Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 1598 – 1614

doi:10.3906/elk-2006-14

Information retrieval-based bug localization approach with adaptive attribute weighting

Mustafa ERŞAHİN[1,*], **Semih UTKU**[1], **Deniz KILINÇ**[2], **Buket ERŞAHİN**[3]
[1]Department of Computer Engineering, The Graduate School of Natural and Applied Sciences, Dokuz Eylül University, İzmir, Turkey
[2]Department of Computer Engineering, Faculty of Engineering, Bakırçay University, İzmir, Turkey
[3]Department of Computer Engineering, Faculty of Engineering, İzmir Institute of Technology, İzmir, Turkey

Received: 03.06.2020 • **Accepted/Published Online:** 06.11.2020 • **Final Version:** 31.05.2021

Abstract: Software quality assurance is one of the crucial factors for the success of software projects. Bug fixing has an essential role in software quality assurance, and bug localization (BL) is the first step of this process. BL is difficult and time-consuming since the developers should understand the flow, coding structure, and the logic of the program. Information retrieval-based bug localization (IRBL) uses the information of bug reports and source code to locate the section of code in which the bug occurs. It is difficult to apply other tools because of the diversity of software development languages, design patterns, and development standards. The aim of this study is to build an adaptive IRBL tool and make it usable by more companies. BugSTAiR solves the aforementioned problem by means of the adaptive attribute weighting (AAW) algorithm and is evaluated on four open-source projects which are well-known benchmark datasets on BL. One of them is BLIA which is the state of the art in bug localization area and another is BLUIR which is a well-known BL tool. According to the promising results of experiments, Top1 rank of BugSTAiR is 2% and MAP is 10% better than BLIA's results on AspectJ and it has localized 4.6% of all bugs in Top1 and its precision is 6.1% better than BLIA on SWT, respectively. On the other side, it is 20% better in the Top1 metric and 30% in precision than BLUIR.

Key words: Software engineering, bug localization, information retrieval, genetic algorithm, software process improvement

1. Introduction

Many studies have been conducted to reduce maintenance costs in software development processes and to improve the quality of software, as evaluated considering different metrics [1, 2]. The typical software development life cycle (SDLC) consists of iterative phases ranging from requirements analysis to maintenance. There can be various issues in each phase which threaten the quality of software. Software bugs are one of the most important threats in this process since they are visible to the end user and reduce customer's confidence in a piece of software. The maintenance phase of the SDLC starts after the release of the software, and its cost is generally more than development costs for large-scale software projects. For larger software projects, catching and fixing implementation errors become more difficult. Therefore, it is important to find buggy source to reduce the maintenance time and cost. Bug localization (BL) is the process of finding portions of a source code associated with the submitted bug report. BL starts with a bug report submission, then a member of the development

*Correspondence: mustafa.ersahin@gmail.com

team proceeds to investigate it. This process consists of understanding the bug report, reproducing steps of bugs and trying to find specific parts of the program that are relevant to the reported issue. Files containing bugs are called buggy files. BL tool is a piece of software which considers source code and bug reports as input and finds similarity between them. All the BL tools can have different approaches to solve localization problems but have the same purpose. This goal is trying to help the developer for finding the actual source code part which causes the reported bug.

BL is one of the ways in which developers use bug reports from bug tracking systems like Bugzilla[1] and Jira[2]. The bug tracking system is a part of the issue tracking, which is dedicated for the software development process. All stakeholders such as developers and quality assurance engineers use these tools to track progress on bug fixing [3]. They have to overcome time-consuming challenges such as reproducing the bug as specified in the bug report, understanding the coding structure, programming logic, and goal of the related flow [4]. BL researches mainly focus to improve the overall process by providing new BL methods [5]. In general, dynamic or static methods are used in BL [6]. Dynamic methods have some processes during execution such as runtime traces, data monitoring, and tracking execution flows. On the other hand, static BL uses bug reports and source code to locate bugs. Static BL methods are easy to apply on any phase of the SDLC since they have few external dependencies and relatively low computational costs owing to information retrieval (IR) algorithms [7, 9]. In this study, BugSTAiR has been evaluated on well-known datasets like Eclipse, AspectJ, and SWT, all of which are developed with Java programming language. Some implementation details of Java can help the IR process to have better accuracy. For example, stack trace of an exception is a valuable input to indicate the buggy file and its function directly. One of the IRBL approaches shows that using stack information can improve the accuracy of BL tool up to 47% [10]. In addition, the file name is always the same as the class name that is publicly declared. Unlike Java, JavaScript (JS) is very flexible and does not force for any naming convention. Moreover, there are no experimental results on JS-based software and datasets in the literature. Another important issue is that development standards and implementation details may vary for the same programming languages depending on the company's coding standards. In addition, a web application may have many files with the same name but with different file extensions such as featurex.html, featurex.css, and featurex.js. This situation causes an extra complexity for all computations in the process. Thus, the proposed study focuses on locating non-UI related bugs such as logic and flow in web applications. Therefore, BugSTAiR evaluates only source files with ".js" extension while working on JS-based applications. The other UI-related project files such as "HTML" and "CSS" are out of our project's scope. Besides, differences in project structures and language-specific keywords have forced us to understand the characteristics of a project. All the previous IRBL tools assign attribute weights intuitively or experimentally while retrieving data. For this reason, none of these tools can be a part of a commercial application or a service. In this study, a new version of IRBL tool named BugSTAiR that is generalized for software products having source codes in AngularJS (front-end) and Java (backend) programming languages is presented. The main contributions of the study are:

- A new adaptive BL model has been proposed.
- BugSTAiR is the first tool that uses an adaptive weight calculation approach based on genetic algorithm (GA).

[1]Bugzilla (2019). Bugzilla [online]. Website https://bugzilla.mozilla.org [accessed 04 March 2019]
[2]Atlassian Jira (2019). Atlassian [online]. Website https://www.atlassian.com/software/jira [accessed 04 March 2019]

- New benchmark datasets have been shared in an open-source platform, Github[3], for further BL research. These are Eclipse dataset which includes source code histories and bug reports of three major repositories, Tomcat dataset which includes source code, bug reports, and cleaned history, and angular-translate dataset which is the first web-based BL benchmark dataset.

