Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Improving software maintenance with improved bug triaging

Chetna Gupta[a,*], Pedro R.M. Inácio[a,b], Mário M Freire[a,b]

[a] *Universidade da Beira Interior, Portugal*
[b] *Instituto de Telecomunicações, Portugal*

ARTICLE INFO

Article history:
Received 10 May 2021
Revised 11 September 2021
Accepted 3 October 2021
Available online 2 November 2021

Keywords:
Automation process
Information processing
Information retrieval
Bug assignment problem
Heuristic algorithm
Fuzzy logic
Open-source software system

ABSTRACT

Bug triaging is a critical and time-consuming activity of software maintenance. This paper aims to present an automated heuristic approach combined with fuzzy multi-criteria decision-making for bug triaging. To date, studies lack consideration of multi-criteria inputs to gather decisive and explicit knowledge of bug reports. The proposed approach builds a bug priority queue using the multi-criteria fuzzy Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method and combines it with Bacterial Foraging Optimization Algorithm (BFOA) and Bar Systems (BAR) optimization to select developers. A relative threshold value is computed and categorization of developers is performed using hybrid optimization techniques to make a distinction between active, inactive, or new developers for bug allocation. The harmonic mean of precision, recall, f-measure, and accuracy obtained is 92.05%, 89.21%, 85.09%, and 93.11% respectively. This indicates increased overall accuracy of 90%±2% when compared with existing approaches. Overall, it is a novel solution to improve the bug assignment process which utilizes intuitive judgment of triagers using fuzzy multi-criteria decision making and is capable of making a distinction between active, inactive, and new developers based on their relative workload categorization.

Contents

1. Introduction

A constant dilemma faced by bug triagers is to decide who among the available pool of developers is most appropriate to handle reported bugs?. Effective bug assignment is a complex multi-criteria decision-making process requiring significant information of their features such as summary, description, assignee, priority, status, comments, etc. (Zhang et al., 2017). Ideally, the bug resolution process should be time-effective and quick in responding to

* Corresponding author.
E-mail address: chetna.gupta@ubi.pt (C. Gupta).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2021.10.011
1319-1578/© 2021 The Authors. Published by Elsevier B.V. on behalf of King Saud University.

the assignment of newly reported bugs (based on their priority) to the most appropriate and the least occupied developer. However, in reality with the advent of issue tracker systems, it's not strange to notice tens of thousands of bugs reporting in a day for fixing. In the last few decades, with the emergence of ubiquitous computing, large open-source software such as OpenStack[1] is becoming progressively important to manage large pools of computing, storage, and networking resources. Additionally, Mozilla[2], Eclipse[3], Jira[4] are other well-known platforms for issue reporting.

Researchers have proposed an array of techniques to automate the process of bug assignment. The majority of these techniques are based on machine learning, social network analysis, graph theory, fuzzy logic, information retrieval, automatic text summarization, and severity/priority prediction, etc.

length, and time required for bug resolution using three experimental design parameters: evaluation metrics, the definition of assignees, and the community of developers' consideration. Approaches based on machine learning match the new bug report closest to the characteristics with a set of bug reports fixed by a developer for a recommendation. Information retrieval techniques use feature vectors to represent textual information of bug reports which is later processed for potential developer selection. The basic idea is to allocate a bug to that developer having comparative skill towards a specific kind of bug by considering the developers' history. (Sajedi-Badashian and Stroulia, 2020) also reports a lack of studies considering fuzzy-based solutions. Available approaches (Tamrawi et al., 2011; Shokripour et al., 2015) computes a score for each pair of "*developer technical terms*" and do not explicitly consider *meta*-features supporting multi-criteria for bug assignment. These approaches use fuzzy logic with similarity measures to classify bug reports into bugs and non-bugs. In reality, not all features are of equal importance and hence an effective bug triaging approach should explicitly weigh available features.

