AlgorithmSeer: A System for Extracting and Searching for Algorithms in Scholarly Big Data

Suppawong Tuarob, *Member, IEEE*, Sumit Bhatia,
Prasenjit Mitra, *Senior Member, IEEE*, and C. Lee Giles, *Fellow, IEEE*

Abstract—Algorithms are usually published in scholarly articles, especially in the computational sciences and related disciplines. The ability to automatically find and extract these algorithms in this increasingly vast collection of scholarly digital documents would enable algorithm indexing, searching, discovery, and analysis. Recently, *AlgorithmSeer*, a search engine for algorithms, has been investigated as part of CiteSeer^X with the intent of providing a large algorithm database. Currently, over 200,000 algorithms have been extracted from over 2 million scholarly documents. This paper proposes a novel set of scalable techniques used by *AlgorithmSeer* to identify and extract algorithm representations in a heterogeneous pool of scholarly documents. Specifically, hybrid machine learning approaches are proposed to discover algorithm representations. Then, techniques to extract textual metadata for each algorithm are discussed. Finally, a demonstration version of *AlgorithmSeer* that is built on Solr/Lucene open source indexing and search system is presented.

Index Terms—Algorithm search engine, ensemble machine learning, scholarly big data

1 Introduction

Computer science and many of its applications are about developing, analyzing, and applying algorithms. Efficient solutions to important problems in various disciplines other than computer science usually involve transforming the problems into algorithmic ones on which standard algorithms are applied. For example, algorithms for stock portfolio optimization are used for diversifying search results in information retrieval systems [37]. In the industrial design disciplines, the Latent Dirichlet Allocation [8] is effectively used to identify notable product features [34], [35]. Likewise, in bioinformatics Hirschberg's algorithm [16] is widely used to find maximal global alignments of DNA and protein sequences. Furthermore, a thorough knowledge of state-of-the-art algorithms is also crucial for developing efficient software systems.

A significant number of scholarly articles in computer science and related disciplines contain high-quality algorithms developed by researchers. Bhatia, et al., estimated the number of algorithms published in some major computer science conferences between 2005-2009 (Table 1). With dozens of new algorithms being reported in these conferences every year, it would be useful to have automated systems that efficiently identify, extract, index and search this ever increasing collection of algorithmic innovations. Such systems could provide an alternative source for researchers and software developers looking for cutting-edge algorithmic solutions to their problems.

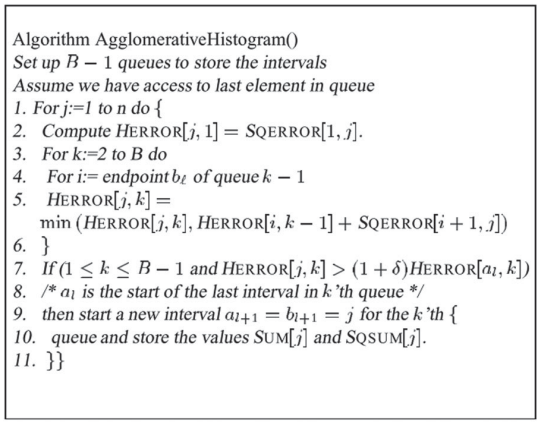
Standard algorithms[1] are usually collected and cataloged manually in algorithm textbooks (e.g., [21]), encyclopedias (especially the ones available online such as Wikipedia[2]), and websites that provide references for computer programmers (e.g., Rosettacode.org[3]). We parsed Wikipedia algorithm pages published in 2010, and found that roughly 1,765 standard algorithms were cataloged. The National Institute of Standards and Technology (NIST)[4] also has a dictionary of over 289 standard algorithms. While most standard algorithms are already cataloged and made searchable, especially those in online catalogs, newly published algorithms only appear in new articles. The explosion of newly developed algorithms in scientific and technical documents makes it infeasible to manually catalog these newly developed algorithms.

Manually searching for these newly published algorithms is a nontrivial task. Researchers and others who aim to discover efficient and innovative algorithms would have to actively search and monitor relevant new publications in their fields of study to keep abreast of latest algorithmic developments. The problem is worse for algorithm searchers who are inexperienced in document search. Ideally, we would like to have a system that automatically discovers and extracts algorithms from scholarly digital documents. Such a system could prove to facilitate algorithm indexing, searching, and a wide range of potential knowledge discovery applications and

- *S. Tuarob is with the Faculty of Information and Communication Technology, Mahidol University, Salaya 73170, Thailand. E-mail: suppawong.tua@mahidol.ac.th.*
- *S. Bhatia is with IBM Almaden Research Center, San Jose, CA. E-mail: sumit.bhatia@us.ibm.com.*
- *P. Mitra is with Qatar Computing Research Institute, Doha, Qatar. E-mail: pmitra@qf.org.qa.*
- *C.L. Giles is with Information Sciences and Technology, The Pennsylvania State University, PA 16801. E-mail: giles@ist.psu.edu.*

Manuscript received 31 Mar. 2015; revised 2 Jan. 2016; accepted 12 Feb. 2016. Date of publication 5 Apr. 2016; date of current version 27 May 2016. Recommended for acceptance by Y.-R. Lin, H. Tong, J. Tang and K.S. Candan. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TBDATA.2016.2546302

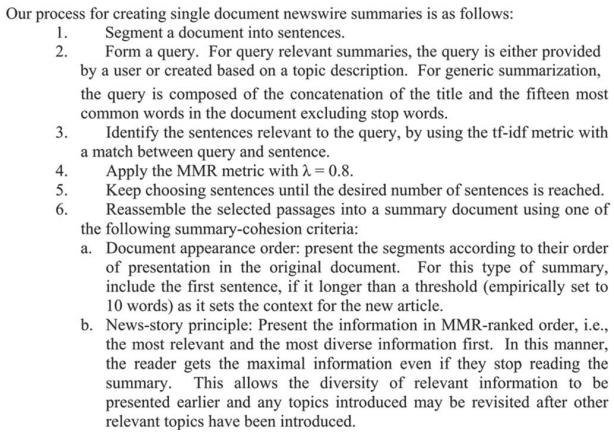
1. We define a standard algorithm to be an algorithm that is well known by people in a field and is usually recognized by its name, including *Dijkstra*'s shortest-path algorithm, *Bellman-Ford* algorithm, *Quicksort* algorithm, etc.
2. http://www.wikipedia.org/
3. http://rosettacode.org/wiki/Rosetta_Code/
4. http://xlinux.nist.gov/dads/



Algorithm AgglomerativeHistogram()
Set up $B-1$ queues to store the intervals
Assume we have access to last element in queue
1. For j:=1 to n do {
2. Compute $\text{HERROR}[j,1] = \text{SQERROR}[1,j]$.
3. For k:=2 to B do
4. For i:= endpoint b_ℓ of queue $k-1$
5. $\text{HERROR}[j,k] =$
$\min(\text{HERROR}[j,k], \text{HERROR}[i,k-1] + \text{SQERROR}[i+1,j])$
6. }
7. If ($1 \le k \le B-1$ and $\text{HERROR}[j,k] > (1+\delta)\text{HERROR}[a_l,k]$)
8. / a_l is the start of the last interval in k'th queue */*
9. then start a new interval $a_{l+1} = b_{l+1} = j$ for the k'th {
10. queue and store the values $\text{SUM}[j]$ and $\text{SQSUM}[j]$.
11. }}

Figure 3. Algorithm AgglomerativeHistogram

Fig. 1. Example pseudo-code (PC), from [1].



Our process for creating single document newswire summaries is as follows:

1. Segment a document into sentences.
2. Form a query. For query relevant summaries, the query is either provided by a user or created based on a topic description. For generic summarization, the query is composed of the concatenation of the title and the fifteen most common words in the document excluding stop words.
3. Identify the sentences relevant to the query, by using the tf-idf metric with a match between query and sentence.
4. Apply the MMR metric with λ = 0.8.
5. Keep choosing sentences until the desired number of sentences is reached.
6. Reassemble the selected passages into a summary document using one of the following summary-cohesion criteria:
   a. Document appearance order: present the segments according to their order of presentation in the original document. For this type of summary, include the first sentence, if it longer than a threshold (empirically set to 10 words) as it sets the context for the new article.
   b. News-story principle: Present the information in MMR-ranked order, i.e., the most relevant and the most diverse information first. In this manner, the reader gets the maximal information even if they stop reading the summary. This allows the diversity of relevant information to be presented earlier and any topics introduced may be revisited after other relevant topics have been introduced.

Fig. 2. Example algorithmic procedure (AP), from [27].

studies of the algorithm evolution, and presumably increase the productivity of scientists.

Identifying and extracting various informative entities from scholarly documents is an active area of research. For algorithm discovery in digital documents, Bhatia, et al., have described a method for automatic detection of *pseudo-codes* (*PCs*) in Computer Science publications [6]. Their method assumes that each PC is accompanied by a caption (e.g., Fig. 1). Such a PC can then be identified using a set of regular expressions to capture the presence of the accompanied caption.

However, such an approach is limited in its coverage due to reliance on the presence of PC captions and wide variations in writing styles followed by different journals and authors. We found that 25.8 percent (71 out of 275) of the PCs (see Section 4.4.1) did not have accompanied captions, and would remain undetected by their proposed approach.

Furthermore, even though PCs are commonly used in scientific documents to represent algorithms, a majority of algorithms are also represented using *algorithmic procedures* (*APs*). An algorithmic procedure (e.g., Fig. 2) is a set of descriptive algorithmic instructions and differs from a PC in the following ways:

- *Writing style.* PCs are usually written in a programming style, with details omitted. Symbols, Greek letters, mathematical operators, and programming keywords (such as 'for', 'begin', 'end', 'return', etc.) are usually used to compose PCs. On the other hand, APs are usually written in a listing style, with a descriptive manner. Each step usually begins with a bullet point, or a number. APs lack the power to express complex nested loops and are less concise than PCs, but they are easier to comprehend by general readers who do not have a programming background.
- *Location in documents.* PCs are usually not part of the running text; they may appear anywhere in the documents. Because of this, most PCs have *identifiers* which the context in the document can refer to. These identifiers include captions (e.g., '*Figure 3: The hill-climbing algorithm.*'), function names (e.g., '*APPROXMAX-SAT(g, S , p)*'), and algorithm names (e.g., '*Algorithm BuildGalledNetwork*'). On the contrary, algorithmic procedures mostly appear as part of the running text, and hence do not have unique identifiers. Hence, detecting APs would require a different set of techniques.

