Finding Algorithms in Scientific Articles

Sumit Bhatia[†], Prasenjit Mitra[†*] and C. Lee Giles[†*]
[†]Department of Computer Science and Engineering, [*]College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA-16802, USA
sumit@cse.psu.edu, {pmitra,giles}@ist.psu.edu

ABSTRACT

Algorithms are an integral part of computer science literature. However, none of the current search engines offer specialized algorithm search facility. We describe a vertical search engine that identifies the algorithms present in documents and extracts and indexes the related metadata and textual description of the identified algorithms. This algorithm specific information is then utilized for algorithm ranking in response to user queries. Experimental results show the superiority of our system on other popular search engines.

Categories and Subject Descriptors:H.3.3[Information Search And Retrieval]Information Search and Retrieval – Search Process

General Terms: Algorithms, Experimentation, Design.

Keywords: Algorithms, Pseudocodes, Vertical Search Engines.

1. INTRODUCTION

Algorithms are ubiquitous in Computer Science literature and offer a precise methodology to solve problems. From sorting a few numbers to ranking billions of web pages, from Human Genome project to Economics and Internet, algorithms have influenced each and every aspect of human life [2]. Researchers are busy improving the current algorithms as well as developing new algorithms for new and unsolved problems. The current state-of-the-art search engines however, are not optimized to search for algorithms. They do not distinguish between documents that contain an algorithm and those that do not. As a result, the search results contain a variety of unwanted, irrelevant documents. For example, a query to search for algorithm for *SVM* will return lots of documents that contain *SVM* in them (application papers using SVM etc.) even though they do not offer any details about the algorithmic aspects of SVMs. Moreover, due to inappropriate ranking schemes the relevant documents might not show up in the top results.

In this paper, we describe a vertical search engine to search for algorithms in scientific documents. It first analyzes a document to check for the presence of an algorithm. If an algorithm is found, the document text is further analyzed to identify sentences that describe the algorithm. In addition, algorithm specific metadata from the document is also extracted and indexed. This algorithm specific information is then utilized to compute the relevance of algorithms to a given user query and the algorithms are presented in decreasing order of their relevance to the user.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

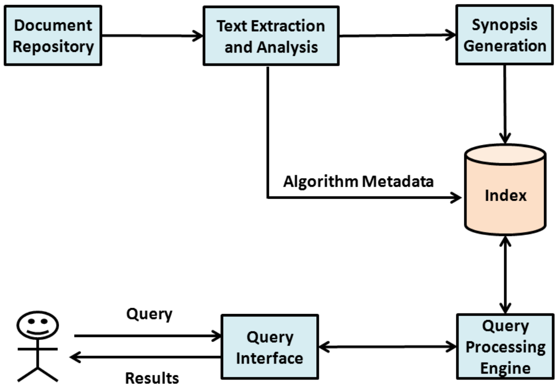


Figure 1: Architecture of the proposed system.

2. SYSTEM ARCHITECTURE

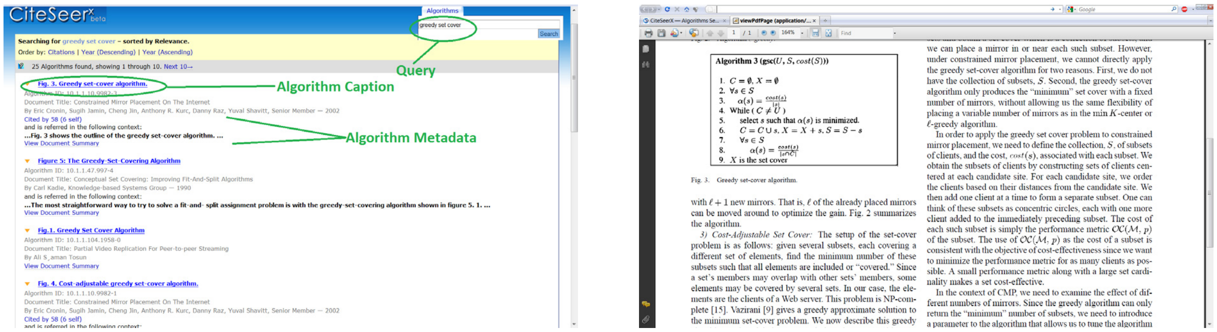
The main components of the proposed system are illustrated in Figure 1 and are described in more detail in the following sections.

