PaMS: A component-based service for finding the missing full text of articles cataloged in a digital library


Department of Computer Science, Federal University of Minas Gerais, 31270-901 Belo Horizonte, MG, Brazil

ARTICLE INFO

Keywords:
Digital libraries
Meta-search
Component-based software development

ABSTRACT

Providing access to the full text of cataloged articles is a highly desirable feature for a digital library. However, in many such systems, not all metadata records have (a direct pointer to) a corresponding full-text document. In this article, we present PaMS: a new service for finding the missing full text of articles cataloged in a digital library. This service is implemented as a software component in order to be readily deployable to existing systems. It works as a parameterized meta-search engine and allows digital library administrators to easily set up a search strategy, i.e., a list of existing search engines to be queried for the missing full text, as well as the filtering and ranking policies to be applied to the results retrieved by each search engine. We evaluate our service with respect to its effectiveness and efficiency with collections from two distinct fields: computer science and biomedical and life sciences. Our results attest the effectiveness of PaMS for finding missing full-text documents as well as other relevant material while keeping its overall execution time at a reasonable level.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

On-line access to the full text of cataloged items is an important requirement for satisfying the needs and expectations of the users of a digital library (DL) of scientific articles [10]. However, in many of such DLs, mainly those built by aggregating metadata from heterogeneous sources, not all (metadata) records have a direct pointer (e.g., a URL) to the corresponding full text. As examples of DLs in the computer science field that suffer from this problem, we can cite the DBLP Computer Science Bibliography1 and the Brazilian Digital Library of Computing—BDBComp.2

Even the existence of a direct pointer to the full text may be useless to the user in some cases. For example, the content of interest may be accessible only by payment and the user may not want to complete the transaction. Also, the link to the full text may be broken—the link was valid at cataloging time but, due to the dynamics of the Web, it became broken. An alternative for users who wish to obtain the full text of articles for which they already have some metadata is to employ these metadata to try to find the desired items on the Web with the aid of existing search engines (whether specialized or not) and to examine their returned results to check whether they correspond to the full texts wanted.

In this article, we propose and evaluate a service, called PaperMetaSearch (PaMS), which automatizes this process thus diminishing the user’s effort while trying to improve the results retrieved by different search engines through customizable search strategies. We also describe the architecture of a software component that implements this service and that can be deployed and reused in several digital libraries.
Our PaMS component is inspired in a search process described by Silva et al. [17] and briefly summarized by Laender et al. [10] that demonstrated to be quite effective in this specific task for collections in the computer science field. This process uses citation information present in the metadata record of a given article to submit queries to one or more search engines requesting for the missing full-text document. Candidate results are extracted from the resulting pages of these search engines and are submitted to a filtering process that tries to remove those potentially irrelevant. Then, before the remaining results are shown to the user in the DL interface, they are re-ranked to prioritize those with higher odds of satisfying the user’s needs.

The architecture of our PaMS component is quite flexible regarding the search strategies it can employ. All the steps of the aforementioned process can be configured and adjusted to match the requirements of a specific digital library in which the component will be employed. Besides its description, we present an evaluation of our component with respect to its effectiveness and efficiency for collections in two distinct fields: computer science articles from the Brazilian Digital Library of Computing and articles cataloged in the PubMed Central, a free digital archive of biomedical and life sciences literature.

In summary, the main contributions of this article are:

1. The proposal of a component-based approach for the adaptive retrieval of full-text documents or any relevant related material for those articles cataloged in a DL for which this information is missing.
2. An effectiveness and efficiency study of the proposed component based on experiments conducted with metadata records of articles from the computer science and biomedical and life sciences fields.

The remainder of this article is organized as follows. Section 2 discusses related work. Section 3 describes the PaMS service architecture and its proposed component-based implementation. Section 4 describes the experimental setup and discusses the results of the experiments conducted with our component. Finally, Section 5 presents our conclusions and directions for future work.

2. Related work

Current approaches to find full-text documents missing from DLs rely mostly on focused crawlers. As introduced by Chakrabarti et al. [3], a focused crawler seeks, acquires, indexes, and maintains pages on a specific set of topics that represent a relatively narrow segment of the Web. This is accomplished by designing a complex priority scheme that guides the crawler through relevant Web documents to build a local focused collection. Several focused crawling algorithms are discussed and evaluated by Menczer et al. [13] and Pant et al. [15]. High-quality focused crawling is important for building vertical search engines. Such systems offer search services on Web documents of a specific topic. However, relying on focused crawlers to maintain collections of scientific articles requires the construction of a complex software infrastructure.

In the last few years, some systems that provide searching or crawling services for scientific articles have been reported in the literature. HPSearch and MOPS, described by Hoff and Mundhenk [8], are able to search for articles close to the homepages of scientists. Paper Search Engine (PaSE), proposed by On and Lee [14], makes use of citation information to locate and crawl copies of articles available throughout the Web. However, these works focus on searching for scientific articles in general. In our work, we restrict the investigation to those articles for which metadata records exist in a DL catalog but a full-text pointer is not available. In an attempt to help users in such situation, we evaluate the effectiveness of our service compared to both a specialized and a general-purpose Web search engine when used for the same task, i.e., searching for the missing full-text documents of scientific articles already cataloged in a DL. Comparative studies evaluating the effectiveness of Web search engines to satisfy general information needs are very common [1,5,6,11,12]. However, few works compare the use of generic and specialized search engines for the task considered in this article.