The proposed solution considers this aspect to fill one of the gaps of existing literature. Overall, this paper proposes an approach to resolve the problem of automating bugs assignment. The proposed solution combines multi-criteria fuzzy TOPSIS method (Chen and Hwang, 1992)and hybrid heuristic algorithms-BFOA (Passino, 2002)with the BAR (Del Acebo, 2008) system optimization method to predict and allocate bugs to appropriate developers. BAR optimization is a swarm intelligence-based approach for task scheduling depicting the social behavior of bartenders in a highly dynamic, asynchronous, and time-critical environment. It will help select developers satisfying the constraint of their current workload and their availability to handle the extra load which to the best of our knowledge is lacking in existing studies. This directly links to one of the major concerns of the identification of active, non-active, and new developers. Active developers are the ones who frequently and actively participate in the bug triaging process whereas inactive developers are those who don't. Furthermore, BFOA has shown competitive performance with other well-known evolutionary algorithms such as differential evolution and genetic algorithms. The hybrid nature of BFOA and BAR optimization will provide a good framework to address the challenge of developing a new adaptive and robust solution. This paper addresses these gaps and proposes the following main contributions:

An integrated solution to allocate bugs to appropriate developers utilizing (i) the intuitive judgment of triagers using fuzzy multi-criteria decision making, (ii) handling constraint of developers availability, and current workload status to make a distinction between active, inactive, and new developers. It computes relative threshold value and categorizes developers using hybrid optimization techniques to make a distinction between active, inactive, or new developers for bug allocation. The performance comparison results with existing baseline approaches conclude that a combination of fuzzy logic multi-criteria and heuristic algorithms for bug assignment becomes a worthy way to minimize the triagers workload and fixing time.

[1] https://www.openstack.org

[2] https://bugzilla.mozilla.org

[3] https://bugs.eclipse.org

[4] https://www.atlassian.com/software/jira

2. Related work

In the past array of bug triaging approaches are proposed which are broadly categorized into the following types namely, machine learning, information retrieval, auction-based, social network, tossing graphs, fuzzy sets, and operational research-based techniques (Sajedi-Badashian and Stroulia, 2020). Approaches based on machine learning (Guo et al., 2020; Bhattacharya et al., 2012; Karim, 2019; Yang et al., 2014a; Govindasamy et al., 2016; Jonsson et al., 2016; Tian et al., 2015; Sharma et al., 2012; Mani et al., 2019; Xia et al., 2017; Johnson and Zhang, 2016; Deng et al., 2019; Guo et al., 2019; Naguib et al., 2013; Xie et al., 2012)-match the new bug report closest to the characteristics with a set of bug reports fixed by a developer for the recommendation. Some existing work has tried to match the expertise of developer's profiles to the set of characteristic features to recommend the developers matching their profile. The biggest challenge faced in these approaches is - how to label bug reports with insufficient or missing label information. Different dimensions such as choice of decision classifier, feature selection, the inclusion of tossing graphs, and incremental learning influences bug triaging efficiency were considered. In addition to this usage of textual description, component, operating system, hardware, version, the developer who owns the code, current workload of developers, and developers effectively taking part in the project were also explored for efficient bug allocations.

Another set of approaches explored by researchers are techniques based on information retrieval to automate the process of bug assignment (Xia et al., 2017; Zamani et al., 2014; Kaur and Jindal, 2019; Gujral et al., 2015; Xuan et al., 2015; Bhattacharya and Neamtiu, 2010). In these approaches, bug reports are considered as documents that are transformed. These approaches use feature vectors to represent textual information of bug reports which is later processed for potential developer selection. The basic idea is to allocate a bug to that developer having comparative skill towards a specific kind of bug by considering the developers' history. Guo et al. (Guo et al., 2020)used Word2vec a natural language processing for bug summary and CNN for implementation. A few researchers have used additional information such as components, products, severity, priority to improve the accuracy of bug assignment (Xia et al., 2017; Somasundaram and Murphy, 2012; Deng et al., 2017; Guo et al., 2018; Zhao et al., 2019). The work conducted in (Zhao et al., 2019)recommends the suitable developer using the topic model Latent Dirichlet Allocation (LDA) method. They compute the similarity of the bug reports using the LDA method and later combine it with multiple attribute information to filter out inconsistent bug reports. Xia et al. (Xia et al., 2017) proposed a model with additional supervision information to the LDA to obtain a closer supervised subject distribution. However, these approaches suffer from concerns of labeling of bug reports with insufficient or missing or duplicate label information which leads to loss of context information. Another concern is the cost related to the execution of machine learning or information retrieval methods. Most of the existing work uses the topic model and classification algorithms which are costly when the size of the data is large.

