Cross-Language Bug Localization

Xin Xia[1]*, David Lo[2], Xingen Wang[1], Chenyi Zhang[1]*, and Xinyu Wang[1]
[1]College of Computer Science and Technology, Zhejiang University, China
[2]School of Information Systems, Singapore Management University, Singapore
xxkidd@zju.edu.cn, davidlo@smu.edu.sg, {newroot, chenyizhang, wangxinyu}@zju.edu.cn

ABSTRACT

Bug localization refers to the process of identifying source code files that contain defects from textual descriptions in bug reports. Existing bug localization techniques work on the assumption that bug reports, and identifiers and comments in source code files, are written in the same language (i.e., English). However, software users from non-English speaking countries (e.g., China) often use their native languages (e.g., Chinese) to write bug reports. For this setting, existing studies on bug localization would not work as the terms that appear in the bug reports do not appear in the source code. We refer to this problem as *cross-language bug localization*. In this paper, we propose a cross-language bug localization algorithm named *CrosLocator*, which is based on language translation.

Since different online translators (e.g., Google and Microsoft translators) have different translation accuracies for various texts, *CrosLocator* uses multiple translators to convert a non-English textual description of a bug report into English – each bug report would then have multiple translated versions. For each translated version, *CrosLocator* applies a bug localization technique to rank source code files. Finally, *CrosLocator* combines the multiple ranked lists of source code files. Our preliminary experiment on Ruby-China shows that *CrosLocator* could achieve mean reciprocal rank (mrr) and mean average precision (map) scores of up to 0.146 and 0.116, which outperforms a baseline approach by an average of 10% and 12% respectively.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Algorithms and Experimentation

*The work was done while the two authors were visiting Singapore Management University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

ICPC'14, June 2–3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2879-1/14/06...$15.00
http://dx.doi.org/10.1145/2597008.2597788

Keywords

Bug Localization, Cross-language, Translator, Rank

1. INTRODUCTION

Bug fixing is one of the most important activities in the whole lifecycle of software development and maintenance. Once a bug report is submitted and confirmed, developers need to spend much time to locate relevant source code files. To address this problem, in recent years, many information retrieval based (IR-based) bug localization techniques have been proposed [8, 6, 9, 11, 10]. IR-based bug localization techniques consider textual description in a bug report as a query, and leverage IR techniques to rank source code files by their relevance to the query (i.e., bug report). Existing techniques work well if the bug report, and their identifiers and comments in the source code files, are written in the same language [11, 9].

However, in practice, for users from a non-English speaking country (e.g., China), they often use their own native languages (e.g., Chinese) to write bug reports. On the other hand, due to various reasons (e.g., international collaboration, etc.), identifiers and comments in source code files are often written in English. In this situation, IR-based bug localization would not work since bug reports and source code files contain different words in different languages. We refer to the bug localization problem where identifiers and comments in source code files and bug reports are written in different languages as *cross-language bug localization*.

One way to solve the *cross-language bug localization* problem is to first translate a non-English bug report (i.e., query) into English by using an online translator (e.g., Google[1] and Microsoft[2] translators), and then use a standard IR-based bug localization technique to rank source code files by using the translated bug report. However, we notice that different online translators have different translation accuracies for different texts, which would translate to different effectiveness for different bug reports. Also, given a non-English text, there could be many correct translations, c.f. [7]. For example, "软件修复" in Chinese is translated as "software fix" by Google translator, and "software repair" by Microsoft translator, and both of them are correct translations. Thus, it is good to make use and combine the results of multiple online translators.

