Are Bug Reports Enough for Text Retrieval-based Bug Localization?

Chris Mills*, Jevgenija Pantiuchina†, Esteban Parra*, Gabriele Bavota†, Sonia Haiduc*
*Department of Computer Science - Florida State University - Tallahassee, FL, USA
{cmills, parrarod, shaiduc}@cs.fsu.edu
†Faculty of Informatics - Università della Svizzera italiana - Lugano, Switzerland
{jevgenija.pantiuchina, gabriele.bavota}@usi.ch

Abstract*—Text Retrieval (TR) has been widely used to support many software engineering tasks, including bug localization (*i.e.,* the activity of localizing buggy code starting from a bug report). Many studies show TR's effectiveness in lowering the manual effort required to perform this maintenance task; however, the actual usefulness of TR-based bug localization has been questioned in recent studies. These studies discuss (i) potential biases in the experimental design usually adopted to evaluate TR-based bug localization techniques and (ii) their poor performance in the scenario when they are needed most: when the bug report, which serves as the *de facto* query in most studies, does not contain *localization hints* (*e.g.,* code snippets, method names, *etc.*) Fundamentally, these studies raise the question: *do bug reports provide sufficient information to perform TR-based localization?

In this work, we approach that question from two perspectives. First, we investigate potential biases in the evaluation of TR-based approaches which artificially boost the performance of these techniques, making them appear more successful than they are. Second, we analyze bug report text with and without *localization hints* using a genetic algorithm to derive a *near-optimal query* that provides insight into the potential of that bug report for use in TR-based localization. Through this analysis we show that in most cases the bug report vocabulary (*i.e.,* the terms contained in the bug title and description) is all we need to formulate effective queries, making TR-based bug localization successful without supplementary query expansion. Most notably, this also holds when localization hints are completely absent from the bug report. In fact, our results suggest that the next major step in improving TR-based bug localization is the ability to formulate these *near-optimal queries*.

I. Introduction

Text Retrieval (TR) techniques have become a popular means of automating numerous software engineering tasks [1], including feature and bug localization in source code [2], traceability link recovery [3], impact analysis [4], bug triaging [5], and software refactoring [6] among others. One of the most popular applications among these has been bug localization. When using TR techniques for this task, the user formulates a natural language query with the goal of describing the observed bug. That query is run through a TR engine, which returns a ranked list of code components (*e.g.,* classes or methods, depending on the desired granularity), containing the most relevant results (*i.e.,* the components likely related to the bug) in the top most positions.

While TR techniques have been successfully applied for this and other software engineering tasks for more than a decade, the *actual usefulness of TR-based techniques* to support some of these tasks has been questioned by recent studies.

In the context of bug localization, the effectiveness of these approaches has generally been demonstrated through experiments using issue reports marked as bugs for TR queries to search source code for relevant, buggy methods, classes, or files. In these experiments, the code components modified in the corresponding bug-fixing commits are used as the ground truth for evaluation. Recent work has called these prior studies into question by identifying key concerns with this experimental design [7]. In particular, three main biases have been identified. First, using miss-classified bugs, *e.g.,* issues that have been classified as bugs, when in fact they represent new features. Second, using bloated ground truths that include code changes irrelevant to a bug fix. Third, using bugs reports that contain localizing information, that is, reports that explicitly point to a code location such as a code snippet, file or class name, or identifier, which we refer to as *localization hints*. Kochhar *et al.* [7] showed that while the first two do not significantly impact the evaluation, bug reports containing localization hints have a major impact on TR results.

Wang *et al.* [8] further analyzed the bias introduced by three types of localization hints: program entity names, test cases, and stack traces. Of the three, they found that program entity names significantly boost retrieval results. Basically, these studies highlight the fact that *TR-based approaches perform very well when the bug report description already includes the bug localization*, by listing all (or some) of the buggy code components. *However, in such scenarios TR-based bug localization techniques are not needed in the first place, as the bug has already been localized* [7]. On the other hand, *when hints to localize the bug are not found in the bug report description* (*e.g.,* when the bug report is written by a user of the system that lacks knowledge of the system's technical implementation), the studies showed that *the performance of TR-based bug localization techniques significantly drops, thus questioning their usefulness in this scenario as well.*

One commonality between the aforementioned studies, as well as many before them, is that they use the whole text of the bug report (*i.e.,* its title and description) as the TR query, without considering alternatives. Since *TR effectiveness strongly depends on the quality of the formulated query* [9], the choice of query is a crucial factor in ensuring the success

of TR. Bug reports, however, can often be lengthy and contain "noise" (*i.e.,* words that do not efficiently describe the bug). This is supported by the findings of a recent study [10] which showed that removing even a few of the "noisiest" terms in a bug report can lead to improved TR results. We therefore conjecture that TR approaches have a lot more *potential* for bug localization than they have been given credit for, so long as they can be provided with an optimized query for a bug.

In this paper we present the results of a large empirical study providing new evidence on the true potential of TR bug localization approaches and the significant impact that optimizing queries can have on their effectiveness. Further, we show that given a bug report, we can often obtain such a query using only words selected from its vocabulary, even when localization hints are not present. In short, bug reports typically contain sufficient information to perform bug localization even without specific *localization hints* or the need for additional sources of context such as dictionaries or historical information.

To perform this study, we first collected 620 bug reports from 13 open source systems used in previous bug localization experiments [8], [10]. To account for the potential biases named above, we manually analyzed each bug report (BR) and its fixing commits. From the BR, we extracted two queries: one containing all terms in BR's title and description (Q_{all}), and one (Q_{noHint}) obtained by removing all code-related terms possibly representing localization hints (*e.g.,* class/method/file names, stack traces, test cases, *etc.*) from Q_{all}. This allowed us to investigate the bias that the presence of localization information can have on the TR results, as discussed by Wang *et al.* [8] and Kochhar *et al.* [7]. Then, in the fixing commit of BR, we also manually analyzed each modified file to verify that two conditions were met. First, the change was actually made to fix the bug rather than being the result of a tangled commit [11]. Second, the change actually modified the functionality of the system and was not purely aesthetic (*e.g.,* changes to comments or code formatting). This manual verification of the changes ensured a reliable ground truth for our experiments, free of the miss-classified bugs and bloated ground truth biases identified by Kochhar *et al.* [7]. With this extracted data, we show that it is almost always possible to formulate an effective query returning relevant results in the top positions of the ranked result list by only using the terms present in Q_{all} or in Q_{noHint}. Therefore, we are able to derive an effective query in either case: with or without localization hints in the BR description and title.

Our study differs from previous work in that we are interested in establishing the potential of TR for bug localization by finding the most effective query that can be extracted from the BR and evaluating the performance of that query, rather than considering the default query composed of the entire bug title and/or description. Finding this query by brute force, however, would require testing all possible queries of any length that can be obtained from the BR's vocabulary. For example, assuming a BR description composed of n distinct terms, the number of possible queries to test is the sum of factorials of descending natural numbers beginning with n: $n! + (n-1)! + \ldots + 1!$. Given that our Q_{noHint} and Q_{all} queries have on average 40 and 133 terms, respectively, this would mean running between 8.2E+47 and 1.5E+226 queries for each bug report, which is clearly computationally infeasible.

