SELF-INDEX COMPRESSION IN ELECTRONIC PUBLISHING

Eduardo R. López, Database Laboratory, University of A Coruña, Spain, eduardo.rodriguez.lopez@udc.es

Ángeles S. Places, Database Laboratory, University of A Coruña, Spain, asplaces@udc.gal

Juan R. López, Database Laboratory, University of A Coruña, Spain, juan.ramon.lopez@udc.es

José R. Paramá, Database Laboratory, University of A Coruña, Spain, jose.parama@udc.es

Antonio Fariña, Database Laboratory, University of A Coruña, Spain, antonio.farina@udc.es

Abstract

Compression has been used typically to save disk space and bandwidth. However, from a decade ago, a new family of data structures allows applying compression to save space in main memory as well. The so-called self-indexes are an example of these new structures. They can store a text in space proportional to that of the compressed version of the same text, and at the same time, they allow searching for a pattern in sublinear (sometimes logarithmic) time. In addition, they allow random access, that is, they can extract a portion of the text without the need of decompressing it from the beginning.

In this work, we present a real project that uses a self-index - in order to explore and exploit its potential advantages - as the structure to store and manage text inside electronic books with digital rights management.

Keywords: Electronic publishing, compression, digital rights management.

1 INTRODUCTION

The Internet and all recent advances in hardware technology have boosted the world of electronic publishing. The process started around the year 2000 when the digital libraries (Lesk, 1997) opened a new way of publishing. This technology advance was based on the World Wide Web, and more specifically in web pages, which were displayed on regular monitors. Obviously, this is not comfortable enough for reading, and soon the e-readers appeared (Golovchinsky, 2008). These handheld devices are easy to carry and use in any situation, that is, they are really useful for the purpose of reading as a leisure activity. This technology change was coined by some as the greatest advance for the literature since the Gutenberg press (Rao, 2003). Later, smartphones and tablets represented one step further in this path, opening a new way of distributing all types of multimedia content.

Nevertheless, the general use of these new technologies posed new issues. Despite the improvements in the bandwidth of the networks, users now use their devices extensively, and then they have to face increased costs and limits of data transmission. In addition, with new functionalities, the devices should have more computational power and memory capacity. To solve in part these issues, compression is usually used in several ways; the most obvious is to save disk space and bandwidth. An example could be the ePub eBook format, which uses Deflate compression (Deutsch, 1996) to reduce the size of the files. Yet compression is not limited to an archival method, as many data transmissions carry compressed data. An example of this is Google's Chrome Web browser (Google Inc.), which can transmit compressed data between a server and a client without the user noticing anything. That is, data were not compressed before transmission, and compression is applied on the fly to save bandwidth.

By managing compressed data in main memory, we obtain additional benefits. The most obvious profit is that larger datasets can be loaded completely into main memory in order to process them. However, most compression methods require a previous decompression of the data from the beginning in order to access them. Even if we need only a portion of the data, the whole dataset must be decompressed, consuming both useless memory space, disk I/O, and processing time. In this way, compression is only useful to save space and/or transmission bandwidth.

However, a new family of data structures, called compact data structures (Jacobson, 1989), aim at storing the data and the structures to access such data in a unique compact data structure which allows accessing the data without decompressing it from the beginning. Thus, we can split the dataset into portions, and process each portion separately. Also, if needed, we can maintain more portions in main memory, possibly the whole dataset, and then we avoid costly disk accesses that slow down the processing. Even if the data fit in main memory, the compact data structures can take advantage of the lower latency and higher bandwidth of the upper levels of the memory hierarchy, resulting in many cases in an improvement in the running times. Therefore, compact data structures allow fitting, efficiently querying, and manipulating much larger datasets in main memory than in the case of representing the data in plain form or in a traditional compressed format.

Regarding text, there exists a family of compact data structures called *self-indexes*. In a single structure, they do not only compress text, but they index it. From a plain text, a building process obtains a data structure (the self-index) that requires space proportional to the size of the compressed text and permits fast indexed searches on it, without any additional structure (Navarro & Mäkinen, 2007). They are able to *extract* any portion of the text and *locate* the occurrence positions of a pattern string in a time that depends on the pattern length and the output size (number of occurrences), but not on the text size (that is, the search process is not sequential).

Note that the original text is no longer needed as its in-memory self-indexed representation is enough to support both the search and rendering processes. Also, by directly storing such self-index in disk a compressed representation of the text is retained.

Despite their advantages, and to the best of our knowledge, these data structures are not currently being used by the industry in the field of text management. The reasons can be found in several restrictions related to self-indexes. First of all, it is not possible to modify or add new text directly over a self-index. To perform any modification, we must decompress the self-index, change the original text, and finally compress/index it again. Moreover, if we pretend to perform searches over a collection of documents, the whole collection should be joined to make up a unique self-index, loaded completely into main memory to allow those searches. The above issues prevented the use of self-indexes and, as a consequence, there are no industrial level tools to exploit them, something which represents an obstacle to the application of these techniques in non-research projects.

