, & H. Hasselgård (Eds.), *Studies in variation, contacts and change in English, volume 12: Aspects of corpus linguistics: Compilation, annotation, analysis*. Varieng.

Manning, C. D. (2009). *An Introduction to Information Retrieval*. Cambridge University Press.

Meurers, W. D. (2005). On the use of electronic corpora for theoretical linguistics: Case studies from the syntax of German. *Lingua*, *115*(11), 1619–1639.

Rayson, P. E., Mariani, J. A., Anderson-Cooper, B., Baron, A., Gullick, D. S., Moore, A., & Wattam, S. (2017). Towards Interactive Multidimensional Visualisations for Corpus Linguistics. *Journal for Language Technology and Computational Linguistics*, *31*(1), 27–49.

Rohde, D. L. T. (2005). TGrep2 user manual.

Schneider, R. (2013). KoGra-DB: Using MapReduce for language corpora. In M. Horbach (Ed.), *INFORMATIK 2013 – Informatik angepasst an Mensch, Organisation und Umwelt* (pp. 140–142). Gesellschaft für Informatik e.V.

Zeldes, A., Lüdeling, A., Ritz, J., & Chiarcos, C. (2009). ANNIS: A search tool for multi-layer annotated corpora.