Abstract

The fast growth of electronic text collections (in particular, the Web) and the diversity of available documents immensely increased the difficulty to retrieve relevant documents in an efficient way. A variety of Web search engines have been built to help users in this task. These systems, however, lack precision in the retrieved documents. Different solutions to improve retrieval precision have been proposed, however they did not show to be satisfactory so far. Investigating a new approach to this problem, we developed the ActiveSearch system, a standalone application for suggesting to the user similar documents to the one being browsed or edited. It processes different document formats (e.g., HTML, DOC) in different repositories, focusing on the Internet, local area networks and local directories. Once activated, the system consults the repository being accessed by the user and reorders the list of retrieved documents according to their similarity to the current document's content and format, considering as well the user's preferences (registered in the user's profile). The documents in the reordered list may further be grouped into dynamic clusters, in order to facilitate the visualization of the results. Undergone tests showed a very good system's performance, with precision rates of 57%. The system's performance was compared to the Google Toolbar, showing a superiority of 34 percentile points in the precision rate.

KEYWORDS: Web Information Retrieval, Approximate Retrieval, Search for Similar Documents.

1. Introduction

In the information age, one of the main problems to be tackled is the precision of the documents' retrieval process in digital repositories. In fact, the fast growth of electronic text collections (in particular, the World Wide Web) and the diversity of available documents increased the difficulty to retrieve documents that are relevant to the user's information needs (IN).

The majority of available Web search engines (e.g., AltaVista
\footnote{http://www.altavista.com}, Google\footnote{http://www.google.com}) are based on keyword indexing systems. Although robust, this technique does not always provide an appropriate documents ordering regarding the user's IN, lacking precision \cite{1}. Some of the main problems with this approach are: the user's difficulty to choose keywords to represent his/her IN (mainly due to Synonymy\footnote{Synonymy: several terms (words or phrases) designating the same concept (e.g., disable, handicap, incapacitate).}); the lack of "semantics" in keyword queries to represent the user's IN context; and the intrinsic ambiguity of natural language (mainly due to Polysemy\footnote{Polysemy: one word with different meanings (e.g., bank - establishment for keeping money, river side).}), which compromises retrieval precision.

Different approaches to improve retrieval precision have been investigated, such as the use of concept hierarchies that can be browsed by the user while searching ( e.g., Yahoo!, Lycos\footnote{Yahoo - http://www.altavista.com/, Lycos - http://www.lycos.com/}). The pages in the system's Index Base are classified within one or more topics in the hierarchy, and this is usually done by hand. Clearly, this approach does not scale to the current Web size and growth rate. Furthermore, the classification is static and occurs previous to the search, which may compromise the system's update, flexibility and transparency.

In previous work, we proposed an alternative approach to enhance precision and recall in information retrieval sessions by the use of ontologies to dynamically associate a context to keyword queries \cite{2}. Via the Ontologies Manager Framework (OMF), the user is able to create ontologies (concept hierarchies) related to his/her interests from where terms can be selected to form a context for each query session. The OMF is a plug-in that can be connected to different search engines. Although tests with the prototype were very encouraging, the decision of leaving the construction of the ontologies in the user's hands (for transparency and flexibility) ended up to be a burden for non-expert users.

Our next step was towards a more automatic process to improve retrieval precision, investigating systems that search for similar documents to the one being accessed. In this case, the document itself is used as a context to the search, and the process is independent of the user's actions, who has only to provide a document as a starting point. Examples of such systems are Google Toolbar \cite{3} and Alexa \cite{4}, which are browsers' plug-ins. Although offering a considerable improvement, these plug-ins
suffer from two main limitations: they are restricted to treat HTML documents, and they deal solely with pages in their Web graph (see section 6). Another example of such system, the Kenjin [5] standalone application, does not present these restrictions. However, it showed low precision rates, probably due to the small size of its documents base.

In this light, we developed the ActiveSearch system, a standalone application designed with the main goal of suggesting to the user similar documents to the one being edited (e.g., via MS-Word) or accessed through a browser. As such, it must be able to process different document formats (e.g., HTML, DOC, PS), as well as to provide on-line access to different repositories, focusing on the Internet (via meta-searches), Local Area Networks (LAN) and local directories. Once activated, the system consults the repository being accessed by the user and reorders the list of retrieved documents according to their similarity to the current document's content and format, considering as well the user's preferences (registered in the user's profile). The reordered list of documents may be further grouped into dynamic clusters, in order to facilitate the visualization of the results.