Most of the previous work lack in addressing the involvement of the developer's activities that is a serious concern in the bug assignment process. This is due to fact that there are a lot many developers and there is no way to identify who is available or has left the job or has some concern related to skill or potential to specific bug resolution. In this direction, Xuan (Xuan et al., 2017)proposed a method for semi-supervised text categorization to recommend developers to mark inaccurate circumstances of the developer in the current bug report data. Their approach uses the naive Bayesian approach with the expectation-maximization approach.

The work conducted in (Xuan et al., 2012) analyses developer information using social networking techniques to prioritize developers. They analyzed three major influential factors: product characteristics, time variation, and noise tolerance to assign bugs based on the priority of the developer. In social network-based approaches, a developer's social network is referred to as a network depicting social interactions and personal relationships among software developers. The expertise of the developer is computed on various influencing factors of the network that utilizes the relationships among software developers and bug reports for the selection of potential developers (Zanetti et al., 2013; Wu et al., 2011; Zhang and Lee, 2012; Yang et al., 2014b; Zhang et al., 2013). Another set of approaches reported in the literature are based on tossing graph-based approaches (Bhattacharya et al., 2012; Bhattacharya and Neamtiu, 2010; Chen et al., 2011; Jeong et al., 2009). In these approaches, the tossing paths of previously fixed bug reports are considered. Jeong et al. (Jeong et al., 2009)used the transfer graph to describe the bugs that the current developer could not fix and the information that the bug report passed to other developers to optimize the accuracy of the bug report assignment. Bhattacharya et al. (Bhattacharya and Neamtiu, 2010)presented an improved assignment accuracy method based on (Sajedi-Badashian and Stroulia, 2020) by analyzing the transfer graphs combined with various attributes. Reports state that bug tossing is one of the major problems in bug triaging constituting roughly 93% of bug reports which get tossed at least once in their lifetime. Some of the researchers have used fuzzy sets-based approaches (Tamrawi et al., 2011; Shokripour et al., 2015)to compute the membership score of developers concerning various topics obtained from bug parameters. These approaches use fuzzy logic with similarity measures to classify bug reports into bugs and non-bugs. Tamrawi et al. (Tamrawi et al., 2011)proposed a method based on fuzzy sets and developer caching for bug distribution.

From the array of approaches listed in the literature, it can be concluded that existing approaches do not consider the ranking of bugs or developers using *meta*-features supporting multi-criteria decision-making for assigning a bug to the most potential developer. In reality, not all features/parameters are of equal importance and hence an effective bug triaging approach should explicitly weight parameters and their prioritization. In addition to this existing studies do not consider the overall developer's capability/expertise concerning critical factors while assigning bugs. Also, literature lack studies considering the developer's availability which is directly linked to one of the major concerns of identification of active and non-active developers. Hence, this paper addresses these gaps and approaches a hybrid bug triaging approach.

3. Proposed approach

The proposed solution has two aspects involving bug assignment and adequate developer identification. Fig. 1 sketches the proposed bug assignment process. The process starts with the extraction of features such as bug id, summary, bug reporter, comments, components, priority, severity, and time stamp from bug reports along with their resolution status (labels) as RESOLVED, FIXED, CLOSED, and VERIFIED.

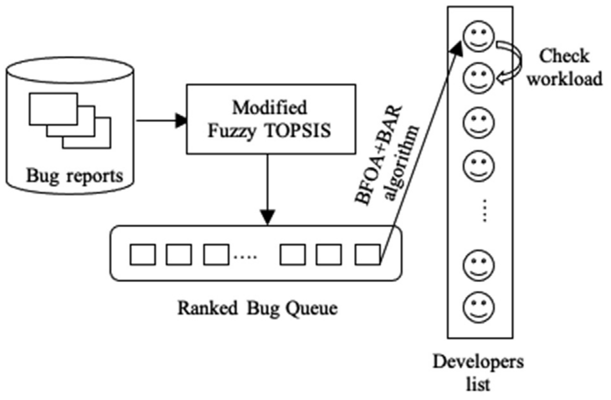

Fig. 1. The proposed bug allocation process.