In this paper, we propose an approach named *CrosLocator*, which leverages multiple online translators, to address the

[1]http://translate.google.com
[2]http://www.bing.com/translator

Figure 1: Bug Report of Ruby-China with BugID = 211.

cross-language bug localization problem. *CrosLocator* first translates bug reports by using multiple online translators. At the end of this step, a bug report would have multiple translated versions. For each translated version, *CrosLocator* applies an IR-based bug localization technique to rank source code files. After this step, we have several ranked lists of files, one for each translated version. Finally, *CrosLocator* uses a learning to rank technique [5] to combine the multiple ranked lists of source code files into one. In this paper, we consider several learning to rank techniques, such as Borda count [1], *CombSUM*, *CombMNZ*, and *CombANZ* [2]. We evaluate *CrosLocator* on 50 bug reports from Ruby-China[3]. The experiment results show that *CrosLocator* could achieve mrr and map scores of 0.146 and 0.116 which are in similar score ranges reported in several recent conventional (single-language) bug localization studies, c.f., [9, 10]. *CrosLocator* also outperforms a baseline approach, which only uses one translator and does not perform learning to rank, by an average of 10% and 12% for mrr and map scores, respectively.

The main contributions of the paper are:

1. To our best knowledge, we are the first to propose the *cross-language bug localization* problem.
2. To solve the problem, we propose *CrosLocator*, which uses multiple online translators and combines multiple translated bug report versions using a learning to rank technique to achieve a better performance. The preliminary experiment results show that *CrosLocator* could achieve mean reciprocal rank (mrr) and mean average precision (map) scores of up to 0.146 and 0.116, which outperform the performance of a baseline approach by a substantial margin.

2. MOTIVATION

Figure 1 presents a bug report of Ruby-China written in Chinese with BugID = 211.[4] The bug report describes a problem in the search functionality. To fix the bug, developers need to modify the source file "search_controller.rb".

We translate the textual description of the bug report by using Google and Microsoft translators. The translated versions of the bug report using Google and Microsoft translators are:

> **Google**: *Search in the search box with "#" character, it will jump to the google homepage, I will jump here specifically to https://www.google.com.hk/#*
>
> **Microsoft**: *Search in the search box contains "#" character, will go to Google Home, I will go to https://www.Google.com.hk/#*

When using each of these two translated versions of the bug report to locate the relevant source code file (i.e., search_controller.rb), we find its rank and score for the translated version produced by Google to be 7 and 0.174 respectively, while its rank and score for the version produced by Microsoft to be 9 and 0.164 respectively.[5] However, if we combine the two versions, the rank would become 5.[6]

From the example, we have the following observations:

1. Different online translators have different translation accuracies, which cause different bug localization effectiveness.
2. Combining multiple translators could improve the effectiveness of bug localization.

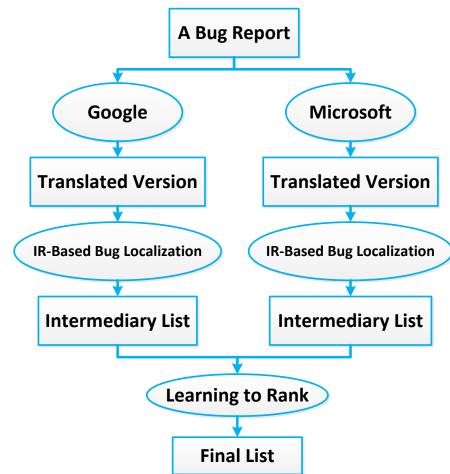

Figure 2: *CrosLocator* Architecture

3. PROPOSED APPROACH

3.1 Overall Architecture

Figure 2 presents the architecture of *CrosLocator*. Notice that *CrosLocator* does not requires any training data. It has 3 components: online translator component, IR-based bug localization component, and learning to rank component. The main task of the online translator component is to translate the textual description of a bug report into English. The main task of the IR-based bug localization component is to compute and output a ranked list of source code files given a translated bug report (intermediary list). The main task of the learning component is to combine multiple intermediary lists produced by the IR-based bug localization component into one list (final list).

When a new bug report is received, *CrosLocator* first uses Google and Microsoft online translators to translate the textual description of the bug report into English. At the end of this process, we have two translated versions of the bug report. Next, these two translated versions would be input into the IR-based bug localization component to get two ranked list of source code files. Finally, we merge the two ranked lists into one using a learning to rank technique [5].

