Corpuscle – a new corpus management platform for annotated corpora

Draft version, please do not circulate
2011-05-14

Paul Meurer

Uni Computing; University of Bergen

Abstract. Corpuscle is a new corpus query engine and Web-based corpus management system. The main design goals were the ability to handle very large corpora, support for structured data (XML), and seamless integration of manual corpus annotation and editing. New algorithms have been developed, among them a technique for running finite state automata from edges with lowest corpus counts, and an implementation of regular expressions on suffix arrays for fast reverse index lookup. These algorithms allow for a clean and elegant implementation of multi-valued and set-valued attributes. The Web interface offers rich functionality for concordancing, collocations, distribution statistics, and more. Queries can be input in a graphical, menu-driven way, freeing the user from dealing with the complexities of the query language.

1 Introduction

In this chapter, I describe Corpuscle, a new corpus query engine and corpus management system that is under active development and already in use for a variety of corpora of different types at Uni Computing and the University of Bergen.

The corpus engines that are most widely used today (e.g. Open Corpus Workbench (CWB), Manatee/SketchEngine) were developed in the 1990s, when corpora still were well below 100 million tokens in size, and harddisk and memory capacity were limiting factors. Consequently, much effort was put into achieving high data compression, whereas search algorithms were not overly sophisticated. Some of those systems also suffer (or have suffered until recently) from missing Unicode support, missing support for structured data, a complex programming interface, and restrictive licensing. In contrast, today's largest corpora are approaching a billion tokens or are already significantly larger, and disk space and memory are cheap commodities. Support for Unicode and structured data (XML) are important concerns.

With these changed circumstances in mind, I started designing and implementing Corpuscle in late 2009.[1] My goal was to create a state-of-the-art corpus

[1] The development of Corpuscle was financed through a grant from the Meltzer Foundation.

system that would be flexible enough to handle very large corpora like the Norwegian Newspaper Corpus (around 900 million tokens) in an efficient way and at the same time be useful for the diversity of corpus projects Uni Computing is involved in. Considerable effort has been put into the design of fast search algorithms, whereas data compression has had no priority. Several novel algorithms[2] were developed, among them a technique for running finite state automata from edges with lowest corpus counts[3], and an implementation of regular expression matching on suffix arrays for fast lexicon lookup.[4]

An evaluation of our system against Corpus Workbench using the Norwegian Newspaper Corpus and other large corpora (see Evaluation section) shows that our system is comparable in speed or significantly faster on most types of queries.

In the design of the new corpus system, the main requirements were the following:

- Full Unicode support[5]
- Support for hierarchically structured data (e.g., XML)
- Clean implementation of multi-valued attributes (used to encode ambiguity) and set-valued attributes (used to encode sets of features).
- Support for very large corpora (order of magnitude 2 billion tokens and more)
- Query execution speed comparable to or better than Corpus Workbench, in particular on queries with high-frequent initial position and on queries involving certain types of regular expressions
- Powerful query syntax, comparable to and modeled along CQL ("Corpus Query Langugage", the querying language of Corpus Workbench), with better support for structured data
- Seamless integration of manual corpus annotation and editing, with live querying, at least for smaller corpora
- Rich functionality for concordancing, collocations, distribution statistics etc.
- Well-defined API[6] (Application Programming Interface)
- Integrated Web interface

[2] These algorithms will be described in detail in a forthcoming paper.

[3] See (Evert: 2007) for a discussion of the shortcomings of invariably running the automata from the start edge.

[4] See next section.

[5] Corpuscle supports presently only the Basic Multilingual Plane (BMP) subset of the Unicode code range. The BMP consists of the first $2^{16} = 65536$ Unicode characters and contains characters for almost all modern languages, whereas some ancient languages (e.g., Gothic) are not fully covered. If there is need, Corpuscle can easily be extended to support all of Unicode. Using the BMP alone has however significant advantages, as every character can be uniformly stored as a two-byte integer (in the UCS-2 format) and does not have to be decoded when read in again. This is in contrast to the popular format UTF-8, which is capable of encoding any Unicode character, but costly decoding is necessary when reading from file.

[6] An API is a formal specification of how other software programs can be built on top of the core functionality of a system (here, Corpuscle).

2 Design principles

The overall design of the query engine of Corpuscle is quite similar to that of Corpus Workbench.

In a technical sense, a corpus can be described as a sequence of corpus positions that are annotated with values for one or several attributes (also called positional attributes). The words of the underlying texts comprise the mandatory attribute *word*, that is, a minimally annotated corpus consists of a sequence of words. Other possible attributes may encode grammatical features (part of speech, morphosyntax etc.), other linguistic annotations and metadata. Attribute values can be atomic or structured. Corpuscle has built-in support for structured attributes, both for multi-valued and set-valued attributes and combinations thereof.

