International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 05, May-2015

Bug Localization using LDACG Approach

Prof. Devendra Kumar
Computer Science & Engineering Department
IEC College of Engineering and Technology
Greater Noida, UP, India

Ritu Sharma
Computer Science and Engineering Department
IEC College of Engineering and Technology
Greater Noida, UP, India

***Abstract* –Bug Localization is the task of locating the area of source code that requires modification to correct that bug. By automating this task, effort of debugger can be considerably reduced. In past, automated bug localization has been done with the help of many IR(Information Retrieval) models that focused on the semantic information. In this paper, we have proposed LDACG approach for bug localization which focuses on both semantic and structural information. In LDACG approach, bugs are located using an IR model i.e. LDA (Latent Dirichlet Allocation) and Call graph. Then the combined score of both methods is calculated to locate bugs in efficient manner. We have compared LDACG based approach with LDA based approach and it has been found that LDACG approach performs better than LDA approach for bug localization. The performance of both approaches has been evaluated on the datasets downloaded from two open source projects i.e. Rhino and ModeShape.**

Keywords – Bug localization, Call graph, Information Retrieval, LDA.

I. INTRODUCTION

In today's epoch, software companies are competing with quality of the software which depends on better software maintenance task. Software plays a significant role in both business related enterprises and daily life. In large and compound software systems; insufficient-documentation and aging of the software makes software project hard to understand and ultimately leads to a complex software maintenance task. In order to overcome this, proper testing and debugging jobs are required. Software testing is a process that involves any activity aimed at assessing an attribute or potentiality of a program or system and determining that it meets its requisite results. Also it may be defined as the method of executing a program or system with the purpose of discovering errors. It aids to take informed decision by providing the relevant information based on the context. Whereas, debugging is a systematic process of finding and reducing the number of bugs, or flaws, in a computer program, therefore making it act as expected.

Bug localization is a process of mapping a bug back to the code that might have caused it. A computer bug is an error, flaw, mistake, failure, or fault in a computer program that stops it from working correctly or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program's source code or its design.

Bug localization can be performed in static and dynamic manner. Static bug localization works on the source code whereas dynamic bug localization works on execution traces i.e. it requires working software and test case that triggers the bug. The foremost drawback of dynamic technique is that a program or software developed for locating bugs cannot be made language independent. Due to these shortcomings of traditional and dynamic bug localization techniques, researchers have started using IR models for locating the bugs. [3]

An Information Retrieval system is a software program that stores and manages information on documents. The system helps users in finding the information they need. It does not return information or answer questions. Instead, it informs the user about the location of documents that might contain the required information. Thus the goal of any IR system is to identify the documents that are relevant to the user's query. Most people equate information retrieval with web search, but the main purpose of Information Retrieval is "the finding of" concept where user or programmer finds data/information which can have endless uses and application.

One of the techniques applied in IR models is topic modeling which can be defined as:- A topic model (or latent topic model or statistical topic model) refers to a model designed to automatically extract topics from a corpus of text documents. A collection of terms that co-occur frequently in the documents of the corpus, for example {mouse, click, drag, right, left} and {user, account, password, authentication} makes a topic.[5] Topic models are algorithms for discovering the main themes that pervade a large and otherwise unstructured collection of documents. Topic models can organize the collection according to the discovered themes. [1]

In rest of this paper, section II describes the previous work done in this field. Section III describes the proposed techniques and algorithm. Experimental results have been discussed in section IV and proposed work has been concluded in section V.

II. PREVIOUS WORK

In recent times, researchers have applied many IR models for the task of bug localization. Various models used for this purpose are LSI, LDA, N-gram etc.

Deerwester et al. [12]proposed LSI model which is based on the vector space model, which is an algebraic model that represents documents as vectors of terms and relationships among terms and documents as a term-document co-occurrence matrix. Latent Dirichlet allocation (LDA) was proposed by Blei et al. [1] as a topic model that explained similarity among data.

Hayes et al. [11] proposed a technique for bug localization in which LSI model has been used. And to improve the efficiency, historical patch data has also been used. For locating a given bug combined result of both previous history and LSI based approach has been used.

Lukins et al.[3] proposed Latent Dirichlet Allocation model for source code retrieval. And it has been shown that LDA based approach performed better than LSI based approach for bug localization.