The current version of the system was implemented in Borland Delphi 5.0 and it runs under Windows OS. Undergone tests showed a very good system's performance, with precision rates of 57%. The system was compared to Google Toolbar, showing a superiority of 34 percentile points in the precision rate. Partial results of this work can be seen in [6][7].

In what follows, section 2 brings an overview of the ActiveSearch system and section 3 gives a brief presentation of similarity measures for text documents. Section 4 shows implementation details of the system and section 5 presents an evaluation of the system's performance. Section 6 brings some of the related work, and section 7 summarizes the contributions and future work.

2. ActiveSearch System's Overview

The ActiveSearch system's main goal is to present to the user similar documents to the one being edited or accessed. For flexibility, the system may work only upon the user's request, or it can run in a continuous mode, retrieving new documents when the current document changes or in regular time intervals (which may be set by the user). When activated, the system's interface captures the current document (or a selected part of it) from the application being run by the user (e.g., Internet Explorer, MS Word), and translates it into a XML (eXtensible Markup Language) internal representation that keeps some of the document's original formatting (e.g., title, bold, italics, font size, hyperpointers) to facilitate the processing of documents and to improve the matching process between documents. Following, the system selects from this XML representation the document's most relevant keywords, which will form queries used to consult the repository being accessed.

When used in the Internet, ActiveSearch does not maintain its own Index Base (IB), to avoid duplication efforts of indexing the Web. Instead, it consults different search engine's IBs through meta-searches. Only when used in LANs or in local directories the system must create and maintain its own documents index base.

The list of pointers (URLs/directories addresses) returned from the consultation is reordered according to each document's similarity to the current one (considering the document's content and format, in order to improve the system's precision). The system also considers the user's preferences in this reordering, which are registered in the user's profile. The algorithm used to measure similarity between documents is based on the Vector Space model [8] (see section 3 for details). The reordered list is then presented via the system's Interface, and the user can access the documents by a mouse click. Alternatively, the user can activate a clustering process which groups the documents in the reordered list into dynamic classes. The aim is to facilitate the visualization of the results by the user, who can focus only on themes of interest. The user can also directly query the available search engines or the system's IB (in the case of LANs or local directories).

3. Measuring Similarity between Documents

In order to select one measure to be used by ActiveSearch, we thoroughly investigated several similarity and dissimilarity measures, among which we highlight [9][10]: Cosine, Sorensen, Jaccard, Dice, Inclusion, Overlap, Spearman, Pearson's correlation, Euclidean Distance, Bray-Curtis, Chord and Taxonomic Distance. The former 8 are similarity measures and the latter 4 are dissimilarity measures.

Since our system was designed to treat different document formats, the performance of the above-cited measures was evaluated for two corpora of documents, one containing only HTML documents and another with documents extracted from the TIPSTER/TREC6, an IR testbed. Our goal was to determine the most appropriate measure to be used by the ActiveSearch system. Due to space constraints, we present here only the two measures that obtained the best results: Cosine and Sorensen. A more comprehensive presentation of this experiment can be seen in [7].

The Cosine measure, also known as Ochiai's Coefficient, computes the cosine of the angle formed by the representation of two documents \( x \) and \( y \) as vectors in a \( t \)-dimensional space, where \( t \) is the number of terms occurring in the documents of the whole corpus (Eq. 1). The similarity between documents increases as the angle between their vector representations decreases (and, consequently, the cosine increases).

---

6 TREC - http://trec.nist.gov
Equation 1

\[
sim^{Co} = \frac{\sum_{i=1}^{n} (x_{i} \cdot y_{i})}{\sqrt{\sum_{i=1}^{n} (x_{i})^2 \cdot \sqrt{\sum_{i=1}^{n} (y_{i})^2}}
\]

The Sorensen measure (or Percent Similarity) is given by Eq. 2. In this case, if we interpret the Sorensen measure geometrically, a document would be represented by a point in the t-dimensional space. The similarity between two documents increases as one gets closer to the point representation of the other, and it will be a maximum exactly at the point (see table 1).