In addition to corpus positions that encode word tokens together with their annotations, Corpuscle also uses corpus positions to encode structural data.[7] Here, Corpuscle differs from CWB, where structural attributes are tied to the corpus position of the following word token. The advantage of giving structural data their own dedicated corpus positions is twofold: First, the original order of consecutive XML tags can always be unambiguously reconstructed from the encoded corpus. And second, structural data can also have positional attributes. This can be an advantage when attributes that scope over more than a single corpus position are coded as positional attributes.[8]

A collection of annotated documents has to be formatted as a *vertical file* before it can be imported into Corpuscle. The vertical file that has a format similar to CWB's vertical file. Each line in the vertical file corresponds to a corpus position, and the values of the positional attributes that are defined for the given corpus are separated by *tab* characters. Structural positions are represented by XML-style tags that may include attribute-value pairs (e.g., <text id="314" lang="kat">). They, too, can have positional attribute values.

A vertical file can be automatically derived from XML-encoded documents; the resulting corpus will then only contain structural positions (corresponding to XML tags) and the *word* attribute, corresponding to the sequence of word tokens in the document. When the corpus documents are linguistically annotated, it is however desirable to encode the linguistic annotation as separate attributes, which necessitates preprocessing of the data in a more sophisticated way.

A corpus cannot be searched efficiently without an *index*. An index is (much like a book index) a set of data structures that encode for every word in the corpus the positions of all its occurences in the corpus. Thanks to the index, it is not necessary to scan the whole corpus position for position to find the occurences

[7] Since structural data is represented in the vertical file as XML start, end and empty tags, possibly containing arbitrary XML-style attribute-value-pairs, I will use the terms XML tag and structural attribute (value) synonymously.

[8] As a consequence, the query language of Corpuscle has a semantics that differs slightly from that of CQL. See also Section 3, where an extension of the query language is described that makes it possible to easily ignore structural positions in queries.

of a given search word. Such an index exists for every corpus attribute; it is called an *inverted index*.

For efficiency reasons, the annotated corpus text itself is encoded as a set of sequences of numerical value *id*s (numbers), one sequence per attribute. For each corpus position, the value of a given attribute is represented by a unique *id*, equal values having equal *id*s.

The data structure that implements the mapping between attribute values and value *id*s and vice versa is called the *attribute lexicon*. In Corpuscle, I have chosen *Suffix arrays* (Manber, Myers: 1991) as the data structure to implement the lexica. A suffix array is an index data structure that indexes every substring of a given (potentially very long) string, which means that the occurrences of any substring can be looked up very efficiently. In our case, this long string is the attribute lexicon.

Suffix arrays are also well suited for the implementation of fast regular expression matching in the lexicon. In CWB and elsewhere (e.g. in the Unix utility *grep*), a regular expression is evaluated by linearly scanning the lexicon for matching strings. Although linear search can be quite fast in many cases when techniques like Boyer-Moore search are used, it is slow when the regular expression contains few consecutive characters. An example is the expression ".*ja.*zz.*", which matches a string containing the substrings 'ja' and 'zz', in that order. Since the lexicon suffix array indexes every substring of all lexicon strings, the strings containing 'ja' and 'zz' are found efficiently without the need to scan the lexicon.[9]

In Fig. 1, vertical file, encoded corpus, inverted indexes and attribute lexica are given for a rather short corpus with attributes *word* and *pos* (part of speech).

vertical file

word	**pos**
<NP>	
the	Det
girl	N
with	PP
<NP>	
the	Det
telescope	N
</NP>	
</NP>	

word corpus

cpos	**id**
0	4
1	2
2	0
3	3
4	4
5	2
6	1
7	5
8	5

word lexicon/inv.i.

id	**value**	**inv.i.**
0	girl	2
1	telescope	6
2	the	1, 6
3	with	3
4	<NP>	0, 4
5	</NP>	7, 8

pos corpus

cpos	**id**
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	0
8	0

pos lexicon/inv.i.

id	**value**	**inv.i.**
0		0, 4, 7, 8
1	Det	1, 5
2	N	2, 6
3	PP	3

Fig. 1. A short corpus with vertical file, encoded corpora, lexica and inverted indexes

[9] For this query, Corpuscle is almost 5 times faster than CWB.

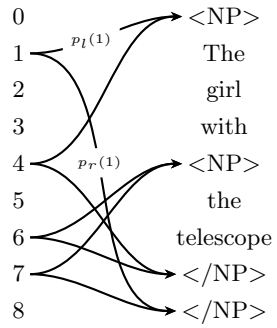

Fig. 2. The corresponding structure tree. Not all pointers are drawn.