Lal et. al. [10] presented technique (which falls into the class of static techniques for bug localization) for fault localization based on a character n-gram based Information Retrieval (IR) model. Problem of bug localization as a relevant document(s) search task for a given query is framed and the application of character-level n-gram based textual features derived from bug reports and source-code file attributes is investigated.

Singh et al.[9] proposed a novel approach for bug localization using call graph reduction where the size of the call graph is reduced without changing the basic structure and no major loss of the information is incurred. The output generated using the proposed methodology showed promising results.

III. PROPOSED WORK

In this work, we have performed Bug localization using LDACG approach. In LDACG approach, bugs are located using LDA and call graph. Then the combined score of both methods is calculated to locate the bugs in efficient manner. Section A provides the background details of the work. While in section B proposed LDACG algorithm has been defined.

A. Background

Bug localization is the process of locating bugs in the source code which are resulting in the abnormal execution of the program or software. Mistakes done in the program are called as bugs. When we locate all the bugs in the program we create its bug report which acts as a query for the user.

Latent Dirichlet Allocation (LDA) is a statistical model that has emerged as a popular technique for discovering topics in a large text document corpus. Thus it can be used in locating bugs in the source code [2]. LDA is implemented on the source code where each v^{th}word can be represented as a V-Vector w such that $w^v = 1$ and $w^u = 0$ for $u \neq v$. The next step involves Document Preprocessing which includes 2 stages. Firstly, Identifier and Comment Excerption: Extract semantic information such as comment and identifier, from each source code element at the desired level of granularity (Classes, Methods, Package etc.) and secondly preprocessing: It involves four further steps like Identifier Separation ,Case Normalization, Stop word elimination and Stemming. The next step involves Document Collection. After the preprocessing is done in previous stages, the data corpus is created, through which the information can be extracted using IR Model. The next stage is LDA Analysis where various parameters are set like the number of topics, the number of iterations, α, a hyper parameter of LDA, determines the amount of smoothing applied to the topic distribution per document. β, a hyper parameter of LDA, determines the amount of smoothing applied to the word distributed per topic. Finally LDA Model is generated.

A call graph (Fig 1) is a software engineering technique which provides a binary relation over selected entities in a program, such as methods, classes, subsystem, modules, files, etc., which represents invocations between those entities. Call graphs are either static or dynamic. A static call graph can be obtained from the source code. It represents all methods of a program as nodes and all possible method invocations as edges. A dynamic call graph is the invocation relation that represents a specific set of runtime executions of a program. A call graph is a directed graph whose nodes represent the functions of program and directed edges symbolize function calls. [11]

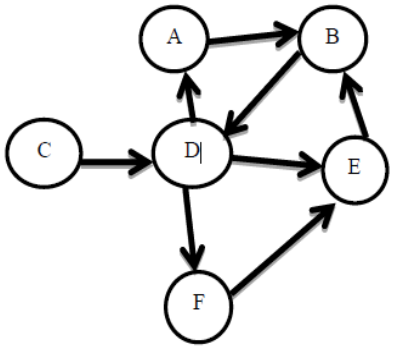

Fig. 1 Call Graph

B. LDACG

Here we have combined the concept of Latent Dirichlet Allocation with Call Graph. Firstly, the user provides a query based on information extracted from a bug report. Then an LDA Model is generated based on the source code listing all the methods in descending order of probability of occurrence (LDASCORE).After that a minimum criterion is selected for the Model. This minimum score is called THRESHOLD. For all the methods under the filtered THRESHOLD score, a call graph is generated and CGSCORE is computed. For every method we compute the new score as follows:

$$NSCORE = \Theta * LDASCORE + (1-\Theta) * CGSCORE \quad (1)$$

Where:

Θ a weight in the range (0,1) that represents weightage of each aspect.

Then we create a new list, LDACG, containing the methods ranked in descending order by NSCORE, and return it to the user. Figure 2 shows the flow chart for proposed LDACG approach.

