An Algorithm Search Engine for Software Developers

Sumit Bhatia*, Suppawong Tuarob*, Prasenjit Mitra*† and C. Lee Giles*†
*Computer Science and Engineering
†Information Sciences and Technology
The Pennsylvania State University
University Park, PA-16802, USA
{sumit,szt5115}@cse.psu.edu, {pmitra,giles}@ist.psu.edu

ABSTRACT

Efficient algorithms are extremely important and can be crucial for certain software projects. Even though many source code search engines have been proposed in the literature to help software developers find source code related to their needs, to our knowledge there has been no effort to develop systems that keep abreast of the latest algorithmic developments. In this paper, we describe our initial effort towards developing such an algorithm search engine. The proposed system extracts and indexes algorithms discussed in academic literature and their associated metadata. Users can search the index through a *free text* query interface. The source code of proposed system, being developed as a part of a larger open source toolkit, SeerSuite, will be released in due course. We also provide directions for further research and improvements of the current system.

Categories and Subject Descriptors

H3.3 [**Information Search and Retrieval**]: Search Process; H4.0 [**Information Systems Applications**]: General

General Terms

Algorithms, Design.

Keywords

Algorithm search, pseudo-code search.

1. INTRODUCTION

Algorithms are ubiquitous in Computer Science. Efficient algorithms play an important role in many software projects. For example, an improved and efficient ranking algorithm can help improve the performance of software used for indexing and searching billions of documents. Hence, it is essential for software developers to keep abreast of latest algorithmic developments related to their projects. There has been much work in past to help software developers search for existing source code [2, 3, 9, 5], however to the best of our knowledge, no effort has been made to develop tools and techniques that can help software developers search for literature about the algorithms used in source code or in documents that describe new algorithmic developments.

SUITE '11, May 28, 2011, Waikiki, Honolulu, HI, USA

Conference	No. of Algorithms
SIGIR	75
SIGMOD	301
STOC	74
VLDB	287
WWW	142

Table 1: Number of Algorithms published in different Computer Science conferences in the last five years (2005–2009).

Here, we describe our effort in building an algorithm search engine using scientific publications as a document collection. Academic documents have been used previously as a document source for various tasks [6, 7] as they offer several advantages –*(i)* academic documents, in general, follow a structure that is easier for a machine to parse and analyze, *(ii)* they are generally peer reviewed ensuring high quality and *(iii)* they are often the best resource for knowing about the latest developments in a field. As an example, Table 1 lists the number[1] of algorithms described in some of the top Computer Science conferences in last five years. Clearly, researchers are busy developing algorithms for new problems as well as continuously improving current state-of-the-art algorithms. As such it behoves one to be able to search the academic literature to be aware of these developments.

Even though popular academic literature search engines, such as Google Scholar[2] and CiteSeer^{X}[3], offer the capability to search academic documents, these systems are not geared towards algorithm search. These systems can not discriminate between a document that contains an algorithm and a document that does not. As a result a user searching for query `shortest path` will be offered many documents that contain the words `shortest` and `path` but do not discuss algorithmic aspects of the shortest path problem.

Our proposed system analyzes a document to identify any algorithms that may be present in the document. If an al-

[1]These numbers were obtained by our parsing methods as discussed in Section 2.
[2]`http://scholar.google.com/`
[3]`http://citeseerx.ist.psu.edu`

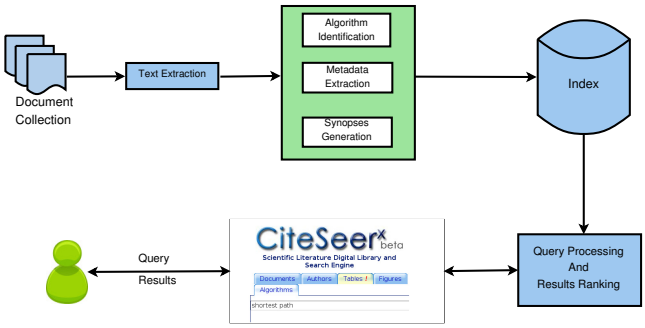

Figure 1: Architecture of the proposed system.

gorithm is found in a document, the document text is further analyzed to extract additional information about the algorithm. All the algorithms thus found and their associated metadata are indexed and made available for searching through a text query interface. For a given user query, the system utilizes evidence from multiple sources to assign relevance scores to algorithms and results are presented to the user in decreasing order of relevance.