The use of the infrastructure provided by search engines has been beneficial in many situations. Qin et al. [16] discuss limitations of traditional focused crawling algorithms and argue that the use of meta-search can help overcome such deficiencies. They also propose that answers of queries submitted to search engines can be used to make more diverse the search space of such algorithms, which are normally limited to the content located close to the seeds selected as initial points for the crawling process. Zhuang et al. [21] investigate the feasibility of using publication metadata to guide the crawler towards author’s homepages to harvest documents that are missing from a digital library catalog. The author’s homepages are compiled based on answers obtained from queries submitted to search engines. The work by Harrison and Nelson [7] describes strategies for finding information related to pages missing from Web sites. Cached versions of the missing pages retrieved from search engines are used for generating a lexical signature—a set of terms that captures the essential information presented in the page—which is then used to find similar documents or alternative copies of the original document. This strategy is the basis of a framework that aims at preserving the information available on the Web.

Silva et al. [17] advocate taking advantage of the current content already indexed by existing search engines for finding documents missing from DL catalogs, having just the small effort of formulating appropriate queries to these search engines. They propose a process based on such approach and report an extensive experimentation of this process using metadata records of articles from the computer science field. Several strategies for query formulation are investigated by using different metadata fields while composing a query (e.g., the title or the list of authors for a given article, either as quoted
phrases or as a bag of keywords). Additionally, five search engines—including both specialized and general-purpose ones—are tested and distinct information needs are considered (e.g., users looking for the free full text only, users also interested in related material, etc.). Moreover, the effectiveness of applying different measures for filtering out potentially irrelevant results and also for re-ranking the results retrieved by each search engine is investigated, as well as the benefits of combining results from distinct search engines.

In the next section, we describe the architecture and the implementation of a fully configurable DL service inspired by the aforementioned process. In particular, we show how the generalization of this process allows for different strategies, possibly better suited for other specific scenarios, to be easily set up as different configurations for the service. In Section 4, we evaluate the effectiveness and the efficiency of our implementation with collections of metadata records of articles from two rather distinct fields, namely, computer science and biomedical and life sciences.

3. The PaMS service: architecture and implementation

3.1. Retrieval flow

The study and the process proposed by Silva et al. [17] served as a basis for our PaMS service. We generalized this process and implemented it as a software component in order to provide a fully configurable, readily deployable service for digital libraries in potentially different domains with very particular characteristics. The working scheme of our component is depicted in Fig. 1.

While interacting with the digital library interface, the user checks the metadata information of an article $q$ and requests our component to search for the corresponding full text that is missing from the DL. By regarding the fields in the article metadata record as a set of potential query arguments $A_q$, the query interface automatically generates and submits queries to one or more search engines requesting for the missing full text. Candidate results are extracted from the resulting pages of each search engine as an ordered list $C = \{r_1, r_2, \ldots, r_n\}$. Each candidate result $r_i$ is a pair $(A_r, s_j)$, where $A_r$ is a set of attributes associated to each corresponding result returned by the search engine $s_j$. The results in $C$ can be fetched in any arbitrary order (e.g., one might choose to query specialized search engines before general-purpose ones). Within the same search engine, the results are retrieved in the order they are originally ranked by that search engine. The maximum number of results allowed from each search engine can be configured, as well as the rules for generating queries from the available metadata fields and the extraction rules for building the attribute sets of each result retrieved by that engine.

Fig. 3 shows an excerpt from the XML configuration document used by our component for describing the setup of the search engines used in our evaluation, namely, Scholar and Google, as discussed in Section 4.1. As shown in the excerpt, each search engine is assigned a filter and a rank group (as discussed following). Also, they carry specifications on the enabled arguments to be used while submitting a query to them (arguments title and creator, in Fig. 3), the path expression to evaluate these arguments from a given input metadata record expressed in XML (e.g., see Fig. 2), whether they should be quoted before submitting the query, and the action URL to where the formed query should be submitted.

The setup of search engines also includes the specification of the limit number of results to be retrieved from each of them and the attributes to be extracted from each of these results (attributes title and url in Fig. 3), including regular expression-based patterns used for extracting

![Fig. 1. PaMS service architecture.](image-url)
these results and for navigating through threads of result pages. Additionally, the number of results to be checked in order for those with broken URLs to be removed can be configured (attribute check in Fig. 3). Finally, the overall efficiency of the service can be further improved by conditioning the request for results from each search engine to the absence of results from previously requested search engines after they have passed the Filter module (attribute conditioned in Fig. 3).

After candidate results are extracted from the configured search engines, the Filter module removes from the list C those results that correspond to documents of potentially little (if any) interest to the user—e.g., authors’ resumes, full text of articles that cite q but without a close relation to the subject of q, etc.—producing a filtered list F. For this, we employ a combination of measures (see below) computed over the attributes of each result and the arguments of the issued query and remove those results for which the value of this combination is below a certain threshold.

