2013 12th International Conference on Document Analysis and Recognition

# Automatic Detection of Pseudocodes in Scholarly Documents Using Machine Learning

Suppawong Tuarob[†], Sumit Bhatia[†], Prasenjit Mitra[†‡], C. Lee Giles[†‡]
[†]Computer Science and Engineering, [‡]Information Sciences and Technology
Pennsylvania State University, University Park, PA 16802, USA
suppawong@psu.edu, sumit@cse.psu.edu, pmitra@ist.psu.edu, giles@ist.psu.edu

***Abstract*—A significant number of scholarly articles in computer science and other disciplines contain algorithms that provide concise descriptions for solving a wide variety of computational problems. For example, Dijkstra's algorithm describes how to find the shortest paths between two nodes in a graph. Automatic identification and extraction of these algorithms from scholarly digital documents would enable automatic algorithm indexing, searching, analysis and discovery. An algorithm search engine, which identifies pseudocodes in scholarly documents and makes them searchable, has been implemented as a part of the CiteSeer$^X$ suite. Here, we illustrate the limitations of start-of-the-art rule based pseudocode detection approach, and present a novel set of machine learning based techniques that extend previous methods.**

## I. Introduction

Algorithms are ubiquitous in Computer Science and related literature. They offer concise stepwise instructions for solving many computational problems such as searching, sorting, hashing, clustering, decoding, machine learning, etc. Furthermore, in various fields other than Computer Science, efficient solutions to important problems involve transforming the problem into an algorithmic one, often use fairly standard algorithms from other fields. For example, algorithms for stock portfolio optimization are used for diversifying search results in information retrieval systems [1]. Likewise, in bioinformatics, Hirschberg's algorithm [2] is widely used to find maximal global alignments of DNA and protein sequences. In addition, a thorough knowledge of state-of-the-art algorithms is also crucial for developing efficient software systems.

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

TABLE I
Approximate number of algorithms published in computer science conferences 2005 - 2009.

### A. Algorithms in Scholarly Documents

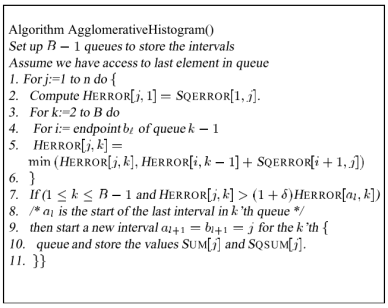
Researchers are constantly developing new algorithms to either solve new problems that have not been solved before, or algorithms that improve upon existing ones. Often, researchers report their new algorithms in scientific publications. Bhatia et al. [3] provide an estimate of the number of algorithms published in some major computer science conferences during 2005 - 2009, which we reproduce here (Table *I*) for reference. With dozens of new algorithms being reported in these conferences every year, it is crucial to have systems that automatically identify, extract, index and search the ever increasing collection of algorithms, both new and old. Such systems can prove useful to researchers and software developers looking for cutting-edge solutions to their problems.

Finding well-known standard algorithms is not difficult, as they are usually already cataloged and made searchable, especially those in online catalogs. We define a standard algorithm as an algorithm that is well known and is usually recognized by its name. Examples of standard algorithms include Dijkstra's shortest-path algorithm, Bellman-Ford algorithm, Quicksort algorithm, and Knuth-Morris-Pratt algorithm. Standard algorithms are usually collected and cataloged manually in algorithm textbooks (e.g. [4]), encyclopedias (especially the ones available online such as Wikipedia[1]), and websites targeted at computer programmers (e.g. Rosettacode.org[2]). As an initial survey, we parsed Wikipedia algorithm pages in 2010, and found that roughly, at the time of this paper, 1,765 standard algorithms are cataloged in Wikipedia.org. The National Institute of Standards and Technology (NIST)[3] also has a dictionary of over 289 standard algorithms. However, unlike these well-known standard algorithms, newly published algorithms are not cataloged by the sources mentioned above, because they are simply too *new* and too *many*, making it also difficult to manually catalog them.

