Daniel Janus
Warsaw University

Adam Przepiórkowski
Institute of Computer Science, Polish Academy of Sciences

POLIQARP 1.0: Some technical aspects of a linguistic search engine for large corpora

1. Introduction

The aim of this article is to present a new universal search engine for large corpora, Poliqarp,[1] whose first official version (1.0) was released in March 2006 under the terms of a free software licence (GNU GPL).[2] Although the tool was developed within a project aiming at creating the first large publicly available morphosyntactically annotated corpus of Polish, cf. http://korpus.pl/, it is universal in the sense that it uses an externally defined tagset and UTF-8 internal encoding, so it could be used for managing a corpus of any language.

The flexibility of Poliqarp and its rich query language has been discussed elsewhere (Przepiórkowski et al. 2004). In this article, we concentrate on some novel features of the current release, i.e., on various aspects of the client-server architecture of Poliqarp, as well as on the techniques used for greatly improving the efficiency of the tool.

2. Overview

2.1. From XML to the binary corpus format

The basic source format of the corpus assumed by Poliqarp is the XML Corpus Encoding Standard (XCES; cf. Ide et al. 2000). However, this representation needs plenty of disk space and is not efficient for the purposes of concordancing. Therefore, before one can search the corpus, it must be compiled to a space-efficient binary form. This is performed by a utility called *bp* (*b*uild *P*oliqarp representation), included in the distribution.

Of particular note about *bp* is the size of corpora produced by it: typically in the range of 10-12 bytes per text segment.[3] For example the 2nd edition of the IPI PAN Corpus, currently the largest corpus of Polish containing over 250 million morphosyntactically annotated segments, as well as structural information and metadata, corresponds to about 2.7 gigabytes of the binary Poliqarp representation, but as much as 7.3 gigabytes in the source format (with all texts compressed with the gzip utility).

Another feature worth mentioning is the ability of *bp* to retrieve metadata from XML header files. XCES requires that each document in the corpus contain a file named `header.xml` with, *inter alia*, bibliographical information about the source document. However, *bp* does not require this file to be fully XCES-compliant. Instead, it makes it possible to define templates of paths to XML elements for each kind of metadata defined for

[1] *Pol*yinterpretation *I*ndexing *Q*uery *a*nd *R*etrieval *P*rocessor.

[2] Poliqarp has been developed within two projects conducted at the Institute of Computer Science, Polish Academy of Sciences, led by by the second author (*The IPI PAN Corpus of Polish* — a national KBN grant, number 7T11C04320, 1 April 2001 — 31 March 2004, and *Automatic extraction of linguistic knowledge from a large corpus of Polish* — a national Ministry of Education and Science grant, number 3T11C00328, 9 March 2005 — 8 September 2007). Currently, the main developer of Poliqarp, responsible for the first official release of the tool, is the first author. The first functional version of Poliqarp, released as a binary executable program in mid-2004, was developed mainly by Zygmunt Krynicki, under the supervision of the second author, with the participation of the first author. Although the current release differs from that early version in many respects, including the general architecture and the user interface, it still contains crucial modules of the first functional version, including the corpus builder described below.

[3] A *segment* is the basic textual token, defined as a unit of text which has a morphosyntactic tag assigned to it, i.e., roughly, a word, but smaller in some cases. Also punctuation marks are separate segments.

the corpus. If an element is found to match such a template, its value is retrieved and stored in the binary corpus. For example, the following three templates are among those used in the IPI PAN Corpus:

```
(multi  "styl" "/cesHeader/profileDesc/textClass/h.keywords/keyTerm")
(single "medium" "/cesHeader/profileDesc/textDesc/channel")
(multi  "autor"
  "/cesHeader/fileDesc/(sourceDesc/biblFull/)*sourceDesc/biblStruct/analytic/h.author"
  "/cesHeader/fileDesc/(sourceDesc/biblFull/)*sourceDesc/biblStruct/monogr/h.author")
```

The first template defines the value of the '`styl`' (genre) attribute as being the content of the XML element defined by the path '`/cesHeader/.../keyTerm`'. It is an attribute of type '`multi`' because many such `keyTerm` elements may be present and the values of all of them should be collected into the value of the attribute '`styl`'. The second template is similar, but it defines a '`single`'-type attribute, i.e., if many XML elements defined by the given path are present, only the contents of the first one is copied to the value of the attribute '`medium`'. Finally, the third template illustrates a more complex case, where the information about the author may be given by a number of different paths. In fact, each of the two path specifications in this template corresponds to an infinite number of paths, with any number of the (recursive) '`sourceDesc/biblFull/`' subpath sequences.

