https://doi.org/10.33919/JCAL.23.1.2

EXTRACTING ALGORITHMIC COMPLEXITY IN SCIENTIFIC LITERATURE FOR ADVANCE SEARCHING

Abu Bakar[1], Raheem Sarwar[2,*], Saeed-Ul Hassan[3], Raheel Nawaz[4]

[1]*Computer Science, Information Technology University, Lahore, Pakistan*
[2] *Department of Operations, Technology, Events and Hospitality Management, Manchester Metropolitan University, United Kingdom*
[3] *Department of Computing and Mathematics, Manchester Metropolitan University, United Kingdom*
[4] *Staffordshire University, United Kingdom*
[*]*Corresponding Author (R.Sarwar@mmu.ac.uk)*

Abstract

Non-textual document elements such as charts, diagrams, algorithms and tables play an important role to present key information in scientific documents. Recent advances in information retrieval systems tap this information to answer more complex user queries by mining text pertaining to non-textual document elements from full text. Algorithms are critically important in computer science. Researchers are working on existing algorithms to improve them for critical application. Moreover, new algorithms for unsolved and newly faced problems are

under development. These enhanced and new algorithms are mostly published in scholarly documents. The complexity of these algorithms is also discussed in the same document by the authors. Complexity of an algorithm is also an important factor for information retrieval (IR) systems. In this paper, we mine the relevant complexities of algorithms from full text document by comparing the metadata of the algorithm, such as caption and function name, with the context of the paragraph in which complexity related discussion is made by the authors. Using the dataset of 256 documents downloaded from CiteSeerX repository, we manually annotate 417 links between algorithms and their complexities. Further, we apply our novel rule-based approach that identifies the desired links with 81% precision, 75% recall, 78% F1-score and 65% accuracy. Overall, our method of identifying the links has potential to improve information retrieval systems that tap the advancements of full text and more specifically non-textual document elements.

Keywords: *algorithm search, algorithm complexity, information retrieval, non-textual document elements (NTDE)*

1. Introduction

Academic literature is growing exponentially in last two decades and flourishing in an unprecedented pace, which has brought new challenges to information retrieval research (Khan, Liu, Shakil and Alam, 2017). Non-Textual Document Elements (NTDEs) such as figures, charts, pseudo-codes, and tables are very common in scientific documents, and they are vital elements for communicating the key information. These elements are sometime placed at the start or at the end of the page instead of following the flow of document text, and the discussion about these elements may or may not be on the same page, the discussion about these elements mostly has the reference of the caption of the entity. Sometimes, these elements are referred multiple times in different sections of the scholarly document.

Algorithms are well-defined methodologies to solve the problems and they are important in every field of science and technology. There are many features of an algorithm such as correctness, elegance, efficiency and scalability. Efficiency of an algorithm is defined as complexity of algorithm, and it is based on time and space. The main aim of estimating the complexity of algorithms is to categorize them according to their efficiency. Complexity of algorithm is described asymptotically using three types of notations as:

(1) O-notation (big oh notation) used for upper bound, (2) Ω-notation (big omega notation) used for lower bound and (3) Θ-notation (big theta notation) used for tight bound. We have used these asymptotic notations to identify the complexity lines in full text scholarly documents.

Scholarly publications host a tremendous number of high-quality algorithms, developed by professionals and researchers. Normally, when new algorithms are published, or existing algorithms are enhanced, their time and/or space complexities are also discussed in the same document by the authors. Most of the time authors are not working on algorithms or not trying to improve existing ones, they are publishing the algorithms that they have used in their research, and sometimes they do not discuss the complexity of the algorithm because it is well-known, or because they do not focus on it.

The complexities of an algorithm (for time and space) can be identified from the document by analyzing the context of the paragraph in which the complexity is mentioned, and the metadata (such as the algorithm caption and the algorithm label) of the algorithm extracted from the same document.

Our Research Contributions. In this research, our contributions are as follows:

- Identification of algorithmic complexity lines in full text document using regular expressions and synopsis generation for each complexity line.
- Algorithmic metadata compilation of algorithms – normally, there are multiple algorithms in a scholarly document and the metadata of each algorithm are compiled separately.
- Linking complexity related textual lines to algorithmic metadata using a novel rule-based approach.

In Section 2, we have discussed related work. In Section 3, a dataset is described, and a system model of our research is discussed in Section 4. Experiments and their results are given in Section 5 and Section 6 concludes the research with future suggestions.

2. Literature Review and Related Work

Many algorithms are being published in research articles on a monthly and yearly basis (Bhatia, Tuarob, Mitra and Giles, 2011). Hundreds of articles are published and/or added to digital archives on a monthly and yearly basis and the arXiv[1] has crossed the boundary of 1.3M full text publications, which shows the importance of this research.

[1] https://arxiv.org/stats/monthly_submissions.

2.1 Algorithmic Representations

Normally, algorithms are explained in pseudo-code (PC) (as shown in Figure 1), in natural language as algorithmic-procedure (AP) (as shown in Figure 2), in mathematical formatting (as shown in Figure 3) or in coding style (as shown in Figure 4) (Tuarob, Bhatia, Mitra and Giles, 2016). Algorithms can be implemented in any programming language. There are a number of document elements in a scholarly document such as figures (Siegel, Horvitz, Levin, Divvala and Farhadi, 2016), tables (Liu, Bai, Mitra and Giles, 2007), forms (Coüasnon and Lemaitre, 2014), algorithms (Bhatia, Mitra and Giles, 2010) mathematical expressions (Baker, Sexton, Sorge and Suzuki, 2011) (Zanibbi and Blostein, 2012), programing codes and a number of textual sections such as abstract, acknowledgments (Khabsa, Treeratpituk and Giles, 2012), collaborations (Chen, Gou, Zhang and Giles, 2011), methodology, results (e.g., precision, recall, or F-measure), conclusion and references. Algorithms are normally given as figures (Bhatia and Mitra, 2012), in mathematical formatting, in coding style, or sparse boxes like other document elements (Tuarob, Bhatia, Mitra and Giles, 2013).

2.2 Information Retrieval Systems and Algorithms

Information retrieval (IR) is a technique for searching required or relevant information form exiting data (Wang, 2009). There are several search engines to search for academic literature such as Google Scholar[2], Microsoft Academic[3], PloS One[4], Semantic Scholar[5], Science Direct[6], ACM Digital Library[7] and CiteSeerX[8], an academic document search (Wu, et al., 2015). There are also some search engines that are optimized for a specific area of science such as BioText Search Engine[9] which is optimized for bioinformatics related search (Hearst, et al., 2007). There are some other IR systems such as TableSeer for searching tables in digital libraries (Liu, Bai, Mitra and

[2] https://scholar.google.com/.

[3] https://academic.microsoft.com/.

[4] http://journals.plos.org/plosone/.

[5] https://www.semanticscholar.org.

[6] https://www.sciencedirect.com/.

[7] https://dl.acm.org.

[8] http://citeseerx.ist.psu.edu/index.

