A Conceptual Dependency Graph Based Keyword Extraction Model for Source Code to API Documentation Mapping

Nakul Sharma, Prasanth Yalla

Abstract*: Natural language processing on software systems usually contain high dimensional noisy and irrelevant features which lead to inaccurate and poor contextual similarity between the project source code and its API documentation. Most of the traditional source code analysis models are independent of finding and extracting the relevant features for contextual similarity. As the size of the project source code and its related API documentation increases, these models incorporate the contextual similarity between the source code and API documentation for code analysis. One of the best solutions for this problem is finding the essential features using the source code dependency graph. In this paper, the dependency graph is used to compute the contextual similarity computation between the source code metrics and its API documents. A novel contextual similarity measure is used to find the relationship between the project source code metrics to the API documents. Proposed model is evaluated on different project source codes and API documents in terms of pre-processing, context similarity and runtime. Experimental results show that the proposed model has high computational efficiency compared to the existing models on the large size datasets.

***Index Terms*: contextual similarity, Natural Language Processing, Text Mining, code analysis, Dependency graph.**

I. INTRODUCTION

The measurement of contextual similarity plays a vital role in many domain applications such as software project analysis and natural language processing. Traditional key feature extraction techniques use terms or sentences from the project source codes to form a unique code structure. So, these techniques usually rank the phrase or sentences in the source code documents according to their predefined characteristics such as term-frequency or inverse term frequency. Almost all traditional document key phrase extraction techniques represent a document collection as the phrase or sentence matrix in which each row denotes the phrase or sentence-id and corresponding column represents the frequency. The main issue with these methods are that they ignore the textual context information and assumes the phrase or sentences as independent to each code document. It is very much difficult to understand the structure of a large and complicated software system. Program comprehension is considered as the most important factor during the development and maintenance phases of software development life cycle. The software developers need more time in order to understand the source code completely. Source code analysis is implemented in order to identify software architecture. This scheme assists software developers for better understanding during the reverse software engineering process. The main objective of Source Code Analysis (SCA) is to implement an appropriate key feature extraction scheme on large source code data. It partitions the large complex software system into smaller and simple sub-systems. Almost all the software SCA techniques use Artifacts Dependency Graph (ADG) in order to determine all the artifacts of the software system along with the essential dependencies. In the SCA dependency graph, software artifacts (class, function, file, and so on.) are denoted by vertices. Again, edges are responsible for representing all dependencies among the above artifacts. SCA models require various domain concepts and the relationships to analyse the structure of the source code in dynamic software systems. Also, the representations of concepts also vary at the time of development and evolution. A large numbers of research works have been carried out in order to analyse the methods and identifiers names using the graph dependency models. In order to perform the (re-)construction of graph model for a particular program, a group of key identifiers are used to extract the domain ontology automatically. Software developers are capable of using SCA in small size projects. However, the complete process of gathering, analysing and implementing of large size project source code is very complicated in key term identification. The developer team has the responsibility to detect the exact source files those contain API key terms. In the subsequent step, they allocate a specific team in order to perform certain modifications. In case complex software projects, it is very hard and time consuming to detect the API key terms from large numbers of source code files. In the above scenario, the development team is required to give extra effort for understanding various portions of the program in order to detect higher level concepts those are explained in a bug report or source code projects.

Revised Manuscript Received on 30 July 2019.
* Correspondence Author
Nakul Sharma*, Research Scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur(Dist), India
Prasanth Yalla, Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur(Dist), India.

Hence, it is very much necessary to develop an automatic system that can assist the development team in order to detect the source code files according to a particular API report. Most of these approaches usually rank the source files based on the relevance of a particular API report. The process of software design is considered as the most difficult process in the complete software development life cycle.

