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

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 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 engine. We also discuss the results of an experimental evaluation of our service with respect to its effectiveness and efficiency with collections from two distinct fields: computer science and biomedical and life sciences.

Key words: Digital libraries, Meta-search, Component-based software development

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)

* Corresponding author.

Email addresses: rodrygo@dcc.ufmg.br (Rodrygo L.T. Santos), allan@dcc.ufmg.br (Allan J.C. Silva), hugocomp@dcc.ufmg.br (Hugo S. Santos), laender@dcc.ufmg.br (Alberto H.F. Laender), mgoncalv@dcc.ufmg.br (Marcos André Gonçalves).

Preprint submitted to Elsevier 9 June 2009
of scientific articles. 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 BDBComp – Brazilian Digital Library of Computing\(^1\) and DBLP – Digital Bibliography & Library Project\(^2\).

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, i.e., 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 the returned results to check whether any of them corresponds to the full texts wanted.

In this article, we propose and evaluate a service called PaperMetaSearch (PaMS) that automatizes this process thus diminishing the user’s effort while trying to maximize the results using the best possible strategy for this specific task. We also describe the architecture of a software component that implements this service and that can be applied and reused in several digital libraries.

Our PaMS component is inspired in a process investigated by Silva (2007) that demonstrated to be quite effective in this specific task for collections in the computer science field. This process uses information present in the metadata record of a given article to submit queries to one or more search engines requesting for the respective full-text document. In this article, we employ a combination of the two best-performing search engines as determined by Silva (2007), namely Scholar\(^3\) and Google\(^4\). 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. Following, before the remaining results are shown to the user in the DL interface, they are re-ranked to prioritize those documents with higher odds of satisfying the user’s needs (e.g., documents coming from free sources).

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 li-

---

\(^1\) [http://www.lbd.dcc.ufmg.br/bdbcomp/](http://www.lbd.dcc.ufmg.br/bdbcomp/)

\(^2\) [http://www.informatik.uni-trier.de/~ley/db/](http://www.informatik.uni-trier.de/~ley/db/)

\(^3\) [http://scholar.google.com](http://scholar.google.com)

\(^4\) [http://www.google.com](http://www.google.com)
brary in which the component will be employed. Besides its description, we also 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\(^5\), a free digital archive of biomedical and life sciences literature. 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.

Summarizing, 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 digital libraries rely mostly on focused crawlers. As introduced by Chakrabarti et al. (1999), 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. (2004) and Pant et al. (2004). 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 (2001), are able to search for articles close to the homepages of scientists. Paper Search Engine (PaSE), proposed by On and Lee (2004), 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, aiming to build a service to help DL users in such situation. Then, we evaluate the effectiveness of querying a specialized search engine to look for the missing full texts of scientific articles already cataloged in a DL compared with applying the same strategy to a general-purpose Web search engine which indexes all kinds of document. Comparative studies evaluating the effectiveness of generic search engines to satisfy general information needs are very common (Bharat and Broder, 1998; Chu and Rosenthal, 1996; Gordon and Pathak, 1999; Lawrence and Giles, 1998, 1999). 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 have been beneficial in many situations. Qin et al. (2004) 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. (2005) 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 (2006) 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 (2007) advocates 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. It proposes a process based on such approach and reports an extensive experimentation of this process using metadata records of articles from the computer science field. This experimentation aimed at obtaining the best configuration for the service for finding full texts missing from a digital library. Several strategies for query submission were evaluated (e.g., with different metadata fields, using quoted and unquoted queries), using five search engines.
including both specialized and general-purpose ones – and considering users with different and distinct requirement levels (e.g., users looking for the free full text only, users also interested in related material, etc.). Moreover, strategies based on the combination of results from different search engines and re-ranking policies were tested and, as shown, they were able to significantly improve the overall quality of the retrieved results when compared with the results from a single search engine. The best strategy reported by Silva (2007) is employed here as the baseline configuration for evaluating our component.