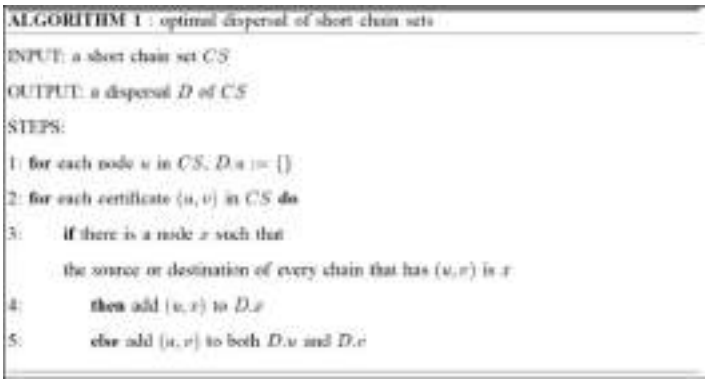
[9] http://biosearch.berkeley.edu/

Giles, 2007), AckSeer for acknowledgments (Khabsa, Treeratpituk and Giles, 2012), CollabSeer for collaborations (Chen, Gou, Zhang and Giles, 2011), FigureSeer for figures (Siegel, Horvitz, Levin, Divvala and Farhadi, 2016) and AlgorithmSeer for searching algorithms in scholarly big data (Tuarob, Bhatia, Mitra and Giles, 2016).

Efficient algorithms are critically important and sometime crucial for certain software projects. Nowadays, there are a number of source code search engines for software developers and researchers to find relevant source code according to their requirements. Information retrieval systems for algorithms in scholarly documents are improving in past recent years to fulfill the search queries by providing relevant information such as Sourerer (Bajracharya, Ossher and Lopes, 2009) and Exemplar (EXEcutable exaMPLes ARchive) (McMillan, Grechanik, Poshyvanyk, Fu and Xie, 2012).

In recent years, a few attempts have been made to extract non textual document elements such as figures, tables, algorithms and charts (Al-Zaidy and Giles, 2017), (Tuarob, Bhatia, Mitra and Giles, 2013), (Safder, Hassan and Aljohani, 2018). These techniques are actively applied for effective document summarization to improve the existing IR systems. A customized search engine AlgorithmSeer is designed for algorithms searching from full text articles (Tuarob, Bhatia, Mitra and Giles, 2016). This system uses some rule-based and machine learning based techniques for automatic extraction of algorithms from full text articles, then it creates a specialized algorithmic summary of a document to match against a user search query.

Moreover, to mine information from results figures present in scholarly articles, FigureSeer, a specialized results figure extractor system has been presented (Siegel, Horvitz, Levin, Divvala and Farhadi, 2016). The designed system leveraged the deep learning-based techniques to identify, class an image as results image. Further, the system mines the information presented on these figures to design a results figure search engine. Likewise, another system Deep-Figures has implemented a similar kind of system using supervised neural network-based technique (Siegel, Lourie, Power and Ammar, 2018). Table search system is also a very prominent work to retrieve complex tables against user queries from massive repositories (Nargesian, Zhu, Pu and Miller, 2018). Additionally, linking these document elements (table, figure, algorithms) with their discussions or reference sentences written in the full body text of an article has its own significance. This additional information about a document element can help to understand its context instead of reading and scrolling the whole paper (Bhatia and Mitra, 2012).



ALGORITHM 1 : optimal dispersal of short chain sets

INPUT: a short chain set CS

OUTPUT: a dispersal D of CS

STEPS:

1: **for** each node u in CS, $D.u := \{\}$

2: **for** each certificate (u, v) in CS **do**

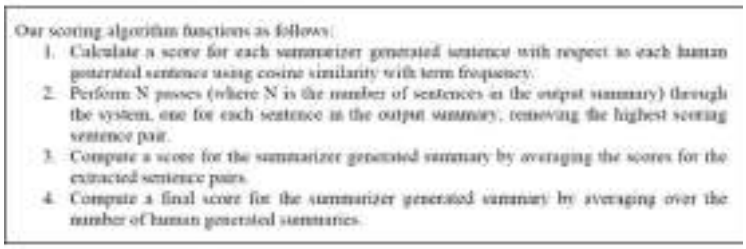
3: **if** there is a node x such that

the source or destination of every chain that has (u, v) is x

4: **then** add (u, v) to $D.x$

5: **else** add (u, v) to both $D.u$ and $D.v$

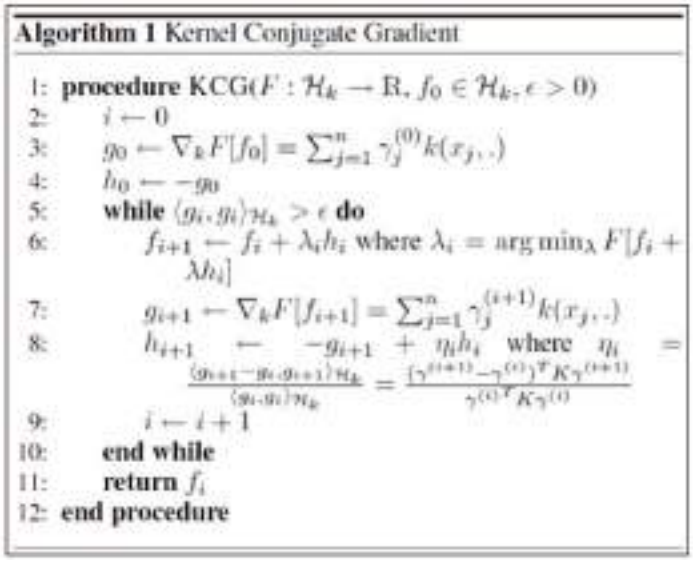
Figure 1: Example of Pseudo-Code (PC) **(Jung, Elmallah and Gouda, 2007)**



Our scoring algorithm functions as follows:

1. Calculate a score for each summarizer generated sentence with respect to each human generated sentence using cosine similarity with term frequency.
2. Perform N passes (where N is the number of sentences in the output summary) through the system, one for each sentence in the output summary, removing the highest scoring sentence pair.
3. Compute a score for the summarizer generated summary by averaging the scores for the extracted sentence pairs.
4. Compute a final score for the summarizer generated summary by averaging over the number of human generated summaries.

Figure 2: Example of Algorithmic Procedure (AP), from **(Stewart and Callan, 2009)**



Algorithm 1 Kernel Conjugate Gradient

procedure KCG($F : \mathcal{H}_k \to \mathrm{R}, f_0 \in \mathcal{H}_k, \epsilon > 0$)
  $i \leftarrow 0$
  $g_0 \leftarrow \nabla_k F[f_0] = \sum_{j=1}^{n} \gamma_j^{(0)} k(x_j, .)$
  $h_0 \leftarrow -g_0$
  while $\langle g_i, g_i \rangle_{\mathcal{H}_k} > \epsilon$ **do**
    $f_{i+1} \leftarrow f_i + \lambda_i h_i$ where $\lambda_i = \arg\min_\lambda F[f_i + \lambda h_i]$
    $g_{i+1} \leftarrow \nabla_k F[f_{i+1}] = \sum_{j=1}^{n} \gamma_j^{(i+1)} k(x_j, .)$
    $h_{i+1} \leftarrow -g_{i+1} + \eta_i h_i$ where $\eta_i = \frac{\langle g_{i+1} - g_i, g_{i+1} \rangle_{\mathcal{H}_k}}{\langle g_i, g_i \rangle_{\mathcal{H}_k}} = \frac{(\gamma^{(i+1)} - \gamma^{(i)})^T K \gamma^{(i+1)}}{\gamma^{(i)T} K \gamma^{(i)}}$
    $i \leftarrow i + 1$
  end while
  return f_i
end procedure