Nevertheless, we had developed in the past a self-index implementation specifically designed for plain text; and, on the other hand, we had also developed innovative industrial level software for the publication and distribution of electronic content. Having total control over both lines of work, we decided to apply our previous research in self-indexes to an industrial project of electronic publishing, in order to develop a concept model of the possibilities of this technology. The target was to demonstrate the feasibility and the advantages of using this technology in the world of electronic publishing. In this paper, we present *elibro-galego*, a real solution for the distribution of digital content that exploits the advantages of self-indexes and analyzes the impact of their limitations, offering different alternatives to mitigate their effects.

The main contribution of this paper is the design of an architecture/solution for the distribution and consumption of multi-format digital contents through the Internet, which make use of state-of-the-art algorithms and compact structures from the compression and indexing research fields. More precisely, we apply self-indexes for the compression/indexing of textual nature content. We discuss the advantages obtained in different tasks: storage and full-text searches at the server, delivery/transfer, storage, and full-text searches and rendering at local devices.

The outline of the paper is as follows. Section 2 describes compact data structures. Section 3 presents the field of electronic publishing. Section 4 introduces the platform elibro-galego and presents our solution for electronic publishing. Finally, Section 5 presents our conclusions and directions for future work.

2 COMPACT DATA STRUCTURES

Although digital content nowadays involves multiple formats (text, image, video, etc.), text remains as a crucial component, and it is the focus of this work.

The outbreak of the web has promoted the development of data structures that have several appealing characteristics to store and/or index text. The research community has worked extensively on inverted indexes (Knuth, 1973) since they prove to be a simple and effective tool to index text (Baeza-Yates & Ribeiro-Neto, 2011; Witten, Moffat, & Bell, 1999), being used in all search engines. Even so, inverted indexes have several problems. First, the consumption of space, since they spend between a 40% and 80% (if represented in plain form) of additional space on top of the original information. Second, the search for phrases, which requires an important amount of computation resources (Demaine, López-Ortiz, & Munro, 2000). Third, problems to index some languages like Chinese or Japanese, and even languages where long words are formed from particles such as Finnish, German, and many others (Navarro & Mäkinen, 2007). The space problem is relevant because it makes difficult to fit large texts in main memory and harms the I/O. Thus, the reduction of space of inverted indexes is an important research line (Ottaviano & Venturini, 2014; Zobel & Moffat, 2006). Nevertheless, the inverted index is the common data structure in large systems that require text indexation.

Suffix arrays (SAs) (Manber & Myers, 1993) were designed to solve the problems of the inverted indexes with phrases and complex languages. However, the problem with space is even worse since suffix arrays occupy around 4 times the space of the original text collection. Again, the problem of the space consumption of this structure boosted the research in succinct versions (Navarro & Mäkinen, 2007). At the beginning of the 2000 decade, suffix arrays evolved into *self-indexes* (Ferragina & Manzini, 2000), which merge compression with indexing. This data structure requires space proportional to that of the compressed text, and permits fast indexed searches on it without any additional structure (Navarro & Mäkinen, 2007). Moreover, it allows extracting any text substring without decompressing from the beginning. Among the most well-known self-indexes, we can find the *Compressed Suffix Array (CSA)* (Sadakane, 2003), which can occupy only around 50-60% of the size of the original text (Fariña, Navarro, & Paramá, 2012). This structure uses characters as the compression unit.

Later, following a tendency in text compression that uses words as units for compressing (Moffat, 1989), a word-based version of the CSA, called *Word-based CSA (WCSA)* was developed (Fariña et al., 2012). The advantage is an improved compression in large texts with respect to the original self-index. The word-based self-index can occupy only 30-40% of the space occupied by the original text. The price is that those versions are no longer capable of searching for arbitrary strings, and now all searches should use words as units.

