Searching the Web: Challenges and Partial Solutions

Ricardo A. Baeza-Yates *

Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile
E-mail: rbaeza@dcc.uchile.cl

Abstract

In this article we analyze the problem of searching the WWW, giving some insight and models to understand its complexity. Then we survey the two main current techniques used to search the WWW. Finally, we present recent results that can help to partially solve the challenges posed.

1. Introduction

The boom in the use of the World Wide Web (WWW) and its exponential growth are well known facts nowadays. Just the amount of textual data available is estimated in the order of one terabyte. In addition, other media, as images, audio and video, are also available. Thus, the WWW can be seen as a very large, unstructured but ubiquitous database. This triggers the need for efficient tools to manage, retrieve, and filter information from this database. This problem is also becoming important in large Intranets, where we want to extract or infer new information to support a decision process. This task is called data mining. We make the important distinction between data and information. The later is processed data that fulfills our needs.

In this article we outline the main problems of searching the WWW and some partial solutions to them. We focus on text, because although there are techniques to search in images and other non-textual data, they cannot be applied (yet) in large scale. We also emphasize syntactic search. That is, we search for WWW documents that have some words or patterns as content, which may (in most cases) or may not reflect the intrinsic semantics of the text. Although there are techniques to preprocess natural language and extract the text semantics, they are not 100% effective and they are also too costly for large amounts of data. In addition, in most cases they work with well structured text, a thesaurus and other contextual information.

We now mention the main problems posed by the WWW. We can divide them in two classes: problems of the data itself and problems of the user. The first are:

- Distributed data: due to the intrinsic nature of the Web, data spans over many computers and platforms. These computers are interconnected with no predefined topology and with very different bandwiths.
- High percentage of volatile data: due to Internet dynamics, new computers and data can be added or removed easily. We also have relocation problems when domain or file names change.
- Large volume: the exponential growth of the WWW poses scaling issues that are difficult to cope with.
- Unstructured data: most people say that the WWW is a distributed hypertext. However, this is incorrect. Any hypertext has a conceptual model behind, which organizes and adds consistency to the data and the hyperlinks. That is hardly true in the WWW, even for individual documents.
- Quality of data: the WWW can be considered as a new publishing media. However, there is, in most cases, no editorial process. So, data can be even false, invalid (for example, because is too old), poorly written or, typically, with many errors from different sources (typos, grammatic mistakes, OCR errors, etc.)

*This work was supported by CYTED Project VII.13: AMYRI.

- Heterogeneous data: in addition to having to deal with multiple media types and hence with multiple formats, when talking only about text, we also have different languages and, what is worse, different alphabets, some of them very large (for example, Chinese or Japanese Kanji).

Most of these problems are not solvable by just software improvements. For example, the cross-language or bad quality issues. Those will not change (and it should not in some cases!) or imply changing working habits. The second class of problems are faced by the user. Given the above, there are basically two problems: how to query and how to manage the answer of the query. Without taking in account the content semantics of a document, it is not easy to precisely specify a query, unless it is very simple. On the other hand, if we are able to pose the query, the answer might be a thousand of WWW pages. How do we handle a large answer? How do we rank the documents? (that is, how we select the documents that really are of interest for the user). In addition, a single document could be large. How do we browse efficiently in such documents?

So, the overall challenge is to, in spite of the intrinsic problems posed by WWW, circumvent all of them and answer the questions above, such that a good query could be sent to the search system, obtaining a manageable and relevant answer. The organization of this paper is as follows. We first describe and model the WWW. This is the first step to understand its complexity and being able to analyze possible solutions. Second, we outline the main ways used today to search the Web, giving some examples. Third, we outline several new results that should help in (partially) solving some of the problems outlined. Between them, we can mention compression techniques allowing random-access, use of available text structure, visual query languages and visual browsing. Some of the results presented are part of an Iberoamerican project funded by CYTED (AMYRI) which has as a goal the research and development of techniques and tools to search the WWW [13].

2. Measuring and Modeling the WWW

In an interesting article, already (sadly) outdated, Tim Bray [15] studied different statistical measures of the WWW. From simple questions as how many servers there are in the WWW to characterizing WWW pages. Currently, there are near one million servers, counting domain names starting with www. However, as not all WWW servers have the www prefix, the real number is higher. On the other hand, the number of independent institutions that have WWW information is much less, because many places have multiple servers. The exact percentage is unknown, but should be more than 30% which was the result back in 1995.