The remaining list F is then submitted to a subsequent mechanism that filters out a predetermined maximum number of unavailable results, i.e., results with a broken (HTTP status code 4xx) URL. The limitation on the number of URLs to be checked can be seen as a tradeoff between retrieval effectiveness and efficiency, since it helps removing unavailable, thus irrelevant results from the final list while incurring an additional overhead to the whole process.

After passing the Filter, the remaining results in the list F are re-ranked by the Ranker module, which generates a new list R. The Ranker aims at placing on the top of the list those documents with higher probability of satisfying the user’s needs. For such, it employs another combination of the measures described below as a function for the pairwise comparison of the results in the list. The list R is then returned to the digital library interface, which presents it to the user.

3.2. Filtering and ranking policies

The core of the Filter and Ranker modules is a set of measures that can be combined to provide conjunctive
filtering policies (i.e., each result must pass the thresholds for all defined measures—lookup exclusion, urldistance, and jaccard in Fig. 4) and multi-level ranking policies (i.e., results are compared against each other with respect to their values for the first defined measure; in case of a tie, they are compared with respect to the second defined measure, and so on—in Fig. 5, results are compared using cosine and, if necessary, lookup infrequency). Additionally, these policies can be different for each search engine or can be applied to groups of search engines. This allows, for example, for results from different search engines to be interwoven in the final ranking.

Each measure may take as input an argument \( a_q \in A_q \) of the query or an attribute \( a_r \in A_r \) of a given result retrieved for that query. As its return value, the measure combines the results from a set of functions calculated based on statistics over the defined inputs (see Fig. 1). Currently, we have implemented three targeted functions, namely, \( \text{IDF} \), \( \text{HostInfrequency} \), and \( \text{HostExclusion} \).

The \( \text{IDF} \) function stands for the inverse document frequency of a given term and represents the scarcity of this term within a collection. It is pre-computed over a defined vocabulary according to the formula:

\[
\text{IDF}(t) = \log \frac{|D|}{n_t}
\]

where \( |D| \) is the number of documents in the collection considered and \( n_t \neq 0 \) is the number of documents in which the term \( t \) appears at least once.

The \( \text{HostInfrequency} \) function is based on a pre-computed list \( I \) of hosts acknowledged to be major suppliers of full texts, sorted by the decreasing order of their relative likelihood of providing full texts. It is particularly interesting for finding documents from free sources, since most “frequent” hosts correspond to publishers that generally offer only paid access to their publications. It is defined as

\[
\text{HostInfrequency}(u) = \begin{cases} \infty & \text{if } host(u) \notin I \\ \text{rank}(host(u)) & \text{otherwise} \end{cases}
\]

where \( host \) is a function that returns the hostname part of a given URL \( u \) and \( \text{rank} \) is a function that returns the position of that hostname in the list \( I \).

The \( \text{HostExclusion} \) function works in a similar way to \( \text{HostInfrequency} \). However, instead of penalizing major suppliers in the ranking phase, it works as a blacklist by filtering out the results coming from hosts in a pre-computed list \( E \). It is defined as follows:

\[
\text{HostExclusion}(u) = \begin{cases} 1 & \text{if } host(u) \notin E \\ 0 & \text{otherwise} \end{cases}
\]
The third and last one, the VectorialDistance measure returns a distance between the vectors representing the issued query and a given result, being each component of such vectors the weighted sum of the defined functions over each term of, respectively, an argument of the query and an attribute of the result (e.g., the IDF-based vectors of the titles of both the query and the result). Currently, we have implemented two of such measures, namely the Cosine distance and the Jaccard coefficient.

This loosely coupled organization allows for the configuration of several filtering and ranking policies based on a plethora of combinations of the currently implemented measures and functions as well as eventually new ones that can be seamlessly integrated into the component implementation with little effort.

4. Experimental evaluation

In this section, we describe the underlying configuration used for evaluating our implemented service. This configuration is employed for testing the effectiveness of the service in the task of finding missing full-text documents from digital library catalogs in two rather distinct domains, namely, computer science and biomedical and life sciences. The efficiency of the implementation with respect to query response time is also investigated.

4.1. Experimental configuration

In our experiments, we employ the best strategy reported by Silva et al. [17] after a thorough investigation of several dimensions related to the task of resorting to Web search engines for finding the missing full text (or related material) of articles from the catalogs of two digital libraries in the computer science domain, namely, BDBComp and a subset of DBLP comprising publications of Brazilian researchers in proceedings of international conferences.

Among the dimensions investigated, this study evaluated the effectiveness of querying different Web search engines (both specialized and general-purpose ones) with different query formulation strategies based on the available fields from the metadata records of cataloged articles. Additionally, different user needs were considered with respect to the content and the accessibility of the retrieved results. For instance, for the most typical scenario (referred to as the strict scenario by Silva et al. [17]), where users are only interested in the missing full-text document of a given article (as opposed to a scenario where they might also be interested in related materials) and it does not matter how this document can be accessed (i.e., free access to the document is not a requirement), Table 1 illustrates the retrieval performance of all investigated query formulation strategies across different search engines in terms of mean average precision (MAP), as reported by Silva et al. [17]. In Table 1, UT and QT stand for the unquoted and the quoted title of the articles, respectively, while FS and AS stand for the surname of the first and of all authors of the articles, respectively. The best strategy for each search engine is highlighted.