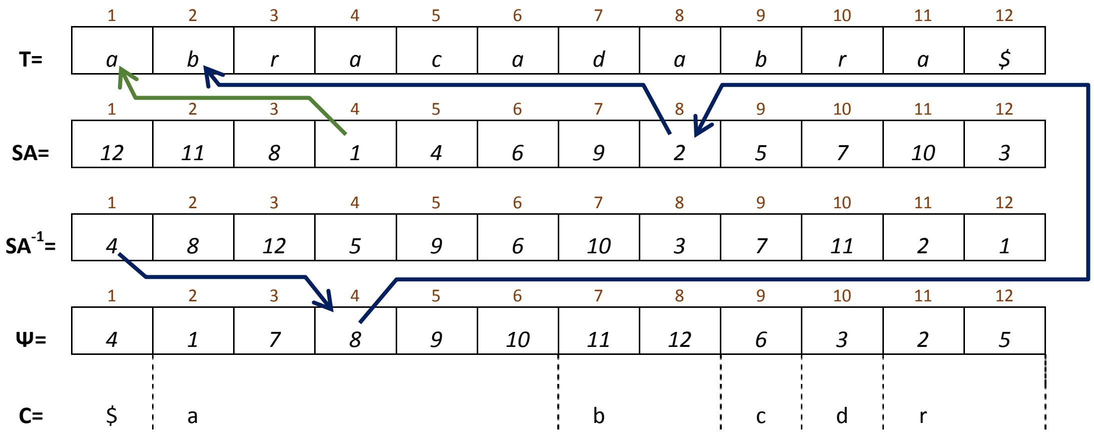


Figure 1. *Structures related to a CSA self-index.*

To briefly illustrate the operation of a self-index, Figure 1 shows an example of a CSA-indexed text. Some of the structures shown are only needed for building the index and are finally discarded. The original plain text (plus an extra terminator character *$*) is represented by the *T* array. Each position in the text is considered as a text *suffix*, that is, a string that starts at that position and goes up to the end of the text. The *SA* array keeps all those suffixes, in lexicographical order, represented by their start position at *T*. For instance, *SA[4]=1*, the position where the suffix *'abracadabra$'* starts at *T*; and *SA[8]=2*, where the suffix *'bracadabra$'* starts. *Ψ* is a very compressible array intended to replace *SA*, by using the property *SA[Ψ(i)]=SA[i]+1*. If *'abracadabra$'* is the suffix starting at position *1* in *T* (and *4* in *SA)*, *Ψ(4)* must point to the position (*8*) at SA of its 'descendant', *'bracadabra$'* (which is precisely the suffix starting at position *2* in *T*). Finally, the *C* array splits *SA* into several slots, each one containing all suffixes starting with the same character. *C* marks the limits of each SA slot and the (shared) initial character of the suffixes that it includes.

Both *Ψ* and *C* can be used to reconstruct *T*. For instance, we know (using *C*) that the suffix in *SA[4]* starts with an '*a*'. The second suffix character is also the start character of suffix *SA[Ψ(4)];* using *C* again, we know that it is a *'b'*. If we recursively continue, we will obtain *SA[4]='abracadabra$'* (the whole original text). Moreover, we can improve the index adding an SA^{-1} array. $SA^{-1}[i]$ must give the position in *SA* of the suffix starting at position *i* in the text (for instance, $SA^{-1}[1] = 4$). Thus, using *Ψ*, *C*, and SA^{-1}, we can decompress the text from any random position. Also, text searches can be performed easily. For instance, binary search over the (recursively reconstructed) *SA* array will allow fast word findings.

3 ELECTRONIC PUBLISHING

The first way to distribute literature in electronic format was through digital libraries (Lesk, 1997). However, although in many cases the target was to make literature available to the general public, broadly digital libraries were only successful in the academic world (Rydberg-Cox, 2005). Probably, the main reason was that they were based on the use of web pages, which were displayed on regular monitors, and this is not comfortable for reading as a leisure activity. As explained, the raising of e-readers (Golovchinsky, 2008), which can be easily used in most circumstances, opened a new way of electronic publishing that drew the attention of publishers and led them to introduce the digital distribution of their catalogs.

In general, there are several ways of building eBooks:

- Traditional multimedia formats, like HTML and PDF.
- Proprietary formats for specific e-readers, like the Mobipocket or KF8 formats of Amazon Kindle.
- Open formats specially designed for eBooks, like ePub.

The formats specially designed for eBooks are capable of managing reflowable content, which means that an e-reader can optimize text visualization for a particular display device. One of the most common formats is ePub. It was created by the International Digital Publishing Forum[1] and it is free and open. An ePub file encapsulates several files that make up the eBook within a ZIP container.

Most devices and applications usually offer a series of common functionalities for text rendering. Among them, we can mention some such pagination, bookmarking, resuming, or text search. All of them may require the ability to access and render some random piece of the text.

Optionally, the eBook can be provided with Digital Rights Management (DRM) protection. The basic aim of DRM is to protect the rights of the content creator. Many definitions include other services, for example, Subramanya and Yi (2006) define a DRM like "*a system that allows the creator (or/and producer) to specify the ownership rights of the content and grant the access according to those rights. In the part of the consumer, a DRM provides mechanisms to specify the desired content, manage it, and in many cases make use of it*".

Nowadays there are many platforms for distributing electronic content. The most well-known are probably Amazon and Apple's iTunes. Most of these platforms not only allow downloading eBooks but also offer additional features such as, for example, keeping a local library in the user's device storage.

Digital libraries are another method to distribute eBooks. For example, Project Gutenberg[2] provides ePub eBooks free of charge. In this case, they include only works that do not have rights to protect, and therefore without any DRM infrastructure. The eBooks are typically managed locally as plain files which are displayed by simple applications capable of reading them. Classic libraries now offer electronic lending of eBooks, yet other platforms now provide the possibility of lending electronic content, after paying a fee.

Regardless of the platform used, it is not unusual the need to perform searches by content over the whole set of works. Not all users who use a library come with a concrete idea of the book they are looking for. In an eBook library, the usual search is by metadata (normally title and/or author search), and search by content is a rare case (but not non-existent).