Unfortunately, *bp* is currently only available for GNU/Linux, because of its heavy use of features that are specific to Linux and the GNU C library. This restriction will hopefully be lifted in future releases.

2.2. The corpus indexer

The *indexer* performs second (optional) stage of corpus building. Given a binary corpus constructed by *bp*, it generates sparse inverted indexes that are used to speed up execution of queries. For simple queries that yield few results (like searching for a word or a sequence of words), the speed boost achieved can amount to two orders of magnitude (from several minutes down to tenths of second). The details of the usage and the construction of indexes are discussed in section 4.2.

2.3. The user interfaces

Currently, there exist two user interfaces to the corpus presenting search results in the usual KWIC format. Both use the same corpus processing engine and support the same query language. One interface is a standalone cross-platform utility written in Java using the Swing toolkit. (Precompiled Windows and Linux versions, as well as the source codes, can be downloaded from http://korpus.pl/ or http://poliqarp.sourceforge.net/.)

The other interface consists of a collection of PHP scripts designed to make corpora available for searching over the WWW; it is less feature-rich than the standalone GUI and currently not included in the source distribution, but available at the Web site of the IPI PAN Corpus (cf. http://korpus.pl/).

The GUI supports incremental displaying of search results as they are found, thus it is no longer necessary to wait for the end of searching to browse the results (as was the case with the preliminary version of Poliqarp). Besides being more efficient and stable, it also allows to browse metadata defined for documents containing the results, allows to impose default restrictions on the queries and has an on-line help facility.

3. The client-server architecture

The concordancer is split into two programs: the server (written in portable C) that performs the actual concordancing, and the client whose job is to present the results of searching to the user. Each of the interfaces introduced in the previous section is in fact a client, and they both communicate with the same server using a simple text-based protocol over TCP.

An obvious benefit of this approach is that it clearly draws the line between the logical and interface modules of the system, but it is not the only advantage of this architecture. First, the server supports multiple connections, so it is possible to connect two instances of a client, or even two different clients, to one server and use the clients concurrently, without the unnecessary duplication of system resources. Second, the protocol supports the notion of a session, which can span multiple connections. It is therefore possible to connect to the server only for a short period of time, to issue a query and/or see whether any new results of an already issued query have arrived. This is crucial in the case of Web-based clients, where the HTML-generating scripts must run quickly, or else the user will be distracted.

Furthermore, the entire functionality of corpus processing is contained in a C library with a well-defined interface. It is thus possible to create another library conforming to the same interface and use it with the rest of the system. The new library does not even have to support the same query language, or the same format of corpora.

In the rest of this chapter, we briefly describe the protocol used by the concordancing clients to communicate with the server. The idea is to demonstrate that Poliqarp may be viewed as a collection of reusable components: even if some of them are not feasible in a given application, others might be. For instance, one might imagine a simple J2ME-based client for mobile devices, or an altogether different corpus library that works on raw text instead of corpora in a dedicated format, etc. The protocol plays an important part in the interoperation between the components, and we hope that it will be more widely supported.

The protocol has been designed with simplicity and flexibility in mind. The client opens a two-way reliable communication channel (Poliqarp uses local TCP for this purpose) and sends requests to the server. Each request consists of exactly one line of text, terminated by a newline character.

The server replies to each request instantly, regardless of how long it might take to process it. The length of each message from the server varies depending on what type of request it replies to, but it always can be deduced from the first line the server sends. In addition, the server can also send asynchronous messages that are not instant replies to a request, but are used to notify clients that, for instance, execution of a query has just finished, or a new bunch of results has been found, and so on.

The client commands can be split into two groups: session manipulation commands and corpus manipulation commands. The first group is used to create sessions, remove them and bind or unbind connections to them. Each command in the second group is executed in the context of a session. At any time, a client can disconnect from the server, which does not result in removing its session or cancelling requests that might be executed in its context. In particular, it can reconnect later and access the session data as if it had not disconnected at all.

The corpus manipulation part of the protocol does not make any implicit assumptions about the query language. It is also independent of the internal structure of corpora, assuming only that a corpus consists of a number of documents that: (i) are mutually disjoint; (ii) sum up to the entire corpus (i.e. every segment belongs to exactly one document); and (iii) can have textual or date-type metadata assigned to them.