From Table 1, it can be seen that the use of the unquoted title and the surname of the first author (UT+FS) provides the best retrieval performance for all but the MSN search engine while searching for the full text of the corresponding article. Though apparently counterintuitive, the use of the unquoted rather than the quoted title presents better results since this type of query is less sensitive to title variations possibly introduced by cataloging problems (e.g., typos, misspellings, OCR errors, PDF extraction errors, etc.).

The results from Table 1 can only be used to compare different query formulation strategies for a given search engine. In order to provide a direct effectiveness comparison among search engines, Table 2 shows the retrieval performance of all search engines investigated by Silva et al. [17] for the strict scenario. Each row in this table presents the MAP performance of a given search engine while using its most effective query formulation strategy for the strict scenario. Relative gains to the search engine
Table 1  
MAP results for different query formulation strategies across different search engines.

<table>
<thead>
<tr>
<th>Query type</th>
<th>Google</th>
<th>Yahoo!</th>
<th>MSN</th>
<th>Scholar</th>
<th>CiteSeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS</td>
<td>23.5 ± 6.5</td>
<td>28.0 ± 8.8</td>
<td>27.3 ± 19.0</td>
<td>30.6 ± 8.4</td>
<td>59.4 ± 15.9</td>
</tr>
<tr>
<td>UT</td>
<td>37.9 ± 7.1</td>
<td>58.8 ± 9.7</td>
<td>25.6 ± 18.9</td>
<td>54.9 ± 9.2</td>
<td>64.3 ± 14.2</td>
</tr>
<tr>
<td>UT+FS</td>
<td>39.0 ± 7.2</td>
<td>63.8 ± 9.7</td>
<td>53.4 ± 20.6</td>
<td>71.6 ± 7.4</td>
<td>66.4 ± 13.7</td>
</tr>
<tr>
<td>UT+AS</td>
<td>36.4 ± 7.6</td>
<td>54.5 ± 10.2</td>
<td>62.5 ± 19.8</td>
<td>66.3 ± 8.1</td>
<td>57.0 ± 15.6</td>
</tr>
<tr>
<td>QT</td>
<td>32.2 ± 7.4</td>
<td>57.4 ± 10.0</td>
<td>75.0 ± 17.8</td>
<td>66.2 ± 8.3</td>
<td>55.3 ± 16.0</td>
</tr>
<tr>
<td>QT+FS</td>
<td>31.8 ± 7.4</td>
<td>54.7 ± 10.0</td>
<td>75.0 ± 17.8</td>
<td>65.9 ± 8.2</td>
<td>45.3 ± 16.5</td>
</tr>
<tr>
<td>QT+AS</td>
<td>28.8 ± 6.6</td>
<td>49.7 ± 10.2</td>
<td>64.8 ± 20.1</td>
<td>63.1 ± 8.5</td>
<td>51.6 ± 16.8</td>
</tr>
</tbody>
</table>

Table 2  
MAP results for different search engines in the strict scenario.

<table>
<thead>
<tr>
<th>Search engine</th>
<th>MAP</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scholar</td>
<td>32.6 ± 6.2</td>
<td>117.9</td>
</tr>
<tr>
<td>Google</td>
<td>15.0 ± 3.6</td>
<td>17.4</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>12.7 ± 4.2</td>
<td>241.3</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>3.7 ± 1.8</td>
<td>14.5</td>
</tr>
<tr>
<td>MSN</td>
<td>3.3 ± 2.2</td>
<td>–</td>
</tr>
</tbody>
</table>

immediately below are also shown. Bold values indicate statistically significant gains.

From Table 2, it can be observed that, for the strict scenario, Scholar significantly outperforms Google which, along with Yahoo!, significantly outperform the other tested search engines. Overall, Silva et al. [17] showed that similar conclusions also hold for other search scenarios, with the combination Scholar+Google being the most effective across different scenarios. These observations are corroborated by Walters [20] who experimentally determined the good coverage of scientific articles by Scholar when compared with other bibliographic sources for a multidisciplinary field.

In the remaining of this section, we employ the best strategy as reported by Silva et al. [17] as the basic configuration for evaluating our service. Given the metadata record of an article, a first query is submitted to Scholar using the UT+FS query formulation strategy. Candidate results are extracted into a semi-structured representation from the resulting pages produced by Scholar for the submitted query, with the relative order of results being preserved. After that, a similarity filter is applied to the extracted results by matching their titles against the original one as cataloged in the metadata record. Silva et al. [17] reported their best performance after removing results for which the Jaccard coefficient [18] between the two titles was smaller than a threshold of 0.36, which we follow in this article. If the list of filtered results is not empty, it is submitted to a re-ranking process based on the cosine distance of result titles with respect to the article title. This is necessary because Scholar’s ranking is primarily based on citation count rather than on similarity. Results for which the title has a higher similarity to the title of the desired article are prioritized. In case of a tie, URLs for which the domain part appears very frequently in the results are penalized. These are normally Web pages of digital libraries of publishers which, in most cases, provide restricted access to the full text of their articles. Finally, if this process still produces a tie, the original order as provided by Scholar is used.