An additional data structure, the *structure tree*, encodes the hierarchical structure of the corpus. The structure tree contains for every corpus position a pair $(p_l(c), p_r(c))$ of pointers that point to the structural element that immediately contains the position. More exactly, if c is a start tag position, $p_l(c)$ points to the start tag of the enclosing element and $p_r(c)$ points to the corresponding end tag; if c is an end tag position, $p_l(c)$ points to the corresponding start tag and $p_r(c)$ points to the end tag of the enclosing element; otherwise, $p_l(c)$ and $p_r(c)$ point to the start and end tags of the enclosing element. In Fig. 2, the structure tree of the corpus in Fig. 1 is displayed.

Using the structure tree, it is straightforward to navigate in the XML tree of the underlying texts and to calculate sentence and other contexts for a given corpus position. The structure tree is also consulted when it has to be checked whether matches of balanced tags (start and end tags with equal name) in a query correspond to matching tags in the corpus.

As CWB and Manatee do, Corpuscle uses a static, file-based representation of the encoded corpus, the inverted indexes and the attribute lexica.

3 Querying the corpus

The query language Corpuscle has a query language in which complex queries can be succinctly formulated. It is based on regular expressions over strings, corpus attributes and corpus positions. The query language is similar to CQL, the query language of Corpus Workbench, and tries to be compatible with it as far as possible. On the other hand, since Corpuscle implements features that are missing in Corpus Workbench, and vice versa, there are necessarily differences in the query languages.

Overview of the query langugage [10] The basic building blocks of a query are *positional constraints*. A positional constraint matches a corpus position if all attribute values in that corpus position satisfy the conditions of the positional constraint. Normally, a positional constraint is written as '[...]'. In the simplest case, the brackets are empty ('[ ]'), they contain no condition, which means that the constraint matches any corpus position. *Attribute constraints* are of the form '*attribute*="*value*"'. A constraint containing an attribute constraint would be '[word="fish"]', which would match every corpus position where the *word* attribute had the value "fish". This constraint can be written abbreviated as '["fish"]' or even "fish". In a positional constraint, attribute constraints can be combined using the boolean operator *and* ('&'), e.g.: '["fish" & pos="V"]'. This query matches all occurrences of "fish" that are marked with part of speech "V".

In addition to a literal string, the value expression in an attribute constraint can be a regular expression, e.g., '[lemma="book.*"]', which would match all corpus words starting with "book". To give characters that have a special meaning in regular expressions their literal meaning, they have to be escaped, e.g., "\." matches a dot, whereas "." would match any character.

Positional constraints can be combined into a regular expression using the sequence operator (juxtaposition of positional constraints) and the following operators that apply to the preceding expression (which can be a positional constraint or a complex expression in parentheses):

$*$	Kleene star: arbitrary many repetitions, including zero
$+$	Kleene plus: at least one repetition
$\{n, m\}$	bounded repetition: between n and m repetitions
?	optionality
|	disjunction

A query expression that illustrates the use of some of those operators would be

(1) "I" [pos="V"]+ [ ]{1, 2} "to" [pos="V"]

This query searches for occurences of "I", followed by some verbs, followed by one or two arbitrary positions, followed by "to", and finally followed by a verb again. Thus, it would match the phrase "I would want these things to happen", but not "I have to go".

Structural positions (XML tags) can also be queried for. They are simply written in standard XML notation (e.g., '<s>' or '</s>'). When a start and an end tag with the same element name appear in a query expression, it is assumed that matching tags should correspond to each other; they should be start and end tag of the same XML element. This means for example that a query like

(2) <s> [ ]* "jeg" [ ]* "deg" [ ]* </s>

[10] An exhaustive description of the query language can be found in the on-line documentation of Corpuscle at <http://iness.uib.no/corpuscle>.

always should find “jeg” and “deg” in the same sentence. It also means that in a situation where XML elements can be embedded in elements of the same type, the retrieved start and end tags always correspond. A query like

(3) <NP> [ ]∗ “girl” [ ]∗ </NP>

would match the phrase

(4) <NP>the girl with <NP>the telescope</NP></NP>

and not the shorter sequence

(5) <NP>the girl with <NP>the telescope</NP>,

since here, the </NP> is not the corresponding end tag of the leftmost <NP>.

Attributes of XML tags can also be queried. The syntax is equal to the XML tag syntax, with the difference that values of attributes can be regular expressions. A valid query would be:

(6) <s type=“main” lang=“nob|nno”>

When writing a query, one has to be aware that structural positions are corpus positions in their own right, in contrast to CWB. A query like

(7) “with” “the” “telescope”

will not, as perhaps intended, match the subsequence “with <NP>the telescope” of (5), because of the intervening <NP>. A query that would match that subsequence can be written like this:

(8) “with” <>∗ “the” <>∗ “telescope”

to allow for XML tags between the words, where ‘<>’ denotes an arbitrary start or end tag. The same query can be formulated in a more succinct and elegant way using the *ignore tag* operator (written as backslash, ‘\’), followed by the tags that should be ignored:

(9) “with” “the” “telescope” \ <>

In some situations, for example when non-textual material is interspersed with text, it is desirable to ignore not only XML tags, but entire XML elements. This can be achieved using the *ignore element* operator (written as double backslash, ‘\\’). In query evaluation, all elements with start tags listed after the *ignore element* operator will be disregarded.