For this reason, we devised a Genetic Algorithm (GA) able to converge towards an optimal query that can be obtained from a bug report vocabulary, knowing *a priori* the ground truth for the query. While our GA **can not be used in a real bug localization scenario where the ground truth is unknown**, it represents a needed tool to run our large-scale study and provide evidence on the potential of TR bug localization with optimal queries. The **contributions** of our study are:

1) *Strong supporting evidence for the potential of TR-based bug localization techniques despite their limitations and biases, using queries extracted from a bug report*. We clearly show that the bug report vocabulary is all we need to formulate optimal queries making TR-based bug localization successful in most cases. We also emphasize the absolute need and importance of applying techniques that automatically formulate good, selective queries starting from a bug report vocabulary, rather than settling for the default query. We also show that such queries are able to mitigate the negative impact that the absence of localization information in bug reports has on TR effectiveness.
2) *A manually curated dataset, reporting a reliable ground truth at the file/class level, as well as the optimal queries that can be obtained using our GA starting from the bug report vocabulary [12]*. This dataset represents an important contribution for researchers interested in evaluating their bug localization techniques. Moreover, the dataset can also be used by researchers working on query reformulation techniques [42], [13], since we provide an "upper-bound" in terms of the optimal queries that can be derived starting from the bug report vocabulary. Finally, having a large set of high-performing queries could help to study their characteristics, in turn fostering the development of approaches supporting automatic query formulation.

II. Background and Related Work

A. TR Approaches to Bug Localization

TR allows users to search in a corpus (*i.e.,* a set of text documents) for documents relevant to a given text-based query. In bug localization, developers must find buggy code given a bug report describing the observed unexpected behavior. Both the system's code and the bug report can be stored as textual documents. Therefore, TR perfectly fits in the bug localization process: the source code can be seen as the document corpus (*e.g.,* by considering each class/method/file as a different textual document), while the text in the bug report can be used to formulate a query aimed at retrieving code documents that are semantically similar to the bug report. The underlying assumption of TR-based bug localization is that code components which contain terms similar to the formulated query are likely related to the observed bug, and thus are recommended for inspection.

Many studies have focused on the application of TR to the bug localization domain [2], [14], [15], providing tool support [16], [17], [18], [19], [20], [21], [22] and improvements over standard TR techniques [23], [24], [25], [26]. Due to the breadth of existing research and space constraints, we direct the interested reader to a survey conducted by Dit *et al.* [27], which contains a detailed account of many approaches to bug localization, including TR-based techniques.

Our study focuses on the actual effectiveness of these techniques and on how they are evaluated in the context of analytical studies based on information extracted from bug repositories. The effectiveness of TR-based bug localization techniques has generally been demonstrated through experiments on issue reports marked as bugs, using the whole text in the bug reports as the textual query, and referencing as ground truth the code components modified in their corresponding bug-fixing commits. However, the data contained in these repositories is not always completely accurate [28] and using bug reports as *de facto* queries for empirical evaluation of TR techniques is the source of potential experimental bias.

B. Potential Biases in Empirical Evaluations of TR-based Bug Localization

Kochhar *et al.* [7] empirically investigated three potential biases: bug misclassification, bloated golden sets, and localized bug reports. In their study, they find that localized bug reports (*i.e.,* bug reports which already list in their text the code location where the bug is present) lead to significantly better results for TR-based bug localization (no significant impact was observed for the other two potential biases). This finding questions the usefulness of TR-based bug localization, as this process is actually needed only when localization hints are **not** present in the bug report (*i.e.,* exactly the scenario in which the performance of TR-based bug localization significantly drops) [7]. We partially replicate this study by investigating TR performance on localized *vs* unlocalized bug reports in our dataset. Our study is conducted on a larger set of software projects and above all it shows that the chosen query matters more than the presence of localization information. More specifically, we show that it is almost always possible to formulate a good query which retrieves buggy code components in the top positions using only terms selected from the bug report vocabulary, even when the bug report description does not include any localization hint.

Kawrykow and Robilliard [29] reported the presence of changes unneeded to fix a bug in bug-fixing commits (*e.g.,* renaming a variable) as a possible source of bias in the evaluation of bug localization techniques. Indeed, these changes artificially increase the number of code components that appear relevant to a bug fix, but are actually unrelated. This is the same form of bias defined by Herzig and Zeller as “tangled commits" [11], meaning commits containing changes addressing a specific bug mixed with unrelated changes (*e.g.,* refactoring). If TR approaches are evaluated on these commits, their performance might be artificially boosted by the fact that there are many more “relevant" files to be found in the search space, even though finding some of these files would not practically assist with localizing the bug. In order to remove such a bias from our study, we manually analyzed all the changed files in each bug fix commit and removed any file whose changes were not directly related to the bug being fixed.

Wang *et al.* [8] provide a detailed analysis of the impact that localized bugs have on evaluating TR-based bug localization. In their study, they consider different types of *identifiable information* that localize a bug explicitly: program entity names, stack traces, and test cases. They found that the presence of program entity names significantly improves the performance of TR-based bug localization. Our study also addresses the impact of localization hints on TR performance, but does so in a different way. First, Wang *et al.* classify bug reports as containing/not containing localization hints on the basis of the information that they report, and then compare the performance of TR-based bug localization on these disjoint sets of bugs. Instead, we manually derive from each bug two versions: one containing the complete title and description, and one from which we manually remove any reference to code components (*i.e.,* any possible localization hint). This allows us to compare the performance of TR-based bug localization with and without localization hints between the same set of bugs, thus removing conflating variables such as bug type or severity that are introduced by comparing disjoint bug sets. Through this analysis, we show that the formulated query is more important than the presence or absence of localization hints in ensuring the success of TR in bug localization. Specifically, we show that it is possible to formulate a near-optimal query from the bug report vocabulary, which leads to successful TR-based bug localization with or without localization hints in that vocabulary.

Finally, according to Bettenburg *et al.* [30], the inclusion of identifiable information is considered to be important when writing a good bug report, yet developers answering a survey indicated that few bugs contain error messages (53%), code examples (36%), or test cases (56%). Moreover, developers who took the survey indicated that the biggest detracting factors in bug reports were unstructured, lengthy text and non-technical language. This further suggests that localization techniques should focus on optimizing performance in these difficult situations. Through investigating queries in the Q_{noHint} set, this study also investigates whether sufficient information to localize a bug exists even for bugs composed of strictly unstructured, natural language. This concept is related to other work which has sought to quantify the quality of a query used for a software engineering task [31], [9] and derive a more effective query when required.

C. Query Reformulation for TR-based Bug Localization Approaches

In the event that a query leads to poor results, there have been numerous techniques devised to reformulate the query [32], [13], [33], [34] using either expansion [35] (*i.e.,* adding additional terms to broaden the query) or reduction [13](*i.e.,* removing words unlikely to contribute to the inherent meaning of the query, in order to reduce noise). Query reformulation

addresses situations in which the available query, whether in the form of a bug report or otherwise, offers an insufficient representation of the information need required to localize a bug. A recent study [10] empirically quantified the improvement in verbose queries achieved by removing words that negatively impact effectiveness. Chaparro *et al.*'s study [10] removes up to six "noisy" terms from the query. These terms are identified though a brute-force approach. In a subsequent study [36], Chaparro *et al.* manually reduce noisy, ineffective queries to reformulated queries that contain only terms that describe *observed behaviors*, and find that the reformulated queries have much improved performance. Since this study is related to the work presented in this paper, we include a *post-hoc* analysis using queries common to both studies in Section IV.