In case Scholar returns no results or all of its returned results are removed by the defined filters, a second query is issued in the same previous format to Google,5 and the results it returns are treated in a similar process, but with a filter threshold of 0.23 for the Jaccard coefficient and without the re-ranking phase, since the reported experiments have shown that the original ranking provided by Google was already a good one.

4. Finding missing full texts for computer science articles

Metadata records of computer science articles were obtained directly from the catalog of BDDBComp as of May 6, 2007. This catalog comprises 4851 titles (discarded entries cataloged through the BDDBComp self-archiving service and still under review) with 11,241 distinct terms. The distribution of the number of terms per title for the BDDBComp catalog is shown in Fig. 6.

In this first experiment, we wanted to better assess the effectiveness of the best strategy reported by Silva et al. [17] for the BDDBComp collection with multi-level, non-dichotomous relevance judgments. For such, we considered three relevance levels, in decreasing order of their relative importance:

1. full text (PDF, PS, HTML, etc.) of the searched article or a direct pointer to it (e.g., a Web page or directory from where it can be downloaded);
2. material related to the searched article (but not its full text), such as related articles/theses/presentations from at least one of the authors of the searched article or even additional metadata for that article;
3. unrelated material.

These relevance levels were used to evaluate retrieved results for queries in four different pools. Such pools were

4 http://scholar.google.com
5 http://www.google.com
obtained from the BDBComp catalog in the following manner:

- random_full: 50 randomly selected articles with a corresponding full text;
- random_miss: 50 randomly selected articles without a corresponding full text;
- top_full: 50 most popular articles with a corresponding full text;
- top_miss: 50 most popular articles without a corresponding full text.

The random_full and random_miss pools were randomly selected from the entire BDBComp catalog, with all its articles having the same probability of being chosen. The top_full and top_miss pools were selected based on the BDBComp access log spanning from April 15, 2007 to May 06, 2007, with a total of 10,921 requests for 2200 unique article pages.

The results for the 200 queries in the four pools were submitted to the evaluation of 17 subjects, all undergraduate or graduate students from the Computer Science Department of the Federal University of Minas Gerais. From their evaluation, we computed the average gain curves for the top 10 rank positions based on the discounted cumulative gain (DCG) measure \[9,19\], as given by the equation

\[
\text{DCG}[i] = \begin{cases} 
G[1] & \text{if } i = 1 \\
\text{DCG}[i - 1] + \frac{G[i]}{\log_2 i} & \text{otherwise} 
\end{cases}
\]

(4)

where the gain vector \(G\) is given by

\[
G[i] = \begin{cases} 
\text{rel}(i) & \text{if } i = 1 \\
\text{G}[i - 1] + \text{rel}(i) & \text{otherwise} 
\end{cases}
\]

(5)

where \(\text{rel}(i)\) corresponds to the relevance level of the result at the \(i\)-th rank position—in our case, 0, 1, or 2. DCG expresses the cumulative gain the user obtains by examining the retrieved results up to a given rank position. It employs a rank-based log-discount factor: the greater the rank, the smaller the share of the document evaluation value added to the cumulated gain.

By selecting the logarithm base \(b\), sharper or smoother discounts can be computed to model varying user behaviors, i.e., the user persistence in examining long ranked lists. In our experiments, we used a logarithm with base 2.

Following, we present the results for two different configurations of our service. The first one, called PaMS-nochk, employs a URL-based filter (the URLDistance measure\(^6\)) and a similarity-based filter (the Jaccard coefficient). Besides these filters, the second configuration, called PaMS-chk, checks every retrieved result in order to filter out unavailable ones, i.e., results with a broken URL.

In Fig. 7, we present the DCG curves along with confidence intervals (\(\alpha = 0.05\)) spanning the top 10 ranked results for both the PaMS-nochk and the PaMS-chk configurations. Additionally, we include reference curves—which we call “best possible” curves, in the sense that they represent an optimal re-ranking of the results retrieved by our service given the relevance levels considered—for each of these configurations and also a theoretical upper bound, which considers a hypothetical ranking comprising only results of our top relevance level, i.e., a ranked list with relevant full-text documents only.

As we can see from Fig. 7, by filtering out broken URLs, the PaMS-chk configuration significantly improves the DCG curve over PaMS-nochk for the top 10 results. Also, our rankings for the top 10 results are not significantly different from the best possible ones for both configurations, what further attests the effectiveness of our implementation.