The remainder of this paper is organized as follows. In Section 2, the general approach of IRBL is demonstrated and the state of art is examined. In Section 3, the architecture and steps of the proposed approach is presented. In Section 4, a case study and the experimental setup such as datasets and evaluation metrics are presented. Then, experimental results are discussed. In Section 5, threats to validity of the proposed study is explained. Finally, in Section 6, conclusions and ideas for future work are explained.

2. Background

2.1. Literature review

IR is a research area that handles the representation, storage and organization of information items [11]. Many researchers have been working on IRBL on different datasets and using different methods. The idea behind IRBL is to find the relevant source files according to the common matching words between source files and bug reports. Query and document collections are two important inputs of IR researches. In this study, each bug report represents a query, and source files are document collection. IR techniques use these inputs to rank documents by similarity and relevance. The ranking process has consecutive phases starting with bug report creation. The user enters a bug report query into the system and IR techniques compute a rank score for all potential source files matching the bug query. Finally, top-ranking candidate source files are listed for developer's consideration. The success of an IR-based technique is highly dependent on the algorithms used in retrieval processes. Rao et al. [12] compared the main IR techniques and some various combinations of them. Poshyvanyk et al. [13] used a probabilistic ranking method and a data acquisition method called latent semantic indexing (LSI) in their work [14]. Zhou et al. [15] proposed BugLocator which uses the revised vector space model (rVSM). BugLocator performs on some high-scale open-source projects using text similarity between source files and bug reports. In addition, it uses the information about fixed bugs to improve BL accuracy. BugLocator has better experimental results than BugScout on the selected datasets. Another approach was introduced by Saha et al. [16]. Bug localization using information retrieval (BLUIR) uses structured information analysis of source code such as class names and method names. It locates more bugs than BugLocator according to the experimental results on the same datasets. The first IRBL tool for JS-based is the first version of BugSTAiR [17]. BugSTAiR is specifically designed for the software products developed using JavaScript and JavaScript-based web frameworks such as AngularJS and ReactJS. Therefore, it does not work for projects developed with Java or other languages. BugSTAiR uses structured information of datasets and TFIDF for IR model. There are some studies which require further information about bugs. For instance, Youm et al. [10] proposed bug localization using integrated analysis (BLIA) which considers stack traces, comments in bug reports, and change history of the source code for better accuracy. Locus is another approach which uses source code and source code history in structured format [18]. In comparison, BLIA evaluates the same datasets which are used to evaluate Locus, BugLocator, and BLUIR, and gets better results than all these approaches. On the other hand, there are three other approaches for BL. Machine learning (ML)-based BL, deep neural network (DNN)-based BL, and hybrid BL solutions which combine DNN and rVSM. DNNLoc is one of the hybrid solutions [19]. The

[3]Github. https://github.com/mustafaersahin/bugstair [accessed 05 September 2020]

BL approach of DeepLoc is based on only DNN [20]. Both of these DNN-based models perform better when the dataset has enough volume of data according to their accuracy results. ML-based BL tools have two approaches such as adopting ML models that are trained to match the topics of bug reports and classifying source files into multiple classes using previously fixed files [9]. Nguyen et al. customized the LDA approach by using a topic-based ML model and proposed BugScout [21].

2.2. IR-based bug localization

There are many open-source software products with datasets that include a lot of bug summaries. However, it is difficult to find web applications which are developed by JS-based frameworks having a bug report dataset. Figure 1 illustrates a real-world bug report from a commercial application developed for a bank. All of the benchmark datasets have structured information which includes bug id, summary, description, and names of changed files as ground truth data (GTD).

Bug ID: 1433
Summary: Account Settings – Next Button is not working after entering **security question**.
Button should trigger a redirect action to the **success state**

Source Code File:securityQuestionPageHelperFactory.js

```
angular.module("WebApp.core").factory("securityQuestionPageHelperFactory",
function (securityQuestionApi, smsOtpConfigFactory, smsOtp, $state) {
    "use strict";
     function securityQuestionPageHelper(securityQuestionPage) {
       var self = this;
       this.validateSecurityQuestionAnswer = function () {
         securityQuestionApi.validateSecurityQuestion({
             "answer": securityQuestionPage.formData.answer
           }, {
             "skipDefaultErrorAlert": false,
             "onSuccess": self.onValidateSecurityQuestionSuccess
           }
        );
    };
    this.onValidateSecurityQuestionSuccess = function (resp) {
      if (securityQuestionPage.config.smsOtp) {
          self.startSmsOtpFlow();
      } else {
          $state.go(securityQuestionPage.config.successState.name,
              securityQuestionPage.config.successState.params,
             {"location": "replace"});
      }
};
```

Figure 1. GTD example of a real-world web application.

IRBL studies rely on calculating similarity scores between bug reports and source code files according to the results of similarity matching algorithms. All source code files have a computed similarity score for each bug report. BL has some steps which should be executed in a predefined order. Detailed information about these steps are given in Section 2.3.

2.3. Common bug localization process

IRBL approaches have five main steps as shown in Figure 2. They are preprocessing, indexing, query construction, similarity computation, and retrieval.