Figure 3: Example of Mathematical Formatting **(Ratliff and Bagnell, 2007)**

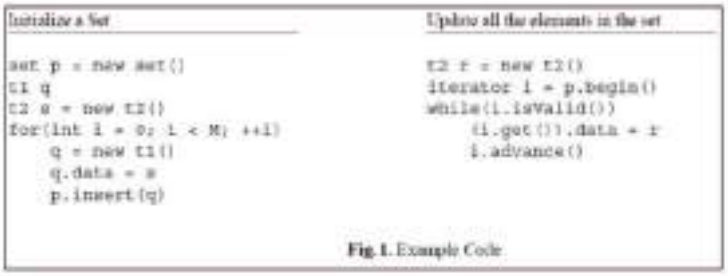


Figure 4: Example of Coding Style Algorithm
(Marron, Stefanovic, Hermenegildo and Kapur, 2007)

Generally, run time complexity related to an algorithm is mentioned in the full body text of a document as algorithmic metadata. In order to find out the above-mentioned complexities of an algorithm, we need to link the algorithm and the paragraphs in which the complexity of that algorithm is discussed. Recently, a few techniques have been designed to extract evolution results lines related to an algorithm from full text articles (Safder, Sarfraz, Hassan, Ali and Tuarob, 2017). However, to the best of our knowledge no work has been done to find run time complexities and to link these run time complexities with their respective algorithm in full text scholarly publications.

2.3 Algorithm Detection in Scholarly Documents

A number of rule-based, machine learning-based, and deep learning-based methods have been designed for detection and extraction of algorithms from full-text scholarly documents (Tuarob, Bhatia, Mitra and Giles, 2016), (Safder, Sarfraz, Hassan, Ali and Tuarob, 2017), (Safder, Hassan and Aljohani, 2018), (Lai, Xu, Liu and Zhao, 2015). Detection of an algorithm in a document is the first step, the aim is to make it retrievable on users' queries. For IR system algorithms metadata are needed, therefore caption lines, indication sentences, function names or algorithm labels are extracted from documents related to the algorithms for metadata.

Results related to algorithmic evaluation performance such as precision, recall, F-measure and accuracy are also extracted from the same documents to improve results for developers and researchers. Complexity of algorithm is also an important factor for IR systems; currently it not directly used in IR systems as effectively as it is important. We aim to improve the feature of time and space complexity identification from scholarly documents in our research. In this paper, we designed a mechanism to identify complexity lines and then to link these complexity lines with their relevant algorithm.

3. Data and Limitations

There are over 100 million scholarly documents in the English language on the web in different fields (Khabsa and Giles, 2014), most of which are indexed by digital libraries. For our research we have selected a small dataset of 258 documents, as discussed below.

Data. The dataset is selected from (Safder, Sarfraz, Hassan, Ali and Tuarob, 2017), which consists of 258 documents originally selected from the CiteSeerX repository (Tuarob, Bhatia, Mitra and Giles, 2016), and has 37,000 lines of text. The data is manually labeled: 2,331 lines for algorithmic efficiency and 80 lines for algorithmic time complexity. There are some limitations to using this dataset which are discussed in Section 3.2. We use algorithmic metadata lines tagging as given in the section below[10]. Data is also tagged for the following type of lines related to algorithmic metadata (as shown in Figure 5): pseudo-code lines, pseudo-code caption lines, function name, algorithm label, indication sentence, algorithm section header, explanation sentence and proposal sentence. We use tagging to identify algorithms and their metadata, and then we compare this metadata with the paragraph in which the complexity is mentioned.

There are 142 documents in the dataset, in which we found algorithms and other tagged lines related to algorithms (tagging is listed above). There are 62 documents in which we found complexity lines, and only 47 documents in which algorithms, an algorithm's related tagged lines and complexities coexist.

Reference Document Preparation. A reference document is prepared manually, by using those 47 documents, in which both algorithms and complexities are found. 471 relations are identified between algorithms, and their time and space complexities in 35 documents out of 47 documents. This dataset is used for results, comparisons and calculations.

Frequent Keywords Set. Frequent Keywords (FK) are extracted from algorithmic metadata lines and are used in matching the synopsis of the complexity line and the algorithmic metadata. The weightage of frequent keywords is less than normal keywords, as given in the following Inequality 1:

$$W_f < W_n \, (1)$$

Where W_f is frequent keywords and W_n is normal or non-frequent keywords.

As frequent keywords are those keywords which are used more commonly than other, normal keywords, they have a smaller relative impact on finding

[10] The data and code used in this research can be downloaded from the following URL: https://github.com/slab-itu/icadl_link_algo.

the relevance of complexity context and algorithmic metadata. The list of frequent keywords is given in Table 1.

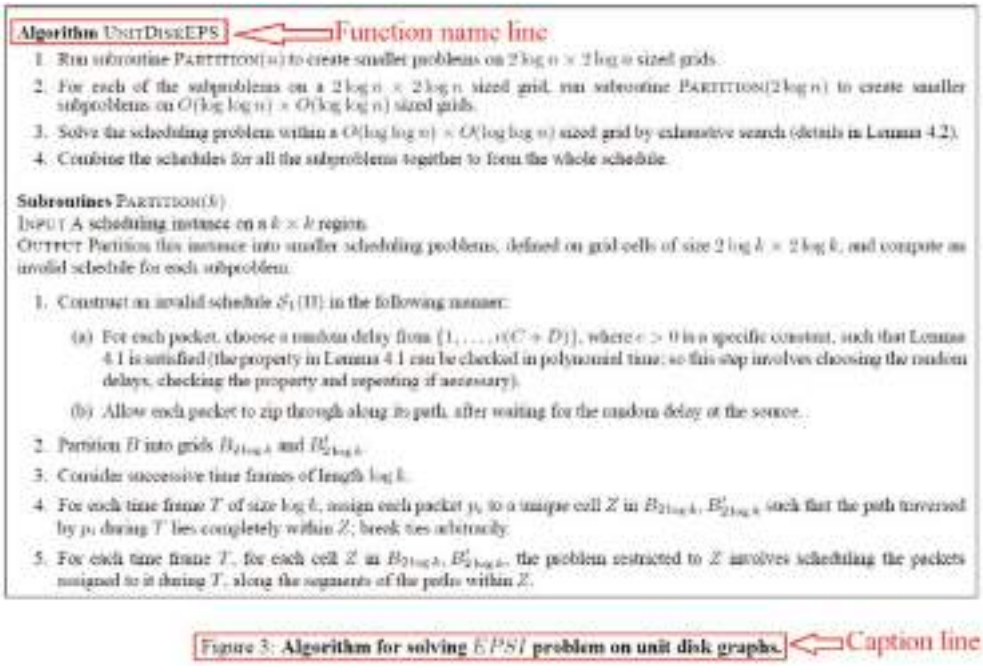


Figure 5: Algorithmic Metadata Lines
(Kumar, Marathe, Parthasarathy and Srinivasan, 2004)

Cue words. We have used two lists of cue words (CW), one for complexity context and the other for the identification of common asymptotic growth rate function names. Cue words for complexity context are listed in Table 1. Cue words to identity common functions growth rate of asymptotic bounds are also listed in Table 1. Cue words are used to weigh the comparison between complexity and the algorithm; they are also used to identify the asymptotic function names of complexity.