Manually searching for these newly published algorithms is a nontrivial task. Researchers and others who aim to discover efficient and innovative algorithms usually actively search and monitor relevant new publications, usually in their fields, to keep abreast of the latest algorithmic developments. Having to read an entire document is tedious. The problem is more problematic if algorithm searchers are novices in document search, especially those who choose poor search keyword(s). Thus to alleviate this problem, we propose automatic identification and extraction of algorithms, in particular their pseudocode since many algorithms are written as such, from digital documents.

[1]http://www.wikipedia.org/
[2]http://rosettacode.org/wiki/Rosetta_Code/
[3]http://xlinux.nist.gov/dads/

1520-5363/13 $26.00 © 2013 IEEE
DOI 10.1109/ICDAR.2013.151



```
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 HERROR[j, 1] = SQERROR[1, j].
3.   For k:=2 to B do
4.     For i:= endpoint b_ℓ of queue k − 1
5.     HERROR[j, k] =
       min (HERROR[j, k], HERROR[i, k − 1] + SQERROR[i + 1, j])
6.   }
7.   If (1 ≤ k ≤ B − 1 and HERROR[j, k] > (1 + δ)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 SUM[j] and SQSUM[j].
11.  }}
```

**Figure 3. Algorithm AgglomerativeHistogram**

Fig. 1. Example pseudocode, taken from [6]

```
DEEPSET(A, c)
1: for g = 1 to c + 1 do
2:    A_{i,j} ← maximum entry of A
3:    x ← (c + 1)-st largest entry in A_{i,⋆}
4:    T̃ ← T̃ ∪ {k : A_{i,k} ≥ x} {* mark c + 1 largest entries in row/column i *}
5:    x ← (c + 1)-st largest entry in A_{j,⋆}
6:    T̃ ← T̃ ∪ {k : A_{k,j} ≥ x} {* mark c + 1 largest entries in row/column j *}
7:    A ← A \ {i, j} {* delete i and j *}
8: return T̃
```

Fig. 2. Example pseudocode without a caption, taken from [7]

### B. State-of-the-Art in Algorithm Detection

Identifying and extracting various informative entities from scholarly documents is an active area of research. For algorithm detection, Bhatia et al. [5] briefly describe methods for automatic detection of *pseudocodes* in Computer Science documents. Their method assumes that each pseudocode is accompanied by a caption. An example of a pseudocode with a caption is given in Figure 1. Such a pseudocode can then be identified using a set of regular expressions to detect the presence of the accompanied caption [3], [5]. Such an approach, however, is limited in its coverage due to its reliance on the presence of pseudocode captions and wide variations in writing styles. From our dataset (DS2) of 258 scholarly documents (see Sect. III), we found 275 pseudocodes, 25.8% (71 out of 275) of which did not have an accompanied caption. Figure 2 shows an example of a pseudocode without a caption. Thus, these pseudocodes will remain undetected by their approach.

Since algorithms represented in documents do not conform to specific styles, and are written in arbitrary formats, this becomes a challenge for effective detection and extraction. Here we improve the performance of pseudocode detection by capturing both pseudocodes with and without captions.

### C. Our Contributions

This work has the following key contributions:

1) We propose three methods for detecting pseudocodes in scholarly documents, including an extension of the existing rule-based method proposed by Bhatia et al. [5], one based on machine learning techniques, and a combination of these two.
2) We use two datasets of scholarly documents selected from Citeseer$^X$ repository: one for identifying rules and features, one for evaluation (the datasets are available for research purposes).
3) We evaluate our proposed methods on a dataset of 258 scholarly PDF documents selected from the Citeseer$^X$ repository.