In this article, we describe a software component for finding the full text of articles cataloged in a DL. We evaluated the effectiveness and efficiency of our component by conducting experiments with collections of metadata records of articles from two fields known to be well represented on the Web. We noticed that, corroborating the generalization of our approach, recent studies (Qin et al., 2004; Walters, 2007) have proved that search engines are also covering the content of other areas of knowledge.

3 The PaMS Service: Architecture and Implementation

3.1 Retrieval Flow

The architecture of the service proposed by Silva (2007) served as a basis for ours. We extended it in order to provide a fully configurable service for finding full-text documents missing from 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 \( C \) are fetched in an order that prioritizes those coming from search engines with higher odds of retrieving the full text of scientific papers. 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 two search engines used in this work, namely Scholar and Google. As shown in the excerpt, each 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 the 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 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 request for the results from each search engine can be conditioned to the absence of results from previously requested search engines after they have passed the Filter module.

The Filter removes from the list \( C \) the candidate results that correspond to documents that are potentially of little (if any) interest to the user – such as authors’ resumes, full text of articles that cite \( q \) but without a close relation.
Fig. 3. Search engines setup from the component XML configuration document to the subject of q, etc. – producing a filtered list $F$. Furthermore, the removal of results from $C$ reduces the processing costs of the subsequent steps in the overall process. 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 defined.

```
<filter id="0">
  <measure type="lookup" limit="1">
    <factors>
      <statistics type="exclusion" weight="1" source="res/bdbcomp.hex"/>
    </factors>
    <inputs>
      <attribute ref="url"/>
    </inputs>
  </measure>
  <measure type="urldistance" limit="1">
    <inputs>
      <argument ref="url"/>
      <attribute ref="url"/>
    </inputs>
  </measure>
  <measure type="jaccard" limit="0.36">
    <factors vocabulary="res/bdbcomp-title.voc">
      <statistics type="idf" weight="1" source="res/bdbcomp-title.idf"/>
    </factors>
    <inputs>
      <argument ref="title"/>
      <attribute ref="title"/>
    </inputs>
  </measure>
</filter>
```

Fig. 4. Filtering excerpt from the component XML configuration document

```
<ranking id="0">
  <measure type="cosine">
    <factors vocabulary="res/bdbcomp-title.voc">
      <statistics type="idf" weight="1" source="res/bdbcomp-title.idf"/>
    </factors>
    <inputs>
      <argument ref="title"/>
      <attribute ref="title"/>
    </inputs>
  </measure>
  <measure type="lookup">
    <factors>
      <statistics type="infrequency" weight="1" source="res/bdbcomp.hin"/>
    </factors>
    <inputs>
      <attribute ref="url"/>
    </inputs>
  </measure>
</ranking>
```

Fig. 5. Ranking excerpt from the component XML configuration document
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 returned for that query and return the weighted sum of a set of functions calculated based on statistics over the defined inputs (see Fig. 1). Currently, we have implemented three targeted functions, namely $IDF$, $HostInfrequency$, and $HostExclusion$.

The $IDF$ function stands for the inverse document frequency of a given term and represents the scarcity of that term within a collection. It is pre-computed over a defined vocabulary according to the formula:

$$IDF(t) = \log \frac{|D|}{n_t}$$

(1)

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 $HostInfrequency$ 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:

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

(2)

where $host$ is a function that returns the hostname part of a given URL $u$ and $rank$ is a function that returns the position of that hostname in the list $I$.

The $HostExclusion$ function works in a similar way to $HostInfrequency$. 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:

$$HostExclusion(u) = \begin{cases} 
1, & \text{if } host(u) \notin E \\
0, & \text{otherwise}
\end{cases}$$

(3)

The values returned by each of these functions can be combined as a weighted sum by three general-purpose measures, namely $Lookup$, $ScalarDistance$, and $VectorialDistance$. The $Lookup$ measure returns the weighted sum of the defined functions for an attribute of a given result (e.g., the $HostInfrequency$ value of the URL of a given result).
The ScalarDistance measure returns a single value based on the direct comparison of the defined inputs (e.g., the cataloged URL of the query – for those articles with an already cataloged URL – and the extracted URL of a given result). Currently, we have implemented only one measure of this kind, called URLDistance, which can be used in a complementary way to the HostExclusion-based Lookup measure, by filtering out results with known URLs. It is useful, for instance, for not retrieving results with URLs already cataloged in the DL running the service.