Different from previous studies, we leverage a GA which allows us to perform any reformulation (using the bug report) that converges to an improved, near-optimal query using effectiveness as a cost function. From this perspective, a major contribution of our study is empirical evidence that more complex techniques leveraging external sources of context [37], or linguistic information [38] are not needed to optimize queries in most cases. Further, our results show that while observable behaviors help with refining a query based on bug report text, there exist even more effective queries that can be derived from the bug report. That is, there exists some combination of terms from the bug report text that serve as an effective query for localization that may or may not have any relationship to the observable behavior of the bug.

III. Study Design

Our main *goal* is to assess the potential effectiveness of TR-based bug localization assuming the ability to formulate an optimal query starting from the bug report vocabulary. In particular, our study addresses the following research questions:

RQ$_0$: *What is the effectiveness of TR-based bug localization techniques when using the whole bug report text as a query?* This is a preliminary research question where we establish a baseline by studying the performance of out-of-the-box TR-based bug localization techniques when using the whole text contained in the bug report (*i.e.,* a concatenation of its title and description) as a query, as usually done in empirical evaluations of these techniques [7]. RQ$_0$ serves as a term of comparison for our main research question (RQ$_1$), in which we study what the *potential* effectiveness of these techniques could be, given the ability to formulate an "*optimal query*" starting from the bug report vocabulary. As part of this research question we also present a differentiated, partial replication of the work by Kochhar *et al.* [7], in which we analyze the impact that localization hints in the bug report text have on the performance of TR-based bug localization.

RQ$_1$: *What is the effectiveness of TR-based bug localization techniques when using an* ***optimal query*** *selected from the vocabulary of the bug report?* We study what the potential effectiveness of out-of-the-box TR-based bug localization techniques truly is. In particular, we devised an experimental design allowing us to formulate an "*optimal query*" from a bug report, (*i.e.,* the most effective query that can be derived from solely the vocabulary of the bug report). We also compare the performance of the formulated *optimal query* when localization hints are present/absent in the bug report vocabulary, to study the impact of this factor on the potential usefulness of TR-based bug localization techniques.

TABLE I
Study Dataset

Project	# Original Bugs	# Cleaned Bugs	Avg. Changed Files	Avg. Q_{all} Size	Avg. Q_{noHint} Size
AspectJ	286	188	2.50	224.35	49.10
BookKeeper	40	24	3.96	83.63	24.83
Derby	96	49	2.64	190.41	44.14
JodaTime	9	7	1.00	126.57	46.14
Lucene	34	32	2.09	232.90	27.43
Mahout	30	25	3.12	97.72	34.24
OpenJpa	18	16	2.06	166.06	41.06
Pig	48	36	1.53	117.44	25.00
Solr	55	45	2.22	87.80	42.93
SWT	98	85	1.68	100.56	42.80
Tika	23	21	2.38	74.33	26.00
ZooKeeper	80	78	1.91	106.09	38.14
ZXing	20	14	1.07	123.31	73.23
Total	837	620	2.16	133.17	39.62

A. Data Collection

We used a set of 620 bug reports manually extracted and verified from 13 software systems. To obtain these bug reports, we initially started from datasets that were used in two previous studies to analyze the effectiveness of TR-based bug/feature localization techniques [8], [10]. We then manually inspected each bug report in these datasets and their corresponding commit and further cleaned the data, as described below.

First, each commit associated with a bug report BR in the datasets was manually analyzed, in order to verify the validity of the ground truth (*i.e.,* the set of files actually modified to fix the bug described in BR). This step was necessary to ensure that only valid files and bugs are being used in our study. To do this verification, two of the authors manually verified the data, each of them focusing on around half of the bug reports in the dataset and their fixes. Moreover, a third author double-checked the manual labeling done by the other two authors, in order to identify any involuntary errors or misunderstandings, as well as to determine any cases in which additional screening or discussion was required. The authors first identified each BR's bug-fixing commit fix_{BR}, by looking in the project's versioning system for commit notes explicitly reporting the BR id (*e.g.,* `DERBY − 6150`). Note that `git` is used by all subject systems except JodaTime, which uses `SVN`; however, the process for JodaTime remains the same using analogous `SVN` features. The mapping between bugs and their respective fixing commits was possible for all the bugs included in our study. Since the presence of a bug id does not always guarantee that the commit contains only the needed bug fix [29], a manual inspection of the actual changes was needed to ensure that only files relevant to the bug are included in the ground truth. Therefore, the two authors manually inspected the changes performed in fix_{BR} by exploiting the `git show` command (or the equivalent `svn diff −c` command), and tagged each modified file as *true positive* (*i.e.,* the file has been actually modified to fix the bug reported in BR) or as *false positive* (*i.e.,* the file has been modified as the result of a tangled commit and/or the changes in the file do not indicate the fixing of a bug). For *false positive* changes, the authors also

tagged the change with a reason why it was considered not relevant to fixing the bug; these tags are available as part of the replication package for our study [12]. In total, 1344 files have been modified in the bug-fixing commits, but only 496 of them were indeed found to be *true positives*. The most common reasons for which the two evaluators excluded modified files from the set of *true positives* are reported in Table II.

This manual verification process resulted in the exclusion of 198 bug reports from our study out of a total of 837 found in the original datasets. These bugs were excluded since no modified files were left for them in the *true positives* set after verification. An additional 19 bug reports were excluded due to insufficient information in the downloaded source files to reconcile the bug's golden set with the code used to construct the corpus (*e.g.,* package migrations with no documentation linking the old and new location, platform-specific code that was not available in the current source download, *etc.*). These cases arose most frequently for the ZXing project, which has been migrated between several version control systems. The final result was a set of 620 bugs that we used in our study.

Such a cleaning process of our dataset was needed to avoid the use of a *bloated ground truth* or *misclassified bugs* in our study [7]. Table I shows the number of bugs before and after the manual verification process in each dataset we used.

Next, for each remaining bug report BR in our data, we extracted a basic query by concatenating BR's title and description. This type of query (from now on called Q_{all}, as it contains all the terms in the bug report) is the one most often used to automatically assess the effectiveness of TR-based bug localization techniques [7]. In addition, we also manually extracted a second query (from now on, Q_{noHint}) obtained from Q_{all} by removing all code-specific terms that could represent localization hints: package, class, method, and identifier names, stack traces, code snippets, file paths, fully qualified names, and version control URLs pointing to code locations. Note that for words embedded in a localization hint (*e.g.,* the word "pointcut" in the localization hint "IfPointCut"), the word itself is still considered relevant and kept. However, the complete localization hint (*i.e.,* "IfPointCut") refers to a specific class and is removed from both the bug title and description. In total, we extracted 1240 queries from the 620 bug reports. Table I reports the systems we considered, the number of bug reports per system, and the average size of the extracted queries in number of words.

We then downloaded the source code of each system and constructed a document corpus by considering each Java file as a document. We applied preprocessing to both the queries we previously extracted and the corpus documents in order to remove English stop words and reserved Java keywords, stem words to their root form, and split identifiers in the source code based on CamelCase and the underscore separator. Each preprocessed query was then run on its corresponding document corpus (*i.e.,* the code files of the related project) by using the `lucene`[1] implementation of the Vector Space Model (VSM). The retrieved ranked list was stored for future analysis aimed at answering RQ_0 (see Section III-B).