Equation 2

\[
sim^{So} = 2 \cdot \frac{\sum_{i=1}^{n} \min(x_{i} \cdot y_{i})}{\sum_{i=1}^{n} x_{i} + \sum_{i=1}^{n} y_{i}}
\]

<table>
<thead>
<tr>
<th>Sorensen</th>
<th>Cosine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corpus HTML</td>
<td>0.145</td>
</tr>
<tr>
<td>Corpus TREC</td>
<td>0.096</td>
</tr>
</tbody>
</table>

Table 1: Cosine and Sorensen similarity measures

According to our experiments, both measures obtained equivalent results. As such, we opted for the Cosine measure for two reasons: it is traditionally used in Information Retrieval systems; and according to it, two documents do not need to be identical in order to obtain the maximum similarity (it is sufficient that their vector representations have the same direction).

4. System's Architecture and Implementation Details

The system's architecture is modular, to safeguard extensibility (see fig. 1). It consists of six main modules, presented in section 4.1: Interface, Pre-processor, Profile Manager, Local Search Engine, Search, and Post-processor.

![Figure 1: System's Architecture](image)

![Figure 2: ActiveSearch's User Interface](image)

In the current version, the system's Interface communicates with two MS applications: Internet Explorer and Word. It uses MS-COM technology [11] to capture the current document being browsed/edited. There are two possibilities to consider here: (1) capturing the whole document, in which case only HTML and DOC styles are covered by the current version; and (2) capturing a selected part of the document (by a drag and drop operation), in which case the system is able to deal with any text format supported by Windows.

Pre-processor

This module is responsible for translating the input document (or the selected part of it) into an internal representation format in XML that keeps some of the document's original formatting (e.g., title, bold, italics, font size, hyperlinks). This module is also used by the Post-processor module to translate the retrieved documents into XML. The use of an uniform internal representation facilitates the processing of documents,
improves the matching process between documents (also because the used representation preserves some of the document's original formatting), and favors the system's extension to cope with other styles, such as XLS and PPT.

**Local Search Engine**

When used in local area networks or in local directories, ActiveSearch must create and maintain its own documents index base. This module is, in fact, an indexing engine for local networks and directories. It includes two sub-modules. The **Crawler** sub-module accesses the local directories selected by the user via the Interface and passes the retrieved documents onto the **Indexing** sub-module. Following, the centroid\(^8\) of each retrieved document is created, and the reference to the document is inserted in the system's index base. The latter sub-module was built based on the Jakarta Lucene search engine\(^9\), an open source free software. Within regular time intervals (e.g., weekly) the Crawler verifies whether new documents have been included in the indicated directories and whether already indexed documents have been changed (by checking the document's update date). The selected documents are passed back to the Indexing sub-module, which proceeds as usual.

**Search**

This module searches the Web or the system's local IB for documents similar to the current one. The **Query maker** sub-module receives the current document's XML representation and creates its centroid, taking into consideration the occurrence frequency of the terms plus formatting features such as title, bold, italics. Following, keyword queries representing the document's content are built based on this centroid. Currently, the system creates three queries\(^10\), one with the document's title, one with the three and another with the six most relevant terms in the document.

Following, these queries are passed onto the **Documents searcher** sub-module, which proceeds with the search for relevant documents in the selected repositories. Two possibilities are offered: (1) for searching the Web, this sub-module runs meta-searches to the selected search engines; and (2) when used in LANs and local directories, the system's IB is directly consulted. This sub-module does not retrieve the documents' content, but only creates a list of pointers to documents based on the search results. Currently, we limit the size of this list to 150 pointers (since they are likely to obtain the most relevant documents to the query, and a higher number of pointers would impact the system's time response). However, this is an easily customizable number. The results of the three queries are combined into one list of pointers, which is passed onto the **Post-processor** module.

---

\(^8\) Centroid: list of terms (words or phrases) appearing in the document, usually ordered by frequency occurrence.

\(^9\) http://jakarta.apache.org/lucene/docs/index.html

\(^10\) In future work, we will investigate the effects of processing a higher number of queries.