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 Results

The experimental study conducted by Silva (2007) with computer science articles reports the following process as the most effective for finding missing full-text documents. Given the metadata record $q$ of an article, a Query Interface automatically generates and submits to Google Scholar a first query containing the unquoted title and the surname of the first author of the article as contained in $q$. 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). Candidate Results are then extracted from the resulting pages and compiled in an ordered list $C = [r_1, r_2, \ldots, r_n]$. Using the Filter, Candidate Results for which the title $t_r \in A_r$ is not similar to the title $t_q \in A_q$ of the desired article are removed, yielding a filtered list $F$. The best results were experimentally obtained by removing results for which the Jaccard coefficient $J(t_r, t_q)$ between the two titles was smaller than a threshold of 0.36.

If the list $F$ of filtered results is not empty, it is submitted to the Ranker that executes a re-ranking based on (1) the Cosine distance of result titles with respect to the query title, and (2) the HostInfrequency between any pair of
result URLs. This is necessary because Scholar’s ranking is primarily based on citation count rather than on similarity. Results for which the title \( t_s \) have a higher similarity with the title \( t_q \) of the desired item are prioritized. In case of a tie, result 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. The new ordered list \( R \) is then presented to the user.

In the case of an empty list \( F \), i.e., if there is no answer to show to the user, a second query is issued in the same previous format to Google, and the results retrieved are treated in a similar process, but with a filter threshold of 0.23 and without the re-ranking phase, since the reported experiments have shown that the original ranking by Google was already a good one. The choice of first going to Scholar is justified by its better coverage of scientific articles when compared to Google, as experimentally determined by Walters (2007).

For our experiments, we employed the best strategy described above to find full texts missing from digital libraries in two distinct domains, namely computer science and biomedical and life sciences.

4.1 Finding Missing Full Texts for Computer Science Articles

Metadata records of computer science articles were obtained directly from the catalog of BDBComp as of May 6, 2007. This catalog comprises 4,851 titles (discarded entries cataloged through the BDBComp self-archiving service and still under review) with 11,241 distinct terms. The distribution of the number of terms per title for the BDBComp catalog is shown in Fig. 6.

In this first experiment, we wanted to better assess the effectiveness of the best strategy reported by Silva (2007) for the BDBComp collection with multi-level, non-dichotomous relevance judgments. For such, we considered three relevance levels, in decreasing order of their relative importance:

2. Full text (PDF, PS, HTML, etc.) of the searched article or a direct pointer to it;
1. 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;
0. Unrelated material.

These relevance levels were used to evaluate retrieved results for queries in four different pools. Such pools were obtained from the BDBComp catalog in
the following manner:

- **random**\textsubscript{full}: 50 randomly selected articles with a corresponding full text URL;
- **random**\textsubscript{miss}: 50 randomly selected articles without a corresponding full text URL;
- **top**\textsubscript{full}: 50 most frequently accessed articles with a corresponding full text URL;
- **top**\textsubscript{miss}: 50 most frequently accessed articles without a corresponding full text URL.

The **random**\textsubscript{full} and **random**\textsubscript{miss} pools were randomly selected from the entire BDBComp catalog, with all its articles having the same probability of being chosen. The **top**\textsubscript{full} and **top**\textsubscript{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 2,200 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) (Järvelin and Kekäläinen, 2000; Voorhees, 2001), which expresses the cumulative gain the user obtains by examining the retrieved results up to a given rank position. DCG 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, 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.