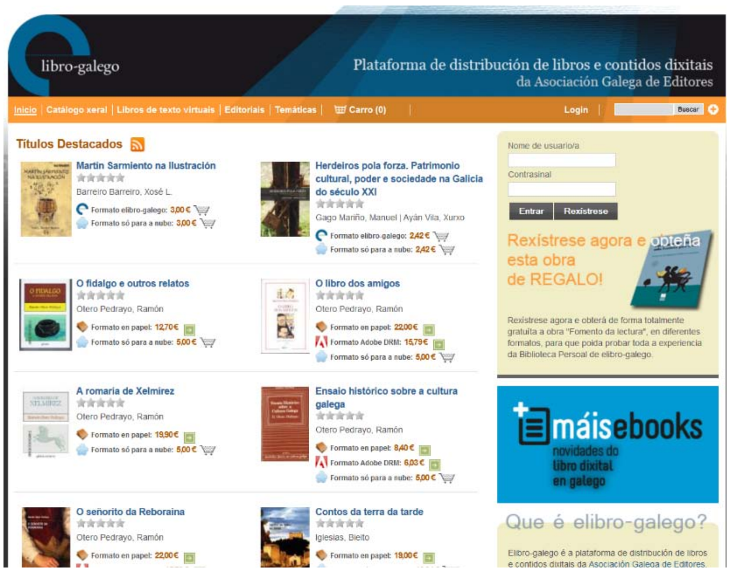
[1] http://idpf.org/epub/30
[2] http://www.gutenberg.org/

4 OUR SOLUTION

4.1 A Case Study: elibro-galego

Galician is a language of a Northwestern region of Spain. It is considered a minority language according to the European Charter for Regional or Minority Languages. In fact, the Galician language is in danger of disappearing, despite the efforts in the last decades of several institutions and organizations. Many factors may apply to this; however one of them is the lack of presence of the language in the on-line world.

The Galician publishing industry consists of many small publishers, with little economic power, and libraries are mostly public institutions that, as usual, have strong budget constraints. In both cases, there exists an interest in offering eBooks through the Internet. To promote the Galician language and to boost the publishing industry, the regional government of Galicia, with the support of other public institutions, awarded a grant to the Galician association of publishers to help them to publish their works in a digital format and offer them through the Internet.


libro-galego

Plataforma de distribución de libros e contidos dixitais
da Asociación Galega de Editores

Inicio | Catálogo xeral | Libros de texto virtuais | Editoriais | Temáticas | Carro (0) | Login | Buscar

Títulos Destacados

Martin Sarmiento na Ilustración
Barreiro Barreiro, Xosé L.
Formato elibro-galego: 3,00 €
Formato só para a nube: 3,00 €

Herdeiros pola forza. Patrimonio cultural, poder e sociedade na Galicia do século XXI
Gago Mariño, Manuel | Ayán Vila, Xurxo
Formato elibro-galego: 2,42 €
Formato só para a nube: 2,42 €

O fidalgo e outros relatos
Otero Pedrayo, Ramón
Formato en papel: 12,70€
Formato só para a nube: 5,00 €

O libro dos amigos
Otero Pedrayo, Ramón
Formato en papel: 22,00€
Formato Adobe DRM: 15,79€
Formato só para a nube: 5,00 €

A romaria de Xelmirez
Otero Pedrayo, Ramón
Formato en papel: 19,90€
Formato só para a nube: 5,00 €

Ensaio histórico sobre a cultura galega
Otero Pedrayo, Ramón
Formato en papel: 8,40 €
Formato Adobe DRM: 6,03 €
Formato só para a nube: 5,00 €

O señorito da Reboraina
Otero Pedrayo, Ramón
Formato en papel: 22,00€

Contos da terra da tarde
Iglesias, Bieito
Formato en papel: 19,00€

Nome de usuario/a
Contrasinal
Entrar Rexistrese

Rexistrese agora e obteña esta obra de REGALO!

Rexistrese agora e obterá de forma totalmente gratuíta a obra "Fomento da lectura", en diferentes formatos, para que poida probar toda a experiencia da Biblioteca Persoal de elibro-galego.

máisebooks
novidades do libro dixital en galego

Que é elibro-galego?

Elibro-galego é a plataforma de distribución de libros e contidos dixitais da Asociación Galega de Editores.

Figure 2. Main page of elibro-galego.com

The publishers could not use that money to contract a system like ADEPT of Adobe since they could not afford the long term costs of that type of systems, which charge a fee to the publisher per each sale of an eBook. So, they contacted with a little software company, a spin-off of a university research lab. This company is devoted to conducting technology transfer between the research lab and the industry. The result is currently available at http://www.elibro-galego.com/ (see Figure 2).

The platform elibro-galego was designed to provide its own format. It is a DRM protected format that works on desktop computers and thus requires downloading a rendering application. This type of protection is available for the formats: PDF, ePub, and HTML.

4.2 Self-indexes in Electronic Publishing

As discussed before, some basic functionality is needed and implemented in most formats, platforms and devices related to electronic publishing:

- Text compression, to save space and bandwidth.
- Text search, both over a single eBook or over the whole library.
- Independent access to specific pieces of text.

As explained, self-indexes have several nice features that can be of interest when comparing with the classic setup. The main advantage is space saving since self-indexes occupy less than the original collection. They also ease searches, as the same structures integrate both text and indexes; and a specific piece of text can be recovered (with no need to decompress the whole text).

In contrast, we have also remarked that self-indexes have several characteristics that prevent their use in many scenarios:

- Given a self-index, it is not possible to modify its text or add new text without decompressing it.
- To search or extract the text in a self-index, it should be loaded completely into memory. This poses a problem; if we want to have the possibility of performing searches by content over the whole collection, we have to compress the whole collection in one unique self-index, which must fit in main memory to perform searches over it.

Therefore, self-indexes are not always the best choice in practice. In particular, the inverted index is still the most common tool to index text. Inverted indexes are used so widely that there are available software packages for the general public, like Lucene (Gospodnetic & Hatcher, 2004).