The most popular formats of WWW documents are HTML followed by GIF, TXT, PS, and JPG, in that order. How is a typical HTML page? First, most of them are not standard, meaning that they do not comply with all the HTML specifications.In addition, although HTML is an instance of SGML, HTML documents seldom start with a formal document type definition. Second, they are small, a few Kbs (around 6 to 10) and no images. The pages that do have images, use them for presentation issues as colored bullets and lines. The average page has between 5 and 15 references (links) and most of them are local (their own WWW server hierarchy). However, on average no external server points to it (commonly there are only are local links). In fact, in 1995, around 80% of the pages had less than 10 links to itself.

The top ten most referenced sites sites are Microsoft, Netscape, Yahoo!, and top US universities. In those cases we are talking about sites being pointed by at least ten thousand places. On the other hand, the site with most external links is Yahoo!. In some sense, Yahoo! is the glue of the WWW. Otherwise, we would have many isolated portions (this is the case with most personal WWW pages). If we assume that each HTML page has 6Kb and there are 100 pages per server, for one million servers we have at least 600Gb of text. The real volume should be larger.

Can we model the document characteristics of the whole WWW? We will make a first attempt. The first problem is the distribution of document sizes, which has been found to have self-similarity [18]. This can be modeled using a "heavy-tail" distribution. That is, the majority of documents are small, but there is non trivial number of large documents. This is intuitive for image or video files, but it is also true for HTML pages. The simplest "heavy-tail" distribution is called the Pareto distribution: the probability of a document of size x is $\alpha k^{\alpha}/(x^{1+\alpha})$ where k and α are parameters of the distribution (see Figure 1). For text files, α is about 1.36, being smaller for images and other binary formats. In fact, for less than 50Kb, images are the typical files, from there to 300Kb we have audio files, and over that to several megabytes we have video files.

For text files, the second important thing is the number of distinct words or vocabulary of each document. We use the *Heaps' Law* [23]. This is a very precise law ruling the growth of the vocabulary in natural language texts. It states that the vocabulary of a text of n words is of size $V = Kn^{\beta} = O(n^{\beta})$, where K and β depend

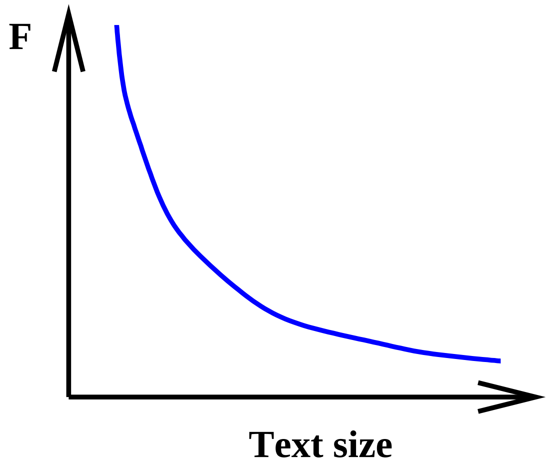

Figure 1. Document size distribution.

on the particular text (see Figure 2). K is normally between 10 and 100, and β is between 0 and 1 (not included). Some recent experiments [8, 12] show that the most common values for β are between 0.4 and 0.6. Hence, the vocabulary of a text grows sublinearly with the text size, in a proportion close to its square root.

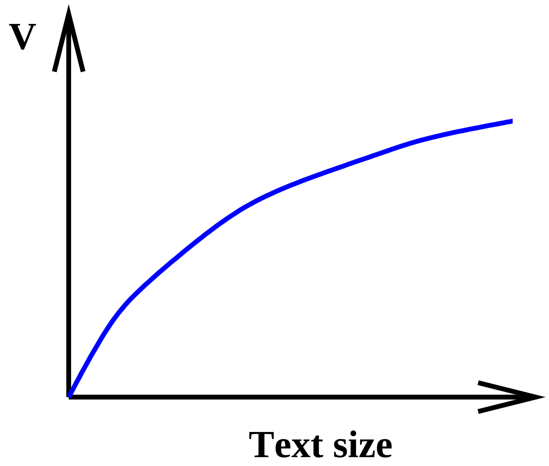

Figure 2. Size of the vocabulary

A first inaccuracy appears immediately. Supposedly, the set of different words of a language is fixed by a constant (e.g. the number of different English words is finite). However, the limit is so high that it is much more accurate to assume that the size of the vocabulary is $O(n^\beta)$ instead of $O(1)$, although the number should stabilize for huge enough texts. On the other hand, many authors argue that the number keeps growing anyway because of the errors