## II. Background and Related Work

Identifying and extracting informative entities such as mathematical expressions [8], [9], tables [10], [11], and figures [12] from documents has been extensively studied. Image processing and Optical Character Recognition (OCR) have been used for automatic extraction of data points and text blocks from 2-D plots in PDFs [12]. They also propose a way to index the extracted information and make it available through a search interface. *TableSeer* [13] automatically identifies and extracts tables in digital documents. BioText[4] search engine, a specialized search engine for biology documents, also offers the capability to extract figures and tables [14]. Bhatia et al. [3], [5] propose a set of methods used for detecting document-elements, e.g. tables, figures, and algorithms. Their methods rely on the assumption that the document-elements are presented along with captions. They detect the presence of a document-element by detecting the corresponding caption using a set of regular expressions. Bhatia et al. [3] propose an algorithm search engine for software developers. Their system collects pseudocodes available in scholarly documents and make them searchable via full text search powered by Solr/Lucene[5]. Our work here extends their pseudocode detection approach.

## III. Datasets

Two datasets are used in this paper. The first dataset (DS1) contains 100 scholarly documents manually selected from the Citeseer$^X$ repository to cover diverse types of pseudocodes. This dataset is used to construct rules and regular expressions for our rule-based methods, and determine feature sets for our machine-learning based methods. The other dataset (DS2) consists of 258 scholarly PDF documents randomly selected from Citeseer$^X$ consisting of 275 pseudocodes. It is used for validation.

### A. Preprocessing

Textual information was extracted from each PDF document using PDFBox[6]. Experiments across text extraction tools (i.e. PDFTextStream[7], Xpdf[8], TET[9], and PDFBox) found PDFBox to be the most suitable because it best preserves line sequences. We modified the source code in PDFBox to also extract font size information from each text line.

[4]http://biosearch.berkeley.edu
[5]http://lucene.apache.org/solr/
[6]http://pdfbox.apache.org/
[7]http://snowtide.com/PDFTextStream
[8]http://www.foolabs.com/xpdf
[9]http://www.pdflib.com/products/tet/

### B. Data Labeling

A document is treated as a sequence of text lines, each identified with a line number. We manually label each line as follow:

0 Not part of pseudocode content
1 Part of pseudocode content

Note that a pseudocode caption is treated as pseudocode content. A line labeled with *1* is said to be a positive line, otherwise it is negative. A pseudocode is defined as a set of consecutive positive lines.

## IV. Our Approaches

```
<CAPTION> ::= <DOC_EL_TYPE> <Integer> <DELIMITER> <TEXT>
<DOC_EL_TYPE> ::= <FIG_TYPE> | <TABLE_TYPE> | <ALGO_TYPE>
<FIG_TYPE> ::= FIGURE|Figure|FIG.|Fig.
<TABLE_TYPE> ::= TABLE|Table
<ALGO_TYPE> ::= Algorithm|algorithm|Algo.|algo.
<DELIMITER> ::= : | .
<TEXT> ::= <A String of Characters>
```

TABLE II
A grammar for document-element captions

Many scientific documents use pseudocodes for compact and concise illustrations of algorithms. Pseudocodes are normally treated as document elements separate from the running text, and usually are accompanied with identifiers such as captions, function names, and/or algorithm names. Since pseudocodes 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 we present three algorithms for detecting pseudocodes in scholarly documents: rule based (PC-RB), machine learning based (PC-ML), and combined (PC-CB) methods.

### A. Rule Based Method (PC-RB)

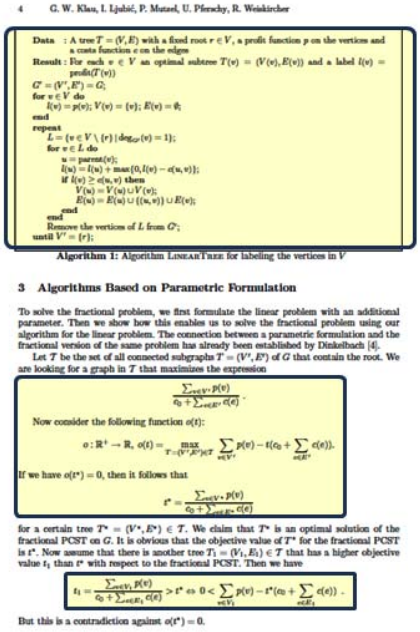
We previously proposed a rule-based pseudocode detection algorithm [3], [15], [16], which utilizes a grammar for document-element captions to detect the presence of pseudocode captions (See Table *II*). Here, we extend the previous approach by adding the following rules to improve the coverage and reduce false positives:

- A pseudocode caption must contain at least one algorithm keyword, namely *pseudocode*, *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 pseudocodes.