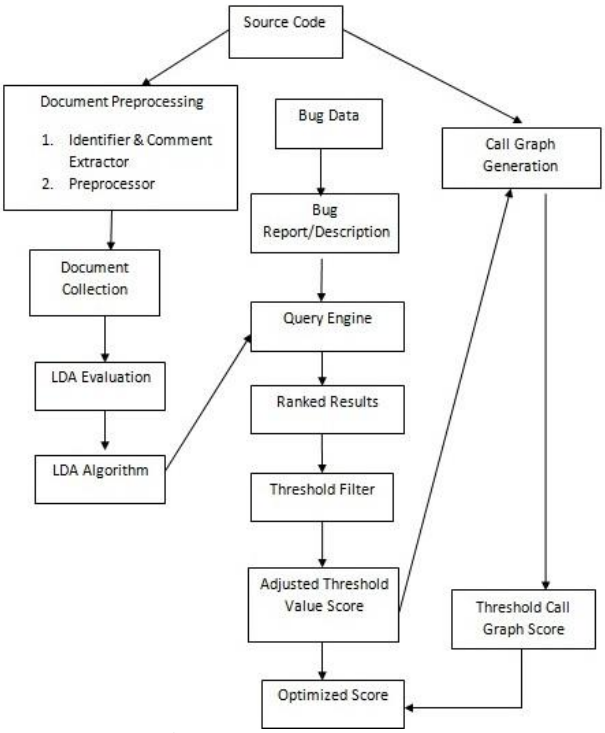

Fig. 2 Flow Chart for LDA using Bug Localization with Call Graph

Algorithm for LDACG

Step 1: User provides query

Using the information provided from a bug report, the user provides a query for the LDA Model generation.

Step 2: LDA Model Generation

After the user query an LDA model is generated based on the source code which contains probability of occurrence of all methods in descending order. This list of score is called as LDASCORE.

Step 3: Selection of Threshold value

Here we select a minimum criterion for the model and select a THRESHOLD value. All those scores which are below THRESHOLD value are neglected.

Step 4: Creation of Call Graph

Now we make a call graph for all those methods that are under the filtered threshold value score and then we compute a CGSCORE.

Step 5: Calculation of Optimized Score

Based on the LDASCORE and CGSCORE we create a new score given by equation:-

NSCORE = Θ*LDASCORE + (1-Θ)*CGSCORE

Step 6: Result to the User

Finally we create a new list in descending order containing the methods ranked by NSCORE and return it to the user.

IV. EXPERIMENTALRESULTS

In this paper, the performance of locating bugs by LDA has been improved by combining it with Call Graph technique. Thus the performance of LDA and LDACG has been compared using the data set of two open source software Rhino (Rhino) and ModeShape (ModeShape Source Code - JBoss Community). For evaluating the performance, we have used two evaluation metrics viz Mean Average Precision(MAP) and Rank of Relevant files.

A.Rhino

For Rhino, it has been observed that the value of MAP for LDA based bug localization is 0.157. While performing bug localization with LDACG it comes out to be 0.177. This comparison has been clearly shown in Fig. 3. It clearly shows the MAP value for LDACG is better than LDA for bug localization.

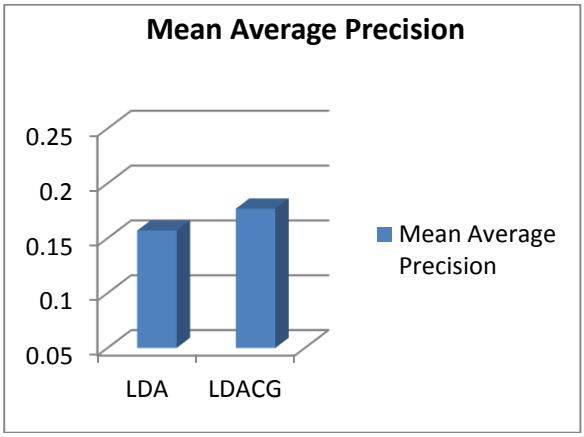

Fig. 3 Comparison between LDA and LDACG approaches using MAP for Rhino

In LDA based approach for locating bugs in Rhino, 20% of bugs are located at Rank less than 5.28% of bugs are located at rank between 6 to 10 and 50% of bugs are located at rank between 11to 20. Fig. 4 illustrates the bugs located with the respective rank ranges.