Another inconsistency is that, as the text grows, the number of different words will grow too, and therefore the number of letters to represent all the different words will be $O(\log(n^\beta)) = O(\log n)$. Therefore, longer and longer words should appear as the text grows. The average length could be kept constant if shorter words are common enough (which is the case). In practice, this effect is not noticeable and we can assume an invariable length, independent of the text size.

The third issue is how the different words are distributed inside each document. A much more inexact law is the *Zipf's Law* [37, 22], which rules the distribution of the frequencies (that is, number of occurrences) of the words. The rule states that the frequency of the i-th most frequent word is $1/i^\theta$ times that of the most frequent word. This implies that in a text of n words with a vocabulary of V words, the i-th most frequent word appears $n/(i^\theta H_V(\theta))$ times, where

$$H_V(\theta) = \sum_{i=1}^{V} \frac{1}{i^\theta}$$

so that the sum of all frequencies is n (see Figure 3). The value of θ depends on the text. In the most simple formulation, $\theta = 1$, and therefore $H_V(\theta) = O(\log n)$. However, this simplified version is very inexact, and the case $\theta > 1$ (more precisely, between 1.5 and 2.0) fits better the real data [8]. This case is very different, since the distribution is much more skewed, and $H_V(\theta) = O(1)$.

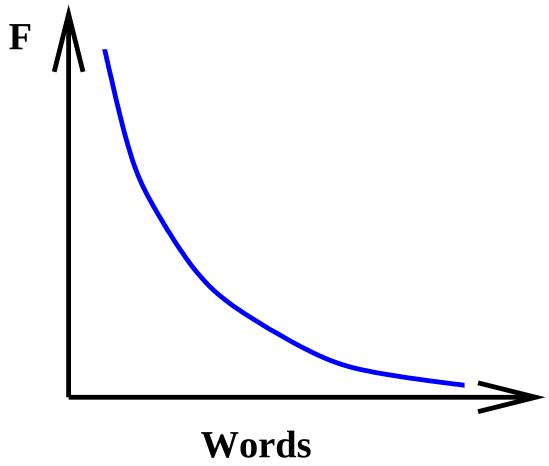

Figure 3. Sorted frequencies of the words.

The fact that the distribution of words is very skewed (that is, there are a few hundreds of words which take up 50% of the text) suggest a concept which is frequently used in full-text retrieval: the *stopwords* [30]. A *stopword* is a word which does not carry meaning in natural language and therefore can be ignored (i.e. made not searchable), such as `"a"`, `"the"`, `"by"`, etc. Fortunately, the most frequent words are stopwords, and therefore half of the words appearing in a text need not be considered. This allows, for instance, to significantly reduce the space overhead of indices for natural language texts.

3. Searching the Web

There are basically three different approaches to search the WWW. Two of them are well known and used frequently. The first, is to use search engines that index all the WWW as a full-text database. The second, is to use Internet directories (catalogues or yellow pages). The third and not yet fully available, is to search the WWW as a graph. In the next paragraphs we outline and exemplify the two main approaches currently available.

3.1. Search Engines

Most search engines use the crawler-indexer architecture. Crawlers are pieces of software that traverse the WWW sending new or updated pages to a main server where they are indexed. That index is used in a centralized fashion to answer queries submitted from different places in Internet. The most large search engines in WWW coverage are Hotbot [3], Altavista [17], Northern Light [4], and Excite [1], in that order. According to recent studies the coverage of the WWW by these engines varies from 28 to 55% [5] or 14 to 34% [26], as the number of WWW pages is estimated from 200 to 320 million. More facts about search engines can be found in the last two references.

