An Automatic Query Reformulation Based Bug Localization Technique using Keyword-Source Code Association Relations.

Shamima Yeasmin Mohammad Masudur Rahman Chanchal K. Roy Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Canada
shy942@mail.usask.ca, {chanchal.roy, kevin.schneider}@usask.ca

ABSTRACT

Bug localization is a software maintenance task in which a developer formulates a query from information presented in a bug report and using this query to search through the software system in order to determine the portion of the source code that needs to be changed or modified for fixing that given bug. Existing studies on bug localization suffer from two issues - one is low accuracy in performance measure and the other is small-scale experiments. The success of existing studies depended much on the quality of the queries as a poorly designed query is prone to retrieve irrelevant documents. However, designing a query is challenging as it requires certain level of knowledge and expertise of the developer regarding the code base. In this paper, we are proposing a bug localization technique utilizing association maps between keywords extracted from previously fixed bug reports and fixed source code files. Our proposed approach also automatically suggest helpful reformulation of a given query based on semantic similarity between terms. We perform a large-scale experiments on 6000 bug reports and 12000 source code files. We compare the performance of our proposed approach with two information retrieval based static bug localization techniques: TF-IDF based and LDA based. Results auggest that...

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Bug Report, Topic Evolution, Summary, Visualization

1. INTRODUCTION

Bug localization is the process of locating the source codes that need to be changed in order to fix a given bug. Locating buggy files is time-consuming and costly if it is done by manual effort, especially for the large software system when the number of bug of that system becomes large. Therefore, effecting methods for locating bugs automatically from bug reports are highly desirable. In automatic bug localization technique, it takes a subject system as an input and produces a list of entities such as classes, methods etc. against a search query. For example, information retrieval based techniques rank the list of entities by predicted relevance and return a ranked list of entities or source codes which may contain the bug. The bug localization techniques are also affected by the fact of designing effective query. If a query contains inadequate information, then the retrieval results will not be relevant at all. One other thing that the performance of existing bug localization approach did not reach to an accepted level and so far those studies showed good results for a small set of bugs. Therefore, in this paper, we apply a bug localization technique on large dataset that exploits an association link established from bug report repository to source code base through commit logs. We also process the query in a reformulation manner so that the semantic importance between terms both in bug reports and code base could be utilized.

In existing studies, information retrieval techniques [?] [?] [?] [?] [?] are applied to automatically search for relevant files based on bug reports. In one case of Text Retrieval (TR) based approaches, ?] propose BugLocator using revised Vector Space Model (rVSM), which requires the information extracted from bug reports and source codes. One of the issue association with TR-based technique is treating source codes as flat text that lacks structure. But exploiting source code structure can be a useful way to improve bug localization accuracy. Due to the fuzzy nature of information retrieval, textually matching bug reports against source files may have to concede noise (less relevant but accidentally matched words). However, large files are more likely to contain noise as usually only a small part of a large file is relevant to the bug report. So, by treating files as single units or favoring large files, the techniques are more likely to be affected by the noise in large files. So, ?] present BLUiR, which retrieve structural information from code constructs. Bug reports often contain stack-trace information, which may provide direct clues for possible faulty files. Most existing approaches directly treat the bug descriptions as plain texts and do not explicitly consider stack-trace information. Here, ?] combine both stack trace similarity and textual similarity to retrieve potential code elements. To deal with two issues associated with source code structure and stack trace information, **?**] proposed a technique that use segmentation i.e., divides each source code file into segments and use stack-trace analysis, which significantly improve the performance of BugLocator. LDA topic-based approaches [**?**] [**?**] assume that the textual contents of a bug report and it's corresponding buggy source files share some technical aspects of the system. Therefore, they develop topic model that represents those technical aspects as topic. However, existing bug localization approaches applied on small-scale data for evaluation so far. Besides the problem of small-scale evaluations, the performance of the existing bug localization methods can be further improved too. For example, using Latent Dirichlet Allocation (LDA), only buggy files for 22% of Eclipse 3.1 bug reports are ranked in the top 10 [25]. But, now it is an important research question to know how effective these approaches are for locating bugs in large-scale (i.e., big data). In the field of query processing **?**] proposed a method that examines the files retrieved for the initial query supplied by a user and then selects from these files only those additional terms that are in close proximity to the terms in the initial query. Their [**?**] experimental evaluation on two large software projects using more than 4,000 queries showed that the proposed approach leads to significant improvements for bug localization and outperforms the well-known QR methods in the literature. There are two common issues associated with existing studies. First, most of the bug localization techniques exploit lexical similarity measure for retrieving relevant files from the code base. So, it is expected that the search query constricted from new bug report should contain keywords similar to code constructs in code base. This issue demands developers or users previous expertise on the given project, which can not be guaranteed in real world. Second, closely related to the first issue there is another well known problem called vocabulary mismatch. In order to convey the same concept on both search query (i.e., new bug report) and source code files the developers tend to use different vocabularies. This issue questions the applicability of exploiting lexical similarity measure.