Since algorithms represented in documents do not conform to specific styles, and are written in arbitrary formats, this becomes a challenge for effective identification and extraction. Here we propose a novel methodology based on ensemble machine learning to discover algorithm representations such as PCs and APs automatically. Moreover, we observe that two or more algorithm representations may be used to describe the same algorithms. Hence, we also propose a simple heuristic that links different algorithm representations that together constitute an algorithm. Automatic discovery and extraction of these algorithm representations will be useful for applications in digital libraries and document engineering.

This paper has the following key contributions:

1) We propose a set of hybrid techniques based on ensemble machine learning to discover PCs and APs in scholarly documents. Specifically, three variations of a methodology for detecting PCs include an extension of the existing rule based method proposed by Bhatia, et al. [6], one based on ensemble machine learning techniques, and a hybrid of these two. The methods for discovering APs include a rule based method and a machine learning based method. Furthermore, a machine learning based strategy to identify sections in a scholarly document by detecting section headers and using them as section boundaries is also discussed. Finally, we propose a heuristic that links different algorithm representations referring to the same algorithm together. A case study of 258 scholarly documents selected from Citeseer^X repository is used to validate the efficacy of these techniques.
2) We find that the textual metadata that can be directly extracted from an algorithm representation (especially PCs), is inadequate for effective retrieval, and propose to use the synopsis generation method [4], which automatically generates a comprehensive description for a document element,

TABLE 1
Approximate Number of Algorithms Published in Different Computer Science Conferences during 2005-2009, Reproduced from [4]

Conference	No. of Algorithms
SIGIR	75
SIGMOD	301
STOC	74
VLDB	278
WWW	142

and the tag-based automatic metadata annotation [32] technique, which automatically textually enriches metadata records, to extract more meaningful textual information for PCs. A case study of 6,285 PCs extracted from randomly sampled 20,567 scholarly documents is used to discuss and validate these methods.

3) We describe how algorithm textual metadata is indexed and made searchable, along with demonstrating the test version of *AlgorithmSeer*.

2 Background and Related Works

Since literature on scholarly information extraction is extensive, only works closely related to ours are discussed.

2.1 Document Element Extraction in Scholarly Documents

Identifying and extracting informative entities such as mathematical expressions [3], [39] , tables [24], figures[12], [17], and tables of contents[38] from documents have been extensively studied. Kataria, et al. [17], employed image processing and Optical Character Recognition (OCR) approaches for automatic extraction of data points and text blocks from 2D plots. They also proposed a method to index and search for the extracted information. Sojka and Líška proposed *MIaS* (Math Indexer and Searcher) that collects and interprets mathematical expressions [26]. Since their system only handles documents in the MathML (Mathematical Markup Language) format where mathematical expressions have already been marked up, their approach is inapplicable to our problem. Bhatia, et al., proposed a set of methods used for detecting document-elements that have accompanied captions, e.g., tables, figures, and pseudocodes [4]. A document-element is identified by detecting the presence of the corresponding caption using a set of regular expressions. In this paper, we use their method as the baseline for the PC detection task.

2.2 Search Systems for Scholarly Information

Aside from well known web search engines such as Google[5] and Microsoft's Bing[6], various vertical search engines have been proposed. CiteSeer[7], now CiteSeer^X, was developed as a scientific literature digital library and search engine which automatically crawls and indexes scientific documents primarily in the field of computer and information science [22].

5. https://www.google.com/
6. http://www.bing.com/
7. http://citeseerx.ist.psu.edu

Fig. 3. Sample search results for the query "`shortest path finding algorithm`" on Citeseer^X.

Liu, et al., presented *TableSeer*, a tool which automatically identifies and extracts tables in digital documents [23]. They used a tailored vector-space model based ranking algorithm, *TableRank*, to rank the search results. An implementation of *TableSeer* that extracts and searches for tables in the CiteSeer^X document repository has been included in the Citeseer^X suite. BioText[8] search engine, a specialized search engine for biology documents, also offers the capability to extract figures and tables, and make them searchable [15]. Khabsa, et al., described *AckSeer*, an acknowledgement search engine that extracts, disambiguates, and indexes more than 4 million mentioned entities from 500,000 acknowledgments from documents in CiteSeer^X [18]. Chen, et al., emphasized the importance of scientific collaboration and introduced *CollabSeer*, a search engine for discovering potential collaborators for a given author or researcher by analyzing the structure of the coauthor network and the user's research interests [11].

Though useful algorithms could be found in many specialized search engines (e.g., *Citeseer^X*[9], *Google Scholar*[10]) and algorithm-related discussion forums (e.g., *Stack Overflow*[11], *Quora*[12]), the search results can be contaminated with irrelevant items. Fig. 3 illustrates an example of a search session (top five results) for *shortest path finding algorithms* using the query "`shortest path finding algorithm`" on *Citeseer^X* search engine that currently hosts 5 million academic publications whose major proportion is computer science and related disciplines. According to the figure, only two out of five top results are relevant (actual papers that describe

8. http://biosearch.berkeley.edu
9. http://citeseerx.ist.psu.edu/
10. https://scholar.google.co.th/
11. http://stackoverflow.com/
12. https://www.quora.com/

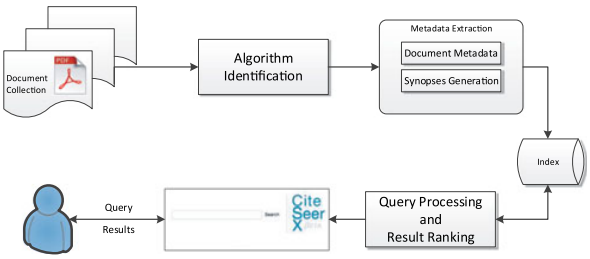


Fig. 4. Architecture of the proposed system.

shortest path algorithms), while the other three results are simply documents containing those search terms. Furthermore, these search engines would return the whole documents (papers or websites) as search results, requiring the users to invest additional, unnecessary effort to read the entire documents to find the desired algorithms. To the best of our knowledge, we are the first to explore the possibility of building a search engine specifically for algorithms extracted from scholarly digital documents.

3 The System

In this article, a prototype of an algorithm search engine, *AlgorithmSeer*, is presented. Fig. 4 illustrates the high level of the proposed system. First, scholarly documents are processed to identify algorithm representations (Section 4). Then, the textual metadata that provides relevant information about each detected algorithm representation is extracted (Section 5). The extracted textual metadata is then indexed, and made searchable (Section 6).

Note that, even though the methods presented in this paper are developed to handle algorithms in scholarly digital documents, the methods could be generalized to other information sources that contain textual representations of algorithms such as pseudo-codes found in web-pages (e.g., Stack Overflow, Wikipedia, etc.). Algorithm extraction in html is different from algorithm extraction in free text, though there are some similarities. For example, webpages are usually composed in HTML which provides markup tags (e.g., `<tr>`,`<td>`, etc.) that can be utilized to detect boundaries of document elements, while such functionalities do not exist in free text data. It is also possible that the ideas presented in this paper could be developed for other document elements such as tables and diagrams, as long as textual information from such entities could be extracted.

4 Algorithm Identification in Scholarly Documents

This section discusses the methods for automatic discovery of PCs and APs in scholarly documents. Fig. 5 displays the high level diagram of the proposed system. The system specifically handles PDF documents since a majority of articles in modern digital libraries including Citeseer^X are in PDF format. First, plain text is extracted from the PDF file. Inspired by Hassan [14], we use PDFBox[13] to extract text and modify the package to also extract object information such as

13. http://pdfbox.apache.org/

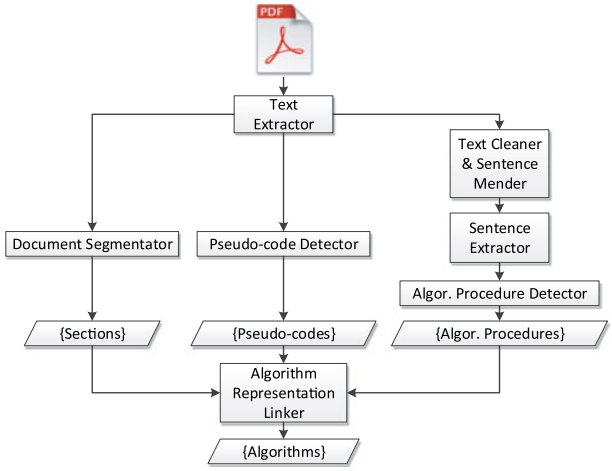


Fig. 5. Diagram of the proposed algorithm identification system.

font and location information from a PDF document. Then, three sub-processes operate in parallel, including document segmentation, PC detection, and AP detection. The document segmentation module identifies sections in the document. The PC detection module detects PCs in the parsed text file. The AP detector first cleans extracted text and repairs broken sentences (since the method assumes that a document is represented with a sequence of sentences), then identifies APs. After PCs and APs are identified, the final step involves linking these algorithm representations referring the same algorithms together. The final output would then be a set of unique algorithms.

4.1 Detecting Pseudo-Codes (PCs)

Most scientific documents use PCs for compact and concise illustrations of algorithms. PCs are normally treated as document elements separated from the running text, and usually are accompanied with identifiers such as captions, function names, and/or algorithm names. Since PCs can appear anywhere in a document, these identifiers usually serve the purpose of being anchors which can be referred to by context in the running text. Here, three approaches for detecting PCs in scholarly documents are presented: a rule based method (*PC-RB*), an ensemble machine learning based method (*PC-ML*), and a combined method (*PC-CB*). Note that though OCR based techniques (where each page of the document is first converted into an image, where image processing based techniques are used to locate the boundary of the PCs, then an OCR technique is used to extract the content within each candidate region) have been explored, textual content can be directly and quite accurately extracted from most PDF files. It seems that converting documents into images and applying OCR algorithms would just add more noise to the extracted text.

4.1.1 Rule Based Method (PC-RB)

Recently, Bhatia et al. proposed a rule based PC detection approach, which utilizes a grammar for document-element captions to detect the presence of PC captions [7]. We refer their method as our baseline (*PC-BL*) for the PC detection task. Here, our proposed *PC-RB* extends the baseline by

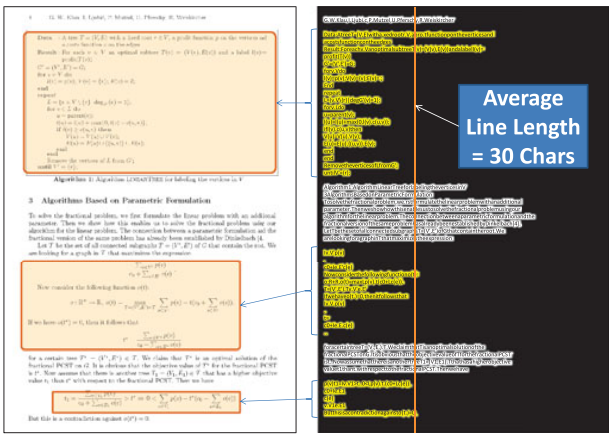


Fig. 6. Example of sparse regions (i.e., *sparse boxes*), taken from [20]. The left figure is an actual PDF page. The right figure illustrates the extracted text.

adding the following rules to improve coverage and reduce false positive rate:

- A PC caption must contain at least one algorithm keyword, namely *pseudo-code*, *algorithm*, and *procedure*.
- Captions in which the algorithm keywords appear after prepositions (e.g., *'Figure 15: The robust envelope obtained by the proposed algorithm'*) are excluded, as these are not likely captions of PCs.