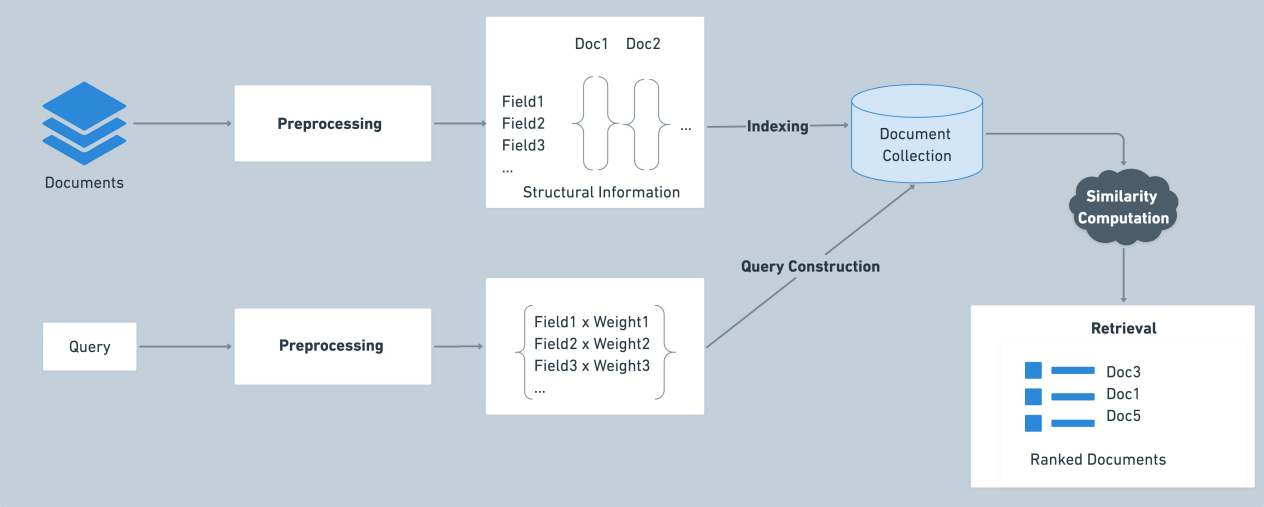

Figure 2. General view of the IRBL process.

- Preprocessing: This step is related with both source code files and bug reports. All of them should be preprocessed to improve the efficiency of the retrieval process. In this step, all stop-words such as language-specific identifiers and punctuations are removed from the source code. In addition, some syntactic operations are performed such as camel case splitting, lowercase transformation, word stemming, and tokenization.
- Indexing: IRBL approaches are used to index a dataset that is ready when both source file and bug reports are preprocessed, and dataset is prepared. The VSM is one of the well-known IR techniques but there are also some other probabilistic models such as LSI.
- Query construction: Query is one of the most important parts in IR processes. In general, summary and description fields of bug reports are used as inputs.
- Similarity computation: There are several methods such as rVSM, TFIDF, LSI to compute the similarity between bug reports and source code files. Every IRBL approach applies one of these methods to compute the relevance.
- Retrieval: After all the steps are performed, each IRBL approach applies its proposed algorithm or method to obtain better accuracy on the retrieval process.

3. Proposed approach

In this section, detailed information about the proposed approach has been given. Then, the main contribution of the paper, which is building adaptive attribute weighting (AAW) algorithm, is described.

3.1. Bug localization process

All the previous researchers that are pointed out in Section 2 studied software projects implemented with Java. According to this fact, all benchmark datasets have Java-specific information. The proposed model has adaptive processes because Java is not the only programming language in the software systems. JS is the leader among

the list of most popular programming, scripting, and markup languages according to Developer Survey Results [4]. Therefore, this study focuses on not only Java-based but also JS-based applications to build an adaptive retrieval model. A new "adaptation" step is defined to provide this IRBL architecture. The aim of this step is building IRBL basis for newcomer software and optimizing the retrieval process.

IRBL is hard to implement in JS-based web applications, and retrieval results are not as accurate as in Java-based software applications. The main reason is that the user interface (UI) of an application is related with more than one file at the same time. Figure 3 shows the general architecture of the approach. In our proposed work, it is considered that change history of the source code is as important as current source code, because any change in the source code has a history. This history can be related to a feature or a bug fix. There may be many source code files depending on the change. Evaluating the information obtained from the history, the impact analysis between the features and source code has been identified. Therefore, the history of source code is valuable, and is used to locate potential buggy files.

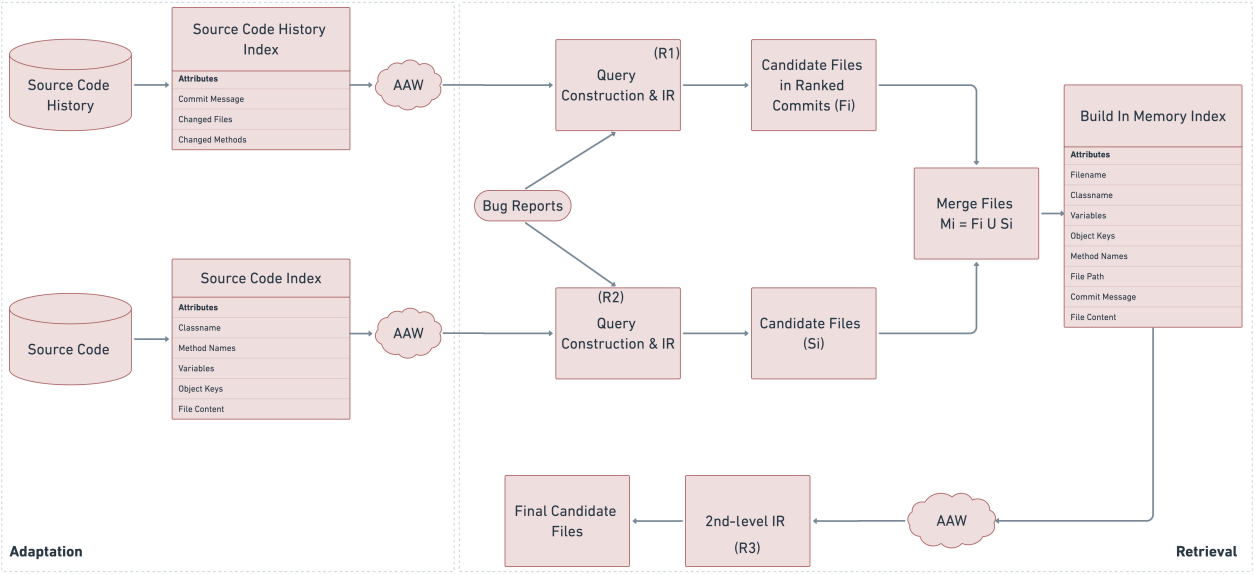

Figure 3. General architecture of the proposed approach.