As of Poliqarp 1.0, the protocol consists of 25 commands, three of which are meta-commands (used to check whether the server is working, to retrieve its version and to terminate the server), three belong to the session manipulation group, and the remaining 19 are used for processing corpora. Due to space constraints, we will not attempt to describe each of these in full detail here,[4] but we will illustrate it with a transcript of the sample dialogue between a client and a server ('-->' indicates messages sent by the client, '<--' marks messages sent by the server):

```
--> MAKE-SESSION user
<-- R OK 0
--> OPEN /usr/share/corpora/samplecorpus
<-- R OK
<-- M OPENED
--> MAKE-QUERY [pos=subst]{5}
<-- R OK
--> RUN-QUERY 100
<-- R OK
--> BUFFER-STATE
<-- R OK 1000 54
<-- M QUERY-DONE 100
--> SET wide-context-width 5
<-- R OK
--> GET-CONTEXT 0
<-- R OK
<-- R  Nie wiem, czy otrzymał
<-- R
<-- R  pan premier list szefów wydziałów
<-- R  w tej sprawie. Czujemy
--> CLOSE
```

[4] The M.Sc. thesis of the first author, currently in preparation (under the supervision of the second author) at the Institute of Informatics, Warsaw University, contains a detailed description.

```
<-- R OK
--> CLOSE-SESSION
<-- R OK
```

Some comments are in order. First of all, the first letter of each message from the server identifies the type of the message: `R` denotes a synchronous message (part of a reply to a request from the client), while `M` stands for an asynchronous message. Second, the `0` sent by the server in response to the `MAKE-QUERY` command is the session identifier which can later be used to reconnect. Third, after making a query, the client is expected to issue a command specifying how many results it wants to receive (cf. the '`RUN-QUERY 100`' above). Fourth, the client may at any point ask about the status of query execution, using the `BUFFER-STATE` command; the first number in the reply to the command denotes the overall capacity of the buffer, while the second indicates the number of results that are currently stored in the buffer (i.e., found so far). After finding the requested 100 results, the server issues an asynchronous message (cf. the '`QUERY-DONE 100`' above). Before requesting the results, the client may specify the widths of left and right contexts, measured in segments (cf. '`SET wide-context-width 5`'). Finally, the `GET-CONTEXT` command, followed by the number of the results (counting from 0), may be used to retrieve a query result (here, the five nouns in a row, '`pan premier list szefów wydziałów`') together with its left and right context, preceding and following the match respectively.[5]

4. Efficiency

4.1. The binary corpus format

The binary corpus consists of several backends, i.e., collections of files that contain information related to the same aspect of the corpus. The backends use two main on-disk data structures to store data, namely vectors (sequences of fixed-size data, e.g., numbers, stored in a single file) and dictionaries (sequences of variable-size data, e.g., strings, stored in two files: the first contains the actual data, while the second contains a vector of offsets to particular elements, which provides constant-time lookup). These structures are mostly accessed using the underlying operating system facility of file to memory mapping (e.g., `mmap()` in Unix).

The main corpus backend is a vector of eight-byte long structures, representing segments and consisting of the following fields:

- a flag specifying whether there is a space before the segment;
- an offset into the dictionary of orthographic words;
- an offset into the dictionary of sets of disambiguated interpretations[6], i.e., those marked as correct in the given context;
- an offset into the dictionary of sets of ambiguous interpretations, i.e., those assigned by the morphological analyser, regardless of whether they were marked as correct in later stages of tagging.

Each of these dictionaries is kept in its own backend; moreover, there are backends that hold structural information about the documents comprising the corpus (offsets of each structural element's borders), document metadata, tagset, predefined aliases, etc. In total, there are nine or ten backends, depending on whether the *indexer* utility has been invoked on the corpus.

The description of the search algorithm is well outside of the scope of this paper, but it is worth mentioning that the queries are compiled to finite-state automata in which edges are labeled with expressions corresponding to specifications of a single segment. In its simplest form, a compiled expression is a vector of bits, with the meaning of the *n*th bit set being that the *n*th word (or set of interpretations) matches this expression. Thus, matching the segment against a simple expression requires only one bit test, without the need of accessing other backends.

4.2. The use of indexes

For typical corpora, the number of segments in a corpus greatly exceeds the number of *distinct* elements in each of the sequences described above (i.e., the sequence of orthographic words, and both sequences of sets of interpretations). Therefore, it is useful to have a data structure that, for each of those distinct elements, assigns to

[5] Actually, the match itself may be split into two parts: left match and right match (see Przepiórkowski 2004 for details). If no such split is requested, the match is returned by the server as the right match, with the left match empty. This explains the line in the sample session above which only contains the single `R`.

[6] An interpretation is defined as a tag with the base form of a given word.

it a list of its occurrences in the corpus. Such structures are called *inverted indexes*, and are what the *indexer* utility produces.

Each of those three sequences has its own index. By default, *indexer* produces all three indexes, but it is possible to tell it to skip some of them, e.g., to produce only the index of orthographic forms. This is useful, for instance, in the case of a corpus devoid of morphological annotation (i.e., one that has a dummy tag assigned to each segment). In such a case, queries regarding morphological information will not be posed anyway, so the irrelevant indexes may be omitted.