So, in order to resolve the above issues, we are proposing a bug localization technique that exploit the association of keywords extracted from fixed bug report with their changed source code location. Our proposed technique employs two heuristics on keyword-source code file association and recommend a ranked list of source code files for a given query. Here, the main idea is to capture information from a collection of bug reports and exploit them for relevant source code location. So our approach (1) does not rely on the lexical similarity measure between big reports and source code files for bug localization, and (2) addresses the vocabulary mismatch problem by using a large vocabulary constructed from fixed bug information. We also replicate two state of the art bug localization techniques in order to compare the performance of our proposed approach with these two.

2. AN EXAMPLE USE CASE SCENARIO

Note: Provide an example such as say for bug B1 source code files F1, F2, F3 are modified or changed. Now a new bug B has similar to B1 in terms of cosine similarity. Then prove that F1, F2, F3 or one or two of them are contained in the list of files which would be changed or modified for fixing that new bug B. Lets consider a bug having ID "" from Eclipse-UI-Platform software. From Git repository it is discoverd that so far n number of file have been modified in order to fix this bug. We also csn find the source code address of those changed files. Now,

In another example, show that the words or keywords of a bug B are aslo presented in the source code files F1, F2 or F3.

3. PROPOSED APPROACH

Our proposed approach consists of two parts - (i) query formulation and (ii) relevant file retrieval. In this paper, we implement two existing bug localization technoques: (i)TF-IDF based bug localization and (ii) LDA topic based bug localization techniques. In the following sections we will discuss both techniques in detail.

4. TF-IDF BASED BUG LOCALIZATION TECHNIQUE:

In this technique each source code file is ranked based on source code file scores and similar bugs information. Source code file contains words those can be also occurred in the bug reports. This is considered as a hint to locate buggy files. If a new bug is similar to a given previously located bug, then there is a possibility that the source code files located for the past bug can provide useful information in finding buggy files for that new bug. Based on these two assumptions, we compute scores for all source code files for a given project. However, we need focus on some concepts which are required to understand the system. They are described as follows:

4.1 Ranking based on TF-IDF calculation

The basic idea of a TF-IDF model is that the weight of a term in a document is increasing with its occurrence frequency in this specific document and decreasing with its occurrence frequency in other documents [**?**]. In this approach we have used a revised Vector Space Model (rVSM) proposed by **?**] in order to index and rank source code files. The main difference between classic VSM and revised VSM is that in case of revised version logarithm variant is used in computing term frequency. The equation is:

$$tf(t+d) = log(f_{td}) + 1 \qquad (1)$$

Here *tf* is the term frequency of each unique term t in a document d and f_{td} is the number of times term t appears in document d. So the new equation of revised VSM model is as follows:

$$rVSMScore(q,d) = g(\#term) \times cos(q,d)$$

$$\frac{1}{1+e^{-N(\#terms))}} \times \frac{1}{\sqrt{\sum_{t\epsilon q}((logf_{tq}+1) \times log(\frac{\#docs}{n_t}))^2}} \times$$

$$\frac{1}{\sqrt{\sum_{t\epsilon d}((logf_{td}+1) \times log(\frac{\#docs}{n_t}))^2}} \times$$

$$\sum_{t\epsilon q \bigcap d} (logf_{tq}+1) \times (logf_{td}+1) \times log(\frac{\#docs}{n_t})^2 \qquad (2)$$