The WWW pages found by the search engine are ranked, usually using the number of occurrences of the query on each page. In most cases this is effective, in others may not have any meaning, because relevance is not fully correlated with query occurrence. The user can refine the query by constructing more complex queries based on the previous answer. As the users receive only a subset of the answer (the first 10 to 100 matches), the search engine should keep each answer in memory, such that is not necessary to recompute it if the user asks for the next subset. Search engines user interfaces in addition to words, allow to filter pages by using boolean operators, and geographic, language or date segmentation.

The main problem faced by these engines is the recollection of data, because of the highly dynamic nature of the WWW, saturated communication links and high loaded WWW servers. Another important problem is the volume of the data. Then, these schema may not be able to cope with WWW growth in the near future.

There other several variants of the crawler-indexer architecture. Between them we can mention Harvest [14] which uses a distributed architecture to gather and distribute data, being more efficient. However, the main drawback is that needs the coordination of several WWW servers. Another variant is WebGlimpse [28] that attaches a small search box to the bottom of every HTML page, and allows the search to cover the neighborhood of that page or the whole site, without having to stop browsing. We also have to mention search engines that specializes in specific topics. For example the Search Broker [27] or the Miner family [34]. Finally, we have the metasearchers. These are WWW servers that use several engines, collect the answers and unify them. Examples are Metacrawler [35] and SavvySearch [19].

3.2. WWW Directories

The best example of WWW directories is Yahoo! [2]. Directories are hierarchical taxonomies (trees) that classify human knowledge. The main advantage of this technique is that if we find what we are looking for, the answer will be in most cases useful. On the other hand, the main disadvantage is that the classification is not specialized enough and that not all WW pages are classified. The last problem is worse every day as the WWW grows. The efforts to do automatic classification in AI are very old. However, until today, natural language processing is not 100% effective to extract relevant terms from a document. Nowadays, classification is done by a limited number of people.

3.3. Finding the Needle in the Haystack

Now we give a couple of search examples. One problem with full-text retrieval is that although many queries can be effective, many others are a total deception. The main reason is that a set of words do not capture all the semantics of a document. There is too much contextual information that can be explicit or even implicit, which we understand when we read. For example, suppose that we want to learn oriental games as Shogi or Go. For the first case, searching for Shogi will give you very fast good WWW pages where we can find what Shogi is (a variant of chess) and its rules. However, for Go the task is complicated, because in opposition to Shogi, is not a unique word in English. We can add more terms to the query, as `game` and `japanese` but still we are out of luck, as the pages found are almost all about Japanese games written in English where the common verb go is used.

The following example taken from [9] explains better this problem, where the ambiguity comes from the same language. Suppose that we want to wind the running speed of the jaguar, a big South American cat. A first naive search in Altavista would be `jaguar speed`. The results are pages that talk about the Jaguar car, an Atari video game, a US football team, a local network server, etc. The first page about the animal is ranked 183 and is a fable, without information about the speed. In a second try, we add the term `cat`. The answers are about the Clans Nova Cat and Smoke Jaguar, LMG Entreprises, fine cars, etc. Only the page ranked 25 has some information on jaguars but not the speed. Suppose we try Yahoo!. We look at `Science:Biology:Zoology:Animals:Cats:Wild_Cats` and `Science:Biology:Animal_Behavior`. No information about jaguars there. We can try to do a more specific search, for example using LiveTopics. However here we also have a shortage of topics, so searching by jaguar only returns cars or football teams.

The lessons learned in the example are that search engines still return too much hay to find the needle while the directories do not have enough deepness to find the needle. So, we can use the following rules of thumb:

- Specific queries: look at an Encyclopedia.
- Broad queries: use directories.
- Vague queries: use search engines.

4 Improvements to Inverted Files

Most indices use variants of the inverted file. An inverted file is a list of sorted words (vocabulary), each one having a set of pointers to the pages where it occurs. As we mentioned before, a set of frequent words or stopwords are not indexed. This reduces the size of the index. Also, it is important to point out that a normalized view of the text is indexed. Normalizing operations include removal of punctuation and multiple spaces to just one space between each word, uppercase to lowercase letters, use of synonyms through a thesaurus, etc. For more information on Information Retrieval algorithms and data structures see [21].