WordNet Library. We used WordNet[11] library in Python for synonyms and semantically related terms along with original keywords to compare the context of the paragraph in which complexity is discussed and the algorithmic metadata, such as caption lines, indication sentences, function names or algorithm labels.

[11] https://wordnet.princeton.edu.

Table 1: Frequent Keywords Set, Cue Words for Complexity Context and for Common Complexity Functions

Sr.	Frequent Keyword	Cue Word for Complexity Context	Cue Word for Common Complexity Functions
1	algorithm	algorithm	polynomial
2	figure	complexity	poly
3	procedure	time complexity	constant
4	fig	space complexity	linear
5	method	run time	sublinear
6	following	best case	quadratic
7	steps	worst case	cubic
8	code	pseudocode	logarithmic
9	program	computational time	linearithmic
10	class	efficiency	exponential
11	function	optimal solution	parallel
12	search	performance	factorial
13	example		approximation
14	based		
15	sequence		
16	given		
17	skeleton		
18	table		
19	using		
20	main		
21	set		
22	list		
23	pseudocode		
24	described		
25	problem		

3.1 Pre-Processing

There are certain limitations in our dataset, some of which are discussed in this section.

3.1.1 PDF to TXT conversion

While extracting plain text from pdf documents, complex functions of complexity was not handled, and they are hard to identify in text document (as shown in Figure 6). Some words are not converted properly; most of them are special words such as the name of algorithm or keywords closely related to algorithmic metadata. The critical thing is that asymptotic notations are not converted properly, as can be observed in line 16 of the text document – "O" is translated to "Cl" – and because of this issue our regular expression will fail to identify the complexity line. If complexity line identification fails, then it will also fail to link that line to any algorithm as it is the base case for further processing.

3.1.2 Algorithmic Metadata Tagging

As we are using a dataset which is already tagged for algorithmic metadata, our results are dependent upon how accurately the metadata is tagged. Accuracy of algorithmic metadata tagging is 76% with 79% precision, 77% recall and 77% F1 scores from (Safder, Sarfraz, Hassan, Ali and Tuarob, 2017).

3.1.3 Multiple Complexity Lines Association

Sometimes multiple algorithms are described in a document or multiple versions of the same algorithm are given, and the complexity of all the algorithms or all versions of the algorithms are discussed in the same paragraph or sometimes in a table (as shown in Figure 7), so it is hard to distinguish which complexity line is related to which algorithm. For the tabular case, it may not associate any of the complexity lines to any algorithm because, when rendering the table from PDF to text, it may convert each cell of the table to a new line, and when we built the context of the complexity line, we only collect text from five lines before and after the complexity line. In the tabular case, this context will have only a few words, and this will not help to link it with algorithmic metadata. In this case, multiple complexity lines will be associated to an algorithm or multiple algorithms will be linked to same complexity line.

Figure 6: PDF to Text Conversion Issues **(Milidiú, Laber and Pessoa, 1999)**

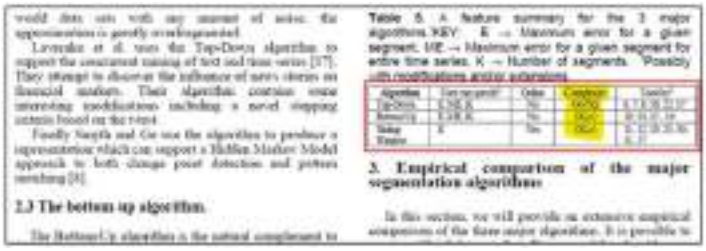


Figure 7: Complexities of Multiple Algorithms in a Table **(Keogh, Chu, Hart and Pazzani, 2001)**

3.2 Error Rate

The error rate of our results may be high because the error will multiply with all error rates, such as the error rate of the pdf to text extraction, the error rate of the algorithmic metadata tagging and the error rate of our model. It can be calculated from the given Equation 2.

$$ER_{total}=ER_{pdfToText}\times ER_{tagging}\times ER_{ourModel} \quad (2)$$

where ER_{total} is the overall total error rate, $ER_{pdfToText}$ is the error rate of pdf to plain text extraction, $ER_{tagging}$ is the error rate of algorithmic metadata tagging and $ER_{ourModel}$ is the error rate of our model.

4. Methodology

Complexity lines are identified, and their context is built. Similarly, algorithmic metadata lines are extracted and combined for each algorithm. After that, by comparing both the complexity context and the algorithmic metadata, a reference file is created in which links between complexity lines and algorithms are listed. A high-level diagram of proposed system is given in Figure 8.

4.1 Complexity Line Identification and Context Building

We use regular expressions in Python to identify complexity lines in plain text documents, and asymptotic notation formats are used in the regular expressions for this purpose. After identification of complexity lines, we have built context of the line, which is identified as a complexity line, to build context – five lines before and after the complexity line are used. Tagged; lines for algorithmic metadata are ignored while building the context. There are multiple complexity lines in the same document, and context of each complexity line is built separately.

The grammar for our Regular Expression (RE) which is used to detect complexity lines from text documents is given in Figure 9, and the Python notation is given below:

$$r'\backslash b\backslash d*[O\Omega\Theta\omega 0]\backslash(.*[nmk\backslash d(log)(ln)].*\backslash)' \quad (3)$$

There are two main parts of this regular expression: the first part is to detect asymptotic bound notations such as O, Ω, Θ and ω; the second part is enclosed in starting parenthesis "(" and closing parenthesis ")", and between these there can be any complexity notation: *n* is mostly used for input size or data size and some other letters such as *m* and *k* are also used for the same purpose; *\d* is used for number in regular expressions, it is used here for power or constant value detection; sometimes complexity is defined in a logarithmic function, *(log)* and *(ln)* are used for logarithmic function detection. All these special characters and symbols are enclosed in a starting bracket "[" and a closing bracket "]" to ignore their sequence and occurrence; case is also ignored to detect upper- and lower-case letters.

4.2 Section Detection

The role of sections is very important when extracting relevant information. In scientific documents, sections can be identified using section headers and their boundaries (Tuarob, Mitra and Giles, 2015). In our case, if a complexity line lies in related work or background section, it may be related to an algorithm which is not discussed in the current document and the author may be comparing the complexities of different algorithms. If it is in the implementation or methodology section, then there are high chances that it is related to algorithms which are described in the current document. Similarly, if it is in the abstract, then chances are very high for that. If it is in future work, then it may be the desired complexity to achieve in future. If it is in a reference section then it will be ignored, because in this case it will be part of some other document's title.

4.3 Algorithm Metadata Extraction and Compilation

As algorithmic metadata is the lines related to an algorithm's description and definition, these lines have already been tagged in our dataset (*e.g.,* caption lines and algorithmic labels). These algorithmic metadata lines are extracted from the plain text document and combined together. Most frequent keywords are also calculated using the frequency of the keywords in all documents. In some documents there are multiple algorithms; the metadata of each of them is combined separately.