The graphs in Fig. 8 show the effectiveness of both PaMS configurations for each query pool individually. The discontinuities at higher ranks are due to the lack of results at these ranks for individual pools averaged separately. In all graphs but the one for the random_miss

---

\(^6\) By definition, the URLDistance measure filters out results with already known URLs; thus, our evaluation comprises only newly discovered results.
pool, it can be observed that the PaMS DCG curves behave similarly to the ones averaged over the results for the whole set of queries, as shown in Fig. 7. This actually comes from the fact that \textit{random\_miss} is the query pool that retrieves the lowest proportion of results with broken URLs (≈ 4% of all results).

A deeper analysis into each pool raises the statistics shown in Table 3. This table presents the correlation between the proportion of broken URLs among the retrieved results and the language of the articles from each pool. From the table, it can be observed that there is a strong inverse correlation (−0.77) between the proportion of articles written in English and the retrieval of results with broken URLs, i.e., the higher the proportion of articles written in English, the lower the proportion of retrieved results with broken URLs. Since BDBComp comprises a majority of articles in Portuguese and is mostly accessed by the Brazilian community, articles in English represent a higher proportion of subsets randomly chosen (\textit{random\_full} and \textit{random\_miss}) than of those selected among the most popular articles (\textit{top\_full} and \textit{top\_miss}).

In Figs. 7 and 8, the best theoretical curves can be seen as an upper bound for the whole process since they consider not only the results actually retrieved, but also others that could have eventually been retrieved in order to enhance the overall performance. Thus, it suggests that loosing the thresholds of the current filtering policies could be a direction for further improvement. This investigation, however, is beyond the scope of this article.

### 4.3. Finding missing full texts for biomedical and life sciences articles

In this second experiment, we wanted to evaluate the effectiveness of the same strategy in a different domain as
well as to perform a broader efficiency test of our implementation. For such, we performed a harvesting from the PubMed Central (PMC) OAI data provider.\(^7\) The PMC catalog snapshot we harvested comprises entries on biomedical and life sciences articles from February 27, 2001 to May 05, 2007. This interval has a total of 972,697 titles (discarded entries corresponding to errata of previously published articles) with 10,591,960 distinct terms. The distribution of terms per title for the PMC catalog is shown in Fig. 9.

Since there is no recall base available for this collection, we resorted to an automatically generated base with one known relevant result for each article to be searched, namely the URL for the full text of the article as cataloged in the PubMed Central on-line archive.\(^8\) This URL is available from the article metadata catalog. This way, we could compare our results against a known relevant result.

Using our PMC harvested catalog with nearly one million records as a population size, and limiting the sampling error to 3%, a sample size of 1066 records would be needed for a 95% confidence interval. Accordingly, we generated a random sample from our catalog with 2000 records and examined the retrieval results in detail.

For each query corresponding to an article metadata record, we calculated the reciprocal rank (RR) of its respective known relevant result. For a given query, the reciprocal rank is defined as the inverse of the rank of its first relevant result [4]. Then, we averaged the reciprocal rank over all 2000 queries, obtaining a mean reciprocal rank (MRR) of 0.41 ± 0.02 (\(\alpha = 0.05\)) for PaMS-nochk and 0.42 ± 0.02 (\(\alpha = 0.05\)) for PaMS-chk. Table 4 shows a comparison of our MRR results to those of Scholar and Google for the same records without the application of any filtering or re-ranking.

As we can see, Google presents a better MRR result than our two PaMS configurations and Scholar. On average, Google places the known relevant result (i.e., the cataloged link to the full text) at the \(\frac{1}{0.41} \approx 1.67\) rank position while our base configuration, PaMS-nochk, places that same result at the \(\frac{1}{0.42} \approx 2.44\) rank position. It is worth noticing, however, that our base configuration retrieved an average of only 3.18 results per query while Google retrieved an average of 27.07 results per query. This is mainly due to the hard filtering policies employed by the strategies used in our service in order to remove irrelevant material from the final ranked list. Also, it can be observed that the MRR results for both PaMS configurations are statistically equivalent; though it has not significantly contributed to increasing MRR, PaMS-chk further helped filtering out irrelevant results from the final list.

In order to better assess the effectiveness of the application of the re-ranking policies in our method, we performed the same comparison as before by discarding those cases in which Scholar, Google, and our PaMS configurations were not able to retrieve any relevant results. Table 5 presents the result of this evaluation with MRRs averaged over the subsets of queries for which each alternative system was able to find the known relevant result.

As shown in Table 5, our performance increases deeply if we consider only the queries for which we are able to retrieve the known relevant result, significantly outperforming Scholar’s and even Google’s performance with their respective retrieved subsets. Our MRR of 0.94 means that, on average, when we are able to retrieve the known relevant result, this result lies around the \(\frac{1}{0.94} \approx 1.06\) position of the ranking. Again, our average final list is much shorter than Google’s (≈ 8% of Google’s list size). However, it should be noted that the subsets of queries considered for each of these alternative systems are not the same. Also, Google still has a higher hit count (i.e., the number of queries for which the known result was retrieved) than all other systems (1395 out of 2000 queries). In order to provide a fairer evaluation, Table 6 shows a paired comparison of these alternatives, by averaging MRRs over the same subset for all of them, i.e., the subset of queries for which all these systems could find the known relevant result.