A typical corpus where the *ignore element* operator is essential is an electronic critical edition of a literary œuvre, like for example Ludvig Holberg’s[11] writings. The XML format of the edited texts is rather complex. In the main text, which basically follows the first edition, notes and deviating readings from later editions are inserted as XML elements. In the example fragment (10), the main text is in boldface; it consists of the top-level text and those parts that are enclosed in <lem> elements inside <app> (apparatus) elements.

[11] Ludvig Holberg was a central figure in the Danish-Norwegian literature of the 18th century. A critical edition of Ludvig Holberg’s writings has been imported into Corpuscle for testing purposes.

(10) **... thi jeg fik aldrig saa mange Hug i** <app type="tc"><lem wit="A A-2 a2 B C">**ti**</lem><rdg resp="Liebenberg">de ti<note type="last">(fulgt af Martensen)</note></rdg></app> **Aar ...**

Thus, in a corpus fulltext search, one would like to search exactly in that part (11) of the encoded document, and exclude text in <rdg> (denoting deviating readings) and <note> elements.

(11) ... thi jeg fik aldrig saa mange Hug i ti Aar ...

A search for "Hug i ti Aar" could then be achieved with query (12), which ignores all XML tags and all <rdg> and <note> elements:

(12) "Hug" "i" "ti" "Aar" \ <> \\ <rdg> | <note>

Attribute values can be *atomic* or *structured*. The '*word*' attribute is atomic, but attributes that encode grammatical features (part of speech, morphosyntax etc.) and other linguistic annotations can be structured in two different ways: attributes can be multi-valued, and they can be set-valued, or even a combination of both.

A typical multi-valued attribute is the '*lemma*' attribute in a lemmatized corpus where the readings are not totally disambiguated. When using a statistical tagger for morphosyntactic parsing, all readings are normally fully disambiguated, but rule-driven parsers like for example Constraint Grammar parsers output ambiguous readings.

The morphosyntactic tags that a Constraint Grammar parser attaches to a reading are a good example for a set-valued attribute ('*morph*'). Since readings can be ambiguous, this attribute is multi-valued at the same time.

Corpuscle has built-in support for structured attributes, both for multi-valued and set-valued attributes and combinations thereof. This support is also reflected in the query language. Plain multi-valued attributes do not need special syntax; a corpus position matches a constraint on such an attribute if at least one of the values matches the constraint.

Consider for example the Norwegian word "fisker", which can mean "fishes" (*pl*) or "fisherman" (*sg*). Thus, a non-fully disambiguated reading could have

(13) word = *fisker*, lemma = *fisk* | *fisker*

Both a search for '[lemma = "fisk"]' and for '[lemma = "fisker"]' will match that position, as one would expect. It is however also possible to search for unambiguous readings only, by using the *unambiguously equal-* or *strict equal-*operator ('=='):

(14) [lemma == "fisk"]

Query (14) would not match the corpus position in (13).

Similarly, there is, besides the *not-equal*-operator ('!='), a *strict-not-equal*-operator ('!!=') which can be used to demand that none of the readings should match the value in the query:

(15) [lemma !!= “fisk”]

Again, query (15) would not match the corpus position in (13), whereas the query

(16) [lemma != “fisk”]

would match (13).

The values of set-valued attributes are stored as strings with separator characters (*space*, ‘|’ or similar) enclosing the set members. Thus, the ‘*morph*’ value set ‘(*N m pl*)’ (i.e., *Noun masculine plural*) would be encoded as the string “␣N␣m␣pl␣”, and in principle, a regular expression could be used to search for subset containment. Corpuscle has however a much more efficient implementation of subset search which is based on suffix arrays, and a syntax extension that makes it easy to formulate queries on set-valued attributes: the values that are searched for can be given in the query expression as a set themselves. Consider the morphosyntactic annotation of the previous example:

(17) word = *fisker*, lemma = *fisk* | *fisker*, morph = (*N m pl*) | (*N m sg*)

A query for plural nouns can be formulated as (18), which would match the corpus position (17).

(18) [morph = (“N” “pl”)]

The *morph* attribute is in fact an example of an attribute that is both multi-valued and set-valued.

The graphical query interface. Since many users are reluctant to learn the syntax of the query language, Corpuscle has a graphical query interface (see Fig. 3) which allows the user to compose most queries in a menu-driven, graphical way. Almost all constructions of the query language are accessible in the query interface; in addition, the user gets all necessary help in chosing admissible attributes and values.