There exist numbers of different design approaches in SDLC, the most common and popular design approach is known as architecture based design approaches. The above mentioned category of design approaches usually uses different software architectural patterns. On the other hand, it can be used as an efficient and effective coordination tools in between different activities of software development process. The most widely accepted architecture based design approach is Attribute Driven Design (ADD). This approach completely depends upon an iterative process in order to implement various architectural methods. A software developer has the responsibility to choose a particular architectural pattern. The implementation process of design patterns in the software development process is considered as a perfect alternative of source code analysis. It has the responsibility to handle the declined quality of software systems. The most common way of document representation in text mining is vector space model. In software systems, each source code is represented in the form of vector for similarity and key term detection process. An alternative to vector space representation is graph dependency model. However, both the vector form and graph dependency models require additional level of complexity on large source code datasets. In most of the software code analysis models, a graph dependency is constructed to find the structural information between the source code metrics. In the existing works, the contextual similarity is computed on the dependency graph for structural similarity identification in software projects. These existing works are independent of finding the contextual relationship between the source code and its related API documents due to high computational memory and large numbers of candidate sets. The work described in this paper addressed this issue by applying a contextual dependency graph measure between the source code files and its related API documents for structural analysis. The main idea behind this approach is that in traditional source code analysis or graph dependency models, all possible candidate sets of the source code metrics are assumed to have equal weightage for similarity computation. In the proposed model, a weighted graph dependency model is used to filter the candidate sets among the vertices for contextual similarity computation. The rest of this paper is organized as follows. In part II a brief summarization of related works is presented. The proposed model is discussed in part III. In the part IV, experimental analysis of the proposed work and existing works are discussed followed by conclusion in part V.

II. RELATED WORK

S. Mohammadi et.al introduced a new approach is presented to extract the knowledge of dependency between artifacts in the source code. Similarity index approaches have been used to retrieve the code structure of a particular software system. An artifact dependency graph is mostly used during the process of similarity computation. Type of artifact dependency graph is usually generated from the source code. Different types of hierarchical and search based clustering approaches are used during the extraction of software architecture. But, these approaches are not efficient to detect the software architecture. On the other hand, search-based techniques are much better as compared to the hierarchical approaches. In case of large-scale software systems, the time and space complexity make the software projects inefficient. V. U. Gómez, et.al, proposed a semantic model on the visually characterizing source code modifications [2]. In the case of the revision control system, automatic and efficient merging approaches are implemented. It usually assists the developers to integrate the modifications with development repositories. These systems are used to detect conflicts in small projects. These are inefficient and incapable to protect the semantic sequences of complex software projects. The release masters or testers are not completely supported in order to make decisions regarding the combination of modifications. The release master is required to read each and every modified code. It integrates the text-based information along with visual representation on several metrics. S. L. Abebe et.al has introduced a new extraction scheme that is sufficiently effective to extract domain concepts from the source code[3]. Program understanding can be defined as a process whereby domain concepts are mapped to the code elements. This type of process of mapping is actually implicit and undocumented. In this work, two different techniques were introduced to exploit the structural and linguistic characteristics of the ontologies of source code. They implemented an advanced filtering approach based on information retrieval in order to carry out the process of filtering domain concepts. The resulting ontologies are evaluated by the manually defined ontology of the domain(ONTOSE). S. Bajracharya, et al, developed a new SCA framework to collect and analyze open source code on a large scale[4]. Open source code search engines in an aggregated repository give access to the source code in the present era. They are unable to see the benefits of information about the structure that is present in the code. Usually search tools require structural and textual information for each source code. They have proposed a new framework for open source code large-scale analysis. R. Gharibi focused on leveraging bug reports ' textual characteristics to locate relevant source files[5]. When a new bug report is received, a developer will need to generate the bug to perform the code review process. The code review process is responsible for identifying this bug. After that, it's necessary to fix the detected bug as soon as possible. The complete process of detecting and fixing bugs has become more complicated and time-consuming due to large numbers of bug reports and increasing software projects. Several bug localization approaches are developed to solve the above mentioned problem. These approaches have a responsibility to rank all of a specific project's source files. In addition, it also assists developers in obtaining relevant source files within a short time.

They presented a multi-component bug localization technique in this research work that includes different textual characteristics of bug reports and source files along with the relationships between already fixed bug reports and the new bug report. This technique includes basic concepts of information recovery approach, text matching technique, stack trace analysis, and multi-label classification to improve overall bug location performance.

A. S. Yumaganov proposed to compare different search models for similarity with limitations on the source code[6]. Keyword search approach is regarded as the most widely accepted approach to querying data sets for tree structure. Most of the keyword search techniques that have been developed before return the lowest common ancestor of a specific keyword. Again, it has the responsibility to match the relevant keyword query to all ranks. As the number of keywords grows along with the complexity, the expected efficiency is very difficult to achieve. To solve the above-mentioned issue, a majority of the approaches to the rankings restrict their ranking to the LCAs subset. They introduced a new top k size stack-based technique for tree structured data in this research work piece. This technique has the responsibility of applying text based on the concept of LCA size for keyword queries.