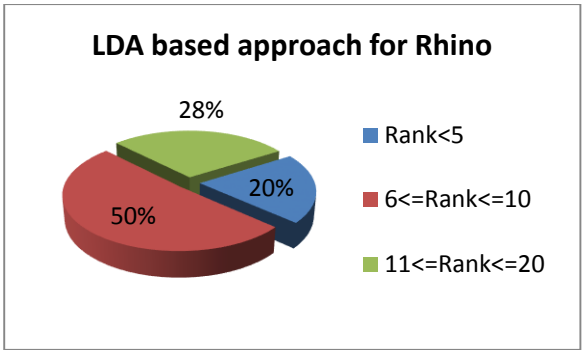

Fig.4 Rank of Relevant files using LDA approach for Rhino

While in LDACG based approach for bug localization in Rhino dataset, 22% of bugs are located at Rank less than 5. 38% of bugs are located at rank between 6 to 10 and 40% of bugs are located at rank between 11to 20. Fig. 5 illustrates the bugs located with the respective rank ranges.

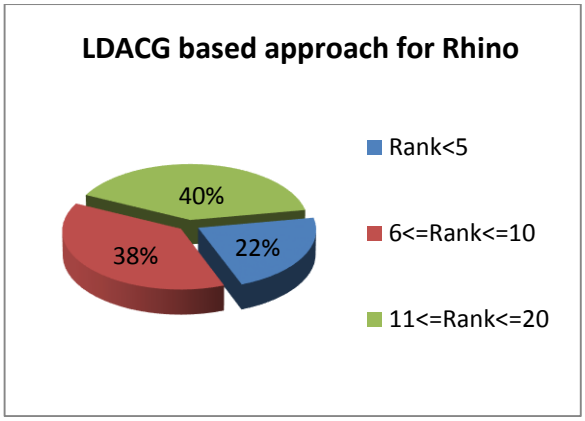

Fig. 5 Rank of Relevant files using LDACG approach for Rhino

B. *ModeShape*

For ModeShape, it has been observed that value of MAP is 0.100 in case of LDA based bug localization. While performing bug localization using LDACG, value of MAP becomes 0.124. This comparison has been clearly shown in Fig. 6. It shows that the MAP of LDACG based approach for bug localization is better than LDA based approach for bug localization.

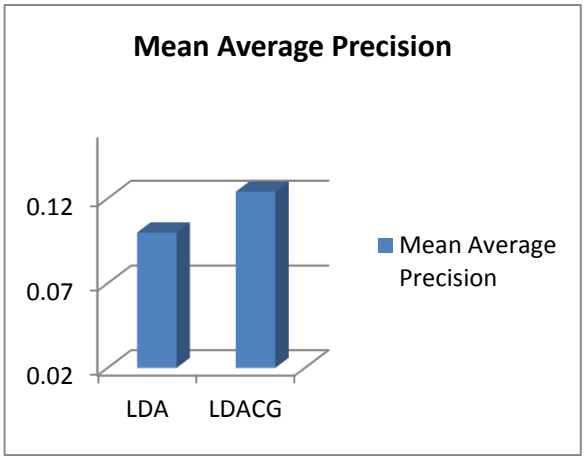

Fig. 6 Comparison between LDA and LDACG approaches using MAP for ModeShape

In LDA based approach, 17% of bugs are located at Rank less than 5.39% of bugs are located at rank between 6 to 10 and 44% of bugs are located at rank between 11to 20. Fig. 7 illustrates the bugs located with the respective rank ranges.

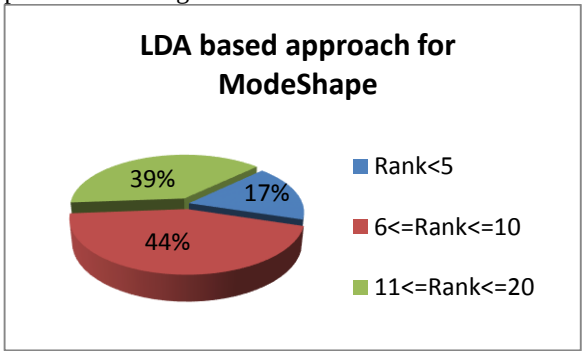

Fig. 7 Rank of Relevant files using LDA approach for ModeShape

In LDACG based approach for bug localization, 15% of bugs are located at Rank less than 5. 37% of bugs are located at rank between 6 to 10 and 48% of bugs are located at rank between 11to20. Fig. 8 illustrates the bugs located with the respective rank ranges.