\(^7\) http://www.pubmedcentral.nih.gov/about/oai.html
\(^8\) http://www.pubmedcentral.nih.gov

![Fig. 9. Title length distribution for PMC.](image-url)
Table 6
MRR over the subset of successful queries for all alternatives.

<table>
<thead>
<tr>
<th></th>
<th>Scholar</th>
<th>Google</th>
<th>PaMS-nochk</th>
<th>PaMS-chk</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR (paired)</td>
<td>0.72</td>
<td>0.93</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>$\Delta (\alpha = 0.05)$</td>
<td>0.03</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Num. queries</td>
<td>709</td>
<td>709</td>
<td>709</td>
<td>709</td>
</tr>
<tr>
<td>Num. results</td>
<td>15,357</td>
<td>20,582</td>
<td>1628</td>
<td>1591</td>
</tr>
<tr>
<td>Avg. results</td>
<td>21.66</td>
<td>29.03</td>
<td>2.30</td>
<td>2.24</td>
</tr>
</tbody>
</table>

Table 7
MRRs for inverse-order strategy.

<table>
<thead>
<tr>
<th></th>
<th>Scholar, then Google</th>
<th>Google, then Scholar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\Delta}{\Delta}$</td>
<td>Scholar, then Google</td>
<td>Google, then Scholar</td>
</tr>
<tr>
<td>MRR (global)</td>
<td>0.41</td>
<td>0.42</td>
</tr>
<tr>
<td>$\Delta (\alpha = 0.05)$</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Num. queries</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Num. results</td>
<td>6352</td>
<td>6171</td>
</tr>
<tr>
<td>Avg. results</td>
<td>3.18</td>
<td>3.09</td>
</tr>
<tr>
<td>MRR (unpaired)</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>$\Delta (\alpha = 0.05)$</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Num. queries</td>
<td>877</td>
<td>886</td>
</tr>
<tr>
<td>Num. results</td>
<td>2078</td>
<td>2069</td>
</tr>
<tr>
<td>Avg. results</td>
<td>2.37</td>
<td>2.34</td>
</tr>
<tr>
<td>MRR (paired)</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>$\Delta (\alpha = 0.05)$</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Num. queries</td>
<td>709</td>
<td>709</td>
</tr>
<tr>
<td>Num. results</td>
<td>1628</td>
<td>1591</td>
</tr>
<tr>
<td>Avg. results</td>
<td>2.30</td>
<td>2.24</td>
</tr>
</tbody>
</table>

In this case, our results are statistically equivalent to those of Google, with a nearly 31% gain over Scholar. Though they represent significant improvements in ranking the retrieved results, these gains cannot be generalized, since we are dealing with a quite small recall base for each query [2]. However, they are valuable to attest the effectiveness of our service for finding full-text documents missing from the catalog of a DL in a domain different from that for which it was originally designed. Also, they suggest that inverting the order in which the two search engines used in the original strategy are queried could be a direction for producing a better search strategy for the biomedical and life sciences domain. In order to investigate this hypothesis, we performed the same evaluation using this suggested strategy, i.e., Scholar is only requested in case no result from Google passes the filtering module. Table 7 compares the retrieval performance of both PaMS configurations using this strategy to their counterparts that use the original one. As shown in Fig. 10, there are two main time-consuming phases in the execution of PaMS-nochk: (1) the loading phase, when the pre-computed lists used for statistics-based calculations are loaded to memory, and (2) the retrieval phase, when result pages are fetched from the selected search engines and the results they list are extracted. The filtering task represents a third time-consuming phase for PaMS-chk, when all the results that passed the defined filtering policies are checked in order to remove those with broken URLs. The loading phase can be easily optimized by running the service as a daemon, so that it loads only once to serve several requests. Optimizing the retrieval phase, on the other hand, has direct implications on the effectiveness of the system with respect to the quality of the results it retrieves. For example, we could set a lower limit on the number of top results extracted per search engine, but this could imply in not extracting potentially relevant results that might have

4.4. Efficiency issues

In this section, we discuss results on the efficiency of our implementation with respect to the time spent in the retrieval task. This study is intended to show that our implementation is feasible and can be deployed to real DLs (whether or not using our suggested strategy) and also to serve as a basis for a more comprehensive assessment in face of several possible different strategies.

Here we briefly describe the behavior of our implementation running with both configurations of the already described search strategy, namely PaMS-nochk and PaMS-chk. According to the process described in Section 3, the total time spent in the process is directly influenced by the number of results retrieved by the service, which, in turn, can be fully determined by the underlying search strategy being used. For example, a search strategy may specify, among other things, the number of search engines to which issue requests, the limit on the number of results to be retrieved from each search engine, the sophistication of the filtering and ranking policies to be applied to the retrieved results, the maximum number of results to be checked in order to remove those with broken URLs, etc. Fig. 10 shows the cumulative time (in milliseconds) spent along a complete execution flow of both configurations of our service, averaged over 2000 independent executions performed to retrieve the results for the sample pool used for the analysis in Section 4.3. For this experiment, we used a Dual Intel Xeon 3 GHz with 2 MB of cache, 4 GB of RAM and a 67 GB SATA hard disk running on a Fast Ethernet LAN.