A. Dimitriou et.al, introduced a new keyword search of top-k-size on tree structured data[7]. Software documents are considered to be the fundamental artifacts generated during the lifecycle of software development. As we know, the knowledge-based techniques have been implemented for many years during the lifecycle of software development. There is no major research work that explains how knowledge-based techniques are more specifically implemented during the software documentation phase during the software architecture design documentation. This research technique's primary aim is to explain how knowledge-based techniques are applied during the process of software documentation. To detect the primary studies from different knowledge-based techniques, they implemented a new approach.

W. Ding proposed a review of software documentation process knowledge-based techniques[8]. It helps software developers select a suitable pattern of design from a large number of patterns to solve the design issue. Most of the traditional approaches that have already existed are limited to semi-formal specification, multiclass problem, etc. A suitable sample size is very important for making the learning process more efficient. It is the responsibility of the individual classifier training to detect a class of design pattern candidates. They presented a technique of text categorization using the fuzzy c means approach to solve all problems.

Using the text categorization technique[9], Hussain et.al proposed a new software design pattern classification and selection scheme. For source code analysis, most of the traditional customization techniques use re-ranked search results. Consequently, all the documents that the user may prefer are usually presented higher. They presented a new scientometric r-ranking technique which depends entirely on a specific source code's scientometric preferences.

To enhance the search results, Ibrahim et.al presented a scientometric re-ranking technique[10]. Using the Bayesian text classification scheme, highly relevant keyword extraction process is implemented. It has the prime objective of performing the keyword extraction feature process during the classification phase. The high relevance keyword extraction technique uses the subsequent probability value of various keywords to perform the keyword extraction process.

L. H. Lee, et.al, used Bayesian text classification to introduce high relevance keyword extraction process[11]. During the software measurement process, source code metrics are considered as the most important components. Usually these components are removed from the software's source code. The component value helps us to assess the quality attributes.

A survey on different source code metrics was conducted by S. Nuñez-Varela, et.al[12]. In case of structured and unstructured data, the Entity resolution process can be defined as the process of disambiguating and resolving entities. During the construction of context graphs, source code documents are used to disclose the indirect co-occurrence relationships between different word numbers. Graph feature vectors can be created using information gain and wordnet (IGEE).

III. PROPOSED MODEL

As the size of the software project data is increasing exponentially, it is practically difficult to analyze the large volumes of individual source code documents due to noise and imbalancing nature. Feature extraction is the process of abstracting the main content from the different document sources and it has become an integral part of the day to day activities in all software projects. Automatic source code feature extraction fulfills certain goals by implementing key phrase extraction techniques to find relevant features of the large document set.

The two research papers which were taken as the basis of the current work are Saroj et. al. [15] and Amir et. al. [17].

The proposed model is summarized in figure 1. In the figure, initially different types of software projects and its API documentation are taken as input to the proposed model. Project source codes are parsed using the class parsing libraries for code dependency graph. Here, the code dependency graph is constructed to extract the methods and fields in each project class file. Similarly, project API documents are pre-processed to remove the un-used text information for contextual graph similarity.

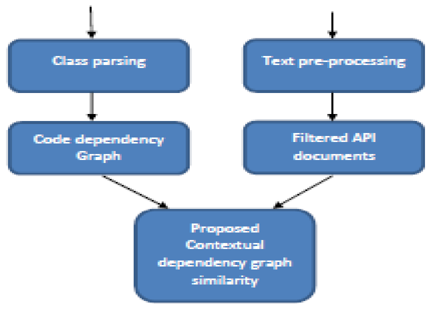


Figure 1. Proposed Model

The proposed model is implemented in two phases for source code to API document similarity extraction. In the first phase of the proposed model, source code metrics are extracted from the project and its API documents are pre-processed.

In the second phase, different contextual relationships are extracted in between the source code and API documents using the proposed contextual similarity measure.