State of the art techniques can reduce an inverted file to about 20% of the text (the Altavista index has around 200Gb and 16 DEC Alpha servers are used to answer the queries, each one with several processors and 8Gb -sic- of RAM). A query is answered by doing a binary search on the sorted list of words. If we are searching multiple words, the results have to be combined to obtain the final answer. This step will be efficient if each word is not too frequent.

Inverted files can also point to actual occurrences. However, that is too costly in space for the WWW, because then each pointer has to specify a page and a position inside the page (word numbers can be used instead of actual bytes). On the other hand, having the positions we can answer phrase searches or proximity queries, by finding words that are after each other or nearby in the same page, respectively.

Finding words starting with a prefix, are solved by doing two binary searches in the sorted list of words. More complex searches, like words with errors, arbitrary wildcards or in general, any regular expression on a word, can be performed by doing a sequential scan over the vocabulary. This may seem slow, but the best sequential algorithms for this type of queries can achieve near 5Mb per second and that is more or less the vocabulary size for 1Gb. Then, for several Gbs we can answer in a few seconds. For the WWW is still too slow (around three minutes for the Altavista index) but not completely out of the question.

Pointing to pages or to word positions is an indication of the granularity of the index. This can be less dense if we point to logical blocks instead of pages. In this way we reduce the variance of the different document sizes, by having all blocks to have roughly the same size. This not only reduces the size of the pointers (because there are less blocks than documents) but also reduces the number of pointers because words have locality of reference (that is, all the occurrences of a non-frequent word will tend to be clustered on the same block). This idea was used in Glimpse [29] which is the core of Harvest [14]. Queries are resolved in the same way in the vocabulary and then are sequentially searched in the corresponding block (exact sequential search can be done over 7Mb per second). Glimpse originally used only 256 blocks, which was efficient up to 200Mb for searching words that were not too frequent, using an index of only 2% of the text. However, tuning the number of blocks and the block size, reasonable space-time trade-offs can be achieved for larger document collections. In fact, in [12] we show that for searching words with errors we can have sublinear space and search time simultaneously. These ideas cannot be used (yet) for the WWW because sequential search cannot be afforded, as it implies a network access. However, in a distributed architecture where the index is also distributed, logical blocks make sense.

The last issue is compression. Inverted lists can be partly compressed, in particular list of occurrences with high granularity. In those cases, the list is a sequence of ascending integers where the differences are much smaller than the values itself. Therefore, we can store just the first value and a sequence of differences. This complicates a bit the query evaluation, but space gains are obtained. Compression can also be applied to the text. However, most compression schemes are context dependent. That is, to decompress we have to decompress the whole file. Nevertheless, by using Huffman byte-coding over words (not letters), compression ratios of 30% coupled with random access to the compressed file are achieved [31]. This compression technique can be combined with the logical block scheme, obtaining an 8-fold improvement over normal sequential search by searching the compressed query word over the compressed text. The improvements came from the fact that one third of the I/O is done and searching over a shorter file is much faster.

5. Visual Query Languages

Traditional systems used words and boolean operations (and, or, butnot) to retrieve information. However, common users many times are confused by these operators, partly due to how we use logical connectives in normal language. That problem still remains today, but search engines have improved the searching interfaces to make things more clear (for example, using "all of", "some of", or "none of" the words). Another solution is to use a visual metaphor to represent the boolean operations. For example, a spatial relation, where the horizontal axis specify groups of words that must be together while the vertical axis specify that a least one of the groups must be present.

Another way to enhance content queries is to use the structure of the document. For example, HTML structure. A query to find a word near an image can be much more precise than just searching the word. For this, the index must include structural elements, adding little space, as structure is usually sparse. There are several proposals for query languages over content and structure [11]. However, most of them are too complicated for the final user. This drawback can be circumvent by using a visual query language. This is very natural as structure is highly correlated with the layout of a document. So, specifying a word near an image using a palette of elements, is not too difficult.

A proposed metaphor for a simple visual query language is given in this proceedings. What the user usually sees is a page of text. So our visual language will be a page where the structure is composed from a set of predefined objects and the content is written where we want to find it. Each structure element will be a rectangle with its name in the top (using the special name Text if it is a content element). Each content element is placed inside the rectangle. Union of queries are obtained by putting rectangles besides each other. Inclusion is obtained by placing rectangles inside rectangles. Figure 4 shows a query where the Text element a must be inside a chapter and should not include documents having the content element c.