2. PROPOSED ALGORITHM SEARCH ENGINE

Figure 1 illustrates the main components of the proposed system. All the documents in the repository are first converted into text using an pdf to text converter. The extracted text is then analyzed to find algorithms which are then indexed along with their associated meta-data. The query processing engine accepts the query from the user through the query interface, searches the index for relevant algorithms, and presents a ranked list of algorithms to the user. We now describe the various components of the proposed system in more detail.

2.1 Text Extraction

The source documents for the proposed system come from CiteSeer^X, a scientific literature digital library and search engine that contains roughly 1.4 million documents from Computer Science and related fields. Since all the documents in the collection are in the PDF or PostScript format, they need to be converted to text for any further analysis. We experimented with a variety of text extraction tools available (PDFBox[4], PDFTextStream[5], XPDF[6] and TET[7]). We found the performance of PDFTextStream 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 the double column format that is common in the computer science literature.

2.2 Algorithm Identification

After document text has been successfully extracted, it needs to be 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 define the sentences referring to the algorithm as *reference sentences*. We utilize these structural properties of scientific documents to identify the algorithms present in a document. The captions and reference sentences are used to check for the presence of algorithms by using a grammar for algorithm captions [4]. We also note our text extractor was unable to extract text from some PDF files. Improperly created PDFs and PDFs containing scanned images of document text were the major reasons for extractor's failure. In order to test the performance of our algorithm extraction method, we selected 500 PDF files at random from the repository and manually identified algorithms present in these files. Neglecting files for which the text extractor failed, we found 46 algorithms in these 500 files. Our algorithm extraction method was able to identify algorithms with a precision of 79.41% and a recall of 58.70%. In total, we found 270367 algorithms in 112836 documents in the repository.

2.3 Metadata Extraction

If an algorithm is present in a document, the document text is then further processed to extract the algorithm's *synopsis* – the set of sentences from the document that are related to the algorithm. We use a Naive Bayes classifier to identify sentences to be included in the synopsis by using a variety of content and context based features [4]. For the documents containing an algorithm, we also extract additional metadata including the document title, author names, publication year and page on which the algorithm is present. For this, we adopt the tools available from the SeerSuite toolkit[8]. All the extracted algorithms from a document and their associated metadata are then indexed using a SOLR [9] based indexer.

2.4 Query Interface and Results Ranking

The proposed system provides a free text based query interface to the user. The user interface is implemented using SeerSuite and extends CiteSeer^X's query interface. The results for a given query are presented to the user as a ranked list of algorithms along with the associated metadata. For algorithm ranking, we use a TF-IDF based cosine similarity ranking function [8, Ch. 6] found in SOLR. The total similarity score for an algorithm is a linear combination of the following three similarity scores, with an equal weight given to each component.

1. Similarity between user query and algorithm caption
2. Similarity between user query and algorithm's reference sentences
3. Similarity between user query and algorithm's synopsis

The algorithms are presented to the user in decreasing order of their scores. Experiments comparing the performance of our proposed system with other state-of-the-art search engine systems have shown superiority of our approach in terms of precision and ranking performance. We selected a set of 20 popular algorithms as test queries (e.g.

[4] http://incubator.apache.org/pdfbox/
[5] http://snowtide.com/PDFTextStream
[6] http://www.foolabs.com/xpdf/about.html
[7] http://www.pdflib.com/products/tet/
[8] http://citeseerx.sourceforge.net/
[9] http://lucene.apache.org/solr/

Figure 2: Screenshot showing results for the query *"shortest path"*. Along with search results, associated metadata is also shown to the user.

topological sort, breadth first search etc.) and tested them with our proposed system, Google Scholar and Google Web Search. A returned result page was considered as relevant if it contained a valid algorithm/pseudo-code. The relevance judgments were provided by two human evaluators not associated with the project. Our proposed system achieves a precision of 81% at top 10 ranks as compared to 41% and 44% achieved by Google Web Search and Google Scholar, respectively.