However, observe that, in the world of the electronic publishing, the above restrictions on self-indexes are not a problem, since each work can be compressed/indexed individually and it never changes, and searches by content over the whole library are very rare. On the contrary, we can fully exploit the advantages of self-indexes.

Due to their ability to compress and the current memory capacity of most devices, virtually any book compressed with a self-index could fit into memory without any problem. It is clear that any kind of compression will produce benefits, and so do a self-index:

- It reduces the size of the eBooks to save data transmission, which is an important factor, especially in handheld devices, since users have connections that can have network data limits.
- It reduces the size of the collection stored at the client (if any), an important factor in cheap handheld devices.
- It saves memory: we can fully keep the eBook, compressed, in main memory.

But the self-index approach provides many more advantages. Since it allows random access (we can extract any portion of the text without decompressing from the beginning), it also provides all these additional benefits:

- It speeds up the use of the content: by keeping all the content in main memory, when a user requests a page of an eBook, it is obtained from main memory, avoiding I/O costs. The required on-line decompression costs are much lower than that of loading the content from disk. A typical CSA self-index requires around 0.5 µsec. to extract a character (Fariña et al., 2012). A page can be decompressed separately from the rest of the text. There is no need to decompress the whole book in order to resume a reading, request a bookmark, or simply turn a page.
- It provides sublinear search time over the whole eBook without the need of an inverted index or any other disk access. As stated before, this is just the novelty offered by self-indexes.
- It makes up an obfuscation of the content. This is useful in DRM systems, as it provides an additional layer to protect the digital content from unauthorized accesses. Self-indexes use complex structures, which are kept as such in main memory. Only when a rendering application requests a part of the text, the *extract* operation obtains the requested portion. Such extraction is a complex procedure (although not very costly in computational time) that depends on the chosen self-index.

All these advantages are important, mostly dealing with cheap devices. Nowadays, software developers are not very worried about the resources wasted by their software since most of the computers and many handheld devices are powerful enough. However, in the world of smartphones and other handheld devices, there is an important amount of cheap devices with limited memory and computation power.

In Figure 3, we show our approach in a system where the eBooks are read on-line, and no copy of the eBook is stored locally at the client. Figure 4 shows the set up in a system where the eBooks are stored locally at the client.

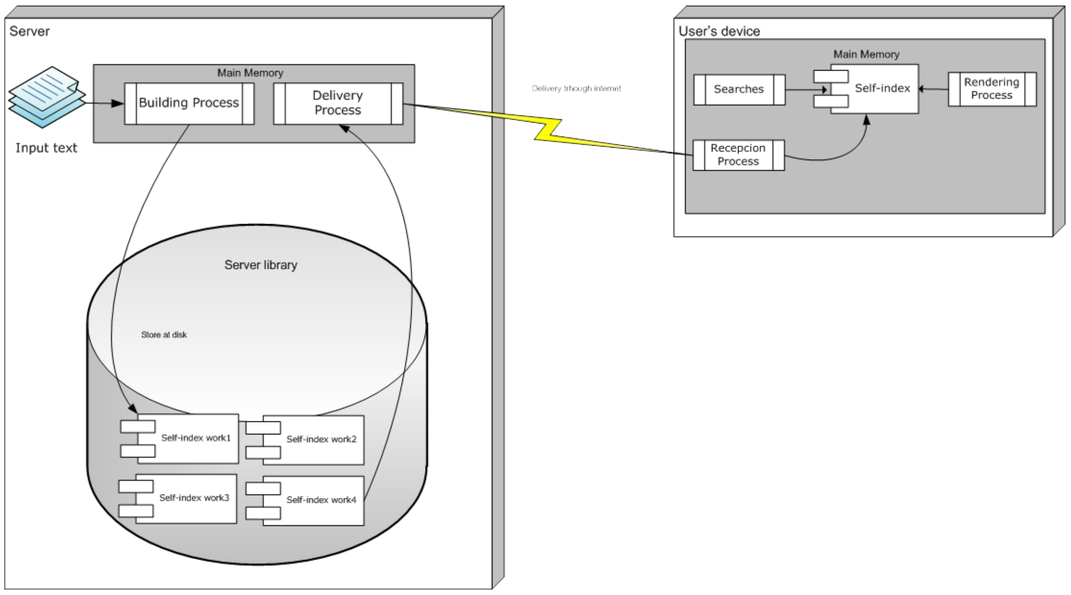


Figure 3. Our approach in system without local storage.