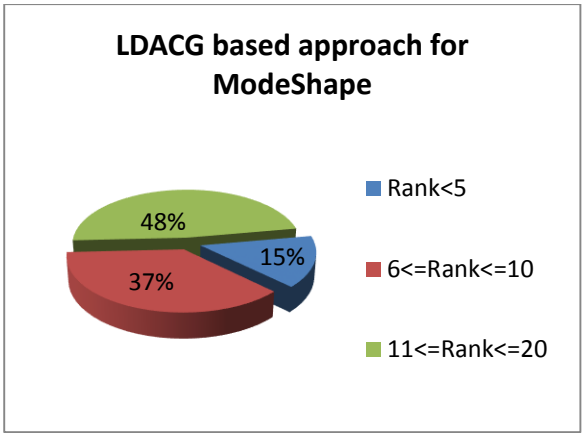

Fig. 8Rank of Relevant files using LDACG for ModeShape

It can be clearly seen from the results that for ModeShape, LDACG based approach has performed better than LDA based approach for bug localization, for both MAP and Ranking metrics.

V. CONCLUSIONS

In present time the quality of software is a major factor in software industry and it can be maintained if software is bug free. In this research paper, we have performed Bug Localization using both lexical (IR Models) and structural (CG) techniques. For locating bugs, LDACG approach using both LDA model and call graph has been used and experimental results have shown that LDACG based approach performs better than LDA based approach for bug localization. For Rhino, it has been observed that the value of MAP for LDA based bug localization is 0.157. While performing bug localization with LDACG it comes out to be 0.177. For ModeShape, it has been observed that value of MAP is 0.100 in case of LDA based bug localization. While performing bug localization using LDACG, value of MAP becomes 0.124.

REFERENCES

[1] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," Journal of Machine Learning Research, vol. 3, pp. 993-1022, 2003.
[2] G. Maskeri, S. Sarkar and K. Heafield, "Mining Business Topics in Source Code using Latent Dirichlet Allocation", ISEC '08 Proceedings of the 1st India software engineering conference, pp. 113-120.
[3]S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation," Proc. 15th Working Conf. Reverse Engineering (WCRE 2008), pp. 155-167, 2008.
[4]M. Beard, N. Kraft, L. Etzkorn and S. Lukins, "Measuring the Accuracy of Information Retrieval," Proc. 15th Working Conf. Reverse Engineering (WCRE 2008), Limerick, pp. 124 – 128, 2008.

[5]S. W. Thomas, "Mining Software Repositories with Topic Models," Proc. 33rd International Conference on Software Engineering (Doctoral Symposium), pp. 1138-1139, 2011.
[6]X. Wang, A. McCallum and X. Wei, "Topical N-grams: Phrase and Topic Discovery," Proc. ICDM'07 Proceedings of the 2007 Seveth IEEE International Conference on Data Mining, pp. 697-702.
[7]J. Zhou, H. Zang and D. Lo, "Where Should the Bugs Be Fixed?," Proc. ICSE 2012 Proceedings of the 2012 International Conference on Software Engineering , pp. 14-24, 2008.
[8]P. Shao, R. k. Smith, N. A. Kraft, T. Atkinson, J. C. Carver and A. S. Parrish, "Combining Information Retrieval Modules And Structural Information For Source Code Bug Localization And Feature Location," Ph.D. dissertation, Department of Computer Science, University of Alabama, 2011.
[9]P. Singh and S. Batra, "A Novel Technique for Call Graph Reduction for Bug Localization," International Journal of Computer Applications (0975 – 888)Volume 47– No.15, June 2012.
[10]S. Lal and A. Sureka, "A Static Technique for Fault Localization Using Character N-Gram Based Information Retrieval Model," Proceedings of ISEC '12, Kanpur, UP, India, February 2012.
[11]C. J. Hayes, B. Nichols,N. A. Kraft and M.D. Anderson,"Improving LSI-Based Bug Localization using Historical Patch data,"The University of Alabama McNair Journal .
[12]S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R.Harshman, "Indexing by Latent Semantic Analysis," Journal Of The American Society For Information Science, 1990.