2.1 Algorithm Extraction and Indexing

We use the collection of about 1.33 million computer science related documents in CiteSeer^X[1]. All the documents in the collection are in PDF/PostScript format and therefore need to be converted into text format for any further analysis. We experimented with a variety of text extraction tools available and found the performance of PDFTextStream[2] to be suitable for our needs. It performed best at preserving the sequence of text streams in the order they appeared in the document, especially for documents in double column format that are common in scientific literature.

The document text is then analyzed to check if the document contains an algorithm. Scientific documents in general have a well defined structure. Often algorithms/pseudocodes are described in the form of a stand alone Text-Box, Figure or Table, along with an associated caption and algorithm number. This *algorithm number* is then used to refer to the algorithm in the running text of the document. We call the sentences referring to the algorithm as a *reference sentences*. We can utilize these captions and reference sentences to check for the presence of algorithms in a document. If an algorithm is present, the document text is then further processed to extract the associated *synopsis* – a set of sentences that describes the

[1] `http://citeseerx.ist.psu.edu`
[2] `http://snowtide.com/PDFTextStream`



greedy set cover

Searching for greedy set cover – sorted by Relevance.

Order by: Citations | Year (Descending) | Year (Ascending)

25 Algorithms found, showing 1 through 10. Next 10→

Fig. 3. Greedy set-cover algorithm.
Algorithm ID: 10.1.1.10.9982-3
Document Title: Constrained Mirror Placement On The Internet
By Eric Cronin, Sugih Jamin, Cheng Jin, Anthony R. Kurc, Danny Raz, Yuval Shavitt, Senior Member — 2002
Cited by 58 (6 self)
and is referred in the following context:
...Fig. 3 shows the outline of the greedy set-cover algorithm. ...
View Document Summary

Figure 5: The Greedy-Set-Covering Algorithm
Algorithm ID: 10.1.1.47.997-4
Document Title: Conceptual Set Covering: Improving Fit-And-Split Algorithms
By Carl Kadie, Knowledge-based Systems Group — 1990
and is referred in the following context:
...The most straightforward way to try to solve a fit-and- split assignment problem is with the greedy-set-covering algorithm shown in figure 5. 1. ...
View Document Summary

Fig.1. Greedy Set Cover Algorithm
Algorithm ID: 10.1.1.104.1958-0
Document Title: Partial Video Replication For Peer-to-peer Streaming
By Ali S¸aman Tosun
View Document Summary

Fig. 4. Cost-adjustable greedy set-cover algorithm.
Algorithm ID: 10.1.1.10.9982-1
Document Title: Constrained Mirror Placement On The Internet
By Eric Cronin, Sugih Jamin, Cheng Jin, Anthony R. Kurc, Danny Raz, Yuval Shavitt, Senior Member — 2002
Cited by 58 (6 self)

(a)

CiteSeerX — Algorithms Se... viewPdfPage (application/...

1 / 1 164% Find

```
Algorithm 3 (gsc($U, S, cost(S)$))

1. $C = \emptyset, X = \emptyset$
2. $\forall s \in S$
3.   $\alpha(s) = \frac{cost(s)}{|s|}$
4. While ( $C \neq U$ )
5.   select $s$ such that $\alpha(s)$ is minimized.
6.   $C = C \cup s, X = X + s, S = S - s$
7.   $\forall s \in S$
8.     $\alpha(s) = \frac{cost(s)}{|s \cap C|}$
9. $X$ is the set cover
```

Fig. 3. Greedy set-cover algorithm.

with $\ell + 1$ new mirrors. That is, ℓ of the already placed mirrors can be moved around to optimize the gain. Fig. 2 summarizes the algorithm.

3) Cost-Adjustable Set Cover: The setup of the set-cover problem is as follows: given several subsets, each covering a different set of elements, find the minimum number of these subsets such that all elements are included or "covered." Since a set's members may overlap with other sets' members, some elements may be covered by several sets. In our case, the elements are the clients of a Web server. This problem is NP-complete [15]. Vazirani [9] gives a greedy approximate solution to the minimum set-cover problem. We now describe this greedy

we can place a mirror in or near each such subset. However, under constrained mirror placement, we cannot directly apply the greedy set-cover algorithm for two reasons. First, we do not have the collection of subsets, S. Second, the greedy set-cover algorithm only produces the "minimum" set cover with a fixed number of mirrors, without allowing us the same flexibility of placing a variable number of mirrors as in the min K-center or ℓ-greedy algorithm.