Hence, given a document, the PC-RB method outputs a set of line numbers, each of which represents a pseudocode caption.

### B. Machine Learning Based Method (PC-ML)

The PC-RB method yields high precision, however it still suffers from low coverage resulting in poor recall. We found that 25.8% of pseudocodes in our dataset DS2 do not have accompanied captions. These pseudocodes would remain undetected by the PC-RB method. To correct this, we propose a machine learning based (PC-ML) method to directly detect the presence of pseudocode content (instead of their captions). This originates from the observation that most pseudocodes are written in a sparse manner, resulting in sparse regions in documents. We call such sparse regions *sparse boxes*. The PC-ML method first detects and extracts these sparse boxes, then classifies each box whether it is a pseudocode or not. The following subsections explain the sparse box identification, the feature sets, and the classification algorithms used. Given a document, the PC-ML method outputs a set of tuples of $\langle start, end \rangle$ line numbers, each of which represents the start and end lines of a pseudocode.


4 G. W. Klau, I. Ljubić, P. Mutzel, U. Pferschy, R. Weiskircher

**Data** : A tree $T = (V, E)$ with a fixed root $r \in V$, a profit function $p$ on the vertices and a costs function $c$ on the edges
**Result** : For each $v \in V$ an optimal subtree $T(v) = (V(v), E(v))$ and a label $l(v) = profit(T(v))$
$G' = (V', E') = G$;
**for** $v \in V$ **do**
  $l(v) = p(v)$; $V(v) = \{v\}$; $E(v) = \emptyset$;
**end**
**repeat**
  $L = \{v \in V \setminus \{r\} \mid \deg_{G'}(v) = 1\}$;
  **for** $v \in L$ **do**
    $u = parent(v)$;
    $l(u) = l(u) + \max\{0, l(v) - c(u, v)\}$;
    **if** $l(v) \geq c(u, v)$ **then**
      $V(u) = V(u) \cup V(v)$;
      $E(u) = E(u) \cup \{(u, v)\} \cup E(v)$;
    **end**
  **end**
  Remove the vertices of $L$ from $G'$;
**until** $V' = \{r\}$;

**Algorithm 1:** Algorithm LinearTree for labeling the vertices in $V$

**3 Algorithms Based on Parametric Formulation**

To solve the fractional problem, we first formulate the linear problem with an additional parameter. Then we show how this enables us to solve the fractional problem using our algorithm for the linear problem. The connection between a parametric formulation and the fractional version of the same problem has already been established by Dinkelbach [4].

Let $\mathcal{T}$ be the set of all connected subgraphs $T = (V', E')$ of $G$ that contain the root. We are looking for a graph in $\mathcal{T}$ that maximizes the expression

$$\frac{\sum_{v \in V'} p(v)}{c_0 + \sum_{e \in E'} c(e)}.$$

Now consider the following function $o(t)$:

$$o : \mathbb{R}^+ \to \mathbb{R},\ o(t) = \max_{T=(V',E') \in \mathcal{T}} \sum_{v \in V'} p(v) - t(c_0 + \sum_{e \in E'} c(e)).$$

If we have $o(t^*) = 0$, then it follows that

$$t^* = \frac{\sum_{v \in V^*} p(v)}{c_0 + \sum_{e \in E^*} c(e)}$$

for a certain tree $T^* = (V^*, E^*) \in \mathcal{T}$. We claim that $T^*$ is an optimal solution of the fractional PCST on $G$. It is obvious that the objective value of $T^*$ for the fractional PCST is $t^*$. Now assume that there is another tree $T_1 = (V_1, E_1) \in \mathcal{T}$ that has a higher objective value $t_1$ than $t^*$ with respect to the fractional PCST. Then we have

$$t_1 = \frac{\sum_{v \in V_1} p(v)}{c_0 + \sum_{e \in E_1} c(e)} > t^* \Leftrightarrow 0 < \sum_{v \in V_1} p(v) - t^*(c_0 + \sum_{e \in E_1} c(e)).$$

But this is a contradiction against $o(t^*) = 0$.