This *rVSM* score is calculated for each query bug report q against every document d in the corpus. However, in the above equation *#terms* refers to the total number of terms in a corpus, n_t is the number of documents where term t occurs.

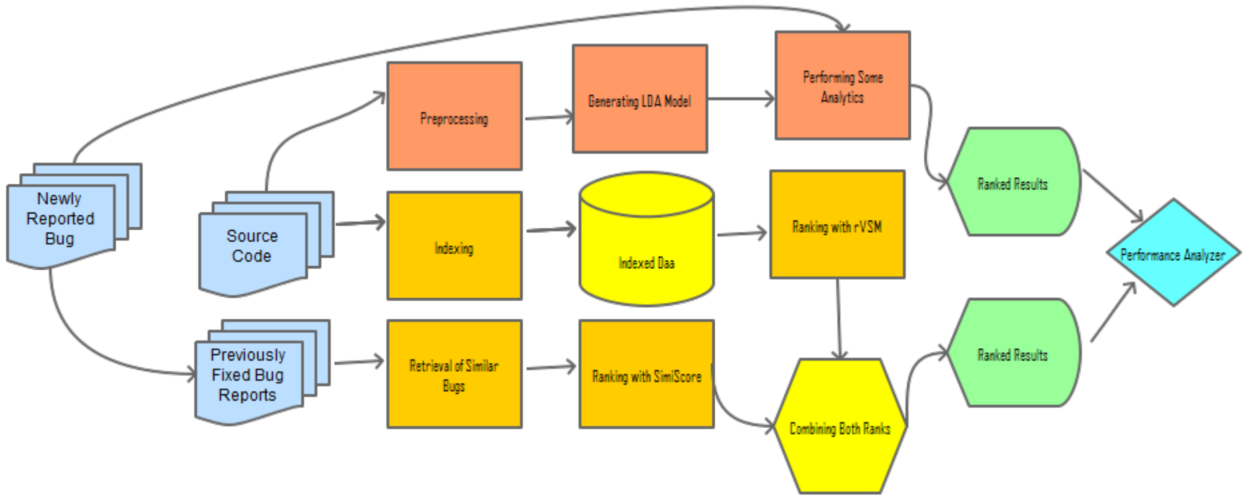

Figure 1: Proposed System Diagram

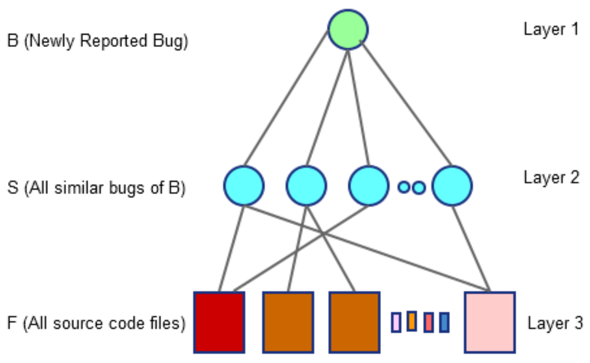

Figure 2: Bug and its Similar Bug Relationship with Source Code Files

4.2 Ranking based on similar bug information

The assumption of this ranking is similar bugs of a given bug tend to modify similar source code files. Here, we construct a 3-layer architecture as described in [**?**]. In the top layer (layer 1) there is a bug B which represents a newly reported bug. All previously fixed bug reports which have non-negative similarity with bug B are presented in second layer. In third layer all source code files are shown. In order to resolve each bug in second layer some files in the corpus were modified or changed, which are indicated by a link between layer 2 and layer 3. Foe all source code files in layer 3, similarity score is computed, which can be referred to as degree of similarity. The score can be defined as:

$$SimiScore = \sum_{AllS_ithatconnecttoF_j} (Similarity(B, S_i)/n_i) \tag{3}$$

Here, similarity between newly reported bug B and previously fixed bug S_i is calculated based on cosine similarity measure and n_i is the total number of link S_i has with source code files in layer 3.

4.3 Combining Both Ranks

We combine the both scores based on source code score and similar bugs score as in [**?**] as follows:

$$FinalScore = (1-\alpha)\times N(rVSMScore)+\alpha\times N(SimiScore) \tag{4}$$

Here, α is a weighting function and the value of α is in between 0 and 1. In our experiment we use 0.2 as the value of α

5. LDA TOPIC BASED BUG LOCALIZATION TECHNIQUE:

The main assumption behind this technique is the textual content of the bug reports and their associated buggy source code files tend to describe some common technical aspects. So, if we could identify the technical topics extracted from both bug reports and source codes, we could recommend the files shared technical topics with the newly reported bug reports. If a newly reported bug has similar technical topics with some previously fixed bug reports, the fixed files could be a good candidate files for newly reported bug.

LDA (Latent Dirichlet Allocation) is a probabilistic and fully generative topic model. It is used to extract latent (i.e., hidden) topics, which are presented in a collection of documents. It also model each document as a finite mixture over the set of topics [add link here]. In LDA, similarity between a document d and a query q is computed as the conditional probability of the query given that document [**?**].

$$Sim(q, d_i) = P(q|d_i) = \prod_{q_k \epsilon q} P(q_k|d_i) \tag{5}$$

Here q_k is the kth word in the query q Thus, a document

Table 1: Description of Data Sets

Project Name	#Source Codes	#Bug Reports
Eclipse Platform Ant	6085	More than 10K

(i.e., source code file) is relevant to a query if it has a high probability of generating the words in the query.

We perform the following steps in order to localize buggy files for newly reported bug using LDA based approach:

- Apply topic modeling on the source code files. The output contains a certain number of topics and some associated keywords for each topic. We also get some other distribution distribution files such as document-topic, word-topic etc.
- Now work with the documents topic distribution file. Make a list of source code documents or files for each topic. So, we wiill have a list that contain all topics and their associated source code documents.
- Here our query is the newly reported bug. This contains information in the bug reports such as title and short description etc. We all do inference for this query using a topic modeling tool. It will extract all topic associated with the query (i.e., newly reported bug).
- Now we need to work with topic keywords. We are going to perform a comparison between newly reported bug or the given query and source code files using topic information. That means we will compare topic-keywords associated with topics inferred for the query with topic-keywords of each topic extracted from source code documents.
- We will rank them based on topic-keyword similarity. So, now we know which are the top most topics, and we already have information regarding topic-document relationship, we will retrieve all source code files associated with all those top most topic as recommended buggy files.

6. EXPERIMENT AND DISCUSSION

6.1 Big Data Set

We work with Eclipse data set. We downloaed a git based Eclipse project from git repository [**?**]. We work with Eclipse Platform UI project. Currently it contains 6085 number of Java source codes. These source codes are contained in our source code repository. On the other hand, currently Eclipse Platform UI project contains more than 10K number of bugs where we only work with the bugs which are fixed. We create quires from each bugs considering their title and short summary.

6.2 Data Collection

We have two parts in our corpus. One is source code files downloaded as git based project and another part is bug reports collection. All bug reports are collected from *Bugzilla.* In order to obtain the links between previously fixed bugs and source code files, we analyze git project commit message. We ran through all commit messages and track Bug IDs associated with examined source code files.