In order to apply the greedy set cover problem to constrained mirror placement, we need to define the collection, S, of subsets of clients, and the cost, $cost(s)$, associated with each subset. We obtain the subsets of clients by constructing sets of clients centered at each candidate site. For each candidate site, we order the clients based on their distances from the candidate site. We then add one client at a time to form a separate subset. One can think of these subsets as concentric circles, each with one more client added to the immediately preceding subset. The cost of each such subset is simply the performance metric $\mathcal{OC}(\mathcal{M}, p)$ of the subset. The use of $\mathcal{OC}(\mathcal{M}, p)$ as the cost of a subset is consistent with the objective of cost-effectiveness since we want to minimize the performance metric for as many clients as possible. A small performance metric along with a large set cardinality makes a set cost-effective.

In the context of CMP, we need to examine the effect of different numbers of mirrors. Since the greedy algorithm can only return the "minimum" number of subsets, we need to introduce a parameter to the algorithm that allows us to tune the algorithm

(b)

Figure 2: Screenshots showing (a) results for the query *"Greedy Set Cover"* and (b) algorithm page displayed on clicking the first result.

algorithm. For further details on algorithm identification and synopsis generation, the interested reader is directed to our previous work [1]. In total, we found 270367 unique algorithms in 112836 documents in the repository. For the documents containing an algorithm, we also extract the document title, author names, publication year and page on which the algorithm is present. We adopt the tools available from the SeerSuite toolkit for this purpose[3]. All the extracted algorithms from a document and their associated metadata are then indexed using an indexer based on SOLR [4].

2.2 Query Interface

Our system offers a free text based query interface to the user and the results are returned along with the associated metadata. For each algorithm, a TF-IDF based cosine similarity score is computed between *(i)* (query, caption), *(ii)* (query, reference sentence) and *(iii)* (query,synopsis). The total similarity score for an algorithm is the sum of these three similarity scores. The algorithms are presented to the user in decreasing order of their scores thus obtained. The user interface is implemented using SeerSuite and query processing and ranking is implemented using SOLR. Figure 2(a) shows the screenshot of the result page for the query *"Greedy Set Cover"*. The top 10 algorithms for the query, along with their associated metadata are presented to the user. The algorithm caption is presented in bold and clicking on it directly takes the user to the PDF page of the concerned document on which the algorithm is present. This is illustrated in Figure 2(b).

3. EXPERIMENTS

We conducted a study using two human annotators to learn how the proposed system and other state-of-the-art search engines fare at satisfying information needs of the users looking for algorithms. We used a set of twenty queries *(eg., greedy set cover, graph isomorphism)* and tested them with Google Scholar, Google Web Search and the proposed system. For Google Web Search and Google Scholar, we added the keyword "algorithm" in hope to get more algorithms in the results. A result that returned a valid algorithm was considered as relevant. We report *precision @ 10* and *NDCG @ 10* for comparison.

We admit that this experiment can not be considered completely *fair* as the other search engines are not fine tuned to search for algorithms. However, we also note that due to the same reason, a user who generally turns to these state of the art, popular search engines to look for algorithms has to work harder to find the desired documents containing relevant algorithms. The results summarized in Table 1 demonstrate the performance gains achieved by the proposed system as compared to other popular search engines. We achieve almost double the precision as compared to other methods as well as an appreciable gain in NDCG values indicating a better ranking of results.

	Precision@10	NDCG@10
Google Web Search	0.41	0.83
Google Scholar	0.44	0.58
Proposed System	**0.81**	**0.94**

Table 1: Comparison between the proposed system and other popular search engines.

4. CONCLUSIONS AND FUTURE WORK

We described a vertical search engine to search for algorithms in digital documents. A user study demonstrated the performance gains achieved by the system as compared to other popular search engines for algorithm search task. For future work, we plan to investigate various strategies for algorithm ranking. Oftentimes, authors discuss about pros and cons of the algorithms and generally, computational complexity of algorithms is also discussed. We plan to investigate how this information can be utilized for algorithm ranking.

5. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 0535656 and 0845487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] S. Bhatia, S. Lahiri, and P. Mitra. Generating synopses for document-element search. In *CIKM '09*, pages 2003–2006, 2009.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[3] http://citeseerx.sourceforge.net/

[4] http://lucene.apache.org/solr/