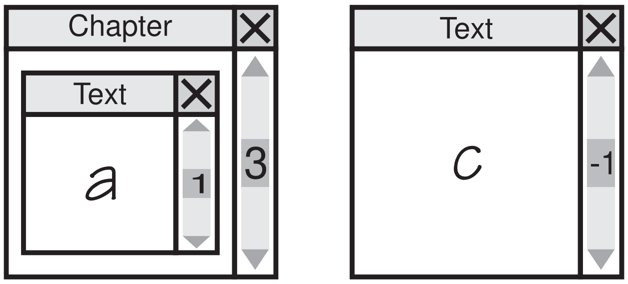

Figure 4. Example of visual query.

6. Visual Browsing

Most visual representations focus on some specific aspects. In text retrieval we can distinguish visualizations for a single document, several documents or queries. Most of the time only one of those elements is visualized. In the last years, several visual metaphors have been designed, for each element, describing next some of them.

6.1. Text Visualization

Possible text visualizations follow, in addition to the normal one, which is the text itself.

- One possible view is a normal text window with an augmented scroll-bar (similar to [32] but in a different context) which has marks where the text positions appear in the document. The scroll-bar can be viewed as a complete compact view of the text [10].
- The text itself is fish-eyed zooming where the query occurs given some adjacent lines to understand the context. The number of lines can be modified by the user. We call this an elastic text [10].
- Only the text layout is given, in multiple columns [20]. Colored lines may indicate where the query occurs.

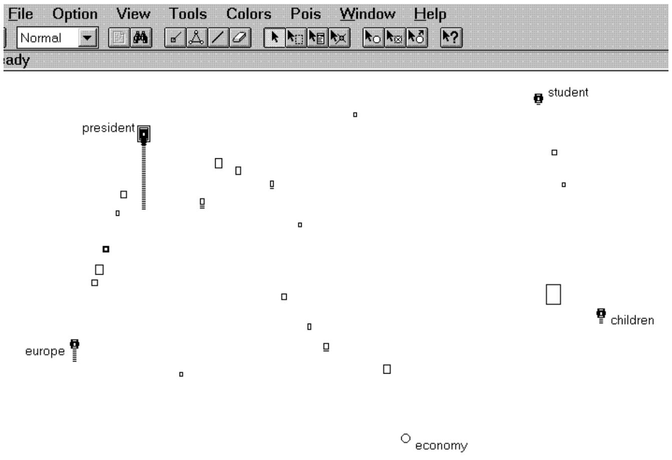

Figure 5. Answer visualized by VIBE.

6.2. Document Visualization

Nowadays there are more than 20 proposals for visualizing a set of documents. The VIBE system [33] is based on user given points of interest of the query (using weighted attributes). These points (query words) act as gravitational forces that attract the documents according to the number of occurrences of each query (see Figure 5). Documents are displayed with different icon types and sizes.

Another metaphor for the document space based on inter-particle forces, as VIBE, is proposed in [16]. In [36] the document space is abstracted from a Venn diagram to an iconic display called InfoCrystal. One advantage of this scheme is that is also a visual query language. Visual tools in three-dimensions to handle the document space are presented in LyberWorld [25]. Visualization of occurrence frequency of terms in different text segments of a document is presented in [24].

Now we present a more elaborate metaphor for manipulating and filtering an answer given by a set of documents [10]. Figure 6 shows an instance of the visual analysis tool that we propose for advanced users. We use a "library" or "bookpile" analogy depending if the tool is used horizontally or vertically, because both are possible. Each document (seen as a book) is represented as a rectangle with a particular color, height, width and position into the set. Each one of this graphical attributes, including the order of the list, can be mapped to a document attribute (occurrence density, size, date, etc). In the example, the order and the color are mapped to the same attribute (for example, the creation date). These mappings allow to study different correlations of attributes on the document set, helping the user to select the desired documents. A select button allows to choose a document subset by using the mouse (the wide border rectangle in the example). A prototype is explained in [7] and available in [6].