2.5 An Example Search Session

Figure 2 shows the screenshot of the result page for the query `shortest path`. The top 10 algorithms for the query, along with their associated metadata are presented to the user. Note that the results returned to the user provide a good coverage of a variety of shortest path algorithms such as the heuristic algorithm for shortest path, the Berge shortest path algorithm, in addition to the standard shortest path algorithm. The algorithm caption is presented in bold and clicking on it directly takes the user to the PDF page of the related document in which the algorithm is present. This is illustrated in Figure 3.

3. DIRECTIONS FOR FUTURE WORK

Despite encouraging initial results, there are several aspects that need further improvements and are the focus of our future research. These are discussed below.

1. **Algorithm Extraction:** Currently, algorithm extraction in the proposed system relies on the presence of captions and reference sentences in document text. This approach, being simple, scales well to large document collections and works well in practice, especially for well structured academic documents. However, the approach fails to identify algorithms which have no associated captions or captions which do not contain the keywords "algorithm" or "pseudo-code". We found many such documents. For such cases, instead of a pure text based approach, it might be useful to adopt techniques such as *page box cutting* [7] that utilize differences in visual appearances of different portions of a document.

2. **Ranking:** For ranking of algorithms, the proposed system utilizes evidence of relevance only from text based sources (i.e., caption, reference sentences and synopsis). However, the importance of various non-textual relevance indicators needs to be explored. For example, one measure of popularity of an algorithm is the number of citations an algorithm receives. Often, when researchers propose a new algorithm, they also discuss various shortcomings and advantages of the proposed algorithm. Furthermore, issues related to space and time efficiency of algorithms are also discussed. Having such information could enable us to assign a higher score to more efficient algorithms. However, how we automatically infer such information from document text is a challenging and unsolved problem.

3. **Algorithm Classification:** Often, algorithms for one problem are used (adopted) to solve problems in other domains. For example, algorithms to find shortest path in a graph, a problem with origins in graph the-