4 API and Web interface

Corpuscle is both a corpus query engine and a corpus management system. As such, it has both a well-defined Application Programming Interface (API) and a flexible and elaborate Web interface.

4.1 The API

The Corpuscle tool is written entirely in Common Lisp. It includes a Web server that makes most of the functionality of the system accessible via a simple HTTP-based protocol. Requests to the query engine and other parts of the tool can be sent to the server as *post*- or *get*-requests with documented parameters. Replies

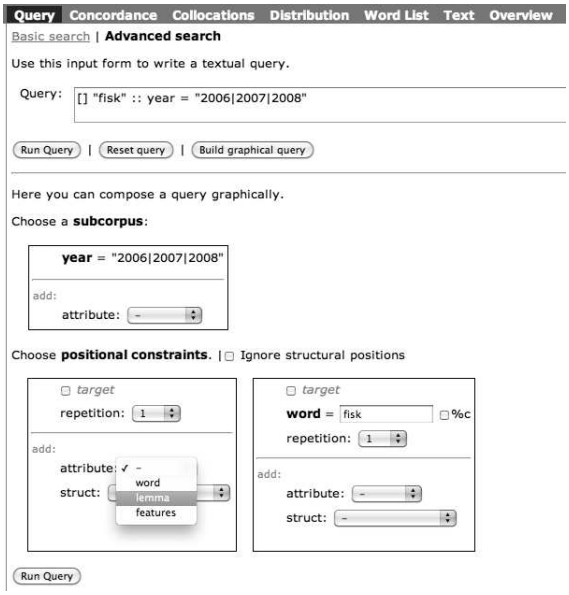

Fig. 3. The graphical query interface

are sent in the form of XML and JSON pages. When the server is contacted for the first time and authentication has taken place, a session object is established in the server that preserves state information of the connection, and a session identifier is sent back. On every following request, the session identifier has to be sent along.

This HTTP/XML/JSON-based interface is presently the only external API that is implemented. It is naturally well suited for building Web applications around the Corpuscle core functionality. When the Corpuscle system is run in default mode, the XML pages are processed by internal XSLT stylesheets that generate the HTML pages and Javascript code for the built-in Web interface. Alternatively, external style sheets can be run; in this way, programmers can adapt the Corpuscle system to their needs without having to touch the Common Lisp code.

4.2 The Web interface

Corpuscle's built-in Web interface offers typical corpus tool functionality: concordances, word lists, collocations, distribution statistics, exporting tools and more.[12]

[12] See G. Andersen (this volume) for a specific study using Corpuscle with the Norwegian Newspaper Corpus, demonstrating the value of some of these features.

Queries can be input either in textual form in Corpuscle's query syntax, or via a flexible menu-driven interface.

When a query is started, the matching corpus positions are displayed as concordance lines in the order they are calculated, already before the query evaluation is finished. The user can choose which attributes to display in the concordance and how to sort the results. Concordances can be displayed either as KWIC-concordances (see Fig. 4) or as matches in a context defined by a structural attribute (e.g., sentence or paragraph). In the concordance view, additional functionality can be available. Fig. 5 for instance shows a concordance from an annotated Georgian corpus; clicking on a context word makes the dictionary entry of the associated lemma pop up. As can be seen from this example, when a structural attribute is queried, token and word counts for the attribute are displayed: the numbers are the number of tokens resp. words between start and corresponding end tag. Those counts are also available when statistics are calculated.

When the corpus can be edited, an edit mode is available in the concordance view. From a concordance line, it is possible to jump to more context, typically the whole document.

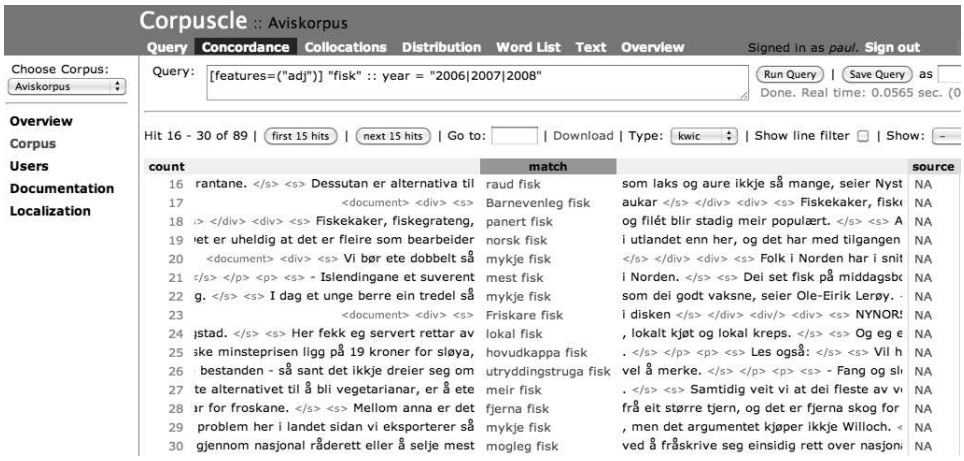

Fig. 4. The KWIC concordance page of the Corpuscle Web interface

On the Word list page, all different query matches are listed together with their counts in the result. Here, it is possible to list the matches for other attributes than *word* only. When an attribute is set-valued, both the matches for the sets as a whole and for the occurring set members can be displayed. Each match word or phrase is linked to a concordance of all contexts the match occurs in.