![Discounted Cumulative Gain](image)

**Fig. 7.** Discounted \((\log_2)\) cumulative gain curves at ranks 1-10 for levels 0-2

As we can see from the graphic, 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 graphics 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 graphics but the one for the random miss pool, it can be observed that the PaMS DCG curves behave similarly to the ones averaged over the results

\(^6\) By definition, the URLDistance measure filters out results with already known URLs; thus, our evaluation comprises only newly discovered results.
Fig. 8. Discounted ($\log_2$) cumulative gain curves for individual query pools for the whole set of queries, as shown in Fig. 7. This actually comes from the fact that `random_miss` is the query pool that retrieves the lowest proportion of results with broken URLs ($\approx 4\%$ of all results). A deeper analysis into each pool raises the statistics shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>Broken</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>random_miss</code></td>
<td>0.04</td>
<td>0.34</td>
</tr>
<tr>
<td><code>random_full</code></td>
<td>0.14</td>
<td>0.36</td>
</tr>
<tr>
<td><code>top_miss</code></td>
<td>0.25</td>
<td>0.22</td>
</tr>
<tr>
<td><code>top_full</code></td>
<td>0.25</td>
<td>0.04</td>
</tr>
<tr>
<td><strong>correlation</strong></td>
<td><strong>-0.77</strong></td>
<td></td>
</tr>
</tbody>
</table>

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 written in Portuguese and is mostly accessed by the Brazilian community, articles in English represent a higher proportion of subsets randomly chosen (random\textsubscript{miss} and random\textsubscript{full}) than of those selected among the most popular articles (top\textsubscript{miss} and top\textsubscript{full}) in this catalog.

At the top of all graphics 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 lowering the thresholds of the current filtering policies could be a direction for further improvement. This investigation, however, is beyond the scope of this article.

\section*{4.2 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\cite{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.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure9.png}
\caption{Title length distribution for PMC}
\end{figure}

\footnote{http://www.pubmedcentral.nih.gov/about/oai.html}
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. 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 1,066 records would be needed for a 95% confidence interval. Accordingly, we generated a random sample from our catalog with 2,000 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. The reciprocal rank is defined as the inverse of the rank of the first relevant result for a query (Chowdhury and Soboroff, 2002). Then, we averaged the reciprocal rank over all 2,000 queries, obtaining a mean reciprocal rank (MRR) of \(0.41 \pm 0.02\) (\(\alpha = 0.05\)) for PaMS-nochk and \(0.42 \pm 0.02\) (\(\alpha = 0.05\)) for PaMS-chk. Table 2 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.

<table>
<thead>
<tr>
<th>MRR over all queries</th>
<th>Scholar</th>
<th>Google</th>
<th>PaMS-nochk</th>
<th>PaMS-chk</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR (global)</td>
<td>0.28</td>
<td><strong>0.60</strong></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>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>num. queries</td>
<td>2,000</td>
<td>2,000</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>num. results</td>
<td>45,758</td>
<td>54,144</td>
<td>6,352</td>
<td>6,171</td>
</tr>
<tr>
<td>avg. results</td>
<td>22.88</td>
<td>27.07</td>
<td>3.18</td>
<td>3.09</td>
</tr>
</tbody>
</table>

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 1/0.60 \(\approx 1.67\) rank position while our base configuration, PaMS-nochk, places that same result at the 1/0.41 \(\approx 2.44\) 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

\[^{8}\text{http://www.pubmedcentral.nih.gov}\]
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 3 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.