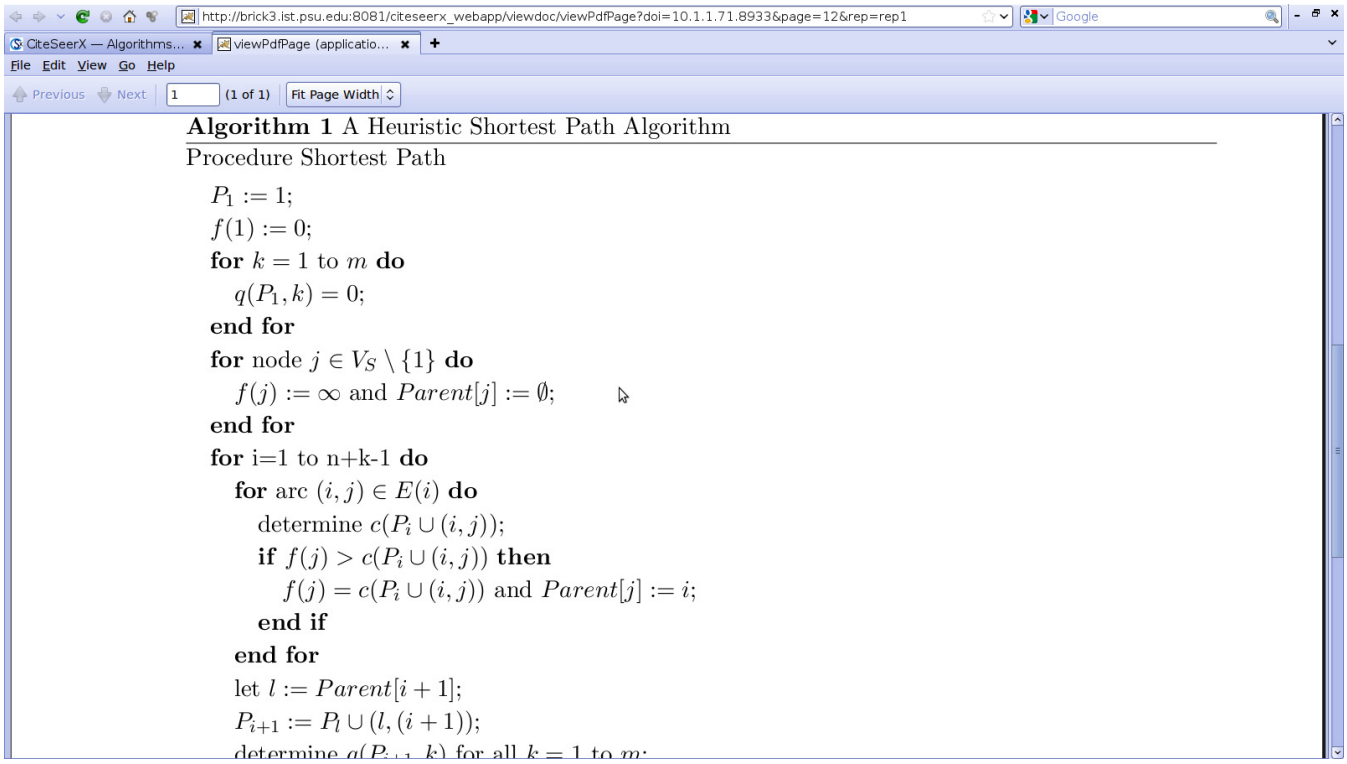

Figure 3: Screenshots showing algorithm page displayed on clicking the first result.

ory, are used extensively in computer networks and operations research. *How can we automatically categorize algorithms based on their applications* is thus an interesting research problem since it will enable us to answer user queries such as `shortest path algorithms for routing`, `SVM for text classification` etc. Having such information will also enable us to diversify search results based on applications. As an example, for the query `shortest path`, the result list can be constructed so as to cover shortest path algorithms as used in various domains. Such diversified result lists help in improving average user satisfaction [1].

4. CONCLUSIONS

We have described our initial efforts towards developing a search engine for algorithms. The proposed system extracts algorithms and associated metadata from academic documents, offers free text based query interface and presents a ranked list of results to the user by utilizing multiple sources of evidence. The source code for the proposed system will also be released as a part of SeerSuite toolkit.

5. ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under Grants 0535656 and 0845487. We gratefully acknowledge help from Juan Pablo Fernández Ramírez and Pradeep B. Teregowda.

6. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14, New York, NY, USA, 2009. ACM.

[2] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An internet-scale software repository. *Search-Driven Development-Users, Infrastructure, Tools and Evaluation, ICSE Workshop on*, 0:1–4, 2009.

[3] S. Bajracharya, J. Ossher, and C. Lopes. Searching api usage examples in code repositories with sourcerer api search. In *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, SUITE '10, pages 5–8, New York, NY, USA, 2010. ACM.

[4] S. Bhatia, S. Lahiri, and P. Mitra. Generating synopses for document-element search. In *Proceeding of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 2003–2006, New York, NY, USA, 2009. ACM.

[5] C. Ghezzi and A. Mocci. Behavior model based component search: an initial assessment. In *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, SUITE '10, pages 9–12, New York, NY, USA, 2010. ACM.

[6] S. Kataria, W. Browuer, P. Mitra, and C. L. Giles. Automatic extraction of data points and text blocks from 2-dimensional plots in digital documents. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, pages 1169–1174. AAAI Press, 2008.

[7] Y. Liu, K. Bai, P. Mitra, and C. L. Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '07, pages 91–100, New York, NY, USA, 2007. ACM.

[8] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[9] R. E. G. Valencia and S. E. Sim. Internet-scale code search. In *Search-Driven Development-Users, Infrastructure, Tools and Evaluation, 2009. SUITE '09. ICSE Workshop on*, pages 49–52, 2009.