While the above-described data is enough to answer RQ_0, it is not sufficient for answering RQ_1. Indeed, we still need to extract from the vocabulary of each bug report an "*optimal query*". Given the vocabulary of BR (*i.e.,* the Q_{all} query) composed of n terms, it would be computationally unfeasible to try all possible queries of any length that can be extracted from Q_{all} in order to observe which one leads to the best results (this would result in running $n! + (n-1)! + \ldots + 1!$ queries through the TR engine). For this reason, we instead opt for an approximation of the "optimal" query obtained using a single-objective Genetic Algorithm (GA) [39] that quickly converges towards a "near-optimal" query[2] composed starting from a given vocabulary (in our case, the terms in Q_{all}).

The solution representation (chromosome) and the GA operators are defined as follows. Given a vocabulary composed of n terms, the chromosome is represented as an n-sized integer array, where the value of the i^{th} element equals 1 if that term is part of the formulated query, and 0 otherwise. The only constraint we set on the generated solutions is that the chromosome must contain at least one "1" (*i.e.,* the query must contain at least one term). The crossover operator is a one-point crossover, while the mutation operator randomly identifies a gene (*i.e.,* a position in the array), and modifies it by randomly assigning it to 0 or 1. This translates to removing/adding a term to the query. The selection operator is the roulette-wheel.

The single-objective GA uses as a fitness function (to be minimized) the rank of the first relevant document in the list of results when running the query represented by the chromosome through `lucene` on the corpus. This fitness function is possible thanks to the fact that the ground truth is know for each bug report. Basically, we look for the query optimizing the retrieval performance as represented by the effectiveness measure, which is often used to evaluate TR-based bug localization [27].

Our GA is built on top of the `jmetal` framework[3] and uses the following parameter configuration: *population size*: 500; *maximum number of generations*: 30,000; *crossover probability*: 0.9; *mutation probability*: $1/n$ (where n is the number of terms in the bug report). Also, given that a run of the GA involves some elements of randomness (*e.g.,* in the roulette-wheel selection), we run the algorithm ten times and average the results to mitigate threats to validity introduced by chance.

We acknowledge that in many cases, modern automatic query formulation techniques would not be able to derive such an optimal query without knowing *a priori* the ground truth. However, our goal in this study is not to present the GA as a query formulation technique. Rather, we seek to identify a near-optimal query that can be formulated from a bug report in order to empirically assess the *potential* of TR-based bug localization, thus answering RQ_1.

We use our GA to extract the near-optimal query from both Q_{all} (resulting in the near-optimal query QGA_{all}) and

[1]https://lucene.apache.org/

[2]We use the term "near-optimal" since the GA will ultimately converge to a local optimal solution which may or may not be the global optima.

[3]http://jmetal.sourceforge.net

TABLE II
CHANGES UNRELATED TO BUG-FIXING IDENTIFIED BY THE TWO EVALUATORS

Change	Description	#Files	%Files
Added code only	A file was changed by only adding new code (*i.e.,* existing code was not modified/deleted)	395	46.58
Test code	Changes to the test code, not impacting the system's behavior	262	30.90
Refactoring	Changes do not affect the system's behavior (*e.g.,* renaming a variable)	74	8.73
Comments	Adding/removing/modifying code comments	72	8.49

Q_{noHint} (QGA_{noHint}), thus also studying the possibility of formulating a high-performing query when localization hints are not present in the starting vocabulary. Note also that we provided to the GA the vocabulary in Q_{all} and in Q_{noHint} after first preprocessing the queries, therefore ensuring the same treatment is applied to both queries and corpus documents.

B. Data Analysis

We answer RQ_0 by presenting the performance of TR-based bug localization when using Q_{all} and Q_{noHint} as queries. For this purpose, we adopt well-established IR metrics:

- *Effectiveness* - The highest rank in the list of results of any relevant document (*i.e.,* Java file) in the golden set. The intuition behind this metric is that given a single relevant file in the top results, developers can then easily navigate from it to the other relevant files using built-in IDE navigation. Because the effectiveness distributions in this study are highly skewed by outlying queries that perform particularly poorly, we report the median scores to provide a fair depiction of the overall effectiveness of queries in each project.
- *HITS@K* - For a set of queries run on a document corpus, it is the percentage of queries that retrieve a relevant file in the top K positions of the ranked list. For example, HITS@1 provides the percentage of queries that return a relevant file as the topmost result. In this study, similar to [8], we use HITS@1, HITS@5, and HITS@10.
- *Mean Average Precision (MAP)* - Average precision is calculated as the mean of precision values at each k such that the document returned at position k is relevant. MAP is the mean across all average precision values within a set of queries.
- *Mean Reciprocal Rank (MRR)* - The reciprocal rank is the inverse of the rank of the first relevant document in a result set. MRR is the average of all reciprocal ranks within a set of queries.

For RQ_0 we also statistically compare the TR-based bug localization performance (as assessed by the four metrics) when using Q_{all} *vs* Q_{noHint} (*i.e.,* when localization hints are present/not present in the bug report). While the Mann-Whitney U or Kolmogorov-Smirnov tests provide a mechanism to compare two potentially non-normal, independent distributions, neither of these tests handle frequent ties in compared distributions well. As such, these tests are not directly suitable for our case where, as will be shown in Section IV, many queries are able to achieve a perfect effectiveness score of one. Therefore, we applied the Asymptotic General Independence test [40] implemented in the `coin` *R* package. This is a generalized permutation test that uses random sampling to determine the independence of two distributions based on mean-differences. It is applicable to non-normal, independent, discrete distributions despite the presence of ties. We also assess the magnitude of the observed difference using Cliff's delta (d) effect size [41], suitable for non-parametric data. Cliff's d ranges in the interval $[-1, 1]$ and is negligible for $|d| < 0.148$, small for $0.148 \leq |d| < 0.33$, medium for $0.33 \leq |d| < 0.474$, and large for $|d| \geq 0.474$. The same four performance metrics/statistical analysis are also computed to answer RQ_1, in which we compare the performance of Q_{all} and Q_{noHint} with the performance of the near-optimal queries QGA_{all} and QGA_{noHint}, obtained by the GA starting from the vocabulary present in Q_{all} and in Q_{noHint}, respectively.

C. Replication Package

All the data from our study are publicly available [12]. In particular, we provide:

- The dataset we manually built, reporting for each bug report: (i) its id, title, description, and link to the issue tracker, (ii) the related bug-fixing commits, (iii) all files modified in the bug-fixing commits, (iv) the list of files modified in the bug-fixing commits tagged as *true positive* (*i.e.,* changed to fix the bug), (v) Q_{all}, (vi) Q_{noHint}, (vii) the near-optimal query QGA_{all} obtained starting from Q_{all}, and (viii) the near-optimal query QGA_{noHint} obtained starting from Q_{noHint};
- The raw data used to answer our two research questions;
- The *R* scripts used to perform statistical analyses;
- The `jar` file implementing our GA accompanied by a `README` file explaining how to use it to obtain the near-optimal query for a given vocabulary and document corpus.