4.4 Comparison of an Algorithm's Metadata and Context of Complexity

We have built algorithmic metadata using tagged lines from plain text document, identified complexity lines and built their context from the same document, as shown in Figure 10. We then compare both of them and use a probabilistic method to compare algorithmic metadata and complexity line context. We have also used weights for direct keywords matching and synonyms and semantically related terms. Synonyms and semantically related terms have been extracted from WordNet library using Python.

Weights for direct keywords are higher than for synonyms and semantically related terms and weights for the most frequent terms are lower than for the less frequent terms in algorithmic metadata. By combining both measures, the following Inequality 4 is applied for matching keywords:

$$W1 > W2 > W3 > W4 \quad (4)$$

where W1 is for direct non-frequent keywords, W2 is for direct frequent keywords, W3 is for synonyms or semantically related non-frequent keywords and W4 is for synonyms or semantically related frequent keywords.

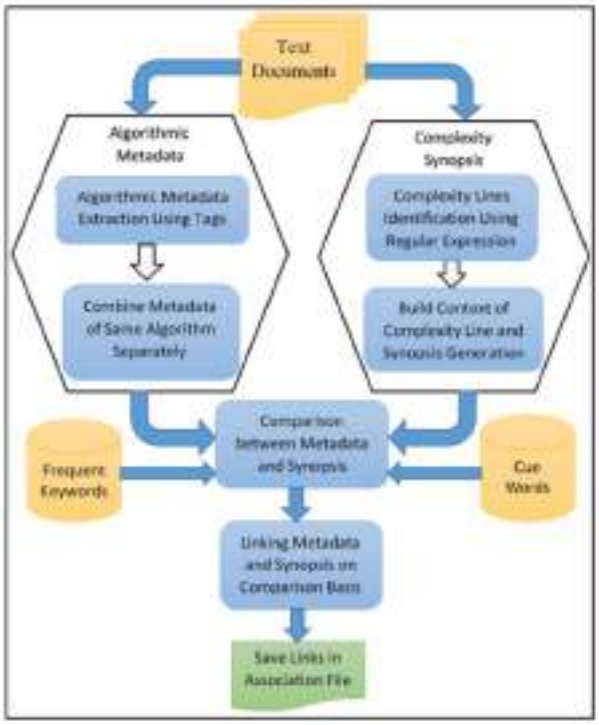


Figure 8: High-Level Diagram of Proposed System

```
     <ALGO_COMP>  ::= <ASYMP_NOTATIONS> <(> <COMP_TEXT> <)>
<ASYMP_NOTATIONS>::= O|Ω|Θ|o|ω
     <COMP_TEXT>  ::= <A String of Characters>
```

Figure 9: Grammar for Algorithmic Complexity

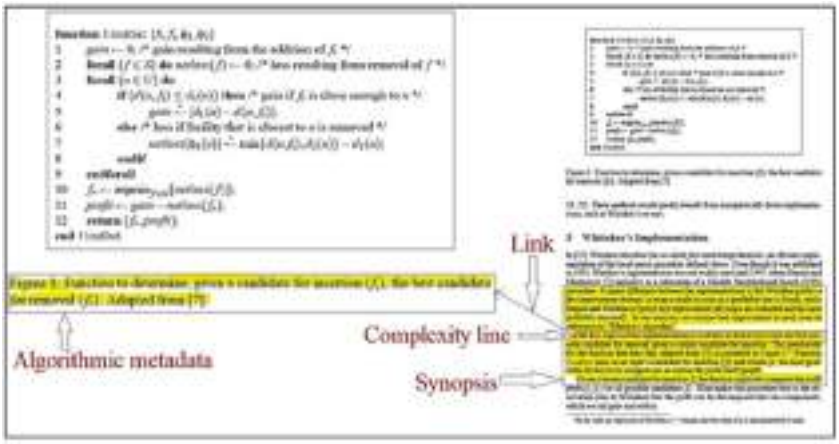


Figure 10: Comparison of Algorithm's Metadata and Context of Complexity

4.5 Reference or Association File for Algorithm and Complexity Relations

A reference file is created to save the links between complexities and algorithms. As an algorithm can be linked to more than one complexity line and one complexity line can be associated with more than one algorithm, we have created a dataset for associations or links between algorithmic metadata and complexity lines. In this dataset we have saved the line numbers of the complexity lines with the text of the line and the line numbers of the algorithmic metadata with the metadata itself. Some other fields have been added in this dataset, such as number of keywords in algorithmic metadata, percentage of matching keywords and cue words that are matched in complexity context (both for complexity context and asymptotic function). We have saved this dataset to a file called reference or association file and this file will be used for ranking and indexing the algorithms for IR systems.

5. Experiments and Results

We have done a number of experiments on data to improve our results by changing matching percentage of algorithmic metadata with complexity synopsis-built form complexity context and with and without considering frequent keywords set and cue words for complexity context and asymptotic growth function names. We have also selected some feasible thresholds using experimental results to improve our results. Some of these experiments and their results are discussed in this section. Graphs for threshold values selection, ROC curves and precision-recall covers are also given.

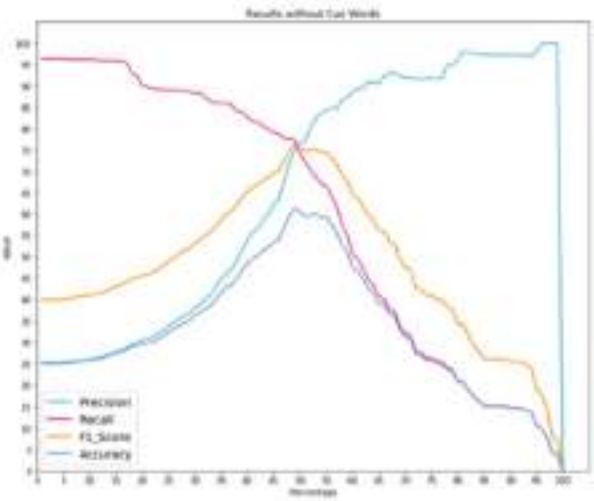


Figure 11: Results without Cue Words for Threshold Selection

5.1 Threshold Selection

Thresholds for matching the percentage ratio of algorithmic metadata and the complexity synopsis are selected using precision, recall, F1-score and accuracy data for all threshold values from 0 to 100 percent.

Graphs for all thresholds are also shown in Figure 11 and Figure 12. In both figures recall is very high at the beginning but goes down as we increase the value of the matching percentage; for the precision value it is low at the beginning but grows when we increase the value of the matching percentage. It is because almost all true links are identified when the percentage is low, but the false links identification ratio is also very high at a low matching percentage. Similarly, at a very high matching percentage only a few true links are identified but the false links identification ratio is also negligible. At the point where precision and recall curves intersect, the ratio of true prediction is maximum.

F1-score and accuracy are low at the beginning, then they grow up to some point (near the intersection of precision and recall curves), and then they go down again, because F1-score is the harmonic average of precision and recall and accuracy is the ratio of correctly identified instances to the total number of instances.