In this study, a new stop-word list and JS parser have been created to process JS-based applications. The stop-word list contains list of words that are commonly found in languages which carry little or no significant semantic context in a sentence [22]. The JS-based stop-word list contains natural language words that are already in Java-based stop-word lists and also JS language-specific keywords. Details of preprocessing are covered in Section 3.2. Adaptation step includes three AAW executions to get ready for retrieval. The first two executions run after the preprocessing step in which two different indexes have been built such as source code index and change history index. Structures of these indexes are different. Source code index has five attributes which are class names, methods, variables, file content, and object keys. Change history index has three attributes which are commit messages, changed files, and changed methods. These two indexes can be created simultaneously. At this point, AAW processes run for both indexes to identify the best weights for

[4]Developer Survey Results (2019). Stackoverflow [online]. Website https://insights.stackoverflow.com/survey/2019/ [accessed 03 June 2020]

retrieval. Next, retrieval query is built as the query construction algorithm given in Section 3.5. The first-level retrieval is performed based on change history index with AAW on all attributes. The retrieval in first level is performed based on source code index. Apache Lucene[5] provides similarity scores between commit messages and bug reports. Every commit may have files that are changed more than once. Therefore, scores of the files that are found in the retrieval processes have to be consolidated. Finally, a buggy file list is gathered according to the first level of the IR process output. The second level of the IR processes starts by creating a structural in-memory index including files in the consolidated list. The proposed structure includes class names, method names, variables, object keys, and file content from the source code index. In addition, file name, file path, and commit messages are included from the source code history index as an attribute to in-memory index to combine all information in one index. The optimum attribute weights are calculated according to the GA's output in the third execution of AAW. Fitness function of GA is based on IR result of dataset and the purpose of fitness function is to reach maximum Top-1 ranked query for the generated population. Details of GA and fitness function are given in Section 3.4. File scores which come from the third execution of AAW are used for reranking between candidate files coming from the first level IR. After the reranking process, a final score is generated for each file. Thus, they can be used for any application that helps software development teams to find bugs earlier in the maintenance period of the SDLC.

3.2. Source code and bug report preprocessing

During the BL process, HTML and CSS files are excluded from source code repository and JS files are the only accepted input to be processed. Moreover, all UI-related bugs are eliminated while getting bug reports from the issue tracking system. Then, all stop-words are removed from the source code files and bug reports. Stop-words include the following keywords:

- English stop-words: "a", "the", "to" etc.
- Syntactic symbols/identifiers: "null", "undefined", "alert", "init" etc.
- Operators/punctuations: "==", "!=", "<",">" etc.

There are some different naming conventions in software companies. Identifiers may consist of more than one word. In order to increase the accuracy of the retrieval, identifiers are tokenized. To achieve this, individual tokens are used, but there might be some conflicts between bug reports and source code with regards to case sensitivity. To resolve these conflicts, all texts are transformed to lowercase. In addition to being the first BL study on the JS-based systems, the model is tested in Java-based systems. The preprocessing step can be done similarly in both systems. A language-based stop-word list has also been created to build a generic infrastructure. The language parsers are used to understand the written code structurally. JS parser and Java parser are included in the proposed tool. In order to support different software development languages, the language-based stop-word list should included in the collection, and a language parser should be added to the project in order to understand the structure of the new language. Therefore, adaptation processes can be executed easily.

3.3. Indexing

In this section, indexing process is presented. Apache Lucene is used to index the source code files. Lucene is one of the most widely used and well-known open-source IR systems. Before the indexing step, the source codes

[5]Apache Lucene (2020). Apache [online]. Website http://lucene.apache.org/core [accessed 04 March 2020]

have to be parsed to index them in a structured way. After the source codes are parsed, important information such as class, method, variable and function names are extracted. All source code files are analyzed structurally by using specific language parsers. Each source code file is called "document" in the index structure and any valuable parts of the source code are called "attributes". These attributes are added to the document as a field. Both file names and attribute names are usually written in camelCase naming convention. In this way, all the file names and attribute names which have more than one word are added to the related field of the document by using the camelCase notation. It is also discovered that these names may be included in the bug records and they are also indexed as a separate field in the document to increase success. For example, "securityQuestionsPageHelper" method name is added to the method field of the index with five inputs which are security, question, page, helper, "securityQuestionPageHelper". In addition to these structural fields in the document, file content is stored in the document.

Three indexing steps are defined in this study. All of them execute the same steps while indexing the related input documents. Source code and source code history index are built for first-level IR and in-memory index for second-level IR. After first-level indexes are formed, the process of finding the weights of the attributes begins. By means of the GA, the optimum values of the attributes on both indexes are determined in order to be used in the next step. The obtained values help in-memory index to achieve the best result on the entire dataset. Details of third index is given in Section 3.6.

3.4. Adaptive attribute weighting (AAW)

In this section, the proposed AAW algorithm is presented. AAW is the key point of the adaptation process. Every software might have different coding structures and styles. Although the structure of the indexes are the same, the weight of the each field must be different according to the software development standards. The multidimensional search on different fields and combining these results conducted using IR techniques require a dynamic calculation of the coefficients/weights that affect the search process and retrieval results. The technique to find and use weights to produce the best results is brute force search algorithm. As the size of dataset grows, it is better to use optimization methods since the time for calculation is high and this has to be repeated in certain periods (new records, daily etc.). Therefore, this subproblem becomes an optimization problem. The idea behind including AAW in the approach is to solve this optimization problem and to reduce the impact of changing project standards and application development standards on the model and to achieve more precise results. There are several optimization algorithms and methods in the literature such as GA and particle swarm optimization. In the proposed approach, GA is selected as the optimization algorithm to solve this problem.

GA is a widely used search and optimization method that works in a manner similar to the evolutionary process observed in nature. It seeks the best holistic solution based on the principle of survival in complex multidimensional search space. A GA has three main steps: crossover, mutation, and selection [23]. In this section, all configurations and strategies which are chosen for implementation are explained in detail. Initially, each chromosome is designed to have eight genes which can take double values (0.0–1.0). Each gene is represented by 16 bits and can have fractions up to two decimal points. A set of chromosomes are defined as population. Population is also a subset of solutions in the current generation. There are some limitations while defining the size of population. The diversity of the population should be maintained, otherwise it might lead to premature convergence. The population size should not be kept very large as it can cause the GA to slow down, while a smaller population might not be enough for a good crossing pool. As mentioned before, diversity of the population affects optimality, and initial population is important. Random initial populations increase the

diversity of the chromosomes in the population. In this study, the initial population is generated completely randomly with the minimum sample size of 100 and the maximum sample size of 200.