IV. RESULTS

A. RQ_0: Evaluation of Q_{all} and Q_{noHint}

Table III shows all evaluation metrics for each project in the dataset when using the complete bug report as query, both including (Q_{all}) and excluding (Q_{noHint}) localization hints. For each metric calculated for each project, a pair of bold values indicates statistical significance between them at 95% confidence with at least a small effect size. Asterisks indicate medium effect sizes. We observed that the impact of localization hints is not statistically significant across all data; however, there are instances where localization hints are seen to substantially boost TR performance. Overall, looking at the last row in Table III, it is clear that while localization hints definitely help TR-based bug localization, the difference in performance between Q_{all} and Q_{noHint} is not as strong as observed in previous studies [26], [7]. Indeed, Wang *et al.* [26] and Kochhar *et al.* [7] found a statistically significant difference in the performance of TR-based bug localization when using queries containing/not containing localization hints, accompanied by a large effect size.

These differences in findings might be due to several reasons. First, it is worth noting that the dataset used in our experiment

TABLE III

RQ$_0$: COMPARISON OF QUERIES Q_{all} AND Q_{noHint}. A PAIR OF BOLD VALUES INDICATES STATISTICAL SIGNIFICANCE BETWEEN THEM AT 95% CONFIDENCE WITH AT LEAST A SMALL EFFECT SIZE AND * INDICATES MEDIUM EFFECT SIZES.

	Median Effectiveness		HITS@1		HITS@5		HITS@10		MAP		MRR	
Project	Q_{all}	Q_{noHint}	Q_{all}	Q_{noHint}	Q_{all}	Q_{noHint}	Q_{all}	Q_{noHint}	Q_{all}	Q_{noHint}	Q_{all}	Q_{noHint}
AspectJ	33	36	0.04	0.06	0.17	0.18	0.28	0.26	0.09	0.10	0.12	0.13
BookKeeper	2	3	0.29	0.21	0.79	0.67	0.83	0.75	0.38	0.33	0.51	0.41
Derby	**15**	**26**	0.16	0.12	0.39	0.31	0.43	0.39	0.17	0.16	0.25	0.21
JodaTime	**2***	**13***	0.14	0.14	**0.57**	**0.14**	**0.57**	**0.29**	0.38	0.21	0.38	0.21
Lucene	**2.5***	**11***	0.31	0.20	**0.69**	**0.33**	0.78	0.47	0.38	0.25	**0.47**	**0.28**
Mahout	**5***	**25***	**0.36**	**0.12**	**0.64**	**0.32**	**0.80**	**0.40**	**0.46***	**0.18***	**0.46***	**0.19***
OpenJpa	**5.5**	**18.5**	0.19	0.13	**0.50***	**0.25***	0.56	0.31	**0.32**	**0.15**	0.35	0.22
Pig	9	10.5	0.25	0.19	0.44	0.39	0.53	0.50	0.30	0.28	0.35	0.30
Solr	**2**	**7**	**0.33**	**0.18**	0.67	0.42	0.76	0.56	0.42	0.26	0.49	0.30
SWT	**3**	**5**	0.35	0.31	0.61	0.53	0.78	0.69	0.43	0.37	0.48	0.43
Tika	**2***	**16***	**0.43***	**0.10***	0.62	0.29	0.81	0.38	**0.38***	**0.16***	**0.53***	**0.21***
ZooKeeper	**2**	**5**	**0.41***	**0.17***	0.73	0.52	0.79	0.64	**0.49**	**0.28**	**0.55**	**0.32**
ZXing	2	6	0.36	0.31	0.57	0.46	0.64	0.54	0.45	0.36	0.49	0.40
Total	**6**	**13**	0.24	0.16	0.48	0.36	0.58	0.46	0.36	0.24	**0.42**	**0.28**

is different from those used in these previous studies [26], [7]. Second, as explained in Section II, we adopt a different experimental design. More specifically, we use the same set of bug reports to derive Q_{all} and Q_{noHint}: the latter is a "worst case scenario" in which all terms matching code components have been manually redacted from the bug report (*i.e.,* from Q_{all}). This allows us to review the performance of TR-based localization in the presence of a potentially elevated vocabulary mismatch (since matching terms between the code identifiers and the bug report have been removed from the latter) and directly compare the same bug report with and without localization hints. Previous studies classify instead the set of bug reports on the basis of the degree of bug localization they contain (*e.g.,* bug is fully localized in the report, partially localized, or not localized) and compare the performance of TR-based bug localization on these (*disjoint*) sets of bugs. These differences make a direct comparison of the results achieved in our study versus previous studies difficult. At minimum, these results underscore the fact that the impact of localization hints on the performance of TR-based bug localization is inextricably linked to the specific project under study and the set of considered bug reports from that project.

To better address this situation, future work will investigate in more detail the impact of specific types of localization hints removed from each query and manually clean a larger set of data to further mitigate any potential sampling bias.

Finally, it is worth commenting on the overall performance achieved by using as query the whole textual content of the bug report, especially when localization hints are not present (*i.e.,* Q_{noHint}, the scenario in which TR-based bug localization is needed the most). The median effectiveness across all projects is 13, meaning that for half of the queries the first relevant document is retrieved after position 13 in the ranked list. Considering that our study is run at file level, the effort required to analyze false positives in the ranked list would likely be too high to be considered acceptable by developers. Also, Q_{noHint} was able to retrieve a buggy file in the top five positions (HITS@5) for only 36% of queries and in the top ten positions (HITS@10) for 46% of queries, on average. This translates to poor performance also across the rest of the metrics.

The achieved results support doubts raised in previous work about the actual usefulness of TR-based bug localization when localization hints are not provided. However, there is an important detail to remember: as done in previous work, we are using here the whole textual content of the bug report as a query (all the terms in the bug title and the bug description that remain after filtering out localization hints and applying preprocessing), which, as previous studies showed [10], can contain noise that hinders TR performance. In the next research question we investigate what the potential effectiveness of TR-based bug localization is given the ability of formulating a near-optimal query instead of using the default one.

Summary for RQ$_0$: The presence of localization hints can boost the results of TR-based bug localization at the project-level, but not necessarily for all bug reports and/or in all projects. We also observed that the performance of TR-based bug localization is poor in the scenario it is needed the most, with less than 50% of queries able to retrieve a relevant result in the top 10 positions of the ranked list.

B. RQ$_1$: Evaluation of the Near-Optimal Queries QGA_{all} and QGA_{noHint}

Table IV shows all evaluation metrics for each project in the dataset when using a near-optimal query generated by our GA based on the bug report vocabulary with and without localization hints. The improvements made by selectively formulating a query rather than using the complete bug report vocabulary are immediately noticeable. Beginning with median effectiveness, we now see that for at least half of the queries a user will arrive at a buggy file after inspecting only the first item in the result set. Note that this holds both when localization hints are present in the bug report (QGA_{all}) and when they are not (QGA_{noHint}). This finding is further supported by the average HITS@1 score over the entire dataset, which shows that overall 69% and 67% of the queries with and without localization hints respectively, are able to return a relevant file in the first position in the list of results. This is a more than three-fold improvement over the queries before applying GA, which are the default queries generally used in the evaluation of TR techniques. Moreover, the MAP and MRR of each dataset also improves dramatically, indicating that not only is one relevant file being pushed to the top of the result set, but many of the other relevant files are moving up the list as well.