Table 3
MRR over the hits for each alternative

<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 (unpaired)</td>
<td>0.71</td>
<td>0.86</td>
<td><strong>0.94</strong></td>
<td><strong>0.94</strong></td>
</tr>
<tr>
<td>$\Delta$ ($\alpha = 0.05$)</td>
<td>0.03</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>num. queries</td>
<td>799</td>
<td>1,395</td>
<td>877</td>
<td>886</td>
</tr>
<tr>
<td>num. results</td>
<td>17,291</td>
<td>39,464</td>
<td>2,078</td>
<td>2,069</td>
</tr>
<tr>
<td>avg. results</td>
<td>21.64</td>
<td>28.29</td>
<td>2.37</td>
<td>2.34</td>
</tr>
</tbody>
</table>

As shown in Table 3, 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 $1/0.94 \approx 1.06$ position of the ranking. Again, our average final list is much shorter than Google’s ($\approx 8\%$ 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 (1,395 out of 2,000 queries). In order to provide a fairer evaluation, Table 4 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.

Table 4
MRR over the hits 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><strong>0.94</strong></td>
<td><strong>0.94</strong></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>1,628</td>
<td>1,591</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>
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 (Buckley and Voorhees, 2004). However, these results are valuable to attest the effectiveness of our service for finding full-text documents missing from the catalog of a digital library 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 employed 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 5 compares the retrieval results for both PaMS configurations using this strategy to their counterparts that use the original strategy.

Table 5
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></td>
<td>PaMS-nochk</td>
<td>PaMS-chk</td>
</tr>
<tr>
<td>MRR (global)</td>
<td>0.41</td>
<td>0.42</td>
</tr>
<tr>
<td>∆ (α = 0.05)</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>num. queries</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>num. results</td>
<td>6,352</td>
<td>6,171</td>
</tr>
<tr>
<td>avg. results</td>
<td>3.18</td>
<td>3.09</td>
</tr>
<tr>
<td>MRR (unpaired)</td>
<td><strong>0.94</strong></td>
<td><strong>0.94</strong></td>
</tr>
<tr>
<td>∆ (α = 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>2,078</td>
<td>2,069</td>
</tr>
<tr>
<td>avg. results</td>
<td>2.37</td>
<td>2.34</td>
</tr>
<tr>
<td>MRR (paired)</td>
<td><strong>0.94</strong></td>
<td><strong>0.94</strong></td>
</tr>
<tr>
<td>∆ (α = 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>1,628</td>
<td>1,591</td>
</tr>
<tr>
<td>avg. results</td>
<td>2.30</td>
<td>2.24</td>
</tr>
</tbody>
</table>

From Table 5, it can be observed that the inverse-order strategy significantly outperforms the original one only in the global scenario (i.e., with MRRs averaged over the whole set of queries), producing worse results in the unpaired
scenario (i.e., when each alternative system disregards the queries for which it could not retrieve the known relevant result) and not significantly differing from the original strategy in the paired scenario (i.e., when considering the subset of queries for which all alternatives could retrieve the known result). It can also be observed that the new strategy increases the hit count for both PaMS configurations in nearly 60%. Once again, it is worth mentioning that although these gains are not significant with respect to the values obtained when querying Google purely, the filtering policies employed by this new strategy as well as the original one can produce quite shorter rankings than those produced by Google and even Scholar individually.

4.3 Efficiency Issues

In this section, we discuss experimental 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 efficiency 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 2,000 independent executions performed to retrieve the results for the sample pool used for the analysis in Section 4.2. For this experiment, we used a Dual Intel Xeon 3GHz with 2MB of cache, 4GB of RAM and a 67GB 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 results retrieved per search engine, but this could imply in not retrieving potentially relevant results that would be retrieved otherwise. 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 to 12. As we can see from the graphics, there is a tendency of growth as the total number of fetched pages increases. The graphic 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 $2,439.52 \pm 367.53$ ($\alpha = 0.05$) 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.

![Graph showing average execution time (ms) per number of pages fetched.](image)

Fig. 11. Average execution time (ms) per number of pages fetched

Overall, in face of the benefits it provides as a value-added service for digital library 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 dis-
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 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.

6 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 fourth and fifth authors also acknowledge their individual research grants from CNPq.

References

Buckley, C., Voorhees, E. M., 2004. Retrieval evaluation with incomplete in-
Science, Federal University of Minas Gerais, Belo Horizonte, MG, Brazil, in Portuguese.