Table 2: The Performance for Top 1, Top 5 and Top 10

Year	#Bugs	Top 1	Top 5	Top 10	MRR
2006	1159	16.37%	56%	62%	0.55
2011	377	44%	54%	58%	0.83
2012	404	35%	55%	62%	0.69
2013	507	26%	54%	60%	0.63

7. EXPERIMENTAL RESULTS AND DISCUSSION

7.1 Evaluation Metrices

Ton N-Rank: It represents the number of bug, for which their associated files are returned in a ranked list. Here, N may be 1, 5 or 10. We assume that if at least one associated file is presented in the resulted ranked list, then the given bug is located.

MRR(Mean Reciprocal Rank) The reciprocal rank of a query is the multiplicative inverse of the rank of the first correct answer. So mean reciprocal rank is the average of the reciprocal ranks of results of a set of queries Q.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{6}$$

MAP(Mean Average Precision)

7.2 Experimental Results

8. RELATED WORK

9. CONCLUSION AND FUTURE WORK

References

[] *Eclipse Platform UI Git Project.* https://git.eclipse.org/c/platform/eclipse.platform.ui.git/.

[] Beard, M. 2011 (Oct). Extending Bug Localization Using Information Retrieval and Code Clone Location Techniques. *Pages 425–428 of: Reverse Engineering (WCRE), 2011 18th Working Conference on.*

[] Chatterji, D., Carver, J.C., Massengil, B., Oslin, J., & Kraft, N.A. 2011 (Sept). Measuring the Efficacy of Code Clone Information in a Bug Localization Task: An Empirical Study. *Pages 20–29 of: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.*

[] Lukins, S.K., Kraft, N.A., & Etzkorn, L.H. 2008 (Oct). Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation. *Pages 155–164 of: Reverse Engineering, 2008. WCRE '08. 15th Working Conference on.*

[] Lukins, Stacy K., Kraft, Nicholas A., & Etzkorn, Letha H. 2010. Bug Localization Using Latent Dirichlet Allocation. *Inf. Softw. Technol.*, **52**(9), 972–990.

[] Moreno, L., Treadway, J.J., Marcus, A., & Shen, Wuwei. 2014 (Sept). On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization. *Pages 151–160 of: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.*

[] Nguyen, Anh Tuan, Nguyen, Tung Thanh, Al-Kofahi, J., Nguyen, Hung Viet, & Nguyen, T.N. 2011 (Nov). A Topic-Based Approach for Narrowing the Search Space of Buggy Files from a Bug Report. *Pages 263–272 of: Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on.*

[] Saha, R.K., Lease, M., Khurshid, S., & Perry, D.E. 2013 (Nov). Improving Bug Localization using Structured Information Retrieval. *Pages 345–355 of: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on.*

[] Saha, R.K., Lawall, J., Khurshid, S., & Perry, D.E. 2014 (Sept). On the Effectiveness of Information Retrieval Based Bug Localization for C Programs. *Pages 161–170 of: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.*

[] Sisman, B., & Kak, A.C. 2013 (May). Assisting Code Search with zAutomatic Query Reformulation for Bug Localization. *Pages 309–318 of: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on.*

[] Wang, Shaowei, & Lo, David. 2014. Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization. *Pages 53–63 of: Proceedings of the 22Nd International Conference on Program Comprehension.* ICPC 2014.

[] Wang, Shaowei, Lo, D., & Lawall, J. 2014 (Sept). Compositional Vector Space Models for Improved Bug Localization. *Pages 171–180 of: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.*

[] Wong, Chu-Pan, Xiong, Yingfei, Zhang, HongYu, Hao, Dan, Zhang, Lu, & Mei, Hong. 2014 (Sept). Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis. *Pages 181–190 of: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.*

[] Zhou, Jian, Zhang, Hongyu, & Lo, D. 2012 (June). Where Should the Bugs be Fixed? More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports. *Pages 14–24 of: Software Engineering (ICSE), 2012 34th International Conference on.*