Given a document (tagged with line numbers), the *PC-RB* then locates PCs by detecting the presence of their captions.

4.1.2 Machine Learning Based Method (PC-ML)

The *PC-RB* yields a high precision, however it still suffers from a low coverage resulting in a poor recall. We found that 25.8 percent of PCs in our dataset do not have accompanied captions. These PCs would remain undetected using the *PC-RB* method. To get around this issue, we propose a machine learning based (*PC-ML*) method that directly detects the presence of PC contents (instead of their captions). Our motivation originated from the observation that most PCs are written in a sparse manner, resulting in sparse regions (we call them *sparse boxes*) in documents. Fig. 6 shows an example of sparse boxes on a sample scientific article page. The *PC-ML* first detects and extracts these sparse boxes, then classifies each box whether it is a PC box or not. A *PC box* is a sparse box that contains at least 80 percent content (in terms of number of lines) of a PC. The following sections explain how sparse boxes are identified, the feature sets, and the classification models. The output of the *PC-ML* is a tuples of $\langle start, end \rangle$ line numbers of the detected PC.

Sparse box extraction. We define a *sparse box* as a set of at least N consecutive sparse lines. A *sparse line* is a line that meets the following criteria: *1)* the ratio of the number of non-space characters to the average number of characters per line is less than the threshold M, *2)* not footers/headers, and *3)* enclosed by sparse lines. We found that N = 4 and M = 0.8 work best for our dataset. The right sub-figure of Fig. 6 illustrates extracted text lines which are sparse lines (highlighted in yellow), and non-sparse lines (highlighted in white). Each set of consecutive sparse lines composes a sparse box as illustrated in the left sub-figure. The efficacy of the sparse box extraction method is evaluated in two perspectives: *coverage* and *accuracy*. Given a set of sparse boxes B extracted from a document d, the *coverage* is defined as following:

$$Coverage = \frac{|\{l | l \in b, b \in B, l \ is \ positive\}|}{|\{l | l \in b, l \ is \ positive\}|}. \quad (1)$$

The line-wise recall is utilized to quantify how much PC content can be captured within each extracted sparse box. Our sparse box extraction method yields a coverage of 92.99 percent. Among all the sparse boxes detected in our *DS2* dataset (See Section 4.4.1), we found 237 (out of 275 (86.18 percent) actual PCs) PC boxes.

The *accuracy* evaluation quantifies how precisely each PC is cut into a sparse box. For each PC box, we measure both the upper boundary delta (the start line number of the actual PC minus the start line number of the sparse box) and lower boundary delta (the end line number of the actual PC minus the end line number of the sparse box). 76.37 and 70.89 percent of PC boxes have upper and lower line deltas of ± 2 lines respectively, suggesting that most PC boxes are fully and precisely extracted by the proposed sparse box cutting technique.

Feature selection for PC box classification. A set of 47 features (listed in Table 2) is extracted from each sparse box. These features are classified into 4 groups: font-style based (*FS*), context based (*CX*), content based (*CN*), and structure based (*ST*). The *FS* features capture various font styles used to compose PCs. The *CX* features detect the presence of PC captions. The *CN* features capture the PC specific keywords and coding styles. The *ST* features characterize the sparsity of PCs and the symbols used.

Classification models. Each sparse box is classified whether it is a PC box or not. We train 12 base machine learning classification algorithms with the features described in Table 2. These algorithms include *Logistic Model Trees* (LMT), *Multinomial Logistic Regression* (MLR), *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER), *Linear Logistic Regression* (LLR), *Support Vector Machine* (SVM), *Random Forest* (RF), *C4.5 decision tree*, *REPTree*, *Decision Table* (DT), *Random Tree* (RT), *Naive Bayes* (NB), and *Decision Stump* (DS).

In addition to the base classifiers listed above, we also combined them using standard ensemble techniques such as *uniform weighted majority voting (VOTE)* and *probability averaging (PAVG)* [19]. First, the 12 base classifiers are tested with 10 percent held-out data from the training data and ranked by their F1 scores. Then, the first 2, 3, ... , 12 ranked classifiers in each ranked list are combined using the ensemble methods. Note that other ensemble techniques such as *Adaboost*, *Bagging*, and *Rotation Forest* were also explored but overall the *VOTE* and *PAVG* methods performed much better, agreeing with a prior empirical study of ensemble classifications by Kittler et al. which found that these two ensemble methods outperformed others (i.e., multi-staging, product, maximum, median, and minimum rules) on the identity verification and handwritten digit recognition tasks [19].

TABLE 2
Features for PC Box Classification Can Be Divided into Four Groups: Font Style Based (FS), Context Based (CX), Content Based (CN), and Structure Based (ST)

Grp	Feature	Description
FS	INDENTATION VARIANCE	Variance of the positions of the first character in each line
	FIRST 4CHARS INDENTATION VARIANCE	Variance of the average of the positions of the first 4 characters
	NUM DIFF FONTSTYLES	# of different font styles. Ex. 'X*X*X*' has 2 font styles.
	NUM FONTSTYLE CHANGES	# of char pairs whose font styles are different. Ex. 'X*X*X*' has 3 font style changes.
	FRAC NUM FONTSTYLE CHANGES	Fraction of number of font style changes to number of lines.
CX	HAS CAPTION NEARBY	Has a caption near (within three upper/lower lines) the sparse box.
	HAS PC CAPTION NEARBY	Whether there is a pseudo-code caption near the sparse box.
	HAS PC NAME NEARBY	Has is an algorithm name (e.g., *'Algorithm ABC'*) near the sparse box.
CN	NUM PC WORDS	# of PC keywords (e.g., *forall, for, if, else, iftrue, endif*, etc.)
	FRAC PC WORDS TO NUMWORDS	Fraction of pseudo-code keywords to number of words.
	FRAC PC WORDS TO NUMLINES	Fraction of pseudo-code keywords to number of lines.
	NUM ALGO WORDS	# of algorithm keywords (e.g., algorithm, pseudo-code,etc.)
	FRAC ALGO WORDS TO NUMWORDS	Fraction of # of algorithm keywords to # of words
	FRAC NUM ALGO WORDS TO NUMLINES	Fraction of # of algorithm keywords to # of lines
	NUM LINES BEGIN WITH PC WORDS	# of lines beginning with a pseudo-code keyword
	FRAC NLINES BEG. W/ PCWORDS TO NLINES	Fraction of # of lines beginning with a PC word to # of lines
	NUM FUNCTIONS	# of functions. Ex. *Scan(f, x)*
	FRACTION NUM FUNCTIONS TO NUMLINES	Fraction of # of functions to # of lines
ST	NUM CHARS	# of characters
	FRAC NUM CHARS TO NUMLINES	Fraction of # of characters to # of lines
	NUM SYMBOLS	# of symbols
	FRAC NUM SYMBOLS TO NUMCHARS	Fraction of # of symbols to # of characters
	NUM ALPHABETS	# of alphabets
	FRAC NUM ALPHABETS TO NUMCHARS	Fraction of # of alphabets to # of characters
	NUM DIGITS	# of digits
	FRAC NUM DIGITS TO NUMCHARS	Fraction of # of digits to # of characters
	NUM ALPHANUMERS	# of alphanumeric characters
	FRAC NUM ALPHANUMBERS TO NUMCHARS	Fraction of # of alphanumeric characters to # of characters
	NUM NON-ALPHANUMBERS	# of non-alphanumeric characters
	FRAC NON-ALPHANUMBERS TO NUMCHARS	Fraction of # of non-alphanumeric characters to # of characters
	NUM GREEKCHARS	# of Greek characters
	FRAC NUM GREEK TO NUMCHARS	Fraction of # of Greek characters to # of characters
	NUM ARROWS	# of arrow symbols
	FRAC NUM ARROWS TO NUMCHARS	Fraction of # of arrow characters to # of all characters
	NUM MATHOPS	# of math operators (e.g., $+, -, \Sigma, \times$, etc.)
	FRAC NUM MATHOPS TO NUMCHARS	Fraction of # of math operators to # of characters
	NUM 1-CHAR WORDS	# of 1-character words (e.g., 'x xx x' has 2 1-character words)
	FRAC NUM 1-CHAR WORDS TO NUMLINES	Fraction of # of single-char words to # of lines
	FRAC NUM 1-CHAR LINES TO NUMLINES	Fraction of # of 1-character lines to # of all lines
	NUM IJK	# of characters 'i', 'j', and 'k'
	FRAC NUM IJK TO NUMLINES	Fraction of # of 'i', 'j', 'k' characters to # of lines
	NUM CODING SYMBOLS	# of coding symbols (e.g., {,},[,],@,/)
	FRAC NUM CODING TO NUMLINES	Fraction of # of coding symbols to # of lines
	NUM LINES END WITH DOT	Number of lines ending with '.'
	FRAC NUMLINES EWDOT TO NUMLINES	Fraction of # of lines ending with '.' to # of lines
	NUM LINES BEGIN WITH NUMBER	# of lines beginning with a number
	FRAC LINES BWNUMBER TO NUMLINES	Fraction of # of lines beginning with anumber to # of lines

4.1.3 Combined Method (PC-CB)

Though the *PC-ML* method can capture PCs even though they do not have accompanied captions, some PCs which are not first captured in a sparse box would still remain undetected. Mostly, such PCs are either written in a descriptive manner (hence do not result in sparse regions in the document), or figures (the text extractor cannot extract images). In our dataset DS2 (See Section 4.4.1), 35 PCs (out of 275 actual PCs) cannot be captured using the sparse box extraction. However, 27 (out of 35) of these undetected PCs have accompanied captions and hence might still be detected using the *PC-RB* method. We propose a combined method (*PC-CB*) of the *PC-RB* and the *PC-ML* using a simple heuristic as follows:
STEP1 For a given document, run both *PC-RB* and *PC-ML*.