After a brief information about the population used in the GA, the crossover strategy and rate are implemented. The crossover step is similar to reproduction and biological crossover. More than one parent are selected and one or more off-springs are produced using the genetic material of the parents. The uniform crossover strategy is used to generate new off-springs with the mix probability of 0.75. In uniform crossover, each gene is evaluated separately while deciding whether it will be included in the off-springs. The second important process that provides diversity is mutation. Flip bit mutation (FBM) strategy is used in this approach. FBM is a mutation approach which has two steps such as selecting one or more bits randomly and flip them with a given probability. In this approach, 0.1 is given as mutation probability. The selection of the genes which occurs with respect to crossover and mutation is called the selection process. Consequently, complex problems are solved to inform the GA of which gene is good using a fitness function (FF) and coding variables.

In this approach, a GA is used to find a common solution for all datasets to solve attribute weighting problems. GA must have an FF to optimize the given problem. Therefore, a customized FF has been implemented to solve the current problem. This function is problem-dependent and each problem has its own FF. Details of the proposed FF are explained. The gene/population selection algorithm is the most important part in GA. Elite selection (elitism) is the most successful and preferred method in the literature. Especially, elitism strategy has been widely used in different evolutionary algorithms [24]. Therefore, the elite selection algorithm is used for the selection process. This selection method is optimized to choose the best chromosomes. The custom FF executes the required number of multiples and keeps the result in memory. After the required number of chromosomes are generated (min 100, max 200 is used in the application), the existing population is evaluated. The evaluation step is performed considering the termination function. This function decides whether the new generation should be produced or not. The application is terminated if the best chromosome value is not changed after the configured number of iterations are executed, otherwise new chromosomes and populations continue to be produced by the GA iteratively.

In this study, a GA is used to optimize IR efficiency by finding the best coefficients for any given dataset. There are three AAW executions to optimize the IRBL process. The main difference of these executions is the number of parameters which will be optimized. In this study, an AAW algorithm is designed to support variable length parameters, and it can take 3 to 8 parameters as input and returns optimized values for each parameter as output. The FF evaluates every bug report in the given dataset and compares the ranked files with GTD to find the number of Top 1 ranked queries. After all bug reports are evaluated, the FF calculates the ratio of Top 1 ranked documents in a given population. The purpose of the FF is to find the maximum ratio for a population. The formula to calculate the $FF(score)$ is represented in Equation 1:

$$FF_{score} = \frac{\#\ of\ Top\ 1\ ranked\ queries}{\#\ of\ all\ queries} \tag{1}$$

To summarize, all of the AAW executions are executed step by step including FF, selections, crossover, and mutation configurations. All the parameters such as crossover, mutation probability, and population size used in the GA are selected with grid search (GS) [25]. It is a technique that scans the dataset to select optimal parameters for the constructed model. GS works iteratively on each data and compares the results for each value. The best value for each parameter is found [26]. The configurable parameters of the applied solution are mutation, crossover rate, and the number of chromosomes. In order to understand whether the

AAW algorithm performs better, basic IR tests have been executed on benchmark datasets. The experiments are performed on four different datasets which are formed as initial structure on source code. The aim is to see better accuracies on benchmark datasets and to apply the AAW to the IRBL model. After applying the AAW algorithm, the retrieval results are better than those without AAW in the same conditions. The results of the IRBL processes for each dataset are given in Table 1. AAW provides better accuracy on all four datasets that are used for benchmarking in BL. AAW achieves better accuracy compared to our previous experiments that have no specific coefficient/weight for each attribute. According to the accuracy results, improvements on benchmark projects such as SWT, Eclipse, and AspectJ are 20%, 21%, and 62%, respectively. These improvements on benchmark datasets show that the AAW improves the IRBL process results significantly.

Table 1. Results before/after AAW.

Dataset	Before AAW(%)	After AAW(%)
SWT	47.00	56.32
Eclipse	26.19	31.61
AspectJ	20.84	33.80
Web Application	27.72	32.18

3.5. Query construction and retrieval

Source code files are called document collection and bug reports are evaluated as query in IRBL process. Since the bug reports are preprocessed in the first step, query construction is performed in the retrieval. It is understood that the query construction process is very important and critical for retrieval accuracy according to the previous studies. Many researchers have used special weights for fields on documents while constructing the retrieval query according to their empirical studies on each dataset. In the present study, outputs of the AAW process are used to set weights while constructing the retrieval queries. In addition, all words of bug report summaries that are tokenized and preprocessed in the previous steps are used in the queries. In general, bug summary contains useful information to localize bugs. Moreover, the description parts of the bug reports are examined to verify whether they have valuable information about bug or not. It is decided to include bug descriptions to queries. Bag of words algorithm (BoW) is applied to extract more valuable information from the bug description and to reduce complexity of query construction. Finally, ten words are selected according to the word counts to be added to the query. Generated queries are executed on indexes to finish retrieval process.

3.6. In-memory structured index (IMSI)

IMSI has an important role in the project. As a result of retrieval on source code history, the most similar change set is determined according to the similarity between commit messages and bug reports. Every commit may include more than one file. Therefore, unique source files are created by evaluating the changes. On the other hand, the code pieces that may potentially contain bugs are determined as a result of the IR process on structured information of source codes. The merge operation in Figure 3 is executed simultaneously. The candidate bug resources selected after the IR process on source file are merged with the candidate file names according to the IR results on the source code history. All the files in the merged list are reindexed in a more complex structured information during the execution of the application. Therefore, lists of candidate source files to index are filtered. Building an IMSI does not affect the retrieval process dramatically. After the IMSI

is constructed, there are more attributes for query construction in the IRBL process. New weights for each attribute are required. By executing the AAW process on the attributes, new weights are determined for all attributes. Thus, all the prerequisites get ready for the second level of IR.