As shown in Fig. 10, there are two main time-consuming phases in the execution of PaMS-nochk: (1) the loading phase, when the pre-computed lists used for statistics-based calculations are loaded to memory, and (2) the retrieval phase, when result pages are fetched from the selected search engines and the results they list are extracted. The filtering task represents a third time-consuming phase for PaMS-chk, when all the results that passed the defined filtering policies are checked in order to remove those with broken URLs. The loading phase can be easily optimized by running the service as a daemon, so that it loads only once to serve several requests. Optimizing the retrieval phase, on the other hand, has direct implications on the effectiveness of the system with respect to the quality of the results it retrieves. For example, we could set a lower limit on the number of top results extracted per search engine, but this could imply in not extracting potentially relevant results that might have
been ranked lower. For PaMS-nochk, the time spent on the filtering and ranking phases is negligible. This is because often either few results are extracted or few results pass the filtering thresholds. For PaMS-chk, however, the mechanism for filtering out results with broken URLs incurs the overhead of making one additional request for each URL to be checked. Even though often only a few results pass the URL and the similarity-based filters, these individual requests may balance or even exceed the number of requests made to extract the results from the selected search engines, hence the importance of setting a low limit on the number of results to be checked by this mechanism.

For reasonably useful configurations, i.e., those aimed at retrieving highly relevant results in a short time frame, we can expect that the overall execution time of PaMS-nochk be always dominated by the time spent on the retrieval phase (for PaMS-chk, a constant proportional to the limit set on the number of URLs to be checked should be added). Fig. 11 illustrates the total execution times of PaMS-nochk (in milliseconds) along with their confidence intervals ($\alpha = 0.05$) averaged over the executions that fetched a total number of pages in the range 1–12. As we can see from the graphs, there is a tendency of growth as the total number of fetched pages increases. The graph also shows a high variability in execution times, particularly for those executions that fetched 8 or more pages, which were quite few in practice. On average, our system took $2439.52 \pm 367.53$ (at $\alpha = 0.05$) milliseconds to fetch each page. An optimization to this could be to employ a caching mechanism in order to reduce the number of pages to be fetched.

Overall, in face of the benefits it provides as a value-added service for DL users in potentially different domains and despite the time spent querying third-party search engines—a problem faced by meta-search engines in general—the above results attest the feasibility of PaMS for running under real scenarios.

5. Conclusions and future work

In this article, we presented PaperMetaSearch (PaMS), a novel component-based service for finding full-text documents missing from digital library catalogs. Since this problem affects DLs in different domains, each one with particular and distinct characteristics, we started by not constraining our service to a pre-defined, hard-coded search strategy. Instead, its implementation was driven with flexibility in mind, in order to allow its deployment to existing systems from different fields with little effort. This flexibility is achieved by the implementation of a parameterized search strategy, which allows the customization of several dimensions involved in the retrieval task, including the selection of the search engines to be queried for the missing documents and, for each selected search engine, the input metadata fields to be employed as arguments to submitting queries to it, the number of results to be retrieved from it, the attributes to be extracted for each of its retrieved results, and the filtering and ranking policies to be applied to these results.

We presented two experiments aimed at assessing the effectiveness of our service with two different configurations of a suggested strategy in two rather distinct fields, namely computer science and biomedical and life sciences. These experiments evaluated the retrieval effectiveness of PaMS using different approaches. For the computer science field, we carried out a smaller but more comprehensive study on our results through the use of three-level relevance judgments and the evaluation of the gain perceived by the users while inspecting these results. We also evaluated the effectiveness of a post-filtering mechanism for removing results with broken URLs and investigated its effectiveness in four distinct query pools. For the biomedical and life sciences field, we performed an automatic effectiveness evaluation using a representative sample from a large catalog based on a known relevant result for each record in this catalog. Additionally, we investigated the effectiveness of employing a slightly modified, more appropriate version of the suggested strategy for this domain. In both cases, our results seemed quite promising and suggested directions for improvements of the employed search strategies. We also discussed some issues concerning the efficiency of our implementation with respect to the time spent on the
retrieval task, which is dominated by the time spent on fetching result pages from the defined search engines. We claimed that the number of results retrieved—which is directly controlled by the underlying strategy—might represent a tradeoff between effectiveness and efficiency.

For future work, we plan to employ the implemented service infrastructure to investigate new search strategies for the domains considered in this work and eventually others. Also, we plan to improve some points of our implementation aiming at enhancing its effectiveness (e.g., by employing stemming and stopword removal techniques) and efficiency (e.g., by extending the query interface to issue requests to different search engines in parallel and by implementing a cache mechanism for fetched pages). Finally, we have just deployed the service to BDBComp in order to evaluate its usage in a real environment.

Acknowledgments

This research was partially funded by the MCT/CNPq/CT-INFO Project 5S-VQ (Grant number 55.1013/2005-2) and by the FAPEMIG Project 5S+ (Grant number CEX-1098/05/Universal). The second and third authors also acknowledge their individual research grants from CNPq.

References