STEP2 For each PC box detected by *PC-ML*, if a PC caption detected by *PC-RB* is in proximity, then the PC box and the caption are combined.

4.2 Detecting Algorithmic Procedures (APs)

An AP is used to represent a relatively simpler algorithm, or describe an algorithm in a higher level, since it lacks the power to precisely express complex computational operations such as nested loops, recursion, and functions. An AP is usually composed in an order-listing style, with descriptive wordings. Most APs usually have a sentence at the beginning to introduce the algorithm enclosed within. Such a sentence is referred to as an *algorithmic procedure indication sentence* (e.g., `To solve the subproblem, we propose a pseudopolynomial dynamic programming`

TABLE 3
Features for Detecting AP Indication Sentences Can Be Divided into Two Groups: Content Based (CN) and Context Based (CX)

Grp	Feature	Description
CN	MATCH AP SENTENCE RULES	Wether it is an AP indication sentence
	HAS ALGO WORDS	Wether it contains an algorithm keyword (e.g., 'algorithm', 'procedure', etc.)
	HAS STEP WORDS	Wether it contains a stepwise indication word (e.g., 'followings', 'steps', etc.)
	END WITH COLON	Whether it ends with a colon (i.e., ':')
	END WITH STEP WORDS	Whether it ends with a stepwise keyword
	END WITH LISTING PLURAL	Whether it ends with plural list-indicating noun (e.g., 'properties', 'results', etc.)
	IS CAPTION	Wether it is a caption
	NUM WORDS	Number of words
	IS NUMWORDS LESSTHAN T	Whether the number of words is less than the threshold T (we use $T = 35$)
CX	NUM STEP SENTENCES	# of sentences starting with a bullet point, a listing number/alphabet, or the word 'Step'
	NUM PC KEYWORDS	# of pseudo-code keywords (e.g., forall,for,if,else,iffalse,iftrue,endif,etc.)
	FRAC PC WORDS TO NUMLINES	Fraction of # of pseudo-code keywords to # of lines
	NUM ALGO WORDS	Number of algorithm keywords (e.g., algorithm,pseudo-code,procedure,etc.)
	FRAC ALGO WORDS NUMWORDS	Fraction of number of algorithm keywords to # of words
	NUM SYMBOLS	Number of symbols
	FRAC SYMBOLS TO NUMLINES	Fraction of number of symbols to number of lines
	NUM GREEKCHARS	Number of Greek characters
	FRAC GREEKCHARS TO NUMCHARS	Fraction of number of Greek characters to # of lines
	NUM 1-CHAR WORDS	Number of single-character words (e.g., 'x xx x' has 2 single-character words)
	FRAC 1-CHAR WORDS NUMLINES	Fraction of number of single-char words to # of lines
	NUM MATHOPS	Number of math operators (e.g., +,-, S, etc.)
	FRAC NUM MATHOPS NUMCHARS	Fraction of number of math operators to # of lines
	NUM NON-ALPHANUMBERS	Number of non-alphanumeric characters
	FRAC N-ALPHANUMBERS	Fraction of number of non-alphanumeric characters to number of lines

`algorithm as follows:`). Our AP detection approaches aim to detect such indication sentences, and use them to locate the accompanied APs.

Two methods are developed for detecting AP indication sentences: a rule based method (*AP-RB*) and a machine learning based method (*AP-ML*). Both methods rely on the sentences correctly extracted from the document. However, most scholarly documents are multi-columned and contain document elements such as tables and diagrams. It is often seen that the extracted text may contain garbage/noisy lines and broken sentences. Hence, before extracting sentences, garbage text fragments are removed from the document. We also develop a heuristic that stitches up an incomplete sentence which are broken into multiple lines. After the extracted text is cleaned and broken sentences are mended, sentences are extracted using LingPipe[14] sentence extractor.

4.2.1 Rule Based Method (AP-RB)

Often, AP indication sentences exhibit certain common properties:

- The sentences usually end with *follows:*, *steps:*, *algorithm:*, *follows:*, *following:*, *follows.*, *steps:*, *below:*.
- The sentences usually contain at least an algorithm keyword.

We create a set of regular expressions to capture sentences according to the rules above.

4.2.2 Machine Learning Based Method (AP-ML)

Unfortunately, some AP indication sentences do not conform to the rules described in Section 4.2.1. Therefore, an alternative approach based on machine learning is proposed to directly learn to capture the characteristics of APs.

Feature selection for AP detection. A set of 26 features is extracted from each sentence. The features can be categorized into two groups: content based features (*CN*) and context based features (*CX*). The content based features are extracted from the sentence itself, and are designed to learn the characteristics of the AP indication sentences. The context based features are extracted from the 28 lines below the line where the sentence appears. The number 28 is the average number of lines of APs observed in our data set. Table 3 lists all the features.

Classification models. We use the same set of base classifiers and ensemble methods as described in Section 4.1.2.

4.3 Linking Algorithm Representations

A simple heuristic is implemented to link algorithm representations referring to the same algorithm together. Specifically, two algorithm representations are linked if:

1) They represent the same algorithm. For example, an AP may be used to provide more detail to a PC.
2) They are part of the same algorithm. For example, an algorithm may be broken into sub-parts, each is represented using a different algorithm representation.

It is assumed that there can be at most one algorithm contained in a document section. Our linking algorithm first assigns each algorithm representation to a section. Algorithm representations which fall into the same section are then linked. We are aware that this assumption needs statistical justification which we leave for future investigation; hence, this section would focus on the general idea of the algorithm linking method, rather than the strict assumption. We first briefly discuss how a document is segmented into

14. http://alias-i.com/lingpipe/

TABLE 4
Notable Classification Results

Classifier	Balancing	Precision	Recall	F-measure	ATT (s)
RandomForest	none	0.9374	0.8912	0.9137	66.47
RandomForest	WEIGHT	0.9374	0.9106	**0.9238**†	52.96
RandomForest	ROver	**0.9403**†	0.9002	0.9198	106.68
RandomForest	RUnder	0.6248	**0.9500**	0.7538	2.58
NaiveBayes	none	**0.4420**	0.8744	**0.5872**	1.19
NaiveBayes	ReS	0.1581	**0.9629**†	0.2717	1.01
SVM	none	**0.8570**	0.8828	**0.8697**	6.74
SVM	RUnder	0.5861	**0.9304**	0.7192	5.68
RIPPER	none	**0.9118**	0.8623	**0.8863**	62.39
RIPPER	RUnder	0.4931	**0.9343**	0.6455	1.75

ATT *denotes Average Training Time per fold*.

sections, then explain how algorithm representations are assigned to corresponding sections and linked.

4.3.1 Identifying Sections in Scholarly Documents

A machine learning based approach is used to identify section boundaries by detecting *section headers*. The algorithm was first proposed by Tuarob et al. [31]. Most scholarly documents have following common properties:

1) Each section has a section header, usually with a section number.
2) Section headers usually have distinct font styles from the surrounding content.
3) A majority of sections are common sections such as *Abstract*, *Introduction*, *Background*, *Conclusions*, and *References*.

From such properties, 22 features are identified to characterize section headers in scholarly documents. The features can be divided into three groups: pattern based (*PAT*), style based (*STY*), and structure based (*STR*). The *PAT* features are used for capturing section headers which are standard sections or section headers with section numbers. The *STY* features filter out lines that look like section headers but are in fact fragments of sentences, lines in tables of contents, or textual fragments from tables/diagrams. The *STR* features concern the locations of the section headers. For example, lines that occur between the *Abstract/Introduction* and the *References* sections are better candidates than those outside this region. These features also filter out footers and headers.

A number of data balancing techniques and classification algorithms were considered, including weighted balancing, random under-sampling, random over-sampling, resampling, and the Synthetic Minority Oversampling TEchnique (SMOTE) [10]. The classifiers are tested on 117 PDF scholarly documents randomly selected from Citeseer^X repository, containing various types of scholarly documents namely conference papers, journals, theses, and academic articles, using document-wise 10 fold cross validation. Table 4 lists notable results of the classification with different combinations of classifiers and data balancing strategies. The bold numbers are the highest among each classifier. The figure with† represents the best performance. We found that the *Random Forest* classification algorithm [9] with 100 trees using a variant of weighting balancing techniques (where minority class instances are given higher weight), achieved the best classification performance with 93.74 percent Precision, 91.06 percent Recall, and 92.38 percent F1. Such a classification model is used in this paper to identify sections in scholarly documents.

4.3.2 Using Document Sections for Linking Algorithm Representations

Assigning an AP to a section is easy, since it is part of the running text. Unlike APs, PCs are normally located separately from the running text; hence, they could appear outside the sections that discuss them. To get around this problem, a PC is mapped to the section that contains the largest number of *reference sentences* that refer to it. Once each algorithm representation is assigned to a section, algorithm representations which are mapped to the same sections are then linked.

4.4 Experiments on Algorithm Identification

The experiments are divided into three parts: PC detection, AP detection, and algorithm representation linking. All the experiments are performed on a Windows machine with an Intel Core i7-2600 CPU (3.4 GHz) and 16 GB of ram. We use the LibSVM[15] implementation for SVM, and Weka[16] implementation for other classification algorithms.

4.4.1 Datasets

Two datasets are used in the algorithm detection tasks:

[DS1] consists of 100 scholarly documents manually selected from the Citeseer^X repository to represent diverse types of scholarly articles and algorithm representations. This data set is used to construct rules and regular expressions for our rule based methods, and determine feature sets for our machine learning based methods.

[DS2] consists of 258 scholarly documents *randomly* selected from the Citeseer^X repository. This data set consists of 275 PCs, 86 APs, and 282 unique algorithms. We use this data set to evaluate our proposed algorithm detection methods.