Percentage threshold without using cue words is selected as 50 percent by using results of precision, recall, F1-score and accuracy, as shown in Figure 11, in which the intersection of precision, recall and F1- score curves is near 50 percent. Similarly, percentage threshold for with using cue words is selected as 55 percent by using results of precision, recall, F1-meaure and accuracy, as shown in Figure 12. Similarly in this figure the intersection of precision, recall and F1-score curves is near 55 percent. These thresholds are used in our experimental setup, which is given in the next sections.

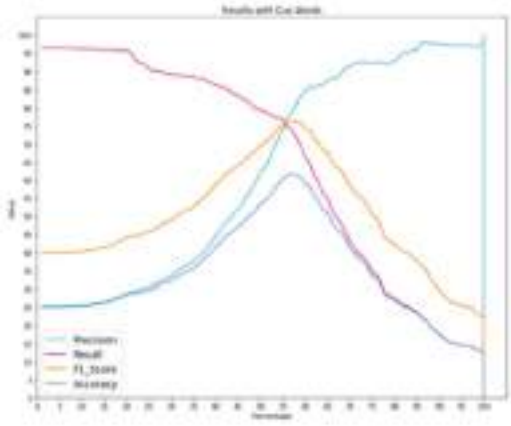


Figure 12: Results with Cue Words Threshold Selection

5. 2 Experimental Setup

We always learn from our experiments, and we have done several experiments on our data to improve our results, but as we discussed earlier, there are some limitations in our data and model, so we can only achieve our results up to some possible level. Four experiments are discussed in the following sections.

Table 2: Truth Table, 50% Matched, with No-Frequent Keywords and No-Cue Words (NFNC50)

Measures	Value
Total True	471
True matched	365
False matched	189
True and False Both matched	554
Not matched	106
Total	660

5.2.1 Experiment 1

In the first experiment, we used a matching percentage ratio of algorithmic metadata up to 50 percent, and in this experiment, we completely ignored the cue words and the frequent keywords from the algorithmic metadata. We named this experiment NFNC50 (No-Frequent keywords and No-Cue words with 50% threshold). Results for experiment 1 are summarized in Table 2.

5.2.2 Experiment 2

In the second experiment, we used a matching percentage ratio of algorithmic metadata up to 50 percent, and in this experiment we ignored the cue words, but we used frequent keywords from the algorithmic metadata. Frequent keywords are considered as low weighted in this case, frequent keywords set is generated from algorithmic metadata as discussed in Section 3.1.2. We named this experiment FNC50 (Frequent keywords and No-Cue words with 50% threshold). Results for experiment 2, are summarized in Table 3.

Table 3: Truth Table, 50% Matched, with Frequent Keywords and No-Cue Words (FNC50)

Measures	Value
Total True	471
True matched	359
False matched	104
True and False Both matched	463
Not matched	112
Total	575

5.2.3 Experiment 3

In this experiment, we used matching percentage ratio of algorithmic metadata along with cue words, greater than 55 percent, and, we added the cue words matching ratio for the overall percentage. Frequent keywords are also considered low weighted in this case. We named this experiment FC55 (Frequent keywords and Cue words with 55% threshold). Results for this experiment are summarized in Table 4.

Table 4: Truth Table, 50% Metadata Matched Overall with Cue Words (FC55)

Measures	Value
Total True	471
True matched	355
False matched	111
True and False Both matched	466
Not matched	116
Total	582

5.2.4 Experiment 4

In this experiment, we used a matching percentage ratio of algorithmic metadata up to 50 percent separately, and we added the cue words matching ratio after that. In other words, we combined the conditions of the second and the third experiment, and in this way, the results were maximized. We named this experiment FNC50FC55. Results for this experiment are summarized in Table 5.

Table 5: Truth Table, 50% Metadata Matched and 55% Overall with Cue Words (FNC50FC55)

Measures	Value
Total True	471
True matched	354
False matched	86
True and False Both matched	440
Not matched	117
Total	557

5.3 Calculations

Results are calculated using standard formulas as discussed in this section.

5.3.1 Precision

Precision is defined as the ratio of correctly identified instances to the total predicted positive instances. The equation to calculate precision is given as follows:

$$Precision = \frac{\text{True matched (TP)}}{\text{True and False Both Matched (TP+FP)}} \qquad (5)$$

The worst precision is recorded for experiment 3 (FC55), which is 61 percent, and the best precision is yielded by experiment 4 (FNC50FC55), which is 81 percent.

5.3.2 Recall

Recall is defined as the ratio of correctly identified instances to all actual observations in the data; it is also called 'sensitivity'. Recall is calculated using the following equation:

$$Recall = \frac{\text{True matched (TP)}}{\text{Total True (TP+FN)}} \qquad (6)$$

Experiment 1 (NFNC50) yielded the best recall, which is 77 percent, and the worst recall is recorded for experiment 3 (FC55), which is 75 percent.

F1 Score

It is the harmonic average of precision and recall, as given below:

$$\text{F1 Score} = \frac{2*(Recall * Precision)}{(Recall + Precision)} \qquad (7)$$

F1 score is more useful than accuracy. The best F1 score is yielded by experiment 4 (FNC50FC55), which is 78 and the worst F1 score is yielded by experiment 3 (FC55), which is 67 percent.

Accuracy

Accuracy is the most common measure to check the performance of results; it is the ratio of correctly identified instances to the total number of instances, as given below:

$$\text{Accuracy} = \frac{\text{True matched (TP+TN)}}{\textit{Total (TP+FP+FN+TN)}} \qquad (8)$$

The best accuracy is yielded by experiment 4 (FNC50FC55), which is 65 and the worst accuracy is yielded by experiment 1 (NFNC50), which is 56 percent.

Results

Results are shown in Table 6, as in our first experiment we did not use frequent keywords from algorithmic metadata and cue words for complexity context and asymptotic function names are completely ignored. Recall was maximum, which is 77%, but accuracy is low. In other words, in this case maximum actual links are identified but false results ratio is also high.

In the second experiment we have considered only frequent keywords for matching weights and we see that results are improved, as precision was improved from 66% to 78% and accuracy – from 56% to 64%; however, recall was down from 77% to 76%,

In our third experiment, we have also considered the cue words along with frequent keywords to evaluate the weights and increase the matching percentage threshold from 50% to 55%. In this case results were not improved, as we can see in the third row of Table 6.

In our last experiment, we have combined the conditions and thresholds of the second and third experiment. By doing this we got maximum results, as precision is improved significantly; F1 score and accuracy is also maximized in this case. Finally, we have achieved 81% precision, 75% recall, 78% F1-score and 65% accuracy.

Precision, recall, F-measure and accuracy for linking the algorithm and complexity line for the different experiments are given in Table 6.

Table 6: Precision, Recall, F-measure and Accuracy for Algorithm and Complexity Line Linking

Name	Method	Precision	Recall	F1 Score	Accuracy
NFNC50	50% matched with no frequent keywords and no cue words	0.66	**0.77**	0.71	*0.56*
FNC50	50% matched with frequent keywords and no cue words	0.78	0.76	0.77	0.64
FC55	55% matched metadata and cue words overall	*0.61*	*0.75*	*0.67*	0.62
FNC50FC55	Combination of the second and the third experiment	**0.81**	0.75	**0.78**	**0.65**

5.4.1 ROC and Precision-Recall Curves

Receiver Operating Characteristic (ROC) curves are shown in Figure 13, in which we can see that the area under the curve for NFNQ50 is 0.90, which is the maximum among the other curves because recall for this experiment was maximum.