Phase 1: Source Code and API documents Pre-processing

```
Step 1: Read project source codes S.
Step 2: Read project API documents D.
Step 3: for each code Ci in S[]
  Do
      Parse source code Ci with methods M and Fields F.
        Mi=ExtractMethods(Ci)
    Fi=ExtractFields(Ci)
    Mapping (Mi , Fi) to Ci
```

C_1	(M_1,F_1)
C_2	(M_2,F_2)
….	……
Cn	(Mn,Fn)

```
done
Step 4: // Remove the duplicate methods and fields in each class
  For each code Ci
  Do
    If( Mi!=0 AND Fi!=0)
    Then
      Remove Mi in Ci or Cj
      Remove Fi in Ci or Cj
    End if
  Done
Step 5: //Pre-processing API documents of related projects.
  For each document di in D
    Do
      T[]=Tokenize(di)
      For each token t in T[]
      Do
        Apply stemming, stopword removal using Stanford NLP library.
      Done
    Parse API document di with methods M and Fields F.
        DMi=ExtractMethods(di)
      DFi=ExtractFields(di)
      Mapping (DMi , DFi) to di
        Done
```

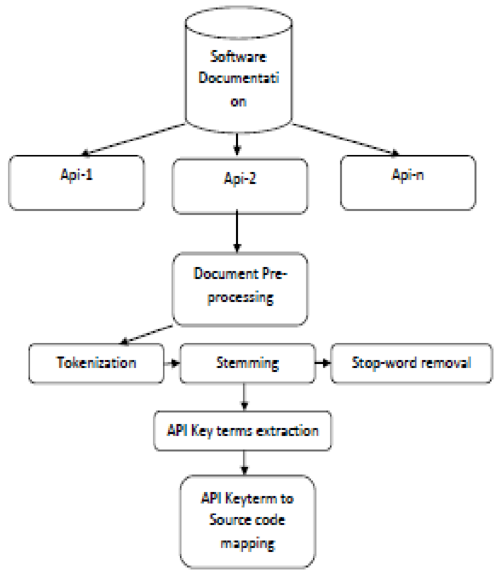


Figure 2. Proposed Model Expanded (Step-5)

In the first phase of the algorithm, project source codes and its API documents are taken as input for pre-processing. To each source code, methods and fields are parsed in the step3. In step 4, duplicate source code metrics are removed to filter the candidate sets using the existence of probability value. In step 5, API project documents are pre-processing using the NLP library.

Phase 2: Query based contextual source code to API document relation mining

Input : Project source codes S, Project API documents D, Project source code metrics (Mi,Fi) and API document metrics (DMi,DFi).

Procedure:

Step 1: Read source code metrics , ci· (Mi,Fi) and API documentation metrics di· (DMi,DFi)

Step 2: Construct dependency graph DG· (V,E) with vertex set V and Edge set E. Here vertex set V is represented with source code methods and fields and edge set E is represented as weighted rank between the vertices.

Step 3: The weights of the edges are computed using the vertex terms ti and tj where and is as shown in equation-1

$$\text{Edgeweight}: w(i,j)=\sum_{\forall i,j}\frac{F(t_i,t_j)}{F(t_i)+F(t_j)-F(t_i,t_j)} \quad ..1$$

$F(t_i,t_j)$ is the number of times both terms occurred together.

$F(t_i)$ is the number of occurrence of in vertex Vi

$F(t_j)$ is the number of occurrence of in vertex Vj

Step 4: The vertices with positive edge weights are sorted in ascending order in the dependency graph to find the contextual similarity computation.

Step 5: Dependency graph DG is used to find the contextual similarity between the vertex nodes to the API documents using the following proposed measure.

Let V(Mi) ← (m1,m2,….mn) denotes the source codes methods vector at vertex i.

V(Fi) ← (f1,f2,….fn) denotes the source code fields vector at vertex i.

DMk← (dm1, dm2, … .dmk) denotes the API document methods vector at index k.

DFi← (df1, df2,….dfk) denotes the API document fields vector at vertex k.

$$|V(M_i)| = \sqrt{V(m_1)^2 + V(m_2)^2 .... V(m_n)^2}$$

$$|V(F_i)| = \sqrt{V(f_1)^2 + V(f_2)^2 .... V(f_n)^2}$$

$$|DM_k| = \sqrt{dm_1^{\ 2} + dm_2^{\ 2} ... d\,m_k^{\ 2}}$$

$$|DF_k| = \sqrt{df_1^{\ 2} + df_2^{\ 2} ... df_k^{\ 2}}$$

$$|V(M_i).DM_j| = V(m_1).df_1 + V(m_2).df_2 ... + V(m_n).df_k$$

Proposed Contextual depenedency graph similarity index is computed as

$$\text{CDGSI} = \frac{\sqrt{V(M_i).DM_j.\text{Max}\{|M_i|,|DM_j|}}{\text{Max}\{|M_i|,|DM_j|}$$

Step 5: User query based source code to API document extraction.