3.2 IR-based Bug Localization Component

In this paper, we use BugLocator proposed by Zhou et al. [11] as the IR-based bug localization component. Given a textual description of a bug report, BugLocator first applies the revised Vector Space Model (rVSM) to return a ranked list of relevant source code files. Next, it also finds similar closed bug reports (i.e., bug reports that have been fixed

[3]https://github.com/ruby-china/ruby-china
[4]https://github.com/ruby-china/ruby-china/issues/211

[5]We use BugLocator [11] to output a ranked list of files from each translated version.
[6]We use *CrosLocator*, with *CombSum* as the learning to rank technique, to combine the two versions.

before) to the input bug report, and computes another ranked list of relevant source code files based on the fixes of these similar bug reports. Finally, these two lists are combined to form one list.

CrosLocator can use various IR-based bug localization techniques. However, in this paper we only experiment with one since BugLocator is recently proposed, has been demonstrated to outperform other techniques, and its implementation is publicly available.

3.3 Learning to Rank Component

Notice that the outputs of the IR-based bug localization component are multiple intermediary lists. In the learning to rank component, we want to merge them into one list using a learning to rank technique. To do this, we first take the top-100 files from each ranked list, and combine these top-100 lists into one list using either one of these techniques: Borda count [1], *CombSUM*, *CombMNZ*, or *CombANZ* [2]. These are well-known learning to rank techniques that do not require any training data. In the following paragraphs, we briefly describe each of these unsupervised learning to rank techniques.

Borda Count: Borda count technique integrates multiple intermediary ranked lists of files into a final ranked list of files in the following way:

1. For each intermediary list of size n, Borda count technique assigns a number from 1 to n to each file in the list. The first ranked file in the list would be assigned n, the second one would be assigned $n-1$, and so on. Let us denote the number assigned to file f appearing in list i as $Rank_i(f)$.
2. For each file f, we compute its Borda count using the following formula: $BordaCount(f) = \sum_i Rank_i(f)$.
3. We create a final ranked list of files based on their Borda counts.

For example, suppose there are two intermediary ranked lists. The first intermediary ranked list is $\langle A, B, C \rangle$, and the second intermediary ranked list is $\langle A, C, B \rangle$. Then, the final Borda count for A, B and C are 6, 3, and 3. This is so since: A is assigned value 3 in both lists, B is assigned value 2 and 1 in the first and second lists respectively, and C is assigned value 1 and 2 in the first and second list respectively.

CombSUM, CombMNZ, and CombANZ: Borda count only uses the rank of the files in the intermediary ranked lists. However, often an IR-based bug localization technique also outputs a ranking score that denotes how likely a file is relevant to a bug report. Different from Borda count, *CombSUM*, *CombMNZ*, and *CombANZ* consider not only the rank of the files in the intermediary ranked lists but also the ranking scores of the files in the lists.

Let f denote a source code file, $score_i(f)$ denotes the ranking score for the source file f in the ith intermediary ranked list, and n_f denotes the number of times the source code file f appears in the multiple intermediary ranked lists. In *CombSUM*, we compute the following score for each file f which is used to rank source code files to produce the final ranked list: $CombSUM(f) = \sum_i Score_i(f)$. In *CombMNZ*, we compute the following score: $CombMNZ(f) = n_f \times \sum_i Score_i(f)$. In *CombANZ*, we compute the following score: $CombANZ(f) = \sum_i Score_i(f)/n_f$.