4. Experimental study

4.1. Subject systems

To evaluate the success of the proposed approach, all experimental results of IR on well-known benchmark datasets such as Eclipse, AspectJ, SWT, and Tomcat are presented. These datasets are used in the BL field by researchers. All of them are open-source software projects that are developed with Java. The source code and change history of the subject projects are collected from Git repository of the projects. All the bug reports which have already been fixed are collected from bug tracking systems. Besides, a commercial web application is used to evaluate the performance of the proposed approach on a JS-based application since there is no open-source benchmark dataset for JS-based applications. Angular-translate project is selected as an open-source project to test BugSTAiR performance in comparison to the commercial web application. Therefore, the first benchmark dataset for web-based project is shared for BL research. Detailed information about the commercial web application and angular-translate project are given. Both projects are developed with AngularJS. Development language (method names, variables etc.), bug summaries, and descriptions are in English. They have enough information and inputs to compare experimental results to other projects.

4.2. Dataset statistics

In this section, information about datasets which are used to test the proposed approach is given. Some important statistics collected from datasets are shown in Table 2.

Indexed source files are different according to the dataset. Javascript files that have "js" extension are indexed in both angular-translate and web application datasets. Additionally, files that have "java" extension are indexed in AspectJ, SWT, Eclipse, and Tomcat dataset. These statistics directly affect the IR result and accuracy. Moreover, they are important for evaluating the success of tools. Some datasets cannot be evaluated depending on the approach adopted by the tool. Therefore, it is important to compare approaches of IRBL tools. BLUIR+, BRTracer+, BLIA, and BugLocator are well-known IRBL tools that have better accuracies on BL. Detailed comparison of these tools and BugSTAiR is shown in Table 3.

Each row of the table is a property of the BL approaches. IR method, information structure, bug similarity, version history usage, stack trace usage, and adaptive attribute weighting are properties evaluated in comparison. "O" means that tools have related property and "X" means they do not have related property in their approaches.

Bug similarity is the standard feature that is used by all IRBL tools. Structured information of source code is another common feature. BRTracer+ and BLIA use information about stack traces to improve IR accuracy. Stack trace is one of the most important features because it mostly contains direct reference of buggy source [27]. Effects of the stack trace is up to 47% according to the reports in BLIA. BLIA and Locus uses the source code version history like BugSTAiR. Source code history threats experimental results on datasets. It is not possible to have source code histories on all benchmark datasets. Specifically, source code architecture of Eclipse is different from any other datasets. Its source code is based on different repositories, so it is difficult to identify which bug is related to which repository. To avoid mismatch, the top three repositories that cover most bug reports are analyzed and selected. The source code history information and bug report dataset are

shared in open-source platforms such as Github. Tomcat has the same issue on source code history. Both of these datasets are cleaned and preprocessed before using them as a benchmark dataset. These datasets are the third contribution of the proposed work. The current states of the IRBL tools, their approaches and details of datasets with statistics are introduced and the experimental results of the retrieval process on these datasets are explained in Section 4.4.

Table 2. Dataset statistics.

Dataset	Development language	Indexed source code files	# of commits	# of bug reports
SWT	Java	738	33.994	98
Eclipse	Java	12.302	37.687	1.174
AspectJ	Java	3.692	8.291	284
Web application	Javascript/AngularJS	724	2.543	313
Tomcat	Java	2.485	62.783	1049
Angular-translate	Javascript/AngularJS	48	1.712	61

4.3. Evaluation metrics

There are some common evaluation metrics on IR research such as Top N rank accuracies, mean average precision (MAP), and mean reciprocal rank (MRR). All of the compared tools [13–15] use the same metrics to evaluate IR results.

- Top N rank: This metric is used to calculate the number of the bug reports in which at least one source file is ranked in list of retrieval results. A higher value for this metric indicates better BL performance [28]. Responsive web application is developed with AngularJs framework.
- MAP: This metric is used to find average precision and it is the major metric in IR evaluation.
- MRR: This metric is based on early precision over recall logic. Reciprocal rank is a value that is inversely proportional to the rank given by the retrieval method to a single relevant item [29]. Shortly, the MRR is the average RR of all queries.

These three metrics are used to evaluate the experiments of this study. Therefore, it is possible to compare our results to those of other tools.

4.4. Experimental results

In this section, experimental results of BugSTAiR are compared to those of the previous BL tools and the overall performance of the proposed approach is presented. The main purpose of this study is to define a generic model to localize bugs in software projects. A commercial JS-based web application is used as dataset to execute the proposed model. The other benchmark tools cannot work for WebApp due to development language limitations. Therefore, there are not any experiment related to WebApp for these tools. All the processes that are already defined in the proposed architecture are executed on both Java-based benchmark datasets and the JS-based web application. The proposed approach performs better than the tools that are created especially for Java applications. In general, BLIA has the highest performance on all datasets except Eclipse. The experimental

Table 3. Comparison of IRBL tools.

Approach	BLUIR+	BRTracer+	BLIA	BugLocator	Locus	BugSTAiR
IR Method	~TF.IDF~	rVSM	rVSM	rVSM	rVSM	~TF.IDF~
Bug similarity	O	O	O	O	O	O
Structured information of source file	O	X	O	X	O	O
Version history	X	X	O	X	O	O
Stack trace analysis	X	O	O	X	X	X
Adapted attribute ~weighting	X	X	X	X	X	O

result shows that BugSTAiR performs better than all the other tools on common evaluation metrics. Top 1 rank of BugSTAiR is 2% and MAP is 10% better than BLIA metrics on AspectJ. BugSTAiR has localized 4.6% bugs in Top 1 and its precision is 6.1% better than BLIA in SWT. There is no performance metric on Eclipse for BLIA, so BLUIR+ has the best scores. Then, BugSTAiR's performance is compared to BLUIR+. Experimental results show that our performance is 20% better than BLUIR+ in Top 1 metric and its precision is also 30% better than BLUIR+. The experimental results are summarized in Table 4. In addition, there is not any time statistics for IRBL approaches in the previous researches. Only DNNLoc has mentioned about timing statistics on both training and predicting phases. Average training time for one fold is 81 min and it might take more time if the training is executed for n-fold. The average time for predicting phase for one report is 2.28 min. Time statistics of BugSTAiR are given in Table 5. One time adaptation process of BugSTAiR takes 2–24 h according to the volume of the dataset and approximately 4.2 h in the experimented datasets. BugSTAiR significantly outperforms DNNLoc in runtime bug prediction.