4.4.2 Evaluation of PC Detection

10-fold document-wise cross validation is used to validate the three variants of PC detection approaches on the dataset DS2. Standard precision, recall, and F1 are used for evaluating the performance. Let T_g be the set of all PCs, T_r be the set of detected PCs, so that the correctly detected PCs are $T_g \bigcap T_r$. These metrics are defined as follows:

$$precision = \frac{|T_g \bigcap T_r|}{|T_r|}, recall = \frac{|T_g \bigcap T_r|}{|T_g|}, F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}. \quad (2)$$

Table 5 lists notable results of our proposed methods against the PC detection baseline (*PC-BL*). As expected, our rule-based method (*PC-RB*) yields high precision with a cost of low recall. Using machine learning techniques (*PC-ML*), the overall performance (in terms of F1) is improved. The combined method (*PC-CB*) of *PC-RB* and a *majority*

15. http://www.csie.ntu.edu.tw/ cjlin/libsvm/
16. http://www.cs.waikato.ac.nz/ml/weka/

TABLE 5
Precision, Recall, and F1 of the Best Classification Models (in Terms of F1) Used for the Pseudo-Code (PC) and Algorithmic Procedure (AP) Methods

Method	Model	Pr%	Re%	F1%
PC-BL	Baseline	70.46	35.96	47.62
PC-RB	RuleBased (Improved Baseline)	87.12	44.57	58.97
PC-ML	!LMT-RF-RIPPER-MLR	85.31	57.04	68.37
PC-CB	!LMT-RF-RIPPER	87.37	67.17	75.95
AP-RB	RuleBased	64.24	28.00	39.00
AP-ML	+NB-LMT-LLR-MLR-RT	69.56	49.00	57.50

('!' denotes Majority Voting (VOTE), '+' denotes Probability Averaging (PAVG)).

voting of *LMT*, *Random Forest*, and *RIPPER* base classifiers performs the best in terms of F1, outperforming the state-of-the-art baseline by 28.33 percent (in terms of F1). It is noticed that base classifiers which inherit from the rule/tree based families perform well in this task as opposed to those from the network and function based families. This is because the features are rule-extracted in nature which result in a fixed feature space (as opposed to dynamic features, such as word features, that grow with the corpus size).

It is worth noting that the ensemble methods result in greater improvement compared to only using individual experts. Fig. 7 compares the performances (in terms of F1) between the ensemble methods and the best base classifiers for *PC-ML* (i.e., with *MLR* classifier) and *PC-CB* (i.e., with *LMT* classifier). The X-axis denotes the first k base classifiers used in each ensemble method. Note that, the fact that the combination of *LMT*, *Random Forest*, and *RIPPER* accumulatively yields the best result means that each individual expert learns a different aspect of the data, and hence is able to correct each other when making collective decisions. Not surprisingly, these base classifiers are inherited from different classes of machine learning algorithms (function, tree, and rule based respectively). We conclude that the ensemble methods are useful for these specific problems, when the best base classifiers are combined. However, the performance of the ensemble methods can decrease as the number of base classifiers grows. This might be because bad base classifiers can impede the collective decisions of the good ones. Unlike traditional document classification techniques

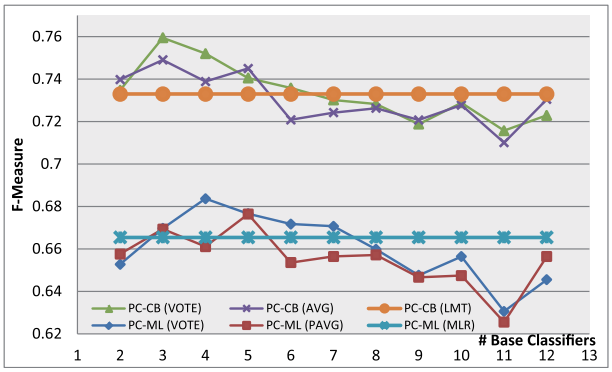


Fig. 7. Comparison of the ensemble methods against the best base classifiers in PC-ML and PC-CB.

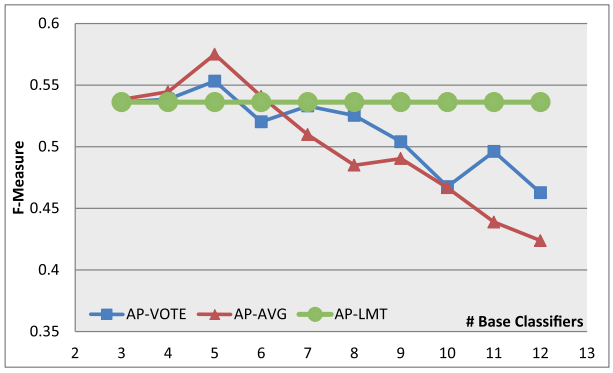


Fig. 8. Comparison of the ensemble methods against the best base classifier in AP-ML.

where the feature space can grow large as the number of documents increases (to handle the pattern and lexical diversity, etc.), all of our proposed methods scale well with document growth as the feature size is fixed.

4.4.3 Evaluation of AP Detection

Each sentence in the dataset *DS2* is labelled whether it is an AP indication sentence or not. The dataset DS2 contains 86 AP indication sentences and 74,278 non-indication sentences. 10-fold document-wise cross validation is used to evaluate our AP detection methods, using standard precision, recall, and F1 defined in Section 4.4.2 as the evaluation metrics. Table 5 lists notable results.

The best performance in terms of F1 is achieved by the ensemble machine learning based method (*AP-ML*) with the probability averaging of *NB*, *LMT*, *LLR*, *MLR*, and *RT* base classifiers, yielding 69.56 percent precision, 49.00 percent recall, and 57.50 percent F1. Fig. 8 compares the performances of the ensemble methods against the best base classifier (i.e., *LMT* classifier). The X-axis denotes the first k base classifiers used in each ensemble method. Similar to the analysis of the ensemble methods used in the PC detection, the ensemble of diverse base classifiers from distinct machine learning families allows individual experts to learn different aspects of the data, resulting in better collective decisions. The classification performance increases up until five base classifiers, then begins to decrease as addition of bad base classifiers could impede the ensemble decisions.

4.4.4 Evaluation on Algorithm Representation Linking

We evaluate our linking strategy on data set *DS2* in terms of accuracy, defined as:

$$Accuracy = \frac{\#\ Algorithm\ Representations\ Correctly\ Grouped}{\#\ All\ Algorithm\ Representations}.$$

We first identify all the PCs and APs from each document, then link them into groups of unique algorithms. Our linking algorithm achieves the accuracy of 85.15 percent.

5 Algorithm Metadata Extraction

The conventional search engine methodology usually handles textual documents. For an AP, corresponding textual information could be the AP itself. However, detected PCs

Pseudo-code

```
procedure INIT-PDA
{Invoked when the router comes up.}
begin
  Initialize all tables;
  call PDA;
end INIT-PDA

procedure PDA
{Executed at each router i. Invoked when an event occurs}
begin
  (1) call NTU;
  (2) call MTU; /* Updates T^i */
  (3) if (there are changes to T^i) then
        Compose an LSU message consisting of topology
        differences using add, delete
        and change link entries;
      endif
  (4) Within a finite amount time, send the
      LSU message to all neighbors;
end PDA
```

Figure 1: The Partial-topology Dissemination Algorithm

Caption Text

```
Figure 1: The Partial-topology Dissemination
```

Reference Sentences

```
The INIT-PDA procedure in Fig. 1 initializes the tables of a router at startup
time; all variables of type distance are initialized to infinity and those
of type node are initialized to null. Sim- ilarly, in Fig. 10, the delays
obtained using MP routing for NET1 are within 28% envelopes of delays obtained
using OPT routing. Fig. 11 compares the average delays of MP and SP for CAIRN.
In Fig. 12, for NET1, MP routing performs even better; average delays of SP
are as much as five to six times those of MP routing which is due to higher
connectivity available in NET1. For CAIRN, Fig. 13 show the effect of increasing
T l when T s and the input traffic is fixed. Similarly, for NET1, delays for SP
increased significantly while there is negligible change in delays of MP as can
be observed in Fig. 14, respectively. This becomes evident in Fig. 15, which
shows a typical response in NET1 when the flow rate is a step function (i.e..,
the flow rate is increased from 0 to a finite amount at time 0). Fig. 18 shows
the delays for SP and MP.
```

Fig. 9. Sample of primary textual metadata (i.e., caption text and reference sentences) associated with Algorithm *INIT-PDA*, represented by a pseudo-code in [36].

have no or little corresponding text. Furthermore, most PDF-to-Text tools are not designed for handling mathematical-like contents such as equations and pseudo-codes, due to complex compositions of symbols, font styles, and untypical decorations. This section describes a set of techniques for generating textual metadata for PCs.

For algorithms represented by PCs, textual metadata could include their captions and reference sentences. Fig. 9 illustrates the textual metadata relevant to the given PC that can be directly extracted, including the PC's caption text and its reference sentences.

Adequate and meaningful textual metadata allows effective retrieval. However, not all PCs have as rich textual

TABLE 6
Statistics about Primary Textual Metadata of the Extracted 6,285 Pseudo-Codes

Total # of Algorithms	6,285
Total # of Words in Caption Text	96,526
Total # of Words in Reference Sentences	605,606
Total # of Words in Primary Textual Metadata	304,233
Total # of Sentences in Primary Metadata	58,661
Avg # of Words in Caption Text	15.36
Avg # of Words in Reference Sentences	96.36
Avg # of Words in Primary Textual Metadata	48.41
Avg # of Sentences in Primary Metadata	9.33

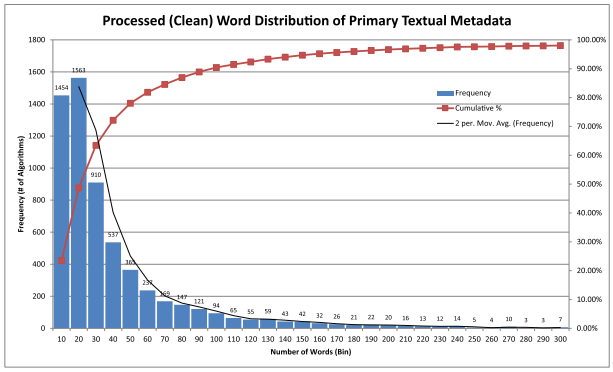


Fig. 10. Distribution of number of words in primary metadata of the extracted pseudo-codes.

metadata as the example in Fig. 9. From here on, we refer to the combination of caption text and reference sentences of a PC as its *primary* textual metadata. Table 6 breaks down the statistics of the primary textual metadata of the collection of 6,285 PCs extracted from randomly sampled 20,567 scholarly documents (Refer to Section 5.3). This data set is also used to validate the algorithm metadata extraction methodology in this section. Fig. 10 plots the distribution of words of the primary textual metadata records. From Table 6, the average number of words in each PC's primary textual metadata is only 48. Even worse, according to Fig. 10, roughly 50 percent of the extracted PCs have only 20 or fewer words in their primary textual metadata. These algorithms with poor textual metadata would likely be missed when being searched.

Recently, Bhatia and Mitra have proposed an algorithm for generating comprehensive descriptions for document elements (e.g., tables, figures, algorithms, etc.) [5]. This comprehensive description is referred to as a *synopsis*. These synopses have been shown to be useful for retrieval of these document elements, as they provide relevant textual information that can be indexed and searched. For a given document element, the algorithm operates in a supervise fashion that includes three main steps: 1) it retrieves the caption and reference sentences from the document, 2) it extracts representative features and trains a classifier, and 3) it classifies the remaining sentences in the document whether they are relevant or irrelevant to the given document element. The relevant sentences are then put together into a synopsis. Interested readers are suggested to consult [5] for further detail about this synopsis generation algorithm. In this paper, this synopsis generation technique is used to generate additional textual information for each PC.