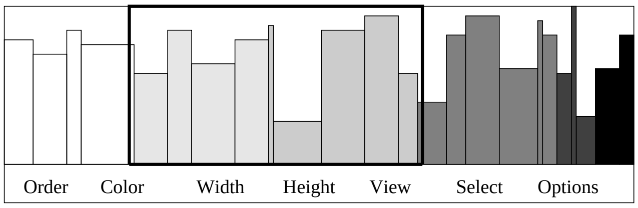

Figure 6. Analyzing and selecting a document set.

The mapping of the attributes is selected by the menu buttons below the book list. The way the books are seen can also be changed. The set of documents can be forced to fit into the window (as in the example), presented using a predefined choice of maximal/minimal widths and heights (using a scroll-bar if bigger than the window). Another view is a fish-eye representation for large sets, focusing where the user wants (by clicking with the mouse the appropriate sector).

6.3. Query Visualization

The relations between query terms and the answer can also be visualized [10]. For example, a pie view showing the percentage of documents of the total database that were selected. Another possibility is to show the distribution of occurrences within terms of the query using boxes of different sizes. This view is useful to know what terms are the best filters in the given query. A third possibility, is to show the distribution of documents selected on the database logical or physical space. This view can show if there is any logical locality of reference associated with the query.

7. Conclusions

A careful integration of some of the new results presented here can help on partially solving the problems of searching the WWW. For example, a truly cooperative and distributed architecture similar to Harvest can diminish WWW traffic and extend the scalability of search engines. In each local index, compression and logical blocks can be used, obtaining smaller indices and documents. This is one of the goals of the AMYRI project [13] mentioned in the introduction. The first step would be a client-server architecture as shown in

Figure 7. The second step would be to distribute the search engine server.

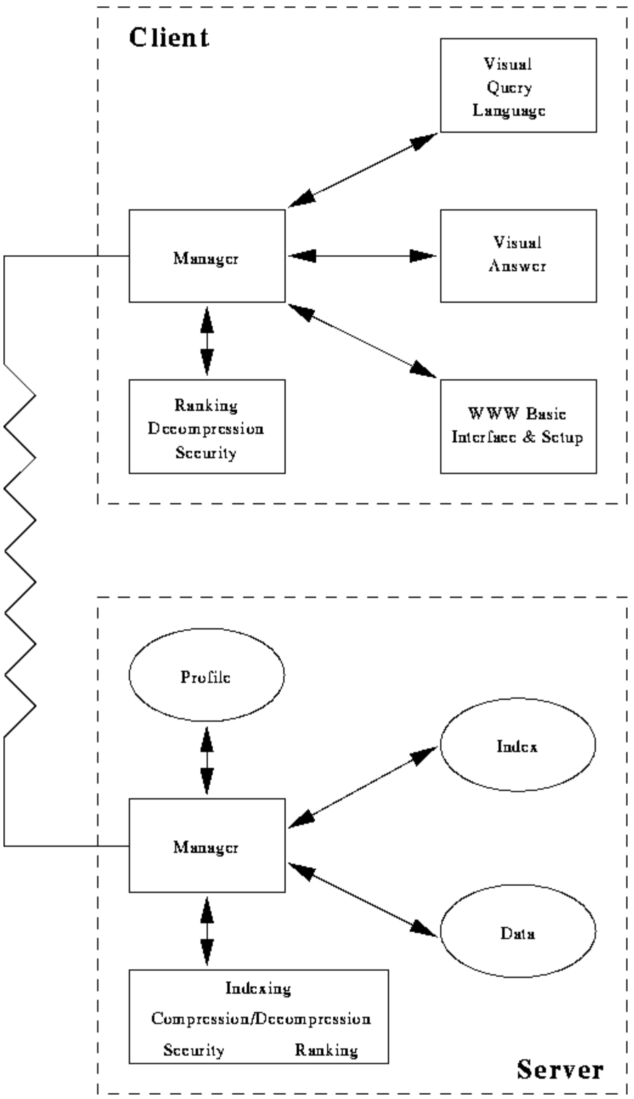

Figure 7. Client-server architecture of AMYRI

Acknowledgments

We thanks the helpful comments of Gonzalo Navarro, in particular his contribution to Section 2.

References

[1] Excite: Main page. `http://www.excite.com`, 1995.

[2] Yahoo!: Main page. `http://www.yahoo.com`, 1995.

[3] Hotbot: Main page. `http://www.hotbot.com`, 1996.