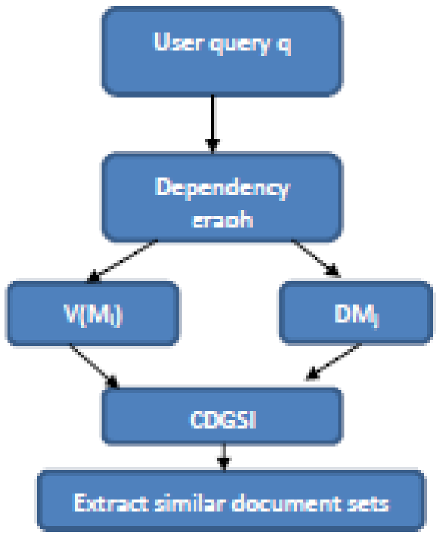


Figure 3. Query based source code to API similarity extraction

```
Input user query q.
 For each vertex Vi in DG
   Do
      Compute CGGS between Vi and DM using equation 1.
      Display similarity results
  Done.
```

IV. EXPERIMENTAL RESULTS

Experimental results are performed on different object oriented software projects and its documentation. A total of five java open source projects and its API documents are taken as input to validate the performance of the proposed model to the existing models. The five open source projects are summarized in table 1.The sizes of the open source projects vary from 100 to 1500 classes and 5134 to 142553 lines of code. Similarly, the sizes of the open source API documents vary from 100 to 1200 for contextual key term identification. For the experimental evaluation, various performance metrics such as accuracy, similarity index and computational time are used to compare the proposed model to the existing models.

$$\text{Accuracy} := \frac{N_C \cap N_{IC}}{N}$$

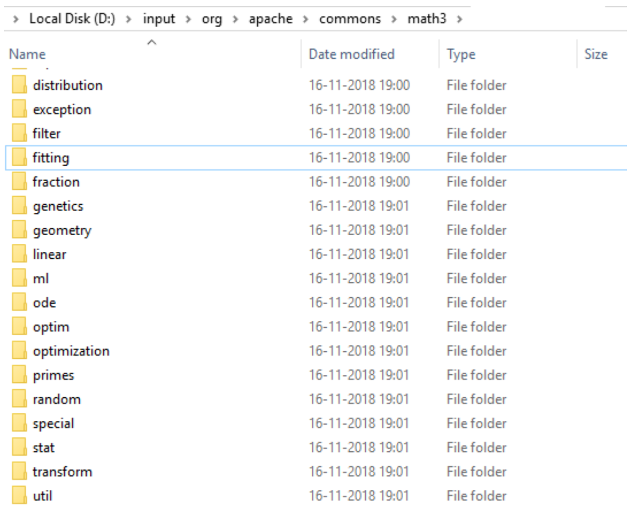
In the above formula NC is source code that predict correctly and NIC is source code reports that predict incorrectly. Accuracy defines the number of the files that predict correctly over the number of files that is matched in the API documents. Context similarity defines the number of source codes that are predicted correctly over the project API documents with high contextual similarity.

Table 1. Summary of selected open source projects with API documentations

Software project	Number of lines	Number of Classes
Apache Pluto	25888	375
Apache Commons Collections	26371	441
JEuclid	12666	230
JFreeChart	95763	1013
Kryo	24144	347

Table 2. Runtime Comparison of data pre-processing on different open source projects

Software project	AvgRuntime(ms)	Number of Classes
Apache Pluto	6364	375
Apache Commons Collections	7831	441
JEuclid	5747	230
JFreeChart	13646	1013
Kryo	6146	347



› Local Disk (D:) › input › org › apache › commons › math3 ›

Name	Date modified	Type	Size
distribution	16-11-2018 19:00	File folder	
exception	16-11-2018 19:00	File folder	
filter	16-11-2018 19:00	File folder	
fitting	16-11-2018 19:00	File folder	
fraction	16-11-2018 19:00	File folder	
genetics	16-11-2018 19:01	File folder	
geometry	16-11-2018 19:01	File folder	
linear	16-11-2018 19:01	File folder	
ml	16-11-2018 19:01	File folder	
ode	16-11-2018 19:01	File folder	
optim	16-11-2018 19:01	File folder	
optimization	16-11-2018 19:01	File folder	
primes	16-11-2018 19:01	File folder	
random	16-11-2018 19:01	File folder	
special	16-11-2018 19:01	File folder	
stat	16-11-2018 19:01	File folder	
transform	16-11-2018 19:01	File folder	
util	16-11-2018 19:01	File folder	

Figure 4. Sample Source code directories of Apache math commons

Table 2, describes the runtime comparison of proposed model on the open source projects. Here, pre-processing model is applied on the source code files and its related API documents. From the table , it is observed that the present model has less runtime on the open source projects.