Precision-Recall curves are shown in Figure 14. As can be seen that the areas under curves for all three experiment 2, 3 and 4 are almost same because recall for these experiments is almost the same, and the area under the curve for experiment 1 (NFNQ50) is the minimum.

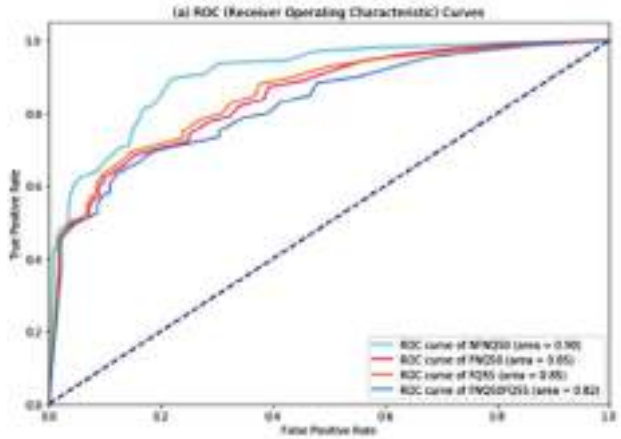


Figure 13: ROC (Receiver Operating Characteristic) Curves

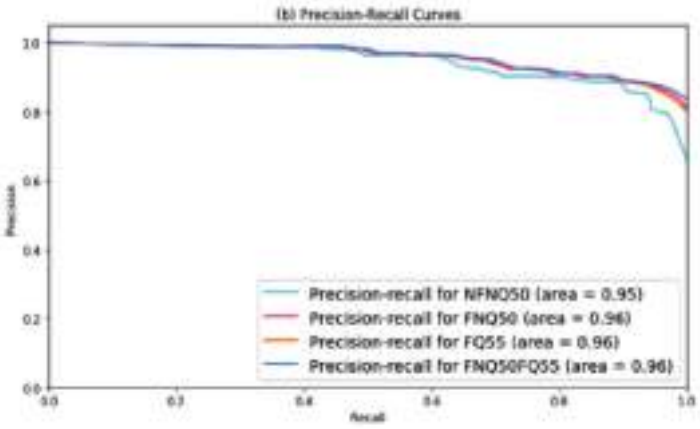


Figure 14: Precision-Recall Curves

6. Conclusion and Future Work

6.1 Concluding Remarks

Linking non-textual document elements (NTDEs), such as charts, diagrams, pseudocodes and tables to their relevant paragraph in a scholarly document is a critical process to improve the relevant results for IR systems which are mainly focused on this type of search. In this research we have focused on algorithms and their complexities linking using complexity lines synopsis.

Scientific publications host a tremendous number of high-quality algorithms developed by professional researchers. In this paper we have linked the complexity lines and the algorithmic metadata in the same scientific document. We have used keywords from algorithmic metadata and a synopsis generated from five lines before and after the complexity line. Complexity of algorithms, for both time and space, is the main concern of developers and researchers. Currently, IR systems for algorithmic search did not directly consider relevant complexity of algorithms to rank and order the results.

In our research, a frequent keywords set and two cue words sets have been used to improve our results. Precision, recall, F1-meaure and accuracy graphs have been used for thresholds selections. Complexity lines have been identified by regular expressions with the use of asymptotic notations. WordNet library has been used for synonyms or related terms. A reference file has been manually annotated for results and comparisons. An associated file has been created to save the links between the complexity lines and their corresponding algorithms. A number of experiments have been performed

with different combinations of frequent keywords, cue words, synonyms or related terms, and thresholds selections for metadata comparison. We have improved our results by combining the best performing experiments.

Future Work

In the future we can use similar linking methodology to link different non-textual document elements, such as figures, tables and charts to their relevant paragraphs in the same document. By using our methodology, we can extract and catalogue relevant algorithms and can introduce several exciting applications including discovering new or enhanced algorithms or analysing different versions of an algorithm. We can also improve algorithmic information retrieval systems by using the complexity of algorithms to index and rank the algorithmic search results. An AI enabled search engine architecture is used in (Safder, Hassan and Aljohani, 2018) and (Safder and Hassan, 2018), where an EMD embedded model is used to improve relevant information retrieval, which is a RCNN based model; we can use our algorithmic metadata and complexity lines context to improve this search engine.

References

Al-Zaidy, R. A. and Giles, C. L. (2017). A Machine Learning Approach for Semantic Structuring of Scientific Charts in Scholarly Documents. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(2), 4644–4649. Avajlable at: https://doi.org/10.1609/aaai.v31i2.19088.

Bajracharya, S., Ossher, J. and Lopes, C. (2009). Sourcerer: An Internet-scale Software Repository. In: *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*. IEEE Computer Society, 1–4.

Baker, J. B., Sexton, A. P., Sorge, V. and Suzuki, M. (2011). Comparing Approaches to Mathematical Document Analysis from PDF. In: *International Conference on Document Analysis and Recognition*. IEEE, 463–467.

Bhatia, S. and Mitra, P. (2012). Summarizing Figures, Tables, and Algorithms in Scientific Publications to Augment Search Results. *ACM Transactions on Information Systems (TOIS),* 30(1), 3.

Bhatia, S., Mitra, P. and Giles, C. L. (2010). Finding Algorithms in Scientific Articles. In: *Proceedings of the 19th International Conference on World Wide Web*. NY: Association for Computing Machinery (ACM), 1061–1062.

Bhatia, S., Tuarob, S., Mitra, P. and Giles, C. L. (2011). An Algorithm Search Engine for Software Developers. In: *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. NY: ACM, 13–16.

Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research,* 3(Jan), 993–1022.

Chen, H.-H., Gou, L., Zhang, X. and Giles, C. L. (2011). Collabseer: A Search Engine for Collaboration Discovery. In: *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*. NY: ACM, 231–240.

Chen, P., Xie, H., Maslov, S. and Redner, S. (2007). Finding Scientific Gems with Google's PageRank Algorithm. *Journal of Informetrics,* 1(1), 8–15.

Coüasnon, B. and Lemaitre, A. (2014). Recognition of Tables and Forms. In: *Handbook of Document Image Processing and Recognition.* London: Springer, 647–677.

Cormen, T. H. (2013). *Algorithms Unlocked.* The MIT Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). *Introduction to Algorithms.* 3rd ed. The MIT Press.

Elminaam, D. S., Abdual-Kader, H. M. and Hadhoud, M. M. (2010). Evaluating the Performance of Symmetric Encryption Algorithms. *IJ Network Security,* 10(3), 216–222.

Hearst, M. A., Divoli, A., Guturu, H., Ksikes, A., Nakov, P., Wooldridge, M. A. and Ye, J. (2007). BioText Search Engine: Beyond Abstract Search. *Bioinformatics,* 23(16), 2196–2197.

Hirschberg, D. S. (1975). A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM,* 18(6), 341–343.

Jung, E., Elmallah, E. S. and Gouda, M. G. (2007). Optimal Dispersal of Certificate Chains. *IEEE Transactions on Parallel and Distributed Systems,* 18(4), 474–484.