More advanced frequency and distribution statistics are available on the Distribution page. Here, match counts can be categorized and grouped along up to

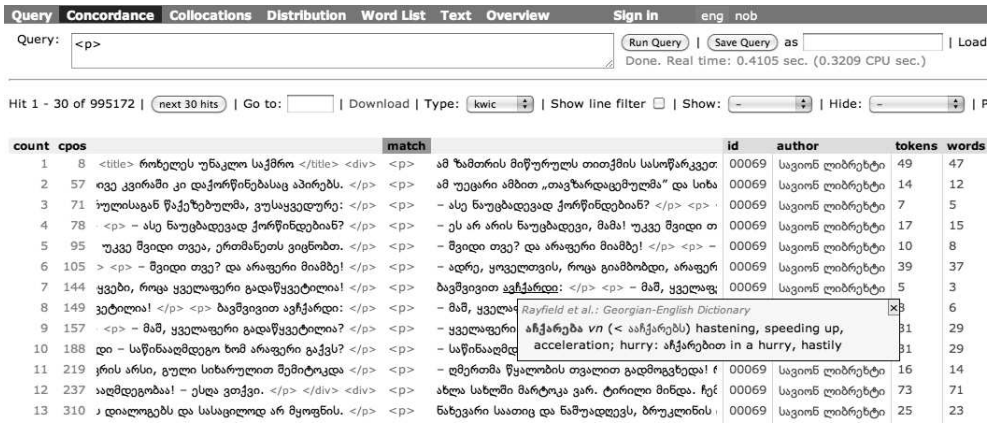

Fig. 5. Query for the structural attribute <p>; clicking on a context word makes the dictionary entry of the associated lemma pop up.

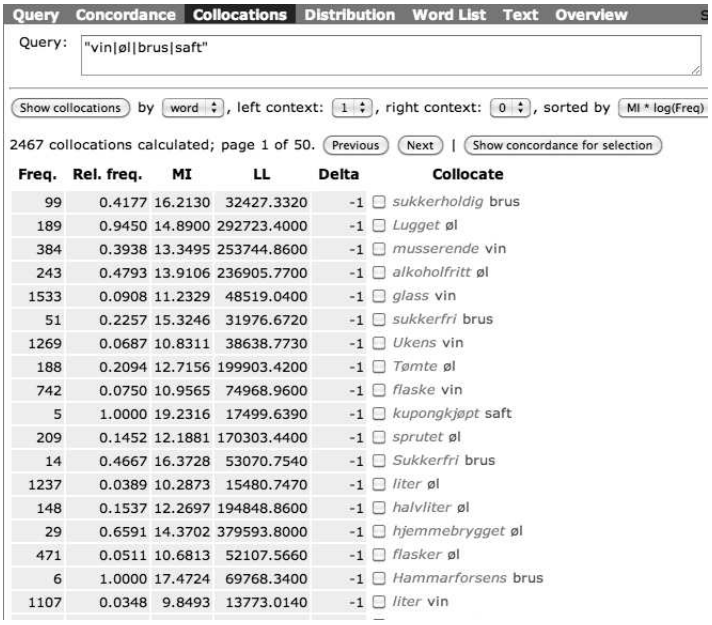

Fig. 6. A collocation for the words *vin 'wine', øl 'beer', brus 'lemonade', saft 'juice'*, sorted by $MI \log(Freq)$.

four attribute axes, and absolute and relative frequencies are calculated. For each attribute, a regular expression filter can be defined that groups attribute values

together. Values that do not match the filter are collected into a rest group. Values can also be manually grouped together via drag-and-drop. In addition, some useful filters that cannot be easily expressed as regular expressions (e.g., numeric ranges) are hard-coded.

All concordances, collocations and distributions can be downloaded in a format suitable for further processing in a spreadsheet or statistical application.

Corpuscle has built-in user management with fine-grained access control: corpora can be open for querying to anybody, or access is restricted to groups of users. Similarly, access to entire texts can be restricted, as well as the possibility to edit and to annotate.

There is extensive on-line documentation. The documentation is written using an integrated Wiki-system that is similar to Redmine's Wiki system or the Wikipedia.

The main language of the user interface is English, but the interface has also been localized to Norwegian. Due to an integrated localization interface, localization to other languages is easy and does not involve editing source code or text files.