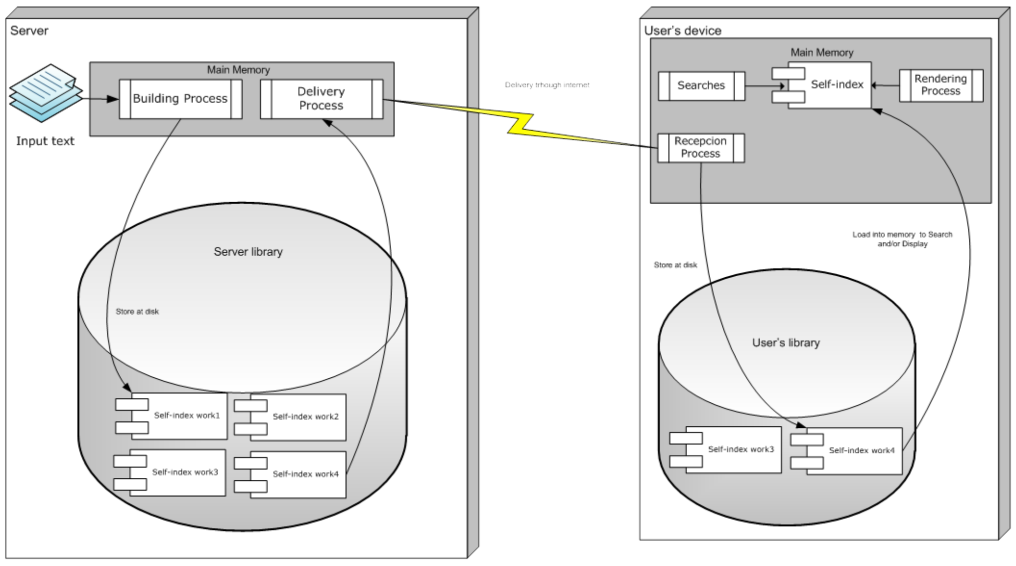


Figure 4. Our approach in a system with local storage.

4.3 Providing Search by Content

Search by content draws much attention due to its key role in the Web, more specifically in the search engines. However, in electronic publishing, it is not as important, since most eBook searches are through metadata. Nevertheless, it is still possible to provide such functionality. We are going to discuss the possible solutions in the case of the server and in the case of the client. These solutions have not been yet implemented in the elibro-galego platform, and thus must be considered as proposals for future extensions.

At the server, the easy solution is to build an inverted index. We have two options:

- To use an inverted index pointing to documents. Figure 5 shows that approach. Each block on the right is an eBook in the form of a self-index. We show some words of each eBook just for illustration purposes. On the left, we show an inverted index pointing to the books. An inverted index is formed by a list of keys, which are the indexed values (words, in our case). For each word there is a posting list, which is the list of eBooks where that key is present. For example, in Figure 5, the key *Cervantes* is present in the eBooks 1, 2, and 5.

  If *document retrieval* (Baeza-Yates & Ribeiro-Neto, 2011) is needed (that is, the search result is a set of documents) this architecture is enough. However, if *full-text retrieval* (Baeza-Yates & Ribeiro-Neto, 2011) is needed (that is, we need the exact positions of the search keys inside the documents), we can load the documents returned by the search on the inverted index into main memory, and perform a search over them taking advantage of the search capabilities of self-indexes.
- An alternative would be to use an inverted index pointing to exact positions. The inverted index can point to exact positions within self-indexes, since we can *extract* a portion of the text in a self-index without starting the decompression from the beginning. With this option, the consumption of space of the inverted index will grow considerably, typically about 3 to 5 times larger than non-positional ones (He & Suel, 2012). It would require huge time savings to justify that growth. Unfortunately, since the search times to *locate* occurrences of a pattern in a self-index depend on the pattern length and the output size (number of occurrences), time saving would be really low (Fariña et al., 2012; Fariña et al., 2012). For example, when searching for patterns of only one word, CSA will spend between 0.06 and 0.005 msec. per occurrence.

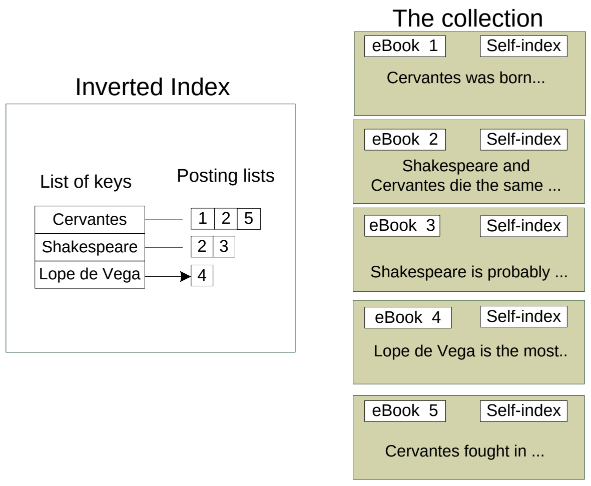


Figure 5. A possible setup of the server with an inverted index.

In the case of the client with a local library of works, we have two options:

- To use the basic architecture of Figure 4, that is, without an inverted index. Then a search must be solved sequentially, searching work by work. Therefore, each work should be loaded into memory and then searched in sublinear time in main memory.
- To use an inverted index at the client, with the same structure explained for the server.

4.4 The elibro-galego Format

The elibro-galego format can manage digital contents in any format (format-agnostic DRM), but applies the self-index technique only over textual data. In general, we will distinguish two types of formats: *indivisible formats* and *packaged formats*. By *indivisible formats*, we will refer to those formats that can only be distributed as a whole, since it is not possible (or it is complicated) to extract components (an example would be the PDF format). By *packed format*, we will refer to those publication formats that can be decomposed into components that can be distributed separately upon demand (for example the ePub format).