For example, suppose there are two intermediary ranked lists. The first intermediary ranked list (along with ranking scores) is $\langle A(0.7), B(0.5), D(0.2) \rangle$, and the second intermediary ranked list (along with ranking scores) is $\langle A(0.9), C(0.7), B(0.4) \rangle$. Then, $n_A = 2, n_B = 2, n_C = 1, n_D = 1$, since A and B appear in the two lists, while C and D only appear in one list. Thus, the *CombSUM* ranking scores for A, B, C, and D are 1.6, 0.9, 0.7, and 0.2 respectively. Also, the *CombMNZ* ranking scores for A, B, C, and D are 3.2, 1.8, 0.7, and 0.2 respectively. Furthermore, the *CombANZ* ranking scores for A, B, C, and D are 0.8, 0.45, 0.7, and 0.2 respectively.

4. PRELIMINARY EXPERIMENT

Experiment Setup. We collect bug reports and their fixes from an open source project named Ruby-China. To collect these, we analyze Ruby-China's bug tracking system (i.e., its bug tracking system in GitHub) and version control system (i.e., git). We extract commit logs from its git. For each commit log, we perform a regular expression check to identify whether a bug report identifier exists in the log. If there is an identifier, we recover the details of the bug (i.e., a bug report) with that identifier from Github. If an identifier is contained in multiple commit logs, we would also group the corresponding commits together. From these pieces of information, for each bug report whose identifier exists in the commit logs, we would recover the code before the fix, and the files that are changed or deleted to fix the bug. We analyze all bug reports of Ruby-China from April 2012 to October 2013, and we find that there are 50 of them that are linked to their corresponding bug fixing commits. We analyze all these 50 reports. Table 1 presents the statistics of the collected dataset. The columns corresponds to the number of bug reports (# Bugs), the number of source code files (# Files), and the time period for the collected bug reports (Time Period).

Table 1: Statistics of Collected Dataset.

Project	# Bugs	# Files	Time Period
Ruby-China	50	529	2012.04 - 2013.10

We compare our algorithm with a baseline approach which simply takes a translated version of a bug report and uses BugLocator to return a ranked list of files. We thus have two variants of this baseline – one using Google translator and another using Microsoft translator. We evaluate the effectiveness of *CrosLocator* in terms of the following measures: recall-rate@1[7] (top1), recall-rate@5 (top5), recall-rate@10 (top10), mean reciprocal rank (mrr), and mean average precision (map), which are widely used in past bug localization studies [11, 6, 9, 10]. Among these measures, mrr and map are the well-known, standard information retrieval measures. Our experimental environment is a Windows 7 32-bit, Intel(R) Core(TM) i5 CPU 3.20GHz server with 4GB RAM.

Experiment Results. We denote *CrosLocator* using Borda count, *CombSUM*, *CombMNZ*, and *CombANZ* as C^{Borda}, C^{Sum}, C^{MNZ}, and C^{ANZ} respectively, and the baseline approach with Google and Microsoft translators as *Goo.* and *Micro.* respectively. Table 2 presents the experiment results of *CrosLocator* compared with the baseline approach. We notice that *CrosLocator* using *CombSUM* and *CombMNZ* achieve the best performance, the recall-rate@1, recall-rate@5, recall-rate@10, mrr, and map scores are 0.08,

[7]It is also known as top n rank in [11].

0.18, 0.30, 0.146, and 0.116, which on average outperform the baseline approach with Google and Microsoft translators by 33.33%, 14.29%, 11.26%, 9.72%, and 11.87% respectively.

Notice that the mrr and map scores of C^{SUM} and C^{MNZ} are 0.146 and 0.116. These scores are in similar score ranges reported in several past bug localization studies. For example, Rao et al. reported that the map scores of the Unigram model and VSM are 0.1454 and 0.0796 for bug reports (written in English) in the iBugs dataset [9]. Zhou et al. reported that the average mrr and map scores of VSM are 0.13 and 0.09 for bug reports from 4 projects, i.e., ZXing, SWT, AspectJ. and Eclipse [11]. Thus, our results are promising and on par with results of many past conventional bug localization techniques.

Still, cross-language bug localization is a more difficult problem than conventional (single-language) bug localization. *CrosLocator* uses BugLocator, and the performance of BugLocator for bug reports (written in English) from 4 projects, as reported in [11], is higher than the performance of *CrosLocator* for the 50 bug reports (written in Chinese) from Ruby-China.