Figure 3, illustrates the source code view of apache math commons project as input to the proposed model.

Table 3. Apache math commons: Dependency graph before Pre-processing.

```
Parsing the source code ...
Finding code structure...
Extracting source code metrics...
UnivariateVectorFunction  0.932
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex DerivativeStructure
 Vertex getFreeParameters
 Vertex getOrder
 Vertex createConstant
 Vertex getReal
 Vertex getValue
 Vertex getPartialDerivative
 Vertex getAllDerivatives
 Vertex add
 Vertex add
 Vertex subtract
 Vertex subtract
Vertex variables
 Vertex order
 Vertex data
 Edge  DerivativeStructure  compiler  0.896
 Edge  DerivativeStructure  data  0.935
 Edge  DerivativeStructure  order  0.959
 Edge  DerivativeStructure  data  0.907
 Edge  DerivativeStructure  order  0.907
 Edge  DerivativeStructure  order  0.928
 Edge  DerivativeStructure  data  0.9
 Edge  DerivativeStructure  compiler  0.895
 Edge  DerivativeStructure  data  0.916
 Edge  DerivativeStructure  compiler  0.912
 Edge  DerivativeStructure  data  0.963
 Edge  DerivativeStructure  compiler  0.95
 Edge  DerivativeStructure  data  0.92
 Edge  DerivativeStructure  order  0.954
 Edge  DerivativeStructure  data  0.921
 Edge  DerivativeStructure  compiler  0.922
 Edge  DerivativeStructure  data  0.924
 Edge  getFreeParameters  compiler  0.918
 Edge  getOrder  compiler  0.913
 Edge  createConstant  getFreeParameters  0.945
 Edge  createConstant  getOrder  0.915
 Edge  getReal  data  0.932
 Edge  getValue  data  0.922
 Edge  getPartialDerivative  data  0.924
 Edge  getPartialDerivative  compiler  0.919
 Edge  getAllDerivatives  data  0.928
 Edge  add  data  0.894
 Edge  add  compiler  0.899
 Edge  add  data  0.948
 Edge  subtract  add  0.928
 Edge  subtract  compiler  0.95
 Edge  subtract  data  0.905
 Edge  multiply  data  0.936
 Edge  multiply  compiler  0.954
 Edge  multiply  data  0.947
 Edge  DataTransferObject  data  0.956
 Edge  readResolve  variables  0.93
 Edge  readResolve  order  0.935
 Edge  readResolve  data  0.938
DerivativeStructure  0.915
Type=DerivativeStructure  Type=DSCompiler
Type=DerivativeStructure  Type=DerivativeStructure
 Vertex DataTransferObject
 Vertex readResolve
 Vertex serialVersionUID
 Vertex variables
 Vertex order
 Vertex data
 Edge  DataTransferObject  variables  0.901
 Edge  DataTransferObject  order  0.912
 Edge  DataTransferObject  data  0.957
 Edge  readResolve  variables  0.944
 Edge  readResolve  order  0.968
 Edge  readResolve  data  0.894
   Edge  divide  multiply  0.939
```

Table 4. Apache Math Commons: Dependency graph after

```
Pre-processing
Parsing the source code ...
Finding code structure...
Extracting source code metrics...
UnivariateVectorFunction  0.932
 Vertex DerivativeStructure
 Vertex getFreeParameters
 Vertex getOrder
 Vertex createConstant
 Vertex getReal
 Vertex getValue
 Vertex getPartialDerivative
 Vertex getAllDerivatives
 Vertex add
 Vertex subtract
 Vertex variables
 Vertex order
 Vertex data
 Edge  DerivativeStructure  compiler  0.896
 Edge  DerivativeStructure  data  0.935
 Edge  DerivativeStructure  order  0.959
 Edge  getFreeParameters  compiler  0.918
 Edge  getOrder  compiler  0.913
 Edge  createConstant  getFreeParameters  0.945
 Edge  createConstant  getOrder  0.915
 Edge  getReal  data  0.932
 Edge  getValue  data  0.922
 Edge     getPartialDerivative
data  0.924
```

```
 Edge  getPartialDerivative  compiler  0.919
 Edge  getAllDerivatives  data  0.928
 Edge  add  data  0.894
 Edge  add  compiler  0.899
 Edge  subtract  add  0.928
 Edge  subtract  compiler  0.95
 Edge  subtract  data  0.905
 Edge  multiply  data  0.936
 Edge  multiply  compiler  0.954
 Edge  multiply  data  0.947
 Edge  DataTransferObject  data  0.956
 Edge  readResolve  variables  0.93
 Edge  readResolve  order  0.935
 Edge  readResolve  data  0.938
DerivativeStructure  0.915
Type=DerivativeStructure  Type=DSCompiler
Type=DerivativeStructure  Type=DerivativeStructure
 Vertex DataTransferObject
 Vertex readResolve
 Vertex serialVersionUID
 Edge  DataTransferObject  variables  0.901
 Edge  DataTransferObject  order  0.912
 Edge  DataTransferObject  data  0.957
 Edge  readResolve  variables  0.944
 Edge  readResolve  order  0.968
    Edge  readResolve  data  0.894
```