**Profile Manager**

In order to improve the retrieval and reordering precision, the system maintains a user profile, containing documents considered relevant by the user. The profile is created during the initial sessions of a particular user, and it can be automatically updated during subsequent sessions or at the user's requirement. Whenever a document is indicated as relevant by the user, the **Pre-processor** module creates its XML internal representation, which is stored in the user's profile.

The profile is only considered for use by ActiveSearch when a minimum number of documents has been informed, to guarantee coverage of the user's preference for different subjects. This is a customizable number, currently set to 15. The **Profile Manager** then extracts the documents' centroids and runs a Nearest Neighbor clustering algorithm [12] to group them into classes, so that each class should represent the user's interest in a particular subject. Following, the centroids in each group are combined to form the class' centroid. These classes are used by the **Post-processor** module to measure the similarity between the retrieved documents and the user's interests.

**Post-processor**

This module is responsible for rearranging the list of pointers obtained from the Search module, thus facilitating the access to the available information. Initially, the **Documents Fetcher** sub-module downloads the content of the documents indicated by the input pointers list, and passes them to the **Pre-processor** module, which creates their XML internal representation. Together with the current document's representation, these representations (and the original pointers) are passed onto the **Documents Rearranger** sub-module. This sub-module extracts the centroids of each document and creates their vector representation. These vectors are used to rearrange the list of pointers in two steps:

- The list is initially reordered based on the similarity of each document to the current one and to the user's profile, both based on the Cosine measure (Eq. 3). The similarity between a document and the profile is measured against each of the profile's classes, and only the class with the higher measure is considered in the reordering process (since each retrieved document usually is related to only one of the user's interests).

\[
\text{Rate}(doc) = k \times \text{sim}(doc,\text{currentDoc}) + (1-k) \times \text{sim}(doc,\text{Profile})
\]  

Equation 3

The rate of the document \( i \) in the reordered list is given by its similarity to the current document times \( k \) plus its similarity to the user's profile times \( (1-k) \). The variable \( k \) represents the weight of the similarity between each document and the current one (see section 5).

- The reordered list may be clustered, in order to group the related documents into classes. The aim is: (1) to facilitate the visualization of the results by the user, who
can focus only on themes of interest; and (2) to improve the precision of the presented results, since it is argued that relevant documents to a query tend to be grouped into 2 or 3 clusters [13]. Each class will provide a brief description of its content (the class' most relevant words), so that the user can deduce the content of a given class without having to look inside it (see section 5 for experiments).

The reordered list (or clusters) is then presented via the system's Interface, and the user can access the documents by a mouse click (see fig. 2).

5. Experiment's Results

In order to evaluate the performance of ActiveSearch, we used a test corpus of 12 Web pages belonging to different domains (e.g., football, Java language, jazz, among others). The search engine chosen to run the meta-searches was Google, to allow for a more realistic comparison of our results with Google Toolbar, which uses Google's index base. It would not be as adequate to compare our system's precision to Alexa or to Kenjin because the index bases used by these systems are not available for external querying.

The methodology deployed in our experiment is presented below:

- ActiveSearch and Google Toolbar were run for each of the 12 Web pages in the corpus, amounting to 24 lists of pointers to be analyzed. The first 150 returned pointers of each list were manually marked as similar or not to the original page. In order to improve the reliability of our tests, each document was analyzed by two users, and the average grade was associated to it.

- The average precision of both systems was measured according to Eq. 4, considering the top 10 pages of each of the 24 list of pointers (we believe that, in this kind of system, users do not usually consult more than the 10 top pointers). The obtained results are shown in the first and second rows in table 2. Observe that the precision improves as it gets closer to 1. At this stage of our experiment, ActiveSearch's Post-processor module was not activated so that we could later measure its impact in the documents reordering.

\[
\text{Average Precision} = \frac{\sum_{i=1}^{n} NRD_i}{10 \cdot n} \quad \text{Equation 4}
\]

where NRD is the number of relevant documents in the top 10 positions of the list returned by each search for each initial page, and n is the number of initial pages in the corpus.

- Following, the 12 lists of pointers returned by ActiveSearch in step 1 were reordered by the Post-processor module solely based on the similarity between the retrieved documents and the current one (i.e., without considering the user's profile). Again, our aim was to be able to later measure the impact of the profile in the documents reordering. Clearly, this simple reordering already improved the system's precision in 9 percentile points (see table 2).