Figure 6 shows the structure of an eBook with our format when protecting *packaged formats*. The raw material is split into two components: the plain text and the rest of multimedia objects (images, sound, etc.). The plain text is compressed/indexed with a self-index, and the multimedia content is compressed with Deflate if required (since some formats like JPEG or MP3 are already compressed). Finally, the resulting package is encrypted by an Advanced Encryption Standard (AES) algorithm of 128 bits (National Institute of Standards and Technology, 2001), by using the public key of the user who purchased the content.

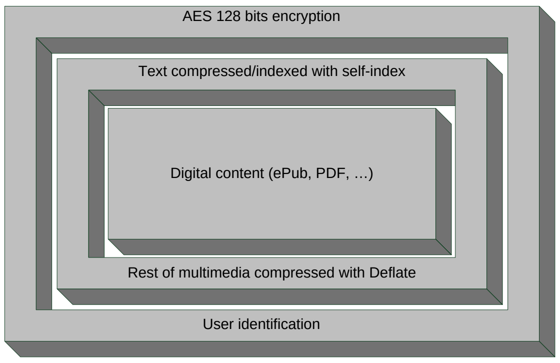

AES 128 bits encryption
Text compressed/indexed with self-index
Digital content (ePub, PDF, ...)
Rest of multimedia compressed with Deflate
User identification

Figure 6. The process of packaging electronic content.

When the content package reaches a client, the private key of the user is used to decrypt it. Since the content was encrypted with the public key of the user who paid the fee, then the decryption will obtain the raw material (text, images, etc.).

At this point, compression is profitable to save bandwidth. But its usefulness does not stop here: all components are kept compressed all the time, even in main memory, providing all the benefits listed in section 4.2.

This real project represents a concept model to check the feasibility of our approach. We have used an implementation of a self-index over which we have all the rights, since the target of the project is a commercial tool. However, we can plug any of the self-indexes available from the Pizza&Chili site, which includes several self-indexes following a common API[3].

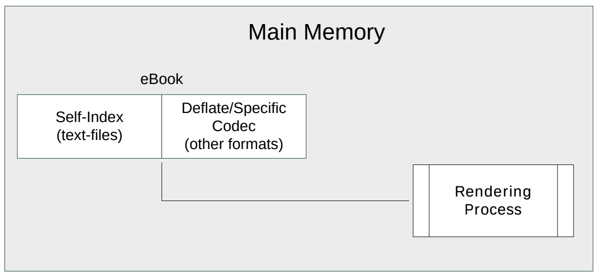


Figure 7. Structure in main memory.

4.5 Performance on Small Documents

We can find several works comparing the extraction speed, compression capabilities, and search performance of several self-indexes (Fariña et al., 2012; Fariña et al., 2012). Yet, those studies deal

[3] http://pizzachili.dcc.uchile.cl/api.html

with large texts, resulting from merging several documents, since they were developed mainly for these type of texts.

To give an idea of the compression power when dealing with individual eBooks, we downloaded several eBooks in plain text from the Gutenberg Project site to compress them with two different self-indexes: a (character-based) CSA implementation available on the Pizza&Chili site, and our own (word-based) WCSA implementation, which is the one used in elibro-galego by default.

We tested them using several works in English of different sizes, namely: the complete works of Shakespeare (5.33 Mbytes), Cervantes's Don Quixote Vol.1 (1.14 Mbytes) and Vol.2 (1.11 Mbytes), Bram Stoker's Dracula (862 Kbytes), Marc Twain's The Adventures of Tom Sawyer (412 Kbytes), Shakespeare's Othello (168 Kbytes), and Shakespeare's Macbeth (117 Kbytes). We also added two books in German as representatives of languages having aggregated words (which are more costly to index with inverted indexes): namely, Shakespeare's Othello (216 Kbytes) and Shakespeare's Macbeth (141 Kbytes).

	Size (bytes)	***CSA***	***WCSA***	***Deflate***	***Huffman***
Shakespeare's works	5,589,889	3.897	3.472	2.914	4.686
Don Quixote Vol. 1	1,200,139	4.104	4.142	3.014	4.510
Don Quixote Vol. 2	1,168,472	4.087	4.130	3.028	4.562
Dracula	883,156	4.222	4.421	3.050	4.559
Tom Sawyer	421,884	4.506	5.304	3.136	4.658
Othello German	221,772	4.604	6.050	3.053	4.817
Othello English	172,033	4.910	6.347	3.171	4.719
Macbeth German	144,566	4.997	6.946	3.177	4.862
Macbeth English	119,988	5.233	7.126	3.274	4.770

Table 1. Compression ratio (in bits per char).

Table 1 shows the compression ratio (in bits per char) using a typical setup for those self-indexes. We also included in the comparison two well-known compressors. Deflate is the format used by the famous ZIP containers. Huffman is the optimal statistical prefix-free compressor (Huffman, 1952).

We can see that WCSA works better over large texts while the character oriented self-indexes have a better behaviour for small eBooks. We can also see that the results do not largely vary when compressing/indexing German, proving the nice behaviour of self-indexes with languages that have complex words. With respect to "normal" compressors, both self-indexes outperform Huffman in large texts. Yet, dealing with the smaller texts, CSA is not far from Huffman. However, Deflate is always the winner. This is not surprising since the only target of Deflate is to obtain compression. Observe that both Deflate and Huffman (like all classical compressors) require to start decompression from the beginning, and thus they are not suitable to keep the compressed data in main memory.

