Corpuscle – a new corpus management platform for annotated corpora

Paul Meurer

Uni Computing; University of Bergen
paul.meurer@uni.no

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, 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 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 were developed, among them a technique for running finite state automata from edges with lowest corpus counts[2], and an implementation of regular expression matching on suffix arrays for fast lexicon lookup.[3]

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 on most types of queries.

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

- Full Unicode support
- Support for hierarchically structured data (e.g., XML)
- Clean implementation of multi-valued and set-valued attributes
- 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 (Application Programming Interface)
- Integrated Web interface

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 (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

[2] See [**?**] for a discussion of the shortcomings of invariably running the automata from the start edge

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

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.[4] 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.[5]

As CWB and Manatee do, Corpuscle uses a static, file-based representation of the attribute lexica (the set of strings that do occur as attribute values for a given attribute). A lexicon file implements a mapping between attribute values and numerical value IDs, and vice versa. In Corpuscle, I have chosen Suffix arrays ([**?**]) as the data structure to implement the lexica. Suffix arrays are well suited for the implementaion of regular expression matching in the lexicon. In addition, an inverted index encodes the mapping from attribute values to the sets of numerical corpus positions where those values are taken. Those two files in tandem allow one to search for corpus positions that match a given attribute value, or, more generally, a boolean combination of regular expressions in corpus attributes.

When a query comprises more than one corpus position, the query evaluation algorithm first tries to find all corpus positions that match the query constraints for one of the positions, and then checks the contexts of all matching corpus positions. This look-up is accomplished via an attribute corpus file for each attribute containing a vector of the value IDs of all corpus positions.

An additional file, the structure tree file, encodes the structure of the structural attribute. 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.

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 in a query correspond to balanced tags in the

[4] 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.

[5] 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.

corpus.

The corpus index files are calculated from a 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. They, too, can have positional attribute values.

Index files 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.

3 Querying the corpus

The query language The query language devised for the Corpuscle system 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. In some cases, Corpuscle's query language also uses a syntax that differs from CQL's for common features.

Overview of the query langugage[6] 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="verb"]'. In addition to a literal string, the value expression in an attribute constraint can be a regular expression, e.g., '[lemma="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.

As in CQL, positional constraints can be combined into a regular expression using the sequence operator (juxtaposition of positional constraints) and operators that apply to the preceding expression (a positional constraint or a complex expression in parentheses): Kleene star ('*', arbitrary many repetions,

[6] An exhaustive description of the query language can be found in the on-line documentation of Corpuscle

including zero) and Kleene plus ('+', at least one repetition), bounded repetition ('$\{n, m\}$', between n and m repetitions), optionality ('?') and disjunction ('|'). A query expression that illustrates the use of some of those operators would be

(1) "I" [pos="V"] [ ]{1, 3} "to" [pos="V"]

which would match the phrase "I 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 they should be balanced in a match. This means for example that a query like

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

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[7] 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.

(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

[7] Ludvig Holberg was a central figure in the Danish-Norwegian literature of the 18th century.

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*'. 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).

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 "␣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 as a set themselves. Consider the morphosyntactic annotation of the previous example in (16):

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

Here, query (17) would match the corpus position (16).

(17) [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 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 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 new 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.[8]

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 or as matches in a context defined by a structural attribute (e.g., sentence or paragraph). When the corpus can be edited, an edit mode is also available in the concordance view. From a concordance line, it is possible to jump to more context, typically the whole document.

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

Fig. 1. The KWIC concordance page of the Corpuscle Web interface

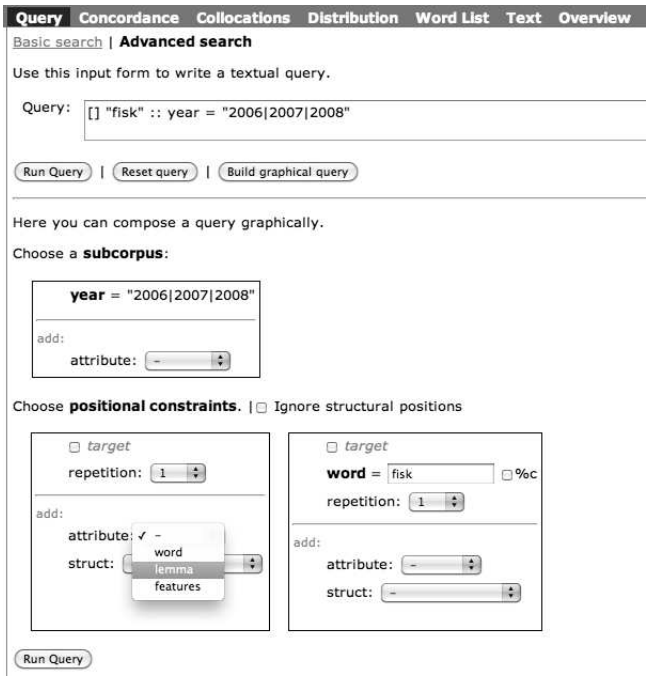
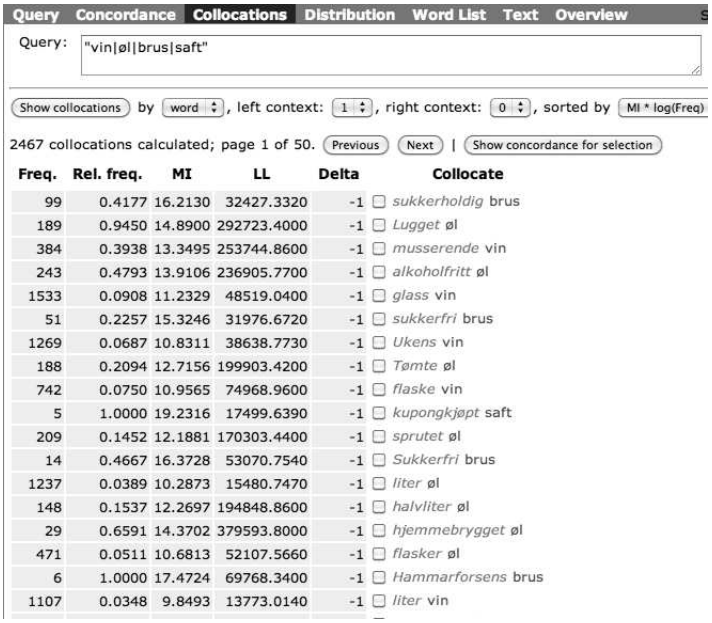