- Finally, the same lists were post-processed, this time considering as well the user's profile. In order to better investigate the role of the profile in our system's performance, we tested three different weights of the profile's similarity to each of the analyzed documents (according to Eq. 3 in section 3). The weights considered to the profile were 20%, 30% and 50%, and the complementary weights of the similarity between each document and the original one were respectively 80%, 70% and 50%. The best results were obtained for the profile's weight set to 30% (see table 2). These results, however, are not conclusive, and demand a more deep investigation of the profile's role in our system.

<table>
<thead>
<tr>
<th>System</th>
<th>Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Toolbar</td>
<td>0.2333</td>
</tr>
<tr>
<td>ActiveSearch (step 2)</td>
<td>0.4667</td>
</tr>
<tr>
<td>ActiveSearch (step 3)</td>
<td>0.5583</td>
</tr>
<tr>
<td>ActiveSearch (step 4)</td>
<td>0.5750</td>
</tr>
</tbody>
</table>

Table 2: ActiveSearch's performance

ActiveSearch revealed a very good performance on these experiments, reaching a superiority of 34 percentile points in the precision rate in relation to Google Toolbar.

Currently, we are investigating the effects of clustering the pointers list, and initial tests already revealed a good precision rate. Initially, a clustering algorithm based on the Iterated Assign-to-Nearest algorithm [14] was run for each of the 12 ActiveSearch's already reordered lists of pointers. For each list of pointers, we measured the precision of each cluster (considering the top 10 pointers), according to Eq. 4. For 8 of the 12 initial pages, the relevant documents were grouped into 2 clusters, confirming the Cluster Hypothesis presented in [13]. Besides, the precision of these clusters was slightly higher than the system's average, varying from 0.6 to 0.7.

6. Related Work

Several are the available systems which tackle different aspects on the assisted use of the Web. In [15], we find a comprehensive review on the (so-called) Intelligent Internet Systems. Focusing on Web browsing, we cite here the Kenjin [5] application, as well as the plug-ins Google Toolbar [3] and Alexa[4] to illustrate the state of the art in this research field. Due to space constraints, our discussion concentrates on the offered similar pages facility, leaving aside the other services provided by these systems. See [6] for more details.

Google Toolbar maintains a graph mirroring the hyperlink structure of the Web pages indexed by Google search engine, which is used by its Similar Pages service. Two pages are considered similar when they are pointed
by a number of common pages in the graph [16][17]. Alexa maintains its own documents index base from which a hyperlink graph is constructed. However, its similar pages service is not only based on the links on the graph, but it also considers the pages' content in the retrieval process [18]. In both cases, the main problems to point out are: they only treat HTML documents; they do not treat an input page which is not in their graph; and the retrieval precision is good only when the current page is linked to a good number of pages in their Web graph.

Kenjin maintains its own documents index base, and the retrieval process is on-line and based on the document's content [19]. As such, it retrieves similar pages also to documents not previously stored in its own index base. It deals with different document formats (e.g., DOC, HTML, PS) on different applications (e.g., Word, PowerPoint, Internet Explorer) and environments (Internet, LANs and local directories). Its main problem concerns the system's low precision, which may be due to the small size of its index base.

7. Conclusions

We conclude this article highlighting the main contributions of our work. As it could be seen, the ActiveSearch system adds to its research area by: running an on-line and content-based retrieval process; providing access to different documents bases (the Web and local IBs); treating different document formats; maintaining a dynamic user's profile; deploying a traditional similarity measure algorithm considering as well each term's relevance within the document; improving the results' precision by rearranging the list of pointers based on the current document and on the user's profile; clustering the reordered list to facilitate visualization; presenting temporal continuity and pro-activity (proceeding with new searches when the current document changes, or in regular time intervals). Besides, the system's architecture is modular and extensible, supporting intercommunication with others applications. The experiments showed a good system's performance when compared with Google Toolbar.

Some of the future works foreseen are: testing the system with other similarity measures; preparing a larger corpus of HTML documents tagged with relevance marks (to allow for a more fine-grained evaluation); evaluating the system with this new corpus; evaluating the system's use on local networks and local directories; investigating in more detail the creation and use of the user's profile.

References