Keogh, E., Chu, S., Hart, D. and Pazzani, M. (2001). An Online Algorithm for Segmenting Time Series. In: *Proceedings 2001 IEEE International Conference on Data Mining.* IEEE, 289–296.

Khabsa, M. and Giles, C. L. (2014). The Number of Scholarly Documents on the Public Web. *PloS one,* 9(5), e93949. Available at: https://doi.org/10.1371/journal.pone.0093949.

Khabsa, M., Treeratpituk, P. and Giles, C. L. (2012). Ackseer: A Repository and Search Engine for Automatically Extracted Acknowledgments from Digital Libraries. In: *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries*. NY: ACM, 185–194. .

Khan, S., Liu, X., Shakil, K. A. and Alam, M. (2017). A Survey on Scholarly Data: From Big Data Perspective. *Information Processing & Management,* 53(4), 923–944.

Kim, H.-S., Lee, J.-H. and Jeong, Y.-S. (2003). Method for Finding Shortest Path to Destination in Traffic Network Using Dijkstra Algorithm or Floyd-warshall Algorithm. Google Patents.

Kleinberg, J. and Tardos, É. (2009). *Algorithm Design.* Boston: Pearson/Addison-Wesley.

Kumar, V., Marathe, M. V., Parthasarathy, S. and Srinivasan, A. (2004). End-to-end Packet-scheduling in Wireless Ad-hoc Networks. In: *Proceedings of the Fifteenth*

Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 1021–1030.

Lai, S., Xu, L., Liu, K. and Zhao, J. (2015). Recurrent Convolutional Neural Networks for Text Classification. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), 2267–2273. Avajlable at: https://doi.org/10.1609/aaai.v29i1.9513.

Li, H. (2013). Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM. *arXiv.org e-Print arXiv:1303.3997.* Available at: https://doi.org/10.48550/arXiv.1303.3997.

Liu, Y., Bai, K., Mitra, P. and Giles, C. L. (2007). Tableseer: Automatic Table Metadata Extraction and Searching in Digital Libraries. In: *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries*. NY: ACM, 91–100.

Marron, M., Stefanovic, D., Hermenegildo, M. and Kapur, D. (2007). Heap Analysis in the Presence of Collection Libraries. In: *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. NY: ACM, 31–36.

McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C. and Xie, Q. (2012). Exemplar: A Source Code Search Engine for Finding Highly Relevant Applications. *IEEE Transactions on Software Engineering,* 38(5), 1069–1087.

Milidiú, R. L., Laber, E. S. and Pessoa, A. A. (1999). A Work-efficient Parallel Algorithm for Constructing Huffman Codes. In: *Data Compression Conference, 1999. Proceedings. DCC'99*. IEEE, 277–286.

Nadeem, A. and Javed, M. Y. (2005). A Performance Comparison of Data Encryption Algorithms. In: *2005 First International Conference on Information and Communication Technologies (ICICT)*. IEEE, 84–89.

Nargesian, F., Zhu, E., Pu, K. Q. and Miller, R. J. (2018). Table Union Search on Open Data. In: *Proceedings of the VLDB Endowment,* 11(7), 813–825.

Ratliff, N. D. and Bagnell, J. A. (2007). Kernel Conjugate Gradient for Fast Kernel Machines. In: *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, 1017–1022.

Safder, I. and Hassan, S.-U. (2018). DS4A: Deep Search System for Algorithms from Full-text Scholarly Big Data. In: *International Conference on Data Mining Workshop (ICDMW),* 1308–1315.

Safder, I., Hassan, S.-U. and Aljohani, N. R. (2018). AI Cognition in Searching for Relevant Knowledge from Scholarly Big Data, Using a Multi-layer Perceptron and Recurrent Convolutional Neural Network Model. In: *Companion of the The Web Conference 2018 on The Web Conference 2018*. International World Wide Web Conferences Steering Committee, 251–258.

Safder, I., Sarfraz, J., Hassan, S.-U., Ali, M. and Tuarob, S. (2017). Detecting Target Text Related to Algorithmic Efficiency in Scholarly Big Data using Recurrent Convolutional Neural Network Model. In: Choemprayong, S., Crestani, F., Cunningham, S., (eds.). *Digital Libraries: Data, Information, and Knowledge for Digital Lives. ICADL 2017.* Cham: Springer, 30–40.

Siegel, N., Horvitz, Z., Levin, R., Divvala, S. and Farhadi, A. (2016). FigureSeer: Parsing Result-figures in Research Papers. In: Leibe, B., Matas, J., Sebe, N., Welling, M., (eds.). *Computer Vision – ECCV 2016,* 9911, 664–680. Available at: https://link.springer.com/chapter/10.1007/978-3-319-46478-7_41.

Siegel, N., Lourie, N., Power, R. and Ammar, W. (2018). Extracting Scientific Figures with Distantly Supervised Neural Networks. In: *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*. NY: ACM, 223–232.

Stewart, J. G. (2009). Genre Oriented Summarization. [PhD diss.]. Carnegie Mellon University, Language Technologies Institute, School of Computer Science.

Ochieng, P. J., Djatna, T. and Kusuma, W. A. (2015). Tandem Repeats Analysis in DNA Sequences Based on Improved Burrows-Wheeler Transform. In: *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 117–122.

Tuarob, S., Bhatia, S., Mitra, P. and Giles, C. L. (2013). Automatic Detection of Pseudocodes in Scholarly Documents Using Machine Learning. In: *12th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 738–742.

Tuarob, S., Bhatia, S., Mitra, P. and Giles, C. L. (2016). AlgorithmSeer: A System for Extracting and Searching for Algorithms in Scholarly Big Data. *IEEE Transactions on Big Data,* 2(1), 3–17.

Tuarob, S., Mitra, P. and Giles, C. L. (2015). A Hybrid Approach to Discover Semantic Hierarchical Sections in Scholarly Documents. In: *13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 1081–1085.

Tyagi, N. and Sharma, S. (2012). Weighted Page Rank Algorithm Based on Number of Visits of Links of Web Page. *International Journal of Soft Computing and Engineering (IJSCE),* 2(3), 2231–2307. Available at: https://www.ijsce.org/wp-content/uploads/papers/v2i3/C0796062312.pdf.

Wang, J. (2009). Mean-variance Analysis: A New Document Ranking Theory in Information Retrieval. *European Conference on Information Retrieval*. Springer, 4–16.

Wise, M. J. (1995). Neweyes: A System for Comparing Biological Sequences Using the Running Karp-Rabin Greedy String-Tiling Algorithm. In: *Proceedings. International Conference on Intelligent Systems for Molecular Biology*, 3, 393–401.

Wu, J., Williams, K. M., Chen, H.-H., Khabsa, M., Caragea, C., Tuarob, S., Ororbia, A., D Jordan, D. and Giles, C. L. (2015). Citeseerx: AI in a Digital Library Search Engine. *AI Magazine,* 36(3), 35–48.

Yang, Y., Yu, P. and Gan, Y. (2011). Experimental Study on the Five Sort Algorithms. In: *2011 Second International Conference on Mechanic Automation and Control Engineering (MACE)*. IEEE, 1314–1317.

Zanibbi, R. and Blostein, D. (2012). Recognition and Retrieval of Mathematical Expressions. *International Journal on Document Analysis and Recognition (IJDAR),* 15(4), 331–357.