However, the main drawback of the synopsis generation method is that it requires the document element to have a caption and at least a reference sentence. In our dataset, we find that 25.8 percent of the extracted PCs do not have accompanied captions. These PCs would not only suffer from poor primary textual metadata, but also from not having a synopsis as an additional textual information. Recently, Tuarob et al. proposed a set of methods that automatically annotates or enriches metadata records with poor textual information [32], [33]. The algorithm works well when these two criterion are met:

1) Each metadata record must have a main meaningful textual content. The main textual component allows the algorithm to find similar documents in the training collection to generate annotation.
2) Each metadata record that needs annotated must have some relationship with the collection of well-annotated metadata. This will guarantee that the annotation will be relevant since the it is modeled from the training data with similar topics.

We examine sample PCs with poor primary textual metadata and found that they have associated small piece of text composed with technical meaningful keywords. For PCs without captions, this would be the function names and surrounding sentences. This characteristic meets the first criteria above. Furthermore, recent studies show that algorithms are not normally composed from scratch, but are rather relevant to each other. Some algorithms inherit properties from existing ones [30]. Likewise, some algorithms are designed to solve similar sets of problems [29]. This also makes the algorithm metadata meets the second criteria.

In this section, we explore the possibility of using the synopsis generation [5] and the metadata annotation [33] methods to generate additional textual metadata for PCs. The following sections first provide brief introductions to the synopsis generation and metadata annotation algorithms, then show how they can be applied to the problem at hand.

5.1 Algorithm Textual Metadata Extraction via Document Element Summarization

Bhatia and Mitra have proposed an algorithm for automatic retrieval of synopsis for a document element that has a caption and reference sentences [5]. We explore the possibility of applying their document summarization technique on the extracted algorithm metadata in order to provide additional meaningful textual information for each algorithm. A document element is defined as an entity in the document that is not part of the running text and representing complementary information to the document. Examples of such document elements include figures, tables, and algorithms. Since not being part of the running text, a document element is usually accompanied with a caption so that the content in the running text can be used to refer to. While some authors compose tailored and detailed captions for document elements, generally, readers are assumed to have read the entire document in order to make sense of the document element within. When the caption is presented out of context as in a document-element-search-engine result, it may not contain enough information to help the end-user to understand what the content of the document element is. Consequently, a short "synopsis" of this information presented along with the document-element that helps end-users to examine the document element would be useful. Having access to the synopsis allows the end-user to quickly understand the content of the document-element without having to download and read the entire document as examining the synopsis takes a shorter time than finding information about a document element by downloading, opening and reading the file. Furthermore, it may allow the end-user to examine more results than they would otherwise. Pseudo-codes are another type of algorithm representation typically used in scholarly documents to condense and express algorithmic instructions. Typically, pseudo-codes are treated as document elements. Like other document elements, oftentimes, captions do not provide much comprehensive information for their accompanied pseudo-codes. Less textual information could impede effective retrieval of algorithms. Hence, obtaining more relevant textual information could promote semantic relatedness between user queries and the algorithms.

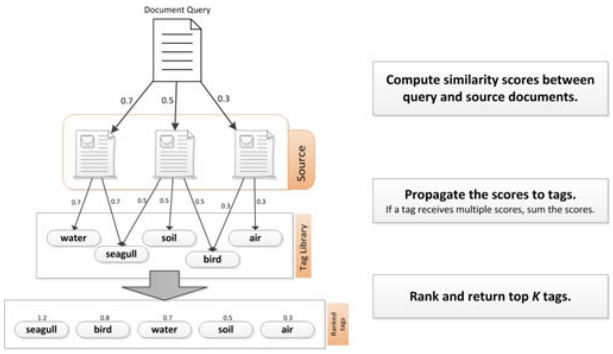


Fig. 11. A high-level illustration of the metadata annotation algorithm proposed by [33].

5.2 Automatic Algorithm Metadata Annotation via Probabilistic Topic Modeling

In this section, we describe the high-level concept of the topic modeling based metadata algorithm proposed in [33]. Interested readers are encouraged to consult the original paper for details.

Fig. 11 illustrates a flow of the score propagation algorithm on a simple example. In the example, we have a document query for which the system will recommend tags. Three documents in the source are annotated with tags {`water`, `seagull`}, {`seagull`, `soil`, `bird`}, and {`bird`, `air`} respectively. The algorithm proceeds as follows:

STEP1 The document similarity score is computed between the document query and each document in the source.

STEP2 The scores then are propagated to the tags in each source document.The scores are combined if a tag receives multiple scores. In the example, tags `seagull` and `bird` obtain multiple scores (0.7+0.5) and (0.5+0.3) respectively.

STEP3 The tags are ranked by the scores. Then the top `K` tags are returned as suggested tags.

Here we use the *topical similarity* to measure the document similarity, as suggested by Tuarob et al. [33]. We use Stanford Topic Modeling Toolbox[17] with the collapsed variational Bayes approximation [2] to identify topics in the source documents. For each document, we generate unigrams, bi-grams, and tri-grams, and combine them to represent the textual content of the document. The algorithm takes two input parameters: the number of topics to be

17. http://nlp.stanford.edu/software/tmt/tmt-0.4/

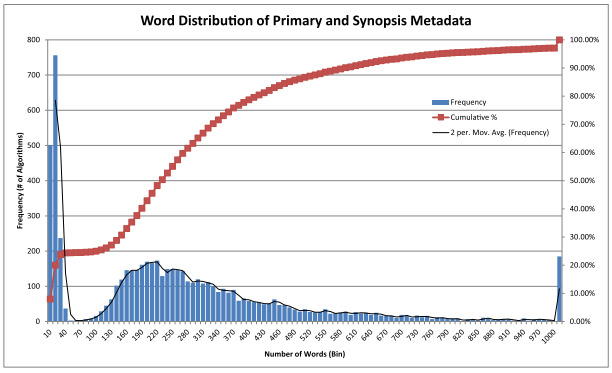


Fig. 12. Distribution of number words in primary and synopsis metadata (combined).

identified and the maximum number of the training iterations. Finally, the inference method proposed by [2] is used to assign a topic distribution to a given document.

Algorithm 1. A simple algorithm that transforms a collection of documents with auxiliary textual metadata (D') into a collection of documents with tag-like metadata (D).

Input: $D' = \{d'_1, d'_2, \ldots, d'_N\}$ where d'=$\langle c, a\rangle$
Output: $D = \{d_1, d_2, \ldots, d_N\}$ where d=$\langle c, e\rangle$, T = Tag Library

```
initialization;
T = ∅ ;
D = ∅ ;
for each d′ ∈ D′ do
    ⟨c, a⟩ = d′;
    a′ ← Clean(a);
    tokens ← Tokenize(a′);
    e ← RemoveDuplicates(tokens);
    Add e to T;
    Add ⟨c, e⟩ to D;
end
return D, T;
```

The metadata annotation algorithm was originally designed to recommend tags. In order to allow the algorithm to annotate textual metadata, we could first transform the textual component into tags. Given a collection of documents with auxiliary textual metadata $D' = \{d'_1, d'_2, \ldots, d'_N\}$ where $d' \in D' = \langle c, a\rangle$ (c is the main textual part of the document, and a is the associated auxiliary textual component), we would like to transform D' into $D = \{d_1, d_2, \ldots, d_N\}$ where $d \in D = \langle c, e\rangle$ (c remains the main textual part of the document, and e is the associated tags). This can be achieved using Algorithm 1.

Algorithm 1 takes the collection of documents with auxiliary textual components (D') as input and outputs the corresponding collection of documents with tag-like metadata (D) along with the tag library (T). Loosely speaking, the algorithm first cleans the auxiliary textual component by removing invalid characters, removing stopwords, and stemming, then treats each cleaned word as a tag. This transformation allows us to apply the tag-based document annotation algorithm on documents with auxiliary textual components, such as psudo-codes and their primary textual metadata.

Generated Synopsis

```
The INIT-PDA procedure in Fig. 1 initializes the tables of a router at startup
time; all variables of type distance are initialized to infinity and those
of type node are initialized to null. ... Similarly, in Fig. 10, the delays
obtained using MP routing for NET1 are within 28% envelopes of delays obtained
using OPT routing. ... Fig. 11 compares the average delays of MP and SP for
CAIRN. ... In Fig. 12, for NET1, MP routing performs even better; average delays
of SP are as much as five to six times those of MP routing which is due to
higher connectivity available in NET1. ... When connectivity is low or network
load is light, MP routing cannot offer any advantage over SP. Avg. Delay in
milliseconds Flow IDs Comparison of MP and SP delays 'OPT' 'MP-TL-10-TS-10'
'MP-TL-10-TS-2' Avg. Delay in milliseconds Flow IDs Comparison of MP and SP
delays 'OPT' 'MP-TL-10-TS-10' 'MP-TL-10-TS-2' 1 and Ts The performance of MP
depends on the update intervals T 1 and T s . ... Just the long-term routes
with load-balancing, without short-term routing parameter updates, 11.5 2 2.5
3 3.5 4 4.5 5 5.5 6 0 1 2 3 4 5 6 7 8 9 10 Avg. delay in milliseconds Flow IDs
Comparison of MP and SP delays 'MP-TL-10-TS-4' 'MP-TL-20-TS-4' 'SP-TL-10' is
kept constant and TL is increased in CAIRN. 2 4 6 8 10 12 14 16 18 20 22 0 1
2 3 4 5 6 7 8 9 Avg. delay in milliseconds Flow IDs Comparison of MP and SP
delays 'MP-TL-10-TS-4' 'MP-TL-20-TS-4' 'SP-TL-10' is kept constant and TL is
increased in NET1. seem to give significant gains; the major gains here are
due to the mere presence of multiple successors and load-balancing. ... This
becomes evident in Fig. 15, which shows a typical response in NET1 when the flow
rate is a step function (i.e., the flow rate is increased from 0 to a finite
amount at time 0). The dampened response of the network using MP indicates the
fast responsiveness of MP, making it suitable for dynamic environments. ...
The reason SP delays of these flows are better than those of MP is because of
uneven distribution of load in the network and low loads in some sections of the
network in low-load environments SP can perform slightly better than MP.
```

Fig. 13. Sample extracted synopsis corresponding to Algorithm *INIT-PDA* in Fig. 9, represented by a pseudo-code in [36].