Table 2: CrosLocator vs. Baseline.

Metric	C^{Borda}	C^{SUM}	C^{MNZ}	C^{ANZ}	*Goo.*	*Micro.*
top1	0.06	**0.08**	**0.08**	0.08	0.06	0.06
top5	0.18	**0.18**	**0.18**	0.18	0.18	0.14
top10	0.28	**0.30**	**0.30**	0.28	0.28	0.26
mrr	0.142	**0.146**	**0.146**	0.145	0.138	0.128
map	0.116	**0.116**	**0.116**	0.115	0.101	0.106

5. RELATED WORK

There have been a number of bug localization techniques proposed in the literature [8, 6, 9, 11]. Poshyvanyk et al. propose PROMESIR, which leverages Latent Semantic Indexing (LSI) and a probabilistic ranking technique to rank source code files [8]. Lukins et al. propose the usage of Latent Dirichlet Allocation (LDA) to locate the relevant source code files [6]. Zhou et al. propose BugLocator which consider two rankings, i.e., ranking based on similar source code files, and ranking based on similar bugs [11].

All the above studies only consider single-language bug localization setting where bug reports are written in English. We consider a new problem namely cross-language bug localization. Our *CrosLocator* can leverage each of the above studies (by using each of them as the IR-Based Bug Localization component) for cross-language bug localization.

Hayes et al. propose a translation-based method for traceability recovery [4]. They use Google translator to translate Italian terms into English and then recover the links. Our paper is different from theirs: 1. We combine 2 different online translators to achieve a better performance, while Hayes et al. only consider one translator, 2. We consider the problem of bug localization instead of traceability recovery.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a new research problem, namely cross-language bug localization, which focuses on ranking source code files given that bug reports, and identifiers and comments written in source code files, are written in *different* natural languages. To solve this problem, we propose *CrosLocator*, which is a bug localization algorithm built upon several online translators, a conventional IR-based bug localization technique, and a learning to rank technique. *CrosLocator* first translates the textual description of a bug report into multiple English versions by using multiple online translators. Next, each of these translated versions is input into an IR-based bug localization technique which outputs an intermediary ranked list. Finally, a learning to rank technique is used to merge the intermediary ranked lists into a final ranked list. The experiment results show that *CrosLocator* achieves mean reciprocal rank (mrr) and mean average precision (map) scores of up to 0.146 and 0.116 respectively on a dataset extracted from Ruby-China. *CrosLocator* on average outperforms a baseline technique, which simply applies BugLocator on a translated version of a bug report, by 10% and 12% for mrr and map respectively.

In the future, we plan use query reformulations methods (e.g., [3]) to reformulate the terms in the bug reports to improve the performance of bug localization.

ACKNOWLEDGMENTS

This research is sponsored in part by NSFC Program (No.6110 3032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (2014BAH24F02).

7. REFERENCES

[1] J. A. Aslam and M. Montague. Models for metasearch. In *SIGIR*, 2001.

[2] E. A. Fox and J. A. Shaw. Combination of multiple searches. *NIST SPECIAL PUBLICATION SP*, 1994.

[3] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *ICSE*, 2013.

[4] J. H. Hayes, H. Sultanov, W.-K. Kong, and W. Li. Software verification and validation research laboratory (svvrl) of the university of kentucky: traceability challenge 2011: language translation. In *TSFSE*, 2011.

[5] H. Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 2011.

[6] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Information and Software Technology*, 2010.

[7] J.-Y. Nie. Cross-language information retrieval. *Synthesis Lectures on Human Language Technologies*, 2010.

[8] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *TSE*, 2007.

[9] S. Rao and A. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *MSR*, 2011.

[10] B. Sisman and A. C. Kak. Incorporating version histories in information retrieval based bug localization. In *MSR*, 2012.

[11] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *ICSE*, 2013.