The pre-processing of collected data is performed and all stop words and noise from the dataset are removed. Next, a bug priority metric is built to generate the bug priority queue using a fuzzy multi-criteria TOPSIS method. It is a powerful method that leverages the benefits of uncertain and vague human decisions. Two features namely, *priority and severity* are extracted from the bug report to assign a rank to each bug. Severity is considered a serious indicator of bug urgency. Severity[5] can be one of the following types: blocker, critical, and major (indicates serious errors and crashes), normal, minor, and trivial (indicates surface errors). Thousands of reported bug reports show that a developer not necessarily treats a bug of critical or blocker severity level at a high priority i.e., P1. Hence, there is no one-to-one mapping between severity and priority levels. Different bug tracking systems have different bug severity levels. Based on the levels of severity, weights will be assigned. For example, in Jira[4] there are only three severity levels namely, critical, major and minor. Hence, the severity weights (S_w) will be 3 for critical, 2 for major and 1 for minor severity levels. Similarly, the severity weight (S_w) will be assigned for different bug severity levels in different bug triaging systems. After generating the bug priority queue, a metric for the developer's availability and current workload is computed for final allocation using the hybrid optimization techniques of BFOA and BAR.

The following steps are followed to rank bug reports and later arrange them into the bug priority queue. The algorithm (refer to pseudo-code 1) takes bug report criteria values and their weights as input. Then it fetches details such as severity level, priority level from the bug report and assigns weight to each parameter. Thereafter, the fuzzy TOPSIS method is used to generate a ranked bug list (steps 1 to 9). Fuzzy TOPSIS is powerful algorithm used to generate selection of the best alternatives for aggregate scores. The underlying principle of this technique rest on the concept that the chosen alternative should have the shortest distance to Positive Ideal Solution (PIS) and the farthest distance to Negative Ideal Solution (NIS). Furthermore, the criteria's importance weights and the ratings of qualitative criteria are taken as linguistic variables. In this paper, the decision makers use the linguistic variables to evaluate the importance of the criteria is considered by decision makers and the ratings of alternatives with respect to various criteria. The linguistic inputs are first mapped to fuzzy values as follows:

[5] https://bugzilla.mozilla.org

- Linguistic Value = Very Low, 5-point scale = 1, Fuzzy Number = 1,1,3.
- Linguistic Value = Low, 5-point scale = 2, Fuzzy Number = 1,3,5.
- Linguistic Value = Average, 5-point scale = 3, Fuzzy Number = 3,5,7.
- Linguistic Value = High, 5-point scale = 4, Fuzzy Number = 5,7,9.
- Linguistic Value = Very High, 5-point scale = 5, Fuzzy Number = 7,9,9.

A triangular function is used as a membership function. With the help of these values, the Fuzzy TOPSIS method will be employed on the inputs to compute and generate bug priority values. A fuzzy decision matrix using the Eq. (1).

$$a_{ij} = max_k\left\{a_{ij}^k\right\}, b_{ij} = \frac{1}{K}\sum_{k=1}^{K} b_{ij}^k, c_{ij} = max_k\left\{c_{ij}^k\right\} \tag{1}$$

where *(a,b,c)* represents the fuzzy values, *k* defines the total number of bugs, *i and j* represent the *i^th^* bug and *j^th^* criteria, respectively.

$$r'_{ij} = \left(\frac{a_{ij}}{c'_j}, \frac{b_{ij}}{c'_j}, \frac{c_{ij}}{c'_j}\right) and$$

$$c'_j = \max\left\{c_{ij}\right\}(prioritycriteria) \tag{2}$$

$$r''_{ij} = \left(\frac{a''_j}{c_{ij}}, \frac{a''_j}{b_{ij}}, \frac{a''_j}{c_{ij}}\right) and$$

$$a''_j = \max\left\{a_{ij}\right\}(severitycriteria) \tag{3}$$

The criteria can be classified into priority criteria or severity criteria. Next, fuzzy positive and fuzzy negative ideal solutions (FPIS and FNIS) are computed using the Eqs. (4) and (5).

$$A' = (\vartheta'_1 \quad , \ \vartheta'_2 \ldots\ldots\ldots.\vartheta'_n \quad) where \ \vartheta'_j = \max\left\{\vartheta_{ij3}\right\} \tag{4}$$

$$A'' = (\vartheta'', \ \vartheta'' \ldots\ldots\ldots.\vartheta'') where \ \vartheta'' = \max\left\{\vartheta_{ij1}\right\} \tag{5}$$