5.3 Experiments on Algorithm Metadata Extraction

The data set used for evaluating and discussing the synopsis generation and metadata annotation comprises 6,285 PCs extracted from randomly sampled 20,567 scholarly documents. The statistics of the dataset is shown in Table 6.

5.3.1 Generating Pseudo-Code Synopses

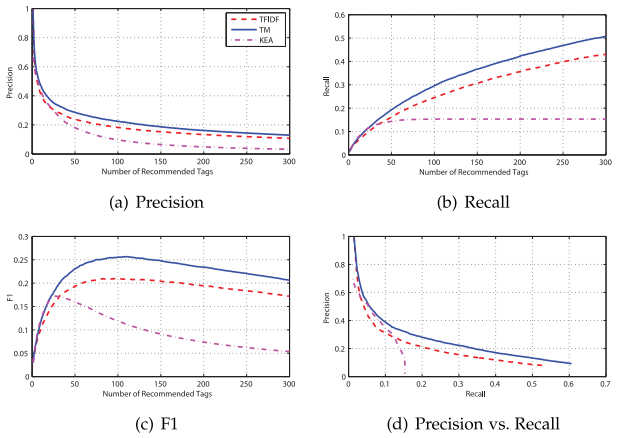
The synopsis generation technique is applied on the extracted 6,285 PCs. The average number of words per each generated synopsis is 190. The average number of words per each overall textual metadata (synopsis+primary textual metadata) has become 302, increasing by 170 percent. Fig. 12 plots the distribution of the sizes of combined textual metadata (in terms of number of words). The method fails to generate synopsis for 1,535 (24.42 percent) PCs. As expected, these are mostly PCs without captions. This proportion also matches with the proportion of PCs without captions (25.8 percent) in the dataset DS2 (See Section 4.4.1).

Fig. 13 shows a sample generated synopsis corresponding to Algorithm *INIT-PDA* in Fig. 9, represented by a PC in [36]. We can see that this particular synopsis provides relevant information to its corresponding PC that extends the information provided by the caption and reference sentences. Note that, the evaluation of the quality of synopsis generation for PCs is omitted since it was already presented in [5] (on a different PC dataset). This dataset (PCs+synopses) will be used to evaluate the metadata annotation algorithm in the next section.

5.3.2 Automatic Algorithm Metadata Annotation

Synopses provide additional meaningful textual information to each PC; however, the PCs whose synopses cannot be generated would remain poorly annotated. Recently, Tuarob et al. proposed a set of methods for automatic annotation of tag-like metadata via transferring topical knowledge from the well-annotated corpus [32], including Term Frequency-Inverse Document Frequency (*TFIDF*) based and Topic Modeling (*TM*) based document annotation algorithms, and validated these proposed algorithms against the baseline Keyphrase Extraction Algorithm (*KEA*) [25].

2,000 PC metadata records with rich synopses are randomly selected for our experiments using document-wise



(a) Precision (b) Recall

(c) F1 (d) Precision vs. Recall

Fig. 14. Precision, Recall, F1, and Precision versus Recall of the TF-IDF, TM, KEA (baseline) algorithms on the Algorithm textual metadata.

10 fold cross validation to evaluate the TFIDF, TM, and KEA metadata annotation methods. The word prediction evaluation protocol is carried out, where synopses are removed from the test documents, so that a system is evaluated based on how well it can predict the missing words in the actual synopses. Standard precision, recall, and F1 are used as evaluation metrics. The TM algorithm is trained with 300 topics and 3,000 iterations. Fig. 14 plots the precision, recall, F1, and precision-versus-recall at K (up to 300). From the results, while it is apparent that the TM algorithm outperforms the other two, the performance on the precision are not significantly different across the three methods. The KEA algorithm has an increasing recall until around $K = 30$, where it starts to remain steady, while both the TFIDF and TM algorithms continue to suggest relevant keywords (hence their recall rates continue to increase). According to Fig. 14c, the diminishing returns (i.e., the peaks) in F1 of the TFIDF, TM, and KEA algorithms start to appear at $K = 96$, 109, and 26 respectively. This suggests that the TFIDF and TM algorithms start to be less effective after roughly 100 recommended terms onwards. The number is relatively smaller for the KEA algorithm.

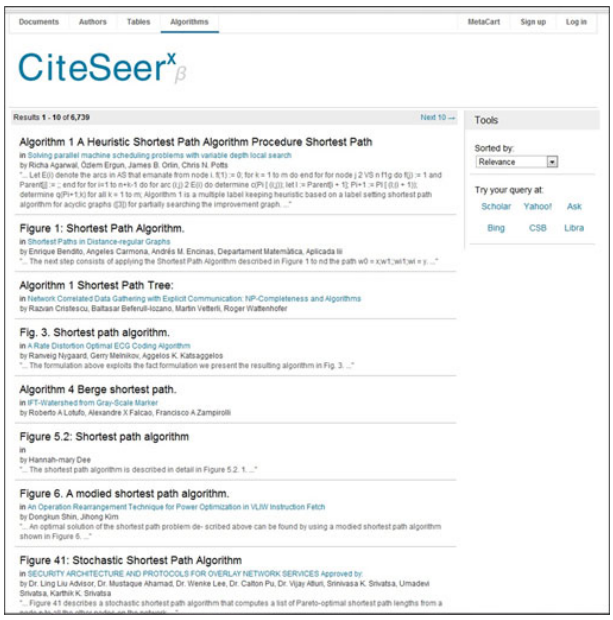


Fig. 16. Screenshot showing results for the query *"shortest path"*. Along with search results, associated metadata is also shown to the user.

Fig. 15 shows sample annotation of the three algorithms on the algorithm metadata of *"Algorithm 3.44 Fixed-base comb method for point multiplication"* that appears in [13]. The top left part of the figure shows the actual PC. The top right part illustrates the partial *actual* synopsis. The bottom part of the figure shows the top 30 terms recommended by the three algorithms which aim to predict the actual synopsis text. Note that these terms are stemmed as part of the preprocessing. The red bold terms are the correctly predicted ones.

In this particular example, the TM algorithm seems to be most effective since it can correctly predict 27/30 words. The KEA algorithm seems to recommend more meaningful terms; however, most of them are incorrectly predicted. Note that, since the tag prediction protocol used for the evaluation measures how well an annotator *guesses* the omitted text, it is not necessary the case that incorrectly predicted

Algorithm 3.44 Fixed-base comb method for point multiplication

INPUT: Window width $w, d = \lceil t/w \rceil, k = (k_{t-1}, \ldots, k_1, k_0)_2, P \in E(\mathbb{F}_q)$.
OUTPUT: kP.

1. *Precomputation*. Compute $[a_{w-1}, \ldots, a_1, a_0]P$ for all bit strings $(a_{w-1}, \ldots, a_1, a_0)$ of length w.
2. By padding k on the left with 0s if necessary, write $k = K^{w-1} \| \cdots \| K^1 \| K^0$, where each K^j is a bit string of length d. Let K_i^j denote the ith bit of K^j.
3. $Q \leftarrow \infty$.
4. For i from $d-1$ downto 0 do
   4.1 $Q \leftarrow 2Q$.
   4.2 $Q \leftarrow Q + [K_i^{w-1}, \ldots, K_i^1, K_i^0]P$.
5. Return(Q).

Actual Synopsys (Partial)

In the fixed-base comb method the binary representation of k is first padded on the left with dw− t 0s, and is then divided into w bit strings each of the same length d so that k = Kw−1l · · · lK 1 lK 0. As an example, LD coordinates in the binary field case give A/D ≃ 2, requiring (roughly) w ≥ 6…In this case, the exponent array for k is processed using interleaving (Algorithm 3.51), with k j given by k =∑wj=1 k j2(j−1)d and points Pj given by Pj = 2(j−1)d P , 1 ≤ j ≤ w … For our case, and for relatively modest amounts of storage, the single-table comb method is among the fastest and can be used to obtain meaningful operation count comparisons. Point multiplication costs 143 Points EC operations Field operations Method Coordinates w stored A D M I Totala Unknown …

TFIDF	TM	KEA
input comput time method set **point** step **base requir number multipl case oper addit line** perform result output **process approxim cost run obtain** section **assum field** function return show **note**	**input multipl comput method curv point binari addit naf mod window time requir field** integ **oper ellipt coordin base case left width precomput** step **approxim kp tnaf fix cost** output	**doubl jacobian** repeat **coordin point field** consecut **addit** perform yield speedup ordinari faster formula trade sion slight squar **comparison applic** modifi **note multipl input ellipt curv** acceler combin strategi improv

Fig. 15. Sample top 30 suggested words by TFIDF, TM, and KEA (Baseline) algorithms on the metadata record of Algorithm 3.44 mentioned in [13].

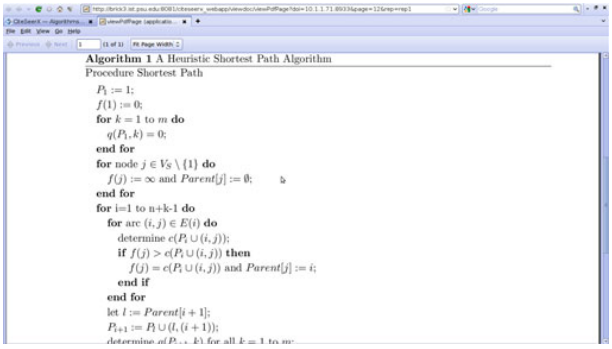


Fig. 17. Screenshots showing algorithm page displayed on clicking the first result.

terms are not relevant. They just do not appear in the actual synopsis. The TFIDF algorithm performs better than the KEA algorithm, but is not as effective as the TM algorithm.

6 Indexing and Searching for Algorithms

All the extracted algorithms and their associated textual metadata are then indexed using SOLR[18], which then makes the algorithms searchable. 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. A TF-IDF based cosine similarity is used to rank search results.

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 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 percent at top 10 ranks as compared to 41 and 44 percent achieved by Google Web Search and Google Scholar, respectively.

Fig. 16 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 Fig. 17.

Note that, though the indexing and searching methodology presented in this section has been successfully employed on general text-based document retrieval tasks, we are aware that searching for algorithms can be much more sophisticated, needing specialized searching capability that narrows down the set of desired algorithms. Hence, the main purpose of this section is to illustrate an initial prototype search system that makes use of the extracted textual metadata of pseudo-codes. Extracting and utilizing algorithm-specific metadata will be explored as the future works.