[4] Northern Light: Main page. `http://www.northernlight.com`, 1997.

[5] Search Engine Watch: Main page. `http://www.searchenginewatch.com`, 1997.

[6] O. Alonso and R. Baeza-Yates. A bookpile applet. `http://www.dcc.uchile.cl/~rbaeza/sem/visual/mio/Bucpil.html`, 1997.

[7] O. Alonso and R. Baeza-Yates. Visualizations of answers in WWW retrieval. Technical report, Department of Computer Science, Univ. of Chile, 1997. `http://nova1.cs.nvgc.vt.edu/ alonso/viswww.html`.

[8] M. Araújo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In *Proc. WSP'97*, pages 2–20. Carleton University Press, 1997.

[9] R. Baeza-Yates. Modeling, browsing and querying large text databases. Technical Report DCC-94-2, Dept. of Computer Science, Univ. of Chile, 1994.

[10] R. Baeza-Yates. Visualizing large answers in text databases. In *Int. Workshop on Advanced User Interfaces (AVI'96)*, pages 101–107, Gubbio, Italy, May 1996. ACM Press.

[11] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.

[12] R. Baeza-Yates and G. Navarro. Block-addressing indices for approximate text retrieval. In *Proc. CIKM'97*, pages 1–8, Las Vegas, USA, 1997.

[13] R. Baeza-Yates and N. Ziviani. AMYRI: Main page. http://www.dcc.ufmg.br/latin/amyri/, 1997.

[14] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28:119–125, 1995.

[15] T. Bray. Measuring the web. In *Fifth International World Wide Web Conference*, Paris, May 1996. `http://www5conf.inria.fr/fich_html/papers/P9/Overview.html`.

[16] M. Chalmers and P. Chitson. BEAD: Exploration in information visualization. In *ACM SIGIR'92*, 1992.

[17] D. E. Corporation. Alta Vista: Main page. `http://altavista.digital.com`, 1996.

[18] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 160–169, May 1996.

[19] D. Dreilinger. Savvysearch home page. 1996. `http://guaraldi.cs.colostate.edu:2000/`.

[20] S. Eick. Graphically displaying text. *Journal of Computational and Graphical Statistics*, 3(2):127–142, 1994.

[21] W. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[22] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 2nd edition, 1991.

[23] J. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, NY, 1978.

[24] M. Hearst. Tilebars: Visualization of term distribution information in full text information access. In *ACM SIGCHI*, Denver, CO, May 1995.

[25] M. Hemmje, C. Kunkel, and A. Willet. Lyberworld - a visualization user interface supporting text retrieval. In *17th ACM SIGIR*, Dublin, Jul 1994.

[26] S. Lawrence and C. Giles. Searching the world wide web (in reports). *Science*, 280(5360):98, April 3 1998.

[27] U. Manber and P. Bigot. Search Broker: Main page. `http://debussy.cs.arizona.edu/sb/`, 1997.

[28] U. Manber, M. Smith, and B. Gopal. Webglimpse: combining browsing and searching. In *Proc. of USENIX Technical Conference*, 1997.

[29] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. In *Proc. USENIX Technical Conference*, pages 23–32. USENIX Association, Berkeley, CA, USA, Winter 1994.

[30] G. Miller, E. Newman, and E. Friedman. Length-frequency statistics for written English. *Information and Control*, 1:370–380, 1958.

[31] E. Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In *Proc. SIGIR'98*, Melbourne, Australia, Aug. 1998. ACM Press.

[32] D. Olsen. Bookmarks: An enhanced scroll bar. *ACM Trans. on Computer Graphics*, 11(3):291–295, 1992.

[33] K. Olsen, R. Korfhage, K. Sochats, M. Spring, and J. Williams. Visualization of a document collection: The VIBE system. *Information Processing and Management*, 29(1):69–81, 1993.

[34] V. Ribeiro and N. Ziviani. Meta Miner: Main page. `http://canela.dcc.ufmg.br:8080/metaminer.html`, 1997.

[35] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, Dec. 1995. `http://www.w3.org/pub/-Conferences/WWW4/Papers/169`.

[36] A. Spoerri. Infocrystal: A visual tool for information retrieval and management. In *Information and Knowledge Management'93*, Washington D.C., 1993.

[37] G. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, 1949.