Fig. 2. The graphical query interface

On the Word list page, all different query matches are listed together with



Freq.	Rel. freq.	MI	LL	Delta	Collocate
99	0.4177	16.2130	32427.3320	-1	*sukkerholdig* brus
189	0.9450	14.8900	292723.4000	-1	*Lugget* øl
384	0.3938	13.3495	253744.8600	-1	*musserende* vin
243	0.4793	13.9106	236905.7700	-1	*alkoholfritt* øl
1533	0.0908	11.2329	48519.0400	-1	*glass* vin
51	0.2257	15.3246	31976.6720	-1	*sukkerfri* brus
1269	0.0687	10.8311	38638.7730	-1	*Ukens* vin
188	0.2094	12.7156	199903.4200	-1	*Tømte* øl
742	0.0750	10.9565	74968.9600	-1	*flaske* vin
5	1.0000	19.2316	17499.6390	-1	*kupongkjøpt* saft
209	0.1452	12.1881	170303.4400	-1	*sprutet* øl
14	0.4667	16.3728	53070.7540	-1	*Sukkerfri* brus
1237	0.0389	10.2873	15480.7470	-1	*liter* øl
148	0.1537	12.2697	194848.8600	-1	*halvliter* øl
29	0.6591	14.3702	379593.8000	-1	*hjemmebrygget* øl
471	0.0511	10.6813	52107.5660	-1	*flasker* øl
6	1.0000	17.4724	69768.3400	-1	*Hammarforsens* brus
1107	0.0348	9.8493	13773.0140	-1	*liter* vin

Fig. 3. A collocation for the words *vin, øl, brus, saft*, sorted by $MI \log(Freq)$.

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 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. The regular expression filter has to contain one or more marked subexpressions (subexpressions that are set in parentheses), and the filtered value is the concatenation of the substrings of the value matched by the subexpressions. 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 are hard-coded.

All concordances, collocations and distributions can be downloaded in a format suitable for further processing in a spreadsheet or statistical application. In addition, an interface to the statistics processing language *R* is under development that allows the graphical display of distribution statistics.

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.

Corpuscle has 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. In [**?**], Smith et al. 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 attribute values and manual annotations are stored in the database, and patches and annotations are searchable in the same way as values that are stored in the regular index. When a query is evaluated, 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. Some annotations can be free text, which is useful for adding all sorts of comments to concordance lines, and they can be enforcibly restricted to a finite set of values. They can be atomic or set valued, and they can contain

a list of attribute-value pairs. 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.

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.[9] 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.[10] 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.

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

[10] 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}.

6 Evaluation

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

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 (18) are in seconds, measured on a 2×3GHz Quad-Core Intel Xeon Macintosh Pro with 26GB of memory. The Corpuscle system was compiled with Clozure Common Lisp. The benchmarks for both systems include query compilation and execution times, and they are taken on a second run of the queries, when the needed index data were fully loaded into main memory.

(18)

Corpuscle	CWB	ratio	query
0.045	0.228	0.197	".*fish.*"
0.221	0.266	0.978	"had"
0.121	0.409	0.296	".*ja.*zz.*"
0.080	0.010	8.000	[word="fish" & path=("title")]
0.327	0.285	1.147	("I" | "you") ("found" | "read") "this"
0.038	7.106	0.005	"and" "fish"
0.072	0.028	2.571	"heavy" "weather"
0.031	0.015	2.067	"why" "not"
0.276	0.212	1.302	"confus.*" [ ]{0,8} "by"
0.175	0.550	0.318	"not" "even"
4.220	0.713	5.919	"look|bring" [word !="i.*"]{0,10} "up|down"
389.580	336.694	1.157	#a:[ ] #b:[ ] :: #a.word = #b.word

7 References

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

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

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