18. http://lucene.apache.org/solr/

7 Conclusions and Future Work

Algorithms play an important part in solving research problems. Scientific publications host a tremendous amount of such high-quality algorithms developed by professional researchers. In digital libraries, being able to extract and catalog these algorithms would introduce a number of exciting applications including algorithm searching, discovering, and analyzing. We discussed the prototype of *AlgorithmSeer*, a search system for algorithms in large scale scholarly documents, along with providing an illustration of the actual demo system. Specifically, we proposed a set of scalable machine learning based methods to detect algorithms in scholarly documents, we discussed using the synopsis generation and document annotation methods to extract textual metadata for pseudo-codes, and finally we explained how algorithms are indexed and made searchable. Future work would be to further explore the semantic analysis of algorithms, their trends, and how algorithms influence each other over time. Such analyses would give rise to multiple applications that could improve algorithm search.

Acknowledgments

We gratefully acknowledge partial support from the National Science Foundation. Suppawong Tuarob is the corresponding author. This manuscript is an extension of the authors' earlier work presented at WWW 2010 [6], SUITE 2011 [7], and ICDAR 2013 [28].

References

[1] S. Guha, N. Koudas, "Approximating a data stream for querying and estimation: Algorithms and performance evaluation," in *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 567–576.
[2] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, "On smoothing and inference for topic models," in *Proc. 25th Conf. Uncertainty Artif. Intell.*, 2009, pp. 27–34.
[3] J. B. Baker, A. P. Sexton, V. Sorge, and M. Suzuki, "Comparing approaches to mathematical document analysis from PDF," in *Proc. Int. Conf. Document Anal. Recog.*, 2011, pp. 463–467.
[4] S. Bhatia and P. Mitra, "Summarizing figures, tables, and algorithms in scientific publications to augment search results," *ACM Trans. Inf. Syst.*, vol. 30, no. 1, pp. 3:1–3:24, Mar. 2012.
[5] S. Bhatia and P. Mitra, "Summarizing figures, tables, and algorithms in scientific publications to augment search results," *ACM Trans. Inf. Syst.*, vol. 30, no. 1, pp. 3:1–3:24, Mar. 2012.
[6] S. Bhatia, P. Mitra, and C. L. Giles, "Finding algorithms in scientific articles," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 1061–1062.
[7] S. Bhatia, S. Tuarob, P. Mitra, and C. L. Giles, "An algorithm search engine for software developers," in *Proc. 3rd Int. Workshop Search-Driven Develop. Users, Infrastructure, Tools, Evaluation*, 2011, pp. 13–16.
[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[9] L. Breiman, "Random forest," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 6, no. 1, pp. 321–357, 2002.
[11] H.-H. Chen, L. Gou, X. Zhang, and C. L. Giles, "Collabseer: A search engine for collaboration discovery," in *Proc. 11th Annu. Int. ACM/IEEE Joint Conf. Digital libraries*, 2011, pp. 231–240.
[12] P. Chiu, F. Chen, and L. Denoue, "Picture detection in document page images," in *Proc. 10th ACM Symp. Document Eng.*, 2010, pp. 211–214.

[13] D. Hankerson, S. Vanstone, and A. J. Menezes, *Guide to Elliptic Curve Cryptography*. Berlin, Germany: Springer, 2004.
[14] T. Hassan, "Object-level document analysis of pdf files," in *Proc. 9th ACM Symp. Document Eng.*, 2009, pp. 47–55.
[15] M. A. Hearst, A. Divoli, H. Guturu, A. Ksikes, P. Nakov, M. A. Wooldridge, and J. Ye, "BioText search engine: Beyond abstract search," *Bioinformatics*, vol. 23, no. 16, pp. 2196–2197, 2007.
[16] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Commun. ACM*, vol. 18, no. 6, pp. 341–343, 1975.
[17] S. Kataria, W. Browuer, P. Mitra, and C. L. Giles, "Automatic extraction of data points and text blocks from two-dimensional plots in digital documents," in *Proc. 23rd Nat. Conf. Artif. Intell. - Volume 2*, 2008, pp. 1169–1174.
[18] M. Khabsa, P. Treeratpituk, and C. L. Giles, "Ackseer: A repository and search engine for automatically extracted acknowledgments from digital libraries," in *Proc. 12th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2012, pp. 185–194.
[19] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 3, pp. 226–239, Mar. 1998.
[20] G. W. Klau, I. Ljubić, P. Mutzel, U. Pferschy, and R. Weiskircher, *The Fractional Prize-Collecting Steiner Tree Problem on Trees*. Berlin, Germany: Springer, 2003.
[21] J. M. Kleinberg and E. Tardos, *Algorithm Design*, volume 30, Reading, MA, USA: Addison Wesley, 2005.
[22] H. Li, I. Councill, W.-C. Lee, and C. L. Giles, "Citeseerx: An architecture and web service design for an academic document search engine," in *Proc. 15th Int. Conf. World Wide Web*, 2006, pp. 883–884.
[23] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, "Tableseer: Automatic table metadata extraction and searching in digital libraries," in *Proc. 7th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2007, pp. 91–100.
[24] S. Mandal, S. P. Chowdhury, A. K. Das, and B. Chanda, "Automated detection and segmentation of table of contents page from document images," in *Proc. 12th Int. Conf. Image Anal. Process.*, 2003, pp. 213–218.
[25] O. Medelyan and I. H. Witten, "Thesaurus based automatic keyphrase indexing," in *Proc. 6th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2006, pp. 296–297.
[26] P. Sojka and M. Líška, "The art of mathematics retrieval," in *Proc. ACM Symp. Document Eng.*, 2011, pp. 57–60.
[27] J. G. Stewart and J. Callan, "Genre oriented summarization," PhD thesis, Carnegie Mellon Univ., Pittsburgh, PA, 2009.
[28] S. Tuarob, S. Bhatia, P. Mitra, and C. L. Giles, "Automatic detection of pseudocodes in scholarly documents using machine learning," in *Proc. 12th Int. Conf. Document Anal. Recog.*, 2013, pp. 738–742.
[29] S. Tuarob, P. Mitra, and C. L. Giles, "Improving algorithm search using the algorithm co-citation network," in *Proc. 12th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2012, pp. 277–280.
[30] S. Tuarob, P. Mitra, and C. L. Giles, "A classification scheme for algorithm citation function in scholarly works," in *Proc. 13th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2013, pp. 367–368.
[31] S. Tuarob, P. Mitra, and C. L. Giles, "A hybrid approach to discover semantic hierarchical sections in scholarly documents," in *Proc. 13th Int. Conf. Document Anal. Recog.* 2015, pp. 1081–1085.
[32] S. Tuarob, L. Pouchard, P. Mitra, and C. Giles, "A generalized topic modeling approach for automatic document annotation," *Int. J. Digital Libraries*, vol. 16, no. 2, pp. 1–18, 2015.
[33] S. Tuarob, L. C. Pouchard, and C. L. Giles, "Automatic tag recommendation for metadata annotation using probabilistic topic modeling," in *Proc. 13th ACM/IEEE-CS Joint Conf. Digital Libraries*, 2013, pp. 239–248.
[34] S. Tuarob and C. S. Tucker, "Fad or here to stay: Predicting product market adoption and longevity using large scale, social media data," in *Proc. ASME Int. Des. Eng. Technical Conf. Comput. Inform. Eng. Conf.*, 2013, Available: http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?articleid=1830257
[35] S. Tuarob and C. S. Tucker, "Quantifying product favorability and extracting notable product features using large scale social media data," *J. Comput. Inform. Sci. Eng.*, vol. 15, no. 3, 2015, Available: http://computingengineering.asmedigitalcollection.asme.org/article.aspx?articleid=2090327
[36] S. Vutukury and J. J. Garcia-Luna-Aceves, "A simple approximation to minimum-delay routing," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 227–238, Aug. 1999.
[37] J. Wang, "Mean-variance analysis: A new document ranking theory in information retrieval," in *Proc. 31th Eur. Conf. IR Res. Adv. Inform. Retrieval*, 2009, pp. 4–16.
[38] Z. Wu, S. Das, Z. Li, P. Mitra, and C. L. Giles, "Searching online book documents and analyzing book citations," in *Proc. ACM Symp. Document Eng.*, 2013, pp. 81–90.
[39] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *Int. J. Document Anal. Recog.*, vol. 15, no. 4, pp. 1–27, 2012.

Suppawong Tuarob received the BSE and MSE degrees in computer science and engineering from the University of Michigan-Ann Arbor, the MS degree in industrial engineering from the Pennsylvania State University, and the PhD degree in computer science and engineering from the Penn State University. He is currently a faculty member at the Faculty of Information and Communication Technology, Mahidol University, Thailand. His research interests include data mining in large scale scholarly, social media, and healthcare domains, by applying multiple cutting-edge techniques such as machine learning, topic modeling, sentiment analysis, etc. He is a member of the IEEE.

Sumit Bhatia is a research scientist in the Watson group, IBM Almaden Research Centre where he is leading the development of cognitive analytic algorithms build on top of Watson's Knowledge Graph. Previously, he was a postdoctoral researcher at Xerox PARC and as a part of CiteSeerX project at Penn State, he developed a search engine that searches for algorithms and pseudo-codes in academic documents. His primary research interests include the fields of information retrieval and text analytics and he has published more than 25 papers in top journals and conferences. He was the organizing chair of the Social Multimedia Data Mining Workshop, collocated with ICDM 2014. He has served as a reviewer for multiple conferences and journals including WWW, CIKM, TKDE, TOIS, WebDB, JASIST, IJCAI, and AAAI.

Prasenjit Mitra received the PhD degree from the Stanford University, Stanford, CA, in electrical engineering in 2004. He is currently a principal scientist at the Qatar Computing Research Institute, Doha, Qatar, and a professor in the College of Information Sciences and Technology, Pennsylvania State University, University Park. He also serves on the graduate faculty of the Department of Computer Sciences and Engineering and is an affiliate faculty member of the Department of Industrial and Manufacturing Engineering there. His research interests include text mining, applied machine learning, digital libraries, artificial intelligence, information retrieval, database systems, social computing, visual analytics, and health informatics. He has also worked as a Senior Member of the Technical Staff at Oracle Corporation and at DBWizards and Narus, Inc. He is the author more than 100 articles in leading journals and peer-reviewed conference proceedings. He is also a member of the ACM and AAAI. He is a senior member of the IEEE.

C. Lee Giles is the David Reese professor at the College of Information Sciences and Technology, Pennsylvania State University. He is also professor of computer science and engineering, professor of supply chain and information systems, and director of the Intelligent Systems Research Laboratory. He directs the CiteSeer^X project and codirects the Chem$_X$Seer project at Penn State. He is a fellow of the IEEE.