Here, *A'* represents the fuzzy positive ideal solution and *A*" represents the fuzzy negative ideal solution where ϑ' and ϑ'' defines the elements of the weighted normalized matrix. Next, the distance between two fuzzy numbers, fuzzy elements of the combined decision matrix, and the fuzzy positive and fuzzy negative ideal solutions are computed. These distances are further used to calculate the weight of each requirement, defining the rank of each requirement. This weight is known as the *closeness coefficient.* The closeness coefficient's highest value represents the first rank, while the lowest represents the last rank.

$$cc_i = \frac{d_i''}{d_i' + d_i''} \tag{6}$$

where, d_i'' and d_i' represents the distance between the total value of the distance between fuzzy elements and FNIS and fuzzy elements and FPIS, respectively. The steps are summarized in Pseudo-code-1 below. In the next stage, a metric for the developer's availability and current workload for final allocation is computed. The estimate of developers' current workload is performed first.

Pseudo-code 1: Modified fuzzy TOPSIS

Input: Bug reports criteria values a_{ij} such that B_n ={$b_1(b_p, b_s)$, $b_2(b_p, b_s)$,... $b_n(b_p, b_s)$}

Pseudo-code 1: Modified fuzzy TOPSIS

Input: Bug reports criteria values a_{ij} such that B_n ={$b_1(b_p, b_s)$, $b_2(b_p, b_s)$,... $b_n(b_p, b_s)$}

Weight of each criterion w_j

Bug Severity level	Severity Weight (S_w)	Bug priority level (P_w)	Priority weight	Bug report Factor (R_f)
Blocker	7	P1	5	1
Critical	6	P2	4	1
Major	5	P3	3	1
Normal	4	P4	2	2
Trivial	3	P5	1	3
Minor	2	--	--	3
Enhancement	1	--	--	3

Output: Best R (Ranked bug) where R is the best tradeoff solution

- Create a decision matrix *D*
- Normalize decision matrix
- Compute weights w_j for each criteria a_{ij}
- Find fuzzy decision matrix *D*∼ and weights *W*∼
- Compute weighted fuzzy decision matrix *V*∼ =*D*∼**W*∼
- Calculate the positive ideal (A^+) and negative ideal (A^-) solution.
- Calculate separation and closeness between each position
- Compute the relative closeness to the ideal solution.
- Rank bug reports according to rank values and creates a queue.

R_f factors represent the complexity of the reports fetched from the bug report. Where, R_f can have three values mapped to severity levels, R_f = 1 represents critical, blocker and major severity level, R_f = 2 represents normal severity level, and R_f = 3 represents trivial, minor and enhancement severity level. In the second step, a metric for the developer's availability and current workload for final allocation is computed. Three attributes of the developer's data are extracted from bug reports: *current workload, average fixing time, and frequency of assignment to compute workload.* Equation (7) is used to compute the workload of each filtered developer.

Steps for workload calculation

1. Generate a list of developers and the number of bugs assigned to each developer in past.
2. For each developer filter the results according to the following attributes:
   a. *current*: the number of currently assigned active bug reports.
   b. *average*-fixing-time: relative time taken to resolve the bug.
   c. *Frequency*: it refers to how often bug reports are assigned to a developer. It is the time frame to be considered for filtering results. In this study, we have considered it for the last three years.
3. Calculate workload on each developer using formula given below:

*workload = current*average-fixing-time*frequency* (7)