Fig. 3. Example of sparse regions (sparse boxes)

*1) Sparse Box Extraction:* We define a sparse box as a set of at least *N* consecutive sparse lines. Figure 3 shows an example of sparse boxes. A sparse line is a line whose ratio of the number of non-space characters to the average number of characters per line is less than threshold *M*. We found that *N* = 4 and *M* = 0.8 work best for our dataset DS2. We evaluate our sparse box extraction method in two perspectives: *coverage* and *accuracy*. Given a set of sparse boxes $B$ extracted from a document $d$, the *coverage* is defined as following:

$$Coverage = \frac{|\{l | l \in b, b \in B, l\ is\ positive\}|}{|\{l | l \in b, b \in B\}|}$$

The coverage utilizes line-wise recall to quantify how much pseudocode content can be captured within the extracted

sparse boxes. Since the sparse box detection process filters in candidates for pseudocodes, it is more favorable to have high coverage to minimize the loss of actual pseudocode boxes. Our sparse box extraction method yields a coverage of 92.99%. Among all the sparse boxes detected in our dataset, we found 237 (out of 275 (86.18%) actual pseudo codes) pseudocode boxes.

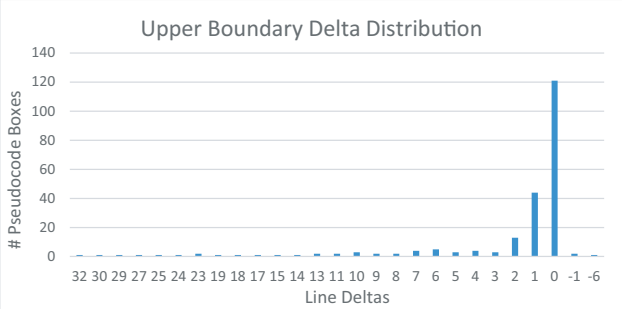



Fig. 4. Distribution of upper boundary deltas of pseudocode boxes

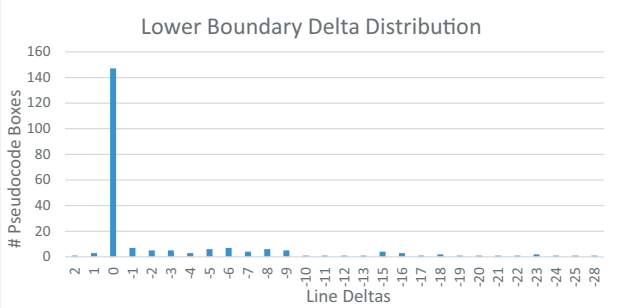



Fig. 5. Distribution of lower boundary deltas of pseudocode boxes

The *accuracy* is measured using the delta evaluation for the pseudocode boxes. For each pseudocode box, we measure both the upper boundary delta (the start line number of the actual pseudocode minus the start line number of the sparse box) and lower boundary delta (the end line number of the actual pseudocode minus the end line number of the sparse box). Figures 4 and 5 show the upper and lower boundary delta distributions of the 237 pseudocode boxes.

*2) Feature Sets:* We extract 47 features from each of the sparse boxes. 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 the various font styles used in pseudocodes. The *CX* features detect the presence of pseudocode captions. The *CN* features capture the pseudocode specific keywords and coding styles. The *ST* features characterize the sparsity of pseudocodes and the symbols used.

*3) Pseudocode Classification:* Each detected sparse box is classified whether it is a pseudocode box or not. We try 12 baseline classification algorithms namely, 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 baseline classifiers listed above, we also try ensemble methods such as uniform weighted majority voting and probability averaging methods among these base classifiers. First, the 12 base classifiers are tested and ranked by their precision, recall, and F1 scores. Then, the first 2, 3, ..., 12 ranked classifiers in each ranked list are used for majority voting and probability averaging methods. Note that we also try other ensemble methods such as AdaBoost, Bagging, and Rotation Forest but overall the majority voting and probability averaging methods perform much better.