For this research question we performed the same statistical tests between queries that contain localization hints and those

TABLE IV
RQ$_1$: PERFORMANCE OF *Near-optimal Queries* QGA_{all} AND QGA_{noHint}

	Median Effectiveness		HITS@1		HITS@5		HITS@10		MAP		MRR	
Project	QGA_{all}	QGA_{noHint}	QGA_{all}	QGA_{noHint}	QGA_{all}	QGA_{noHint}	QGA_{all}	QGA_{noHint}	QGA_{all}	QGA_{noHint}	QGA_{all}	QGA_{noHint}
AspectJ	1	1	0.46	0.60	0.72	0.79	0.78	0.86	0.41	0.49	0.58	0.68
BookKeeper	1	1	0.88	0.75	1.00	0.96	1.00	0.96	0.56	0.48	0.92	0.85
Derby	1	1	0.55	0.53	0.67	0.65	0.67	0.73	0.40	0.40	0.61	0.60
JodaTime	1	1	0.57	0.57	1.00	0.86	1.00	0.86	0.79	0.73	0.79	0.73
Lucene	1	1	0.80	0.63	0.93	0.80	0.97	0.80	0.63	0.49	0.85	0.72
Mahout	1	1	0.88	0.56	0.96	0.72	1.00	0.76	0.84	0.52	0.91	0.63
OpenJpa	1	1	0.75	0.75	0.88	0.94	0.94	1.00	0.63	0.61	0.82	0.83
Pig	1	1	0.69	0.64	0.86	0.81	0.89	0.86	0.65	0.61	0.77	0.71
Solr	1	1	0.84	0.71	0.91	0.84	0.93	0.89	0.68	0.57	0.88	0.77
SWT	1	1	0.79	0.76	0.89	0.91	0.94	0.93	0.69	0.67	0.85	0.83
Tika	1	1	0.90	0.71	0.90	0.86	0.90	0.86	0.59	0.42	0.91	0.78
ZooKeeper	1	1	0.81	0.74	0.95	0.88	0.96	0.94	0.74	0.70	0.86	0.80
Zxing	1	1	0.93	0.85	1.00	0.92	1.00	1.00	0.91	0.86	0.95	0.90
Total	1	1	0.69	0.67	0.84	0.83	0.88	0.88	0.66	0.58	0.82	0.76

that do not. We found that in the case of queries formulated by the GA, the difference in values other than median effectiveness are statistically significant only for Mahout, but even then only with a negligible effect size. Otherwise, near-optimal queries manage to mitigate the lack of localization hints every time. In addition, we calculated the same statistical tests to measure differences between the queries composed by using the whole text in the bug report (*i.e.,* those used in RQ$_0$) and the near-optimal queries formulated by the GA, finding a statistically significant difference with medium or large effect sizes in every case. Summarizing these findings, **the achieved results show that the vocabulary of the bug report is all we need to formulate a good query and make the application of TR-based bug localization successful in most cases**.

It is also worth noting that these results have been achieved by using a simple Vector Space Model (VSM) as the TR engine, configured with default parameters and publicly available in an open source library (*i.e.,* `lucene`). While our goal in this study was to determine the best results we can get by only manipulating the query, a more robust TR engine capable of better handling issues such as vocabulary mismatch, may lead to even better performance. Further, these results show that even though there are numerous studies which have itemized and dissected the limitations of applying TR to source code, specifically for bug localization, the query itself is a tremendously important aspect of optimizing TR, potentially more than internal parameters or even the choice of TR engine (a claim supported by the fact that we are able to achieve such good results even with a rudimentary TR approach).

The results achieved in RQ$_1$ are very promising for researchers working on TR-based bug localization and its many derivatives, as well as practitioners that have implemented or are interested in implementing these systems to improve their development process. However, these findings also come with some practical implications that need to be addressed in the evaluation of future TR approaches. Mainly, evaluating TR techniques using the standard approach of building queries by concatenating the contents of the bug report title and description is an additional factor that could introduce bias in a TR experiment's results. Indeed, our data suggests that the particular query formulated starting from a given bug report is a major factor in the performance of even a rudimentary TR approach. Therefore, great care should be taken when formulating queries from bug reports for evaluation to not only address potential biases such as the presence of localization hints and bloated ground truths, but also biases imposed by the query itself. For example, it has been suggested that queries which contain a large volume of text are also likely to contain a large volume of noise [10], which innately lowers performance. This is supported also by the results of our study, where the GA queries that obtained better results than the original queries were also shorter (see Table V).

Clearly, it is often a non-trivial task to read a bug report and convert its text into a meaningful query that TR approaches can use to locate buggy files. In fact, previous work on iterative query refinement through user feedback [42] found that developers were often unable to reach a good query given a sufficiently bad starting point. Because the quality of a query is not always readily apparent, one can imagine situations in which studies have an imbalance of queries with various degrees of quality. It is then entirely possible that this balance, whether in favor of poor or high quality queries, injects bias into the evaluation results.

Given that initial query formulation is so important to the outcome of a TR approach, it is natural to ask questions about how the research community can best support query formulation given a bug report vocabulary without *a priori* knowledge of the ground truth. As mentioned in Section II, previous work has focused on how to reformulate a query from an initial query. We see the set of near-optimal queries publicly available in our replication package [12] as a precious source that can represent the starting point of further research on this topic, such as studying what the characteristics of high-performing queries are. This is part of our future work. As a hint on how different the near-optimal queries formulated by our GA are with respect to the queries including the whole textual content of the bug report, Table V shows the average size of Q_{all} and Q_{noHint} before and after the application of the GA. We report the size in terms of unique and non-unique terms. In a vast majority of cases, the GA reduces the terms used in the original query by more than 50%, and in each case it results in a higher ratio of unique to non-unique terms. This supports the idea that there are words in the original queries in both Q_{all} and Q_{noHint} whose relevance is diminished by other noisy terms, such that the TR approach is not able to appropriately leverage the information represented by meaningful terms.

TABLE V
AVERAGE QUERY SIZES IN NUMBER OF WORDS BEFORE AND AFTER APPLYING THE GA

	Q_{all}				Q_{noHelp}			
	Before GA		After GA		Before GA		After GA	
project	Non-Unique	Unique	Non-Unique	Unique	Non-Unique	Unique	Non-Unique	Unique
AspectJ	224.35	61.10	104.63	39.34	49.10	29.89	20.83	14.95
BookKeeper	83.63	35.83	40.25	22.75	24.83	17.92	11.13	9.29
Derby	190.41	53.29	89.94	35.33	44.14	26.65	19.43	14.43
JodaTime	126.57	44.00	59.86	27.00	46.14	29.43	19.43	15.43
Lucene	232.90	98.00	114.27	57.53	27.43	21.63	11.53	10.17
Mahout	97.72	45.72	47.28	28.36	34.24	24.96	13.96	11.60
OpenJpa	166.06	61.00	77.69	39.63	41.06	29.94	17.50	15.19
Pig	117.44	46.25	53.28	28.61	25.00	19.11	10.50	8.72
Solr	87.80	41.09	41.51	25.00	42.93	29.13	19.16	15.22
SWT	100.56	41.01	47.80	25.13	42.80	27.84	19.31	14.91
Tika	74.33	34.76	35.05	20.57	26.00	19.48	10.90	9.57
ZooKeeper	106.09	43.36	52.18	27.65	38.14	25.58	17.92	14.35
ZXing	123.31	72.92	58.62	40.54	73.23	54.92	33.23	27.92
Total (Avg.)	133.17	52.18	63.26	32.11	39.62	27.42	17.29	12.82

Table VI also shows some examples of queries before and after the GA application. These queries possess some interesting properties. From this data, we can see extreme cases in which the GA significantly reduces the number of terms in the query, by one order of magnitude. Finding the right query in these cases could prove challenging for a human, given so many term combinations, which stresses the need for automatic techniques to help with this task.