Table 3 and 4 illustrate the results of graph dependency model before and after applying the pre-processing model. The duplication in the table 3 is filtered in the table 4.

Table 5. Comparative analysis of source code retrieval accuracy on open source projects

Project	LDA	ONTOSE	Proposed Method
Apache Pluto	0.846	0.835	0.9436
Apache Commons Collections	0.736	0.753	0.879
JEuclid	0.794	0.825	0.962
JFreeChart	0.773	0.874	0.921
Kyro	0.874	0.915	0.948

Table 5, describes the accuracy of the proposed model to the existing models for source code to API documents extraction process. From the table 5, it is observed that the proposed model has high computational accuracy between the source code to API documents patterns extraction compared to the existing models.

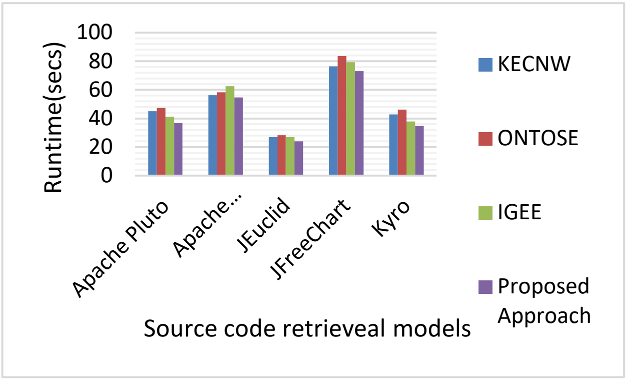


Figure 5. Comparative analysis of source code average retrieval runtime(secs) on open source projects

Figure 5, describes the runtime of the proposed model to the existing models for source code to API documents extraction process. From the figure it is observed that the proposed model has less computational runtime between the source code to API documents patterns extraction compared to the existing models.

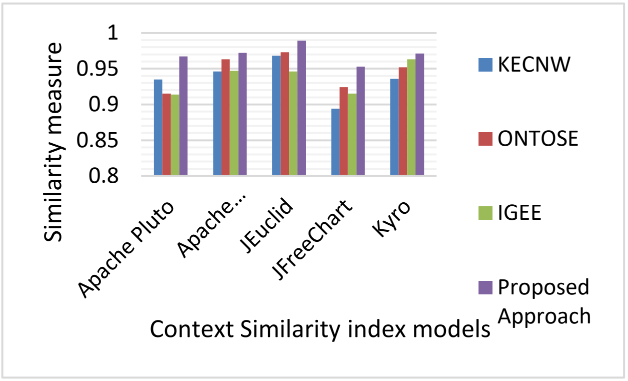


Figure 6. Comparative analysis of average context similarity measure on open source projects

Figure 6, describes the average context similarity of the proposed model to the existing models for source code to API documents extraction process. From the figure, it is observed that the proposed model has high computational average similarity index between the source code to API documents patterns extraction compared to the existing models.

V. CONCULSION

The current paper proposed a novel approach to find the relationship between the source code to API documents using the contextual dependency graph. A two pronged approach is used in the proposed method. The project source code is scanned for the relevant metrics. On the other hand, from the API documentation, necessary information is extracted. Here, the dependency graph is used to compute the contextual similarity computation between the source code metrics and its API documents. A novel contextual similarity measure is used to find the relationship between the project source code metrics to the API documents. Proposed model is evaluated on different project source codes and API documents in terms of pre-processing, context similarity and runtime. Experimental results show that the proposed model has high computational efficiency compared to the existing models on the large size datasets.