### C. Combined Method (PC-CB)

Though the PC-ML method can capture the pseudocodes that do not have accompanied captions, some pseudocodes that are not first captured in one of the sparse boxes would still remain undetected. Mostly, such pseudocodes are either written in a descriptive manner (hence do not result in sparse regions in the document), or are figures (the text extractor cannot extract images). In our dataset DS2, 35 pseudocodes (out of 275 actual pseudocodes) cannot be captured using the sparse box extraction. However, these undetected pseudocodes may 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 pseudocode box detected by PC-ML, check whether there is a pseudocode caption detected by PC-RB nearby. If there is, the pseudocode box and the caption are combined.

## V. Evaluation and Discussion

We evaluate the three pseudocode detection algorithms on dataset DS2, using 10-fold document-wise cross validation.

### A. Evaluation Metrics

Standard precision, recall, and F1 are used for evaluating the performance. Let $T_g$ be the set of all pseudocodes, $T_r$ be the set of detected pseudocodes, so that the correctly detected pseudocodes are $T_g \bigcap T_r$. These metrics are defined as follows:

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

### B. Results

Table III lists notable results. As expected, the rule-based method (PC-RB) yields high precision with a cost of low recall. Using machine learning techniques (PC-ML), the overall performances (in terms of F1) are improved. The combine method (PC-CB) of PC-RB and a majority voting of LMT, Random Forest, and RIPPER classification models[8] performs

| Method | Model | Pr% | Re% | F1% |
|---|---|---|---|---|
| PC-RB | RuleBased | 87.12 | 44.57 | 58.97 |
| PC-ML | MLR$^{\dagger}$ | 80.35 | 56.78 | 66.54 |
| PC-ML | !LMT-RF-RIPPER-MLR | 85.31 | 57.04 | 68.37 |
| PC-ML | +NB-RIPPER-LMT-MLR | 79.61 | 59.37 | 68.02 |
| PC-ML | !NB-RIPPER-LMT-MLR | 78.26 | 60.05 | 67.96 |
| PC-ML | !LMT-RF-RIPPER | **88.84** | 53.74 | 66.97 |
| PC-CB | LMT$^{\ddagger}$ | 79.64 | 67.89 | 73.30 |
| PC-CB | !LMT-RF-RIPPER$^{\S}$ | 87.37 | 67.17 | **75.95** |
| PC-CB | +LMT-RF-RIPPER | 83.49 | 67.92 | 74.90 |
| PC-CB | !NB-RIPPER-LMT-MLR | 78.28 | 70.72 | 74.31 |
| PC-CB | NB | 37.86 | **75.89** | 50.52 |

TABLE III
PRECISION, RECALL, AND F1 OF THE PSEUDOCODE DETECTION METHODS USING DIFFERENT CLASSIFICATION METHODS. ('**!**' DENOTES MAJORITY VOTING, '**+**' DENOTES PROBABILITY AVERAGING)

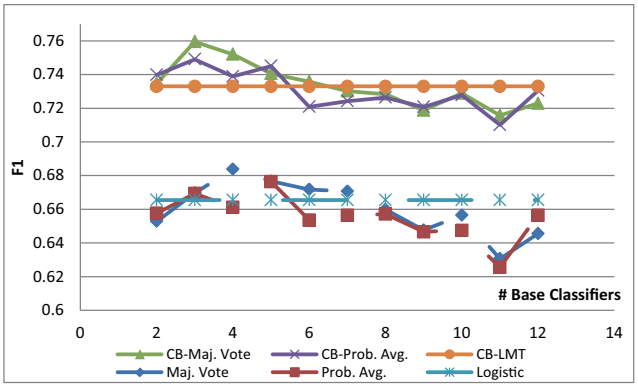



Fig. 6. Comparison of the ensemble methods against the best baseline classifiers in PC-ML and PC-CB. The dash lines and solid lines compare the performance between single classifier, majority vote, and probability averaging performance of PC-ML and PC-CB methods respectively.

the best in terms of F1, improving the performance over the state-of-the-art (the rule based method) by 16.98% (The recall is improved by 22.6%, while the precisions are on par.).