The main problem with inverted indexes is that they can be very large when stored naïvely. Poliqarp overcomes this problem in two ways. First, not every occurrence of each word[7] is stored in the index. Instead, the corpus is artificially split into chunks (pseudo-documents) of equal size (by default, 1024 segments, but this is configurable). Then, for each item, only the numbers of the chunks in which it occurs are stored in the index, regardless of how many times it occurs in each chunk. This allows for compaction of those lists that contain a very large number of elements, while retaining most of the effectiveness gained through the use of indexes.

Second, a mechanism of index compression is employed. The compression schema used provides very fast decompression, while maintaining a reasonable compression ratio. The main idea is to store differences between elements of the lists with varying number of bits per element: thus, the lists corresponding to stop words, containing a very large number of elements, will be coded using few bits per element, while those corresponding to *hapax legomena* will contain many bits per element (but will not occupy much space anyway, thanks to their length).

This technique is known as Golomb encoding. The algorithms used to store and compress the indexes have been adapted (with minor changes to make them fit well with Poliqarp's method of storing corpora) from Witten et al. 1999.

Typically, the total size of indexes produced by the *indexer* utility is about 15% of the total binary corpus size, but it varies depending on the corpus structure and the indexes' granularity (i.e., size of the artificial chunks). For instance, in the case of the preliminary version of the IPI PAN corpus (over 70 million segments, 654 megabytes unindexed), the total size of indexes was 95 megabytes when indexed with granularity of 1024 segments, but only 67 megabytes when indexed with granularity of 4096 segments.

5. Concluding remarks

A question may arise as to why we decided to implement our own search engine and concordancer instead of either using the standard relational databases (the viability of such an approach has been demonstrated within the PELCRA Corpus of Polish project, http://korpus.ia.uni.lodz.pl/) or existing corpus management tools, such as the IMS Corpus Workbench (http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/).

One reason for this decision is already alluded to in Przepiórkowski et al. 2004. Even though Poliqarp was designed as a universal corpus search tool, it also had to meet the specific requirements of the IPI PAN Corpus of Polish, esp., its handling of ambiguities and the fact that it contains much richer morphosyntactic information than other corpora that we were familiar with. In particular, we needed a tool that would make it possible to represent and query two layers of morphosyntactic information: all (possibly many-way ambiguous) information provided by the morphological analyser, or only the morphosyntactic information judged contextually correct by the tagger (but possibly still ambiguous, as some ambiguities cannot be resolved even on the basis of infinite context). To the best of our knowledge no tool offering such expressive power was available at the time, and an attempt at confining this complex hierarchical information into the straight-jacket of relational databases struck us as unmaintainable.

Even if such an attempt had succeeded, it would be still very hard to map Poliqarp's query language to the equivalent SQL queries. The high-level regular expression feature of Poliqarp's query language would make the corresponding SQL queries unnecessarily complex and hard for the optimizers to cope with. Apart from that, the standalone installation of a corpus on a user's workstation for local usage would be painful and very disk-consuming — Poliqarp makes it easy (to the best of our knowledge, this is a unique feature among programs that deal with corpora of this size).

Another reason for this approach was our need for a corpus search tool that would grow with the annotation layers of the IPI PAN Corpus. At the time no such open source tool that would make IPI PAN Corpus-specific modifications possible was available.[8] This need becomes important now, as the IPI PAN Corpus enters the syntactic annotation phase.

[7] Or, more generally speaking, each item.

[8] In the meantime, the tool used by BNC, Xaira, has been released as open source (cf. http://xaira.sourceforge.net/), and we have been informed that there are also plans to make the sources of the IMS Corpus Workbench available.

REFERENCES

Ide, N., Bonhomme, P. and L. Romary. (2000). "XCES: An XML-based standard for linguistic corpora." In: *Proceedings of the Second International Conference on Language Resources and Evaluation, LREC 2000*. Athens: ELRA. 825-830.

Janus, D. (2006). *Metody przeszukiwania i obrazowania jego wyników w dużych korpusach tekstów.* M.Sc. thesis (in preparation). Warsaw: Faculty of Mathematics, Informatics and Mechanics, Warsaw University.

Przepiórkowski, A. (2004). *The IPI PAN Corpus: Preliminary version*. Warsaw: Instytut Podstaw Informatyki PAN.

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

Witten, I. H., Moffat, A. and T. C. Bell. (1999). *Managing gigabytes: compressing and indexing documents and images*. San Francisco: Morgan Kaufmann Publishers Inc.

Keywords: **concordancer**, **corpora, search engine**, **client-server architecture**, **efficiency**, **XCES**, **XML**