REFERENCES

1. G S. Mohammadi and H. Izadkhah , A new algorithm for software clustering considering the knowledge of dependency between artifacts in the source code, Information and Software Technology.
2. V. U. Gómez, S. Ducasse and T. D'Hondta, Visually characterizing source code changes, Science of Computer Programming.
3. S. L. Abebe and P. Tonella, Extraction of domain concepts from the source code, Science ofComputerProgramming98(2015)680–706.
4. S. Bajracharya, J. Ossher and C. Lopes, Sourcerer: An infrastructure for large-scale collection and analysis of open-source code, Science of Computer Programming 79 (2014) 241–259.
5. R. Gharibi, A. H. Rasekh, Md. H. Sadreddini and S. M. Fakhrahmad , Leveraging textual properties of bug reports to localize relevant source files, Information Processing and Management 54 (2018) 1058–1076.
6. A. S. Yumaganov and V. V. Myasnikov , Comparison of the ways of the program's code initial description in the problem of similar code sequences search, 3rd International Conference "Information Technology and Nanotechnology", ITNT-2017, 25-27 April 2017, Samara, Russia, Procedia Engineering 201 (2017) 445–452.
7. A. Dimitriou, D. Theodoratos and T. Sellis , Top-k-size keyword search on tree structured data, Information Systems47(2015)178–193.
8. W. Ding, P. Liang, A. Tang and H. van Vliet , Knowledge-based approaches in software documentation: A systematic literature review, Information and Software Technology 56 (2014) 545–567.
9. S. Hussain, J. Keung and A. A. Khan , Software design patterns classification and selection using text categorization approach, Applied Soft Computing.
10. N. Ibrahim, A. H. Chaibi and H. B. Ghézala , Scientometric re-ranking approach to improve search results, International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2017, 6-8 September 2017, Marseille, France, Procedia Computer Science 112 (2017) 447–456.
11. L. H. Lee, D. Isa, W. O. Choo and W. Y. Chue , High Relevance Keyword Extraction facility for Bayesian text classification on different domains of varying characteristic, Expert Systems with Applications 39 (2012) 1147–1155.
12. A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez and C. Soubervielle-Montalvo , Source code metrics: A systematic mapping study, The Journal of Systems and Software 128 (2017) 164–197.
13. C. Huang, J. Zhu, X. Huang, M. Yang, G. Fung and Q. Hu, A Novel Approach for Entity Resolution in ScientiÞc Documents Using Context Graphs, Information Sciences.
14. W. Qu , Y. Jia and M. Jiang , Pattern mining of cloned codes in software systems, Information Sciences 259 (2014) 544–554.
15. Saroj Kr. Biswas, Monali Bordoloi , Jacob Shreya, A graph based keyword extraction model using collective node weight, Expert Systems With Applications, 97 (2018) 51–59.
16. Amir Hossein Rasekh, Amir Hossein Arshia, Seyed Mostafa Fakhrahmad, Mohammad Hadi Sadreddini; Mining and discovery of hidden relationships between software source codes and related textual documents, Digital Scholarship in the Humanities, Volume 33, Issue 3, 1 September 2018, Pages 651–669, https://doi.org/10.1093/llc/fqx052

AUTHORS PROFILE

Prasanth Yalla received his B.Tech Degree from Acharya Nagarjuna University, Guntur (Dist), India in 2001, M.Tech degree in Computer Science and Engineering from Acharya Nagarjuna University in 2004, and received his Ph.D. degree in CSE titled "A Generic Framework to identify and execute functional test cases for services based on Web Service Description Language" from Acharya Nagarjuna University, Guntur (Dist), India in April 2013. He was an associate professor, with Department of Information Science and Technology in KL University, from 2004 to 2010. Later he worked as Associate professor, with the department of Freshman Engineering from 2011 in KL University. Presently he is working as Professor in the department of Computer Science & Engineering in KL University and also Associate Dean (R&D) looking after the faculty publications. Till now he has published 28 papers in various international journals and 4 papers in conferences. His research interests include Software Engineering, Web services and SOA. He taught several subjects like Multimedia technologies, Distributed Systems, Advanced Software Engineering, Object Oriented Analysis and design, C programming, Object-Oriented programming with C++, Operating Systems , Database management systems, UML etc. He is the Life member of CSI and received "Active Participation- Young Member" Award on 13-12-13 from CSI. He has applied a project to SERB very recently.