5 Editing and manual annotation

In many applications, it is essential to be able to edit or to annotate a corpus. It can be necessary to edit a corpus in order to correct errors of different types, like OCR or transcription errors, or to manually disambiguate or correct automatic morpho-syntactic annotation. Editing can mean to correct the values of attributes in certain corpus positions or even to delete or to add corpus positions. When structural positions are added or deleted, it is important to make sure that the structure of the corpus remains valid; e.g. when start tags are added or deleted, corresponding end tags should also be added or deleted.

Interactive manual annotation gives the corpus user the possibility to add interpretive information to a concordance or query result. This can be very valuable when query results have to be cleaned up or further annotated with a specific research question in mind. The manual annotation can be private or accessible to all users of the corpus. Smith et al. (2008: 175) argue convincingly for the importance of manual corpus annotation. They review existing corpus tools, discuss their functionality and shortcomings with respect to manual annotation and set up a list of "requirements for future tools". They demand greater flexibility in entering and manipulating manual annotations, and they stress the importance of data exchange between the corpus tool and other (spreadsheet and statistical) applications in both directions.

In Corpuscle, a relational database (PostgreSQL) is used to implement editing and manual annotation. The database is fully integrated into the tool: patched (edited) attribute values and manual annotations are stored in the database, and patches and annotations are searchable in the same way as attributes and values that are stored in the regular index. When a query is run,

both the static index and the database are considered; the index does not have to be recompiled to make patches or annotations visible in the query result.

Manual annotations are treated in most respects like ordinary corpus attributes. There can be different types of manual annotations as there can be different types of attributes, depending on the structure of the corpus and the user's needs. Annotations can be free text, which is useful for adding all sorts of comments to concordance lines. Annotations can also be enforcibly restricted to a finite set of values and then be used to classify matches according to a finite set of classes. If more than one classifying criterium is used, annotations are set valued, each set member being an attribute-value pair on its own. In addition, annotations can be private and only visible to the user that created them, and they can be public and thus be used to enrich the corpus with specialized information that is not available through automatic linguistic annotation.

Fig. 7 shows the use of annotations in the Norwegian dialect corpus project "Talebanken". The corpus consists of transcribed dialect recordings that are annotated grammatically and with sociolinguistic metadata. The text is linked to the sound files, and any text segment can be listened to via a keyboard shortcut in the concordance page. In Talebanken, annotations are used to classify the pronunciation of dialect words according to a set of phonetic features or variables (in the figure: F21, L02 and L12) that can have a finite set of possible values. The features and their admissible values can be edited in a separate page of Corpuscle.

After having searched for words to annotate, the annotator can put a concordance line into focus and listen to the match word in its utterance context. Then he can pick a value for a selected feature simply by using a keyboard shortcut, and the feature-value pair is added to the annotation of that word, or the annotation is changed accordingly if the feature had been present before. More than one concordance line can be put into focus simultaneously (the grayed lines in the screenshot) and a feature value can be assigned for all focussed words at once.

Fig. 7. The annotation interface of the "Talebanken" project

Implementation issues Since lookup in a relational database is intrinsically slower than lookup in a static index, extensive annotation and patching can lead to decreased query performance. This is especially true for large corpora. One way to deal with this problem is to merge the patches with the main index when the patch database has grown too large, and in the same way to maintain and update a static index for annotations.[13] A disadvantage of this approach is that the index cannot any longer be recompiled from the original texts when patches are incorporated into the index.

The structure of the tables containing the patches and the annotations has to be designed carefully. We have seen that the corpus positions are coded as consecutive integers. Since this is a problem when new positions are to be inserted, corpus positions are stored arithmetically shifted by a fixed positive integer Δ.[14] New (integer-valued) corpus positions are then inserted in the middle between existing or added (shifted) corpus positions. This allows for the insertion of many new corpus positions; even if all new positions are inserted consecutively behind an existing position, up to Δ new positions can be inserted before there is no gap available to insert the next position. Many more positions can be accommodated if new positions are inserted in a balanced way, and if inserted positions are rebalanced if necessary. There is room for up to 2^{Δ} new corpus positions between two consecutive positions that are coded in the index. It is obvious that all algorithms that deal with corpus positions have to be adapted to this extended coding scheme.

Another difficulty arises when documents are added to or deleted from an existing annotated or patched corpus. When the corpus is reindexed, all documents that follow the added or deleted document will have changed corpus positions. Since annotations and patches are tied to corpus positions, the corpus positions of the annotations and patches will have to be recomputed. This can be done by storing, in addition to the corpus positions, a combination of document name, a document-internal unique identifier of a structural (XML) element containing the corpus position and the offset of the corpus position relative to that element's start tag. If the documents have no structural coding, the offset can be taken relative to document start.