5. Threats to validity

This section considers threats to validity. Three types of them are explained: threats to internal validity, threats to external validity, and threats to construct validity.

Threats to internal validity is biases that may be done by experimenters. In the proposed approach, the same datasets as BugLocator and BLIA have been used. These are well-known datasets used to minimize threats to internal validity. The source files and change histories are downloaded from Git repositories of the projects. Afterwards, all extended properties such as fixed files and commit logs are verified for each dataset.

Threats to external validity is about the generalizability of the results. Most BL tools only work for well-known open-source datasets. Our approach is tested on six datasets of different sizes, different domains, and different languages. One of these datasets is from an internally developed project for commercial use. Therefore, our approach is generalizable to any other open-source or commercial project with different languages. A potential threat to validity is the quality of bug reports. Bug reports contain a lot of crucial information about the issue for developers to fix the bugs. If a bug report has misleading information or does not provide enough information, the accuracy of the BugSTAiR is adversely affected.

Threats to construct validity is about the qualification of the evaluation metrics. In our experiments, three evaluation metrics such as Top N rank, MAP, and MRR are used. These metrics have been widely used for BL benchmarks and are well-known IR metrics. Therefore, it is obvious that our research has strong construct

Table 4. Experimental results of BugSTAiR.

Dataset	Approach	Top 1%	Top 5%	Top 10%	MAP	MRR
SWT	BLIA	67.3	86.7	89.8	0.65	0.75
	BLUIR+	56.1	76.5	87.8	0.58	0.66
	BRTracer+	46.9	79.6	88.8	0.53	0.60
	BugLocator	39.8	67.4	81.6	0.45	0.53
	BugSTAiR	70.41	80.61	83.67	0.69	0.74
	DNNLoc	35.2	69.0	80.3	0.45	0.37
	Locus	64.3	84.7	91.8	0.64	0.73
Eclipse	BLIA	N/A	N/A	N/A	N/A	N/A
	BLUIR+	32.9	56.2	65.4	0.33	0.44
	BRTracer+	32.6	55.9	65.2	0.33	0.43
	BugLocator	29.14	53.76	62.60	0.22	0.41
	BugSTAiR	39.52	53.06	58.09	0.43	0.46
	DNNLoc	45.8	70.5	78.2	0.51	0.41
	Locus	N/A	N/A	N/A	N/A	N/A
AspectJ	BLIA	41.5	71.1	80.6	0.39	0.55
	BLUIR+	33.9	52.4	61.5	0.25	0.43
	BRTracer+	39.5	60.5	68.9	0.29	0.49
	BugLocator	30.8	51.1	59.4	0.22	0.41
	BugSTAiR	42.3	63.4	70.2	0.43	0.51
	DNNLoc	47.8	71.2	85.0	0.52	0.32
	Locus	25.0	56.6	63.9	0.32	0.38
Tomcat	BLIA	N/A	N/A	N/A	N/A	N/A
	BLUIR+	N/A	N/A	N/A	N/A	N/A
	BRTracer+	N/A	N/A	N/A	N/A	N/A
	BugLocator	N/A	N/A	N/A	N/A	N/A
	BugSTAiR	55.11	80.4	82.77	0.61	0.65
	DNNLoC	53.9	72.9	80.4	0.60	0.52
	Locus	53.9	77.7	81.9	0.57	0.64
Angular-translate	BugSTAiR	50.81	80.33	85.25	0.46	0.50
Web application	BugSTAiR	40.23	62.34	72.63	0.46	0.50

validity.

In previous BL tools, various combinations of control parameters have been used to find the best accuracy for each project. Every parameter has been defined according to the number of experiments for each dataset. In our approach, the AAW process is proposed to optimize the control parameters. All the parameters are automatically selected via GA. Therefore, there are no heuristic or experimental tests in our approach.

6. Conclusion and future work

In this study, the proposed approach uses IR and GA on both JS-based web applications and Java-based back-end applications. To the best of our knowledge, it is the first BL work that works for JS-based web applications

Table 5. Time statistics of BugSTAiR.

Dataset	# of source code files	Source code indexing time (s)	# of commit history items	History indexing time (s)	Avg query retrieval time (s)
SWT	738	18.4	33.994	618.8	0.72
Eclipse	12.302	123.6	37.687	11544.5	2.07
AspectJ	3.692	27.5	8.291	286.8	1.7
Web application	724	2.5	2.543	35.6	0.38
Tomcat	2.485	27.6	62.783	382.5	0.35
Angular-translate	48	1	1.272	6.751	0.13

by using IR and ML. Lack of the BL dataset of open-source web applications has made us to use one of our commercial web applications. Its bug report dataset is used to experiment the proposed approach. In addition, an open-source angular-translate dataset is used to compare the results. Thus, there is not any more results to compare the success of the proposed approach in this area. On the other hand, the results of experiments show that BugSTAiR has promising performance on Java-based applications, so BugSTAiR outperforms any other BL tool. An overall comparison of BugSTAiR with other tools has been presented in Table 4. The results indicate that BugSTAiR has better performance than all of IRBL-based approaches such as BLUIR+, BLIA, and Locus. In addition, BugSTAiR performs better than DNNLoc which is a hybrid solution with IR and DNN in some datasets. The proposed tool localized 20% bugs in Eclipse, 4.6% bugs in SWT, and 2% bugs in AspectJ. Besides, our system has better performance (on the MAP metric) than any other tool with all datasets. MAP metrics of BugSTAiR are 6.1%, 10%, and 30% higher compared to BLIA and BLUIR+. In addition to this, BugSTAiR is the first generic BL tool that has the AAW process, which is the most valuable contribution to the state of the art on BL. The adaptation step prevents numerous manual experiments to reach optimum weights for all datasets. This generic implementation of the BL process provides us to enlarge our datasets easily. In the future, we would like to integrate image processing features to handle screenshots that are taken when the error occurred. As it is possible to extract more valuable text information from images to localize bugs, this feature will help us to improve localization accuracy in two ways. First, we can localize only JS files in web applications. Second, it will be possible to localize UI-related bugs more accurately with the help of this feature. In addition to this, we would like to integrate the proposed model with ML algorithms such as clustering. It is possible for a bug to be related with another bug which was fixed before. Therefore, clusters which are created according to the relevance of textual similarity between bug reports can help to improve the accuracy of IR results. Finally, configurable software systems and microservice architectures will be preferred in both front-end and back-end in the future. If there are any open-source projects like this, we will test the BugSTAiR and share experimental results.