Although with small files, self-indexes do not reach the compression level obtained for large texts (around 30-40% in the case of WCSA), reducing the space to around 50-60% is not a negligible achievement. In addition, one might think that saving space in small works does not actually worth it, but it is a question in the long term. For example, in commercial Database Management Systems is possible to compress the content of tables to save space and to increase I/O performance (Garmany, Karam, Hartmann, Jain, & Carr, 2008), although the size of an individual row is typically very small. Also, most web browsers, like Google Chrome, can compress the data exchanged between a server and the web browser in certain cases, although each individual request can be small.

5 CONCLUSIONS

In this paper, we have presented an architecture for electronic publishing through the Internet that was actually implemented on a new completely functional platform for the distribution of digital content. It includes a new important technological solution, specifically, we used a new family of data structures called self-indexes. They replace the original text, occupy space proportional to the compressed text, provide sublinear search time, and can extract any portion of the text upon request. The target was to save bandwidth in the distribution process, to save disk space both at the server and at the client, and to save I/O bandwidth and main memory at the client.

To the best of our knowledge, this is the first real project that employs self-indexes. We used it to prove the feasibility and the robustness of these data structures in a real project. This is a concept project where we show that the problems that prevent their use in search engines or other systems do not apply to this scenario. Specifically, since we use the book as the unit of compression/indexing, we do not have the problem of modifying and/or adding text.

As future work, we wish to add an inverted index pointing to documents, in order to be able to solve searches over the whole library in a faster way.

6 ACKNOWLEDGEMENTS

This work was funded in part by Xunta de Galicia (co-funded by FEDER) [GRC2013/053]; MINECO (PGE and FEDER) [TIN2013-47090-C3-3-P, TIN2013-46238-C4-3-R]; and CDTI-MINECO [IDI-20141259, ITC-20151305, ITC-20151247]

References

Baeza-Yates, R., & Ribeiro-Neto, B. (2011). *Modern information retrieval*. Essex England: Addison-Wesley.

Demaine, E. D., López-Ortiz, A., & Munro, J. I. (2000). Adaptive set intersections, unions, and differences. *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* pp. 743-752.

Deutsch, L. P. (1996). *DEFLATE compressed data format specification version 1.3* No. RFC 1951)

Fariña, A., Brisaboa, N. R., Navarro, G., Claude, F., Places, A. S., & Rodríguez, E. (2012). Word-based self-indexes for natural language text. *ACM Transactions on Information Systems,* , 1-34.

Fariña, A., Navarro, G., & Paramá, J. R. (2012). Boosting text compression with word-based statistical encoding. *The Computer Journal, 55*(1), 111-131.

Ferragina, P., & Manzini, G. (2000). Opportunistic data structures with applications. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science,* pp. 390-398.

Garmany, J., Karam, S., Hartmann, L., Jain, V. J., & Carr, B. (2008). *Oracle 11g new features get started fast with Oracle11g enhancements* Shroff Publishers/Rampant.

Golovchinsky, G. (2008). Reading in the office. *Proceedings of the 2008 ACM Workshop on Research Advances in Large Digital Book Repositories,* Napa Valley, California, USA. pp. 21-24.

Google Inc. *Chrome data compression proxy for network administrators, carriers, and ISPs.* Retrieved 13 March, 2015, from https://support.google.com/chrome/answer/3517349?hl=en

Gospodnetic, O., & Hatcher, E. (2004). *Lucene in action*. Greenwich, CT, USA: Manning Publishers.

He, J., & Suel, T. (2012). Optimizing positional index structures for versioned document collections. *The 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'12,* pp. 245-254.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proc.Inst.Radio Eng., 40*(9), 1098-1101.

Jacobson, G. (1989). Succinct static data structures. Carnegie-Mellon).

Knuth, D. E. (1973,). *The art of computer programming. vol. 3: Sorting and searching* Addison-Wesley.

Lesk, M. (1997). *Practical digital libraries: Books, bytes, and bucks* Morgan Kaufmann.

Manber, U., & Myers, G. (1993). Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing, 22*(5), 935-948.

Moffat, A. (1989). Word-based text compression. *Software Practice and Experience, 19*(2), 185-198.

Advanced Encryption Standard (AES), (2001).

Navarro, G., & Mäkinen, V. (2007). Compressed full-text indexes. *ACM Computing Surveys, 39*(1), Article No. 2.

Ottaviano, G., & Venturini, R. (2014). Partitioned elias-fano indexes. *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14,* pp. 273-282.

Rao, S. S. (2003). Electronic books: A review and evaluation. *Library Hi Tech, 21*(1), 85-93.

Rydberg-Cox, J. A. (2005). *Digital libraries and the challenges of digital humanities*. Oxford, UK: Chandos Publishing.

Sadakane, K. (2003). New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms, 48*(2), 294-313.

Subramanya, S. R., & Yi, B. K. (2006). Digital rights management. *IEEE Potentials, 25*(2), 31-34.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes: Compressing and indexing documents and images* Morgan Kauffman.

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys, 38,* Article No 6.