### C. Impact of Ensemble Methods

The ensemble methods result in a greater improvement compared to only using baseline classifiers. Figure 6 compares the performances (in terms of F1) between the ensemble methods and the best baseline classifiers for PC-ML (MLR$^{\dagger}$) and PC-CB (LMT$^{\ddagger}$). The X-axis denotes the first $k$ baseline classifiers, ranked by their *F1* scores, used in each ensemble method. We conclude that the ensemble methods are useful when the best baseline classifiers are combined. However, the performance of ensemble methods can decrease as the number of classifiers grows. This might be because bad classifiers impede the collective decision of the good ones. Unlike traditional document classification techniques wherein 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.

## VI. CONCLUSIONS

We have presented three methods for detecting pseudocodes in scholarly documents: rule based (PC-RB), machine learning based (PC-ML), and combined (PC-CB) methods. Our *PC-RB* method extends the state-of-the-art approach. The *PC-ML* method employs machine learning techniques to extract sparse boxes from a document and classifies each of them whether it is pseudocode or not using a novel set of 47 features. *PC-CB* captures the benefits of the both former methods. The best performance in terms of F1 is achieved by the *PC-CB* method with the combination of the rule-based method and the majority vote of LMT, RF, and RIPPER classifiers. Moreover, we present an analysis of the performance increase using the ensemble methods. Future work could investigate scalability for large datasets such as the over 3 million documents in Citeseer$^{X}$ repository.

## VII. ACKNOWLEDGMENT

This material is based upon work supported by the United States National Science Foundation.

## REFERENCES

[1] J. Wang, "Mean-Variance Analysis: A New Document Ranking Theory in Information Retrieval," *Proceedings of the European Conference on Information Retrieval ECIR*, pp. 4–16, 2009.
[2] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, pp. 341–343, 1975.
[3] S. Bhatia, S. Tuarob, P. Mitra, and C. L. Giles, "An algorithm search engine for software developers," ser. SUITE '11, 2011, pp. 13–16.
[4] J. M. Kleinberg and E. Tardos, *Algorithm Design*. Addison Wesley, 2005.
[5] S. Bhatia, P. Mitra, and C. L. Giles, "Finding algorithms in scientific articles," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10, 2010, pp. 1061–1062.
[6] S. Guha and N. Koudas, "Approximating a data stream for querying and estimation: algorithms and performance evaluation," *Proceedings 18th International Conference on Data Engineering*, pp. 567–576, 2002.
[7] R. Atanassov, P. Bose, M. Couture, A. Maheshwari, P. Morin, M. Paquette, M. Smid, and S. Wuhrer, "Algorithms for optimal outlier removal," *Journal of Discrete Algorithms*, vol. 7, no. 2, pp. 239–248, 2009.
[8] J. B. Baker, A. P. Sexton, V. Sorge, and M. Suzuki, "Comparing approaches to mathematical document analysis from pdf," ser. ICDAR '11, 2011, pp. 463–467.
[9] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *International Journal on Document Analysis and Recognition*, pp. 1–27, 2012.
[10] S. Mandal, S. P. Chowdhury, A. K. Das, and B. Chanda, "Automated detection and segmentation of table of contents page from document images," ser. ICDAR '03, 2003, pp. 398–.
[11] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, "Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines," ser. ICDAR '09, 2009, pp. 1006–1010.
[12] 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," ser. AAAI'08, 2008, pp. 1169–1174.
[13] Y. Liu, K. Bai, P. Mitra, and C. Giles, "Searching for tables in digital documents," ser. ICDAR '07, 2007, pp. 934–938.
[14] 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.
[15] S. Tuarob, P. Mitra, and C. L. Giles, "Improving algorithm search using the algorithm co-citation network," in *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, ser. JCDL '12, 2012, pp. 277–280.
[16] ——, "A classification scheme for algorithm citation function in scholarly works," in *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital Libraries*, ser. JCDL '13, 2013.