For example, for bug 40257 in AspectJ, the GA is able to derive a two-word query from an original eleven-word query which boosts the effectiveness from 124 to 9. Similarly, for the bug tika-1083, the GA is able to improve the effectiveness from 208 to 24 by reducing an originally 17-word query to the much simpler "add xml".

For the bug lucene-4469, the algorithm derives a query with maximum effectiveness from the original query with an effectiveness of 249 by reducing the 31-word query to a seven-word one. Each of these cases illustrate the amount of information buried in bug reports that could potentially be uncovered by automated formulation techniques, thus increasing the performance of TR-based bug localization.

It is also interesting to note that given a query that is already able to retrieve a relevant document in position one (*i.e.,* the effectiveness of the query before applying the GA is 1), the GA is still able to reduce the number of words in the query while maintaining the same, perfect effectiveness. For example, bug pig-3327 results in an original query of 21 words having perfect effectiveness. The GA is able to derive from it a query still exhibiting perfect effectiveness but using as few as seven words. This indicates that not all noisy words in a query are equivalent. There are some sources of benign noise that do not affect the overall effectiveness of the query and are just ancillary words not really required to represent the information sought in the document corpus. Our future work will focus on determining factors that can lead to automatically identifying these terms without knowing the ground truth *a priori*.

Additionally, we note that there is often not a single and unique *near-optimal query* derived by the GA. In fact, for bug 37576 in AspectJ (see table VI), the algorithm is able to derive ten distinct queries (*i.e.,* one for each iteration), with lengths ranging from three to ten words, all having perfect effectiveness. This suggests that automatic formulation techniques should not always look for one single, ideal query. Rather, many alternative queries built from the text in the bug report could all lead to exceptional retrieval results.

Finally, it is important to compare these results to those reported by Chaparro *et al.* [36]. Table VII shows the median effectiveness and HITS@k metrics for the subset of 102 bugs from 10 different projects that were common between the two studies. To perform this analysis, we obtained the observable behavior-based queries from Chaparro *et al.*'s replication package, and used our ground truth and TR engine to evaluate query effectiveness. We see that while considering only terms related to observable behavior does improve query performance compared to the original queries Q_{all} and Q_{noHint}, there is still room for improvement, as the GA is able to derive queries with even better performance. This is not unexpected, however, since as compared to the observable behaviors approach, the GA is able to derive queries based on patterns in data that might be unrecognizable to a human user. For example, for bug bookkeeper-326, the query derived from observable behaviors contains 9 terms with an effectiveness of 151 and is easily recognized by a human: "deadlock during ledger recovery deadlock found during ledger recovery." Alternatively, the largest query derived by the GA for the same bug report consists of 5 terms with an effectiveness of 3, and is much more opaque from the perspective of human comprehension: "lock ledger deadlock ledger thread".

Summary for RQ$_1$: We find strong evidence that bug reports themselves provide sufficient information to perform bug localization and a near-optimal query extracted from the bug report vocabulary significantly improves the performance of TR-based bug localization compared to the default query. This indicates the potential usefulness of TR bug localization, even in cases where the bug report does not contain localization hints, which is an especially problematic situation for developers [30]. We strongly believe that the definition of techniques supporting the formulation of an *optimal query* should be the main research direction to investigate in TR-based bug localization.

V. THREATS TO VALIDITY

Internal validity. We reduced these threats by considering bug reports used in previous studies [8], [10]. Also, we performed a manual verification and cleaning process, which

TABLE VI
SELECTED GA FORMULATIONS

BugId	Initial Eff	Initial Query	GA Eff	GA Query
AspectJ-40257	124	pars path lst file broken rel path longer parser properli ajd	9	lst ajd
tika-1083	208	add link uti valu tika metadata xml tika 1012 ad tika link tika uti patch fill valu	24	add xml
lucene-4469	249	test appear useless reason guess debug explicitli disabl mdw call dont check valu test current fail wouldnt rememb right catch throwabl record insid statu test chang let mdw run search test	1	appear disabl check valu test wouldnt rememb
pig-3327	1	pig hit oom fetch task report gc overhead limit exceed hit 23 script launch job ha 80k map arrai caus oom	1	oom task report map arrai caus oom
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant rc2 ajc iajc take entiti versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch rc2 iajc bootclasspath entiti classpath
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch iajc nest entiti classpath versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant classpath entiti rc2 iajc take bootclasspath entiti vice versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch rc2 iajc take nest bootclasspath versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	classpath iajc take
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch boot classpath ajc iajc nest entiti versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch classpath iajc
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	rc2 ajc iajc take classpath vice versa
AspectJ-37576	4	ant task switch boot classpath entiti rc2 ajc task iajc task take nest bootclasspath entiti classpath vice versa	1	ant switch boot classpath ajc iajc take nest entiti

TABLE VII
COMPARISON WITH OBSERVABLE BEHAVIORS QUERIES

	Q_{all}	Q_{noHint}	Obs. Behavior	QGA_{all}	QGA_{noHint}
Med. Effectiveness	20	29	23.50	1.05	1
HITS@1	0.12	0.06	0.11	0.50	0.53
HITS@5	0.25	0.22	0.29	0.75	0.75
HITS@10	0.34	0.31	0.36	0.78	0.81

resulted in slightly less data for addressing external validity, but provides more confidence in the correctness of our data.

To lessen the likelihood of errors in the manual cleaning process, the two authors in charge of (i) labeling files changed in bug-fixing commits as true or false positive and (ii) removing localization hints from the text of the bug report, followed an agreed upon definition of localization hints and of the distinction between relevant and irrelevant code changes, as described in the study design. Moreover, a third author verified the correctness of the labeling, identifying cases in which additional screening or discussion was required.

There is also a threat introduced by the randomness of the GA. Because two executions of the GA on the same input can lead to different results, we performed ten trials for each application of the GA, averaging across the resulting metrics. For 80% of the Q_{all} queries and 95% of the Q_{noHint} queries there was no change in effectiveness between trials.

External validity. Our results may not be generalizable to software projects different than the ones used in this study. Most notably, we use only Java open source projects in our study. Thus, results may not generalize to commercial systems or those written in other languages. Also, while we focus on file-level bug localization, in future we plan to assess whether our results also hold when working at method-level granularity.

Construct validity. We mitigated these threats by employing four different metrics widely used to measure the performance of TR-based bug localization [7]. Also, we used the Asymptotic General Independence test to determine the statistical significance of our results because, when looking at the effectiveness of individual queries within a system, we are comparing non-normal distributions with a high percentage of ties. For continuity and comparability of results, we use the same generalized test for all data analysis, as its statistical assumptions hold even for data having lower instances of ties.

VI. ACKNOWLEDGEMENTS

Sonia Haiduc and Esteban Parra are supported in part by the National Science Foundation (NSF) grant CCF-1526929. Gabriele Bavota and Jevgenija Pantiuchina acknowledge the support by the SNF through the JITRA project, No. 172479.