Acknowledgement

Funding for this work was partially supported by the Research and Development Center of Commencis Technology accredited on Turkey - Ministry of Industry and Technology. The modules and services of this work were parts of a Commencis project named BugStair, which was funded by governments including the Scientific and Technological Research Council of Turkey (TÜBİTAK) having Award No. 3180803.

References

[1] Banker RD, Datar SM, Kemerer CF. Software complexity and maintenance costs. Communications of the ACM 1993; 36 (11): 81-94. doi: 10.1145/163359.163375

[2] Sousa MJC, Moreira HM. A survey on the software maintenance process. In: International Conference on Software Maintenance; Bethesda, MD, USA; Nov 1998, pp. 265-274.

[3] Puranik GM. Design of bug tracking system. International Journal of Innovative Research in Science, Engineering and Technology 2014; 3 (7): 14693-14696.

[4] Selby RW. Enabling reuse-based software development of large-scale systems. IEEE Trans. Software Engineering; 2005. 31(6): 495-510.

[5] Saha RK, Lawall J, Khurshid S. On the effectiveness of information retrieval based bug localization for c programmes. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14); Victoria, BC, Canada; 2014. pp. 161-170.

[6] Pathak DP, Dharavath S. A Survey paper for bug localization. International Journal of Science and Research 2014; 3(11): 2835-2838.

[7] Moreno L, Treadway JJ, Marcus A. On the use of stack traces to improve text retrieval-based bug localization. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14); Victoria, BC, Canada; 2014. pp. 151-160.

[8] Wang S, Lo D, Lawall J. Compositional vector space models for improved bug localization. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14); Victoria, BC, Canada; 2014. pp. 171-180.

[9] Kim D, Tao Y, Kim S. Where should we fix this bug a two-phase recommendation model, IEEE Transactions on Software Engineering 2013; 39(11): 1597-1610.

[10] Youm KC, Ahn J, Eunseok L. Improved bug localization based on code change histories and bug reports. Information and Software Technology 2017; pp. 177-192.

[11] Ricardo BY, Berthier RN. Modern Information Retrieval. USA: ACM Press, 1999.

[12] Rao S, Kak A. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In: Proceedings of the 15th IEEE International Conference on Program Comprehension; New York, NY, USA; 2011. pp. 37-48.

[13] Poshyvanyk D, Gueheneuc YG, Marcus A. Combining probabilistic ranking and latent semantic indexing for feature identification. International Conference on Program Comprehension(ICPC); Greece, Athens; 2006. pp. 137-146.

[14] Poshyvanyk D, Gueheneuc YG, Marcus A. Feature location using probabilistic ranking methods based on execution scenarios and information retrieval. Transactions on Software Engineering 2007; 33: 320-342

[15] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 34th International Conference on Software Engineering (ICSE) 2012; Zurich, Switzerland; pp. 14–24.

[16] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: 28th International Conference on Automated Software Engineering (ASE) 2013; California, USA; pp. 345-355.

[17] Mustafa E, Semih U. Bug localization by using information retrieval and machine learning algorithms. In: Proceedings of the 1st International Conference and Advanced Technologies, Computer Engineering and Science; Karabük, Turkey; 2018. pp. 298-602

[18] Ming W, Rongxin W, Shing-Chi C. Locus: locating bugs from software changes. In: Proceedings of the 31th IEEE/ACM International Conference on Automated Software Engineering 2016; Singapore, Singapore; pp. 262-273.

[19] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proceedings of the 26th International Conference on Program Comprehension; Buenos Aires, Argentina; 2017. pp. 218-229.

[20] Xiao Y, Keung J, Bennin KE, Qing M. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology; Elsevier, 2019. pp. 17-29.

[21] Nguyen AT, Nguyen TT, Al-Kofahi J. A topic-based approach for narrowing the search space of buggy files from a bug report. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering; Washington, USA; 2011. pp. 263-272.

[22] Raulji JK, Jatinderkumar RS. Stop-word removal algorithm and its implementation for Sanskrit Language. International Journal of Computer Applications (0975-8887) 2016; 150(2): 15-17.

[23] Goldberg DE. Genetic Algorithms in Search Optimization and Machine Learning. Boston, MA, USA: Addison–Wesley Longman Publishing Co, Inc, 1989

[24] Du H, Wang Z, Zhan W. Elitism and distance strategy for selection of evolutionary algorithms. IEEE Access Digital Object Identifier; 2018 (6): 44531-44541.

[25] Thisted RA. Elements of Statistical Computing: Numerical Computation. New York, USA; Routledge, 1988.

[26] Ersahin B, Aktaş O, Kılınç D, Erşahin M. A hybrid sentiment analysis method for Turkish. Turkish Journal of Electrical Engineering & Computer Science; 2019. 27 (3): 1780-1793. doi: 10.3906/elk-1808-189

[27] Schroter A, Bettenburg N, Premraj R. Do stack traces help developers fix bugs? Mining Software Repositories (MSR), 7th IEEE Working conference; Cape Town, South Africa; May, 2010. pp. 118-121.

[28] Kılınc D, Yucalar F, Borandag E. Multi-level reranking approach for bug localization, Expert Systems; 2016, 33 (3): 286-294.

[29] Voorhees EM, Harman DK. Chapter appendix: common evaluation measures. In: The Eleventh Text Retrieval Conference (TREC 2002), National Institute for Standards and Technology.