This technique allows one to recode the corpus positions of the annotations and patches by searching for the position of the document and the unique identifier using the new index, provided that both document name and unique identifier are indexed.

6 Evaluation and concluding remarks

In this section, I give an informal evaluation of the query execution speed of Corpuscle measured against Corpus Workbench.

[13] This is however not yet implemented in Corpuscle.

[14] This integer can be chosen on a per-corpus basis; in Corpuscle, $\Delta = 28$ is chosen for technical reasons. An arithmetic shift by Δ is equivalent to a multiplication with 2^{Δ}.

It is difficult to benchmark two corpus engines against each other in a systematic and fair way. The best strategy to compare their performance would be to measure query execution times on the basis of a typical usage pattern collected from the queries of several users over a longer time period. Since I do not have such data, I benchmarked some arbitrary queries that exemplify the strengths and weaknesses of both query engines. The test corpus is the English Wikipedia, containing 1,200,904,250 corpus positions. The runtimes in Table 1 are in seconds, measured on a 2×3GHz Quad-Core Intel Xeon Macintosh Pro with 26GB of memory. Queries are ordered by relative performance; those performing best in Corpuscle being on top. The Corpuscle system was compiled with Steelbank Common Lisp (SBCL). The benchmark times include both query compilation and execution and matchlist construction, and they are taken on a second run of the queries, when the needed index data were fully loaded into main memory. Ratios higher than 1 are in favor of Corpuscle.

CWB	**Corpuscle**	**ratio**	**query**
7.106	0.020	355.3	“and” “fish”
10.227	0.450	22.7	“I|you” “found|read” “this”
0.228	0.038	6.0	“.*fish.*”
0.550	0.093	5.9	“not” “even”
0.409	0.086	4.8	“.*ja.*zz.*”
0.028	0.014	2.0	“heavy” “weather”
0.285	0.183	1.6	(“I” | “you”) (“found” | “read”) “this”
0.266	0.217	1.2	“had”
336.694	290.300	1.2	#a:[ ] #b:[ ] :: #a.word = #b.word
0.018	0.018	1.0	“fish” “and”
0.212	0.218	1.0	“confus.*” [ ]{0,8} “by”
0.713	1.688	0.4	“look|bring” [word !=“i.*”]{0,10} “up|down”
0.010	0.030	0.3	[word=“fish” & path=(“title”)]

Table 1. Running times of selected queries in Corpuscle and CWB

Some remarks to the benchmark numbers: As expected, Corpuscle significantly outperforms CWB on queries that have a high-frequent token in initial position: for the query “and” “fish”, Corpuscle is 355 times faster than CWB. Whereas CWB has to check which of the 29 million occurrences of “and” are followed by “fish”, Corpuscle tests which of the 61.000 occurrences of “fish” are preceded by “and”. If we reverse the order of the query tokens (“fish” “and”), both systems use the same index lookup and perform equally well. In general, all queries involving only sequences of regular expressions on words are in Corpuscle as least as fast as in CWB.

The long running time of the query searching for two equal words in a row is explained by the fact that the whole corpus has to be scanned to find the

matches; here, no indexes can be used to speed up the search.

The Corpuscle system has now been in use for over a year in several projects at the University of Bergen and has proven to be a versatile, robust and fast corpus management system. It is used to host diverse corpora of small to large size, among which are:

- The Norwegian Newspaper corpus (940 million tokens)
- The ASK Corpus, a Norwegian Learners' corpus (1.1 million tokens)
- Talebanken, a Norwegian dialect corpus (1.3 million tokens)
- A Georgian corpus of fiction and non-fiction texts (125 million tokens)
- The English Wikipedia (1.2 billion tokens)

The development of the Corpuscle software is not completed. Among the plans for the future development of the system are:

- implementation of more advanced statistics capabilities through the integration of the R statistics package
- graphical visualization of statistics
- extensions for handling of parallel corpora
- integration of multimedia capabilities that go beyond the linking of text to sound

In addition, the Corpuscle platform is becoming an integral part of INESS, the Norwegian Infrastructure for the Exploration of Syntax and Semantics.[15] Although the main focus of INESS is on treebanks, databases of syntactically annotated sentences, the INESS infrastructure will also host traditional corpora. Finally, it is planned to make Corpuscle open source in the near future, such that interested parties will have the possibility to try and use the Corpuscle system with their own corpora.

7 References

Evert, Stefan. 2007. Inside the IMS Corpus Workbench. <http://www.iula.upf.edu/materials/080925evert_hdout.pdf>

Manber, Udi, Myers, G. 1991. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing* 22(5): 935 – 948.

Smith, Nicholas, Hoffmann, S., Rayson, P. 2008. Corpus Tools and Methods, Today and Tomorrow: Incorporating Linguists' Manual Annotations. *Literary and Linguistic Computing* 23(2): 163 – 180.

[15] See V. Rosén, this volume.