VII. CONCLUSIONS AND FUTURE WORK

In this study we first show that TR-based approaches to bug localization exhibit poor performance when using the full text in a bug report as query, particularly in the case when localization hints are not present. This is consistent with previous work that raised localization hints as a possible bias in evaluations [7], [8]. More importantly, we show that given only the vocabulary of a bug report, there exists a near-optimal query capable of drastically improved performance compared to a query containing the entire bug vocabulary. This holds even in the absence of localization hints and when using a rudimentary TR implementation. As a result, we show the potential of TR-based bug localization in the presence of a near-optimal query and the importance of research seeking to formulate a good initial query given only a bug report vocabulary.

In future work we will focus on manually cleaning a larger set of bug reports from other systems and extending our analysis also to method-level. Additionally, we plan to use the data obtained from our GA implementation to better understand the properties of optimal queries and how they can be used to improve query formulation and quality prediction.

REFERENCES

[1] A. Marcus and G. Antoniol, "On the use of text retrieval techniques in software engineering," in *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering (ICSE'12), Technical Briefing*. Zurich, Switzerland: IEEE, 2012.

[2] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proceedings of the 11th IEEE Working Conference on Reverse Engineering (WCRE'04)*. Delft, The Netherlands: IEEE, 2004, pp. 214–223.

[3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[4] G. Canfora and L. Cerulo, "Fine grained indexing of software repositories to support impact analysis," in *Proceedings of the 4th IEEE International Workshop on Mining Software Repositories (MSR'06)*. Shanghai, China: ACM, 2006, pp. 105–111.

[5] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Proceedings of the 28th IEEE/ACM International Conference on Software Engineering (ICSE'06)*. Shanghai, China: IEEE, 2006, pp. 361–370.

[6] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Automating extract class refactoring: An improved method and its evaluation," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1617–1664, 2013.

[7] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*. Vasteras, Sweden: ACM, 2014, pp. 803–814.

[8] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of ir-based fault localization techniques," in *Proceedings of the 24th International Symposium on Software Testing and Analysis (ISSTA'15)*. Baltimore, MD, USA: ACM, 2015, pp. 1–11.

[9] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. D. Lucia, "Predicting query quality for applications of text retrieval to software engineering tasks," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 1, p. 3, 2017.

[10] O. Chaparro and A. Marcus, "On the reduction of verbose queries in text retrieval based software maintenance," in *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering (ICSE'16)*, Austin, TX, USA, 2016, pp. 716–718.

[11] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR'13)*. San Francisco, CA, USA: IEEE, 2013, pp. 121–130.

[12] *Replication Package*, 2018. [Online]. Available: http://www.cs.fsu.edu/~serene/mills_icsme_18_bugs/

[13] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proceedings of the 35th ACM/IEEE International Conference on Software Engineering (ICSE'13)*. San Francisco, CA, USA: IEEE, 2013, pp. 842–851.

[14] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *Proceedings of the 15th IEEE Working Conference on Reverse Engineering (WCRE'08)*, Koblenz-Landau, Germany, 2008, pp. 155–164.

[15] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in *Proceedings of the 8th IEEE Working Conference on Mining Software Repositories (MSR'11)*. Waikiki, HI, USA: ACM, 2011, pp. 43–52.

[16] D. Poshyvanyk, A. Marcus, Y. Dong, and A. Sergeyev, "Iriss-a source code exploration tool," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*. Budapest, Hungary: IEEE, 2005, pp. 25–30.

[17] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "SNIAFL: Towards a static non-interactive approach to feature location," *ACM Transactions on Software Engineering and Methodologies*, vol. 15, no. 2, pp. 195–226, 2006.

[18] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: Mining and searching internet-scale software repositories," *Data Mining and Knowledge Discovery*, vol. 18, no. 2, pp. 300–336, Oct. 2008.

[19] T. Savage, M. Revelle, and D. Poshyvanyk, "Flat3: Feature location and textual tracing tool," in *Proceedings of the 32nd IEEE/ACM International Conference on Software Engineering (ICSE'10)*, vol. 2. Cape Town, South Africa: IEEE, 2010, pp. 255–258.

[20] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering (ICSE'11)*. Waikiki, HI, USA: ACM, 2011, pp. 111–120.

[21] 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 *Proceedings of the 34th IEEE International Conference on Software Engineering (ICSE'12)*. Zurich, Switzerland: IEEE, 2012, pp. 14–24.

[22] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: An extensible local code search framework," in *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE'12)*. Cary, NC, USA: ACM, 2012, pp. 15:1–15:2.

[23] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, Sep. 2010.

[24] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proceedings of the 28th IEEE International Conference on Automated Software Engineering (ASE'13)*. Palo Alto, CA, USA: IEEE, 2013, pp. 345–355.

[25] S. Wang, D. Lo, and J. Lawall, "Compositional vector space models for improved bug localization," in *Proceeedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*. Victoria, Canada: IEEE, Sep. 2014, pp. 171–180.

[26] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *Proceedings of the 22nd IEEE International Conference on Program Comprehension (ICPC'14)*. Hyderabad, India: IEEE, 2014, pp. 53–63.

[27] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[28] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Science China Information Sciences*, vol. 58, no. 2, pp. 1–24, 2015.

[29] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *Proceedings of the 33rd IEEE/ACM International Conference on Software Engineering (ICSE,11)*. Waikiki, HI, USA: IEEE, 2011, pp. 351–360.

[30] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'08)*. Atlanta, GA, USA: ACM, 2008, pp. 308–318.

[31] S. Haiduc, G. Bavota, R. Oliveto, A. Marcus, and A. De Lucia, "Evaluating the specificity of text retrieval queries to support software engineering tasks," in *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, Jun. 2012, pp. 1273–1276.

[32] D. Shepherd, Z. Fry, E. Gibson, L. Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *Proceedings of the 6th International Conference on Aspect Oriented Software Development (AOSD'07)*. Vancouver, Canada: ACM, 2007, pp. 212–224.

[33] M. Roldan-vega, G. Mallet, E. Hill, and J. A. Fails, "Conquer: A tool for nl-based query refinement and contextualizing source code search results," in *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM'13)*. Eindhoven, The Netherlands: IEEE, 2013, pp. 512–515.

[34] M. M. Rahman and C. K. Roy, "Improved query reformulation for concept location using coderank and document structures," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. Urbana-Champaign, Illinois, USA: IEEE, 2017, pp. 428–439.

[35] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *ACM Computing Surveys (CSUR)*, vol. 44, no. 1, p. 1, Jan. 2012.

[36] O. Chaparro, J. M. Florez, and A. Marcus, "Using observed behavior to reformulate queries during text retrieval-based bug localization," in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME'17)*. Shangai, China: IEEE, 2017, pp. 376–387.

[37] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Information and Software Technology*, vol. 82, pp. 177–192, 2017.

[38] Y. Zhou, Y. Tong, T. Chen, and J. Han, "Augmenting bug localization with part-of-speech and invocation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 06, pp. 925–949, 2017.

[39] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[40] H. Strasser and C. Weber, "On the asymptotic theory of permutation statistics," *Mathematical Methods of Statistics*, vol. 2, 1999.

[41] R. J. Grissom and J. J. Kim, *Effect Sizes for Research: A Broad Practical Approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[42] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location," in *Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM'09)*, Edmonton, Canada, Sep. 2009, pp. 351–360.