Finally, the assignment of bugs to adequate developers with the relative threshold value is computed using the hybrid optimization techniques of BFOA and BAR optimizations (refer to pseudo-code 2). is a population-based numerical optimization algorithm presented by Passino (Bhattacharya et al., 2012). BFO is a simple yet powerful recently introduced optimization algorithm that depicts the foraging behavior of motile bacteria such as *E. coli* and *salmonella* that propels themselves by rotation of the flagella. It has successfully solved many engineering problems and many BFO variants have been developed to improve its optimization performance. In this paper, the BFO's performance is improved by integrating it with BAR optimization algorithm. It is a very simple algorithms that are loosely inspired in the behavior a staff of bartenders showing the pattern of serving drinks to a crowd of customers in a bar or pub. BAR algorithm can be easily used to solve complex NP-hard scheduling problem, and other generalized optimization problems. The BAR algorithm can produce much better results in comparison to other optimization algorithms such as greedy algorithms in the "*nearest neighbor*" style. With the integration of BFO and BAR, the proposed algorithm can efficiently create a balance between the exploration and the exploitation of the search space during the bacteria evolution, with the significant improvement in performance. The algorithm collects bug information of all bugs from the bug priority queue and a counter continuously keeps track of the workload status of each developer using (7). Relative threshold allows bug triagers to interpret the workload results based on each developer's performance from the given pool of developers (step 3 and step 4 of pseudo-code 2). Bug reports having an R_f factor = 3 does not mean they are not complex (R_f = complexity of bug reports). They may take a little longer time to be fixed, and hence a new developer can be given those bug reports.

```
Pseudo-code 2: Modified BFOA+BAR optimization algorithm
1. BFOA first determines the number of bugs for allocation.
2. Next, it refers to the bug priority queue computed in the
   previous step for allocation.
3. The allocation is performed according to the current work-
   load status computed using (7). The following steps are
   followed:
   a. Initialize the population
   b. Evaluate the individuals using the following three loops
      of optimization
Inner loop: for the chemotactic event;
Middle loop: for reproduction event;
Outer loop: for elimination–dispersal event
   c. Decode the optimal individual to obtain the first level
      solution which will be taken as input in the BAR
      algorithm.
4. To make the allocation adaptive and robust BAR optimiza-
   tion is used to obtain the final solution. The following steps
   are followed:
   a. Categories workload in following classes and compute
      threshold relatively as follows:
   b. Calculate the mean (average) score and standard devia-
      tion of workload scores of all developers in the pool.
   c. Set average as a mid-range value pointing to Middle
      (A3) from the following list. Separate the next levels
      using the standard deviation value. Adding standard
      deviation to A3 will set the limit for A2 and further add-
      ing it to A2 will set the limit to the Top 10% i.e. A1. Sim-
      ilarly, subtraction of standard deviation value from A3
      will set the limit for A4 and further subtraction will
      result in A5.
A1= Top 10%
A2= Next 25%
A3= Middle 30%
A4= Next 25%
A5= Bottom 10%
   d. Assign workload according to one of the following
      conditions:
   i. if(workload = = A1)
don't assign→check condition for next developer //satisfying
  one of the condition (4d (i to v)).
   ii. if(workload = = A2)
assign a bug report to d where d∈D having R_f = 1
   iii. if(workload = = A3)
assign a bug report to d where d∈D having R_f = 2
   iv. if(workload = =A4)
assign a bug report to d where d∈D having R_f = 3
   v. if(workload = = A5)
case I: don't assign →inactive developer
case II: new developer (workload value will be 0)
assign a bug report to d where d∈D having R_f = 3
```

4. Empirical validation

To validate the proposed approach we have compared our results with a total of 11 datasets. Out of these 11, a total of 6 datasets are used (as is) in Badashian et al. (Sajedi-Badashian and Stroulia, 2020). These datasets are obtained from various contributors of users in Github[6]. Details of these 6 projects are as follows:

- http://github.com/yui/yui3[7]
- http://github.com/julialang/Julia[8]
- http://github.com/elastic/elasticsearch[9]
- http://github.com/saltstack/salt[10]
- http://github.com/Angular.js/Angular.js[11]
- http://github.com/Rails/Rails[12]

Additionally, 5 open-source projects namely, OpenStack, Mozilla, Eclipse, NetBeans, and Jira are used to test our results with other existing approaches. Literature reports that existing studies use different assumptions, settings and hence the result performance of their techniques reported vary a lot on different metrics. This makes it difficult to compare the results of new approaches against existing approaches (Sajedi-Badashian and Stroulia, 2020). From the literature, we have selected 12 approaches (Tamrawi et al., 2011; Guo et al., 2020; Sajedi-Badashian and Stroulia, 2020; Xuan et al., 2015; Bhattacharya and Neamtiu, 2010; Xuan et al., 2012; Anvik and Murphy, 2011; Shokripour et al., 2013; Wang et al., 2014; Badashian et al., 2015; Zhang et al., 2016) which are used for comparison and *meta*-analysis. These approaches have used large datasets and did not have majorly filtered their data. Results are also compared with Badashian et al. (Sajedi-Badashian and Stroulia, 2020) because is the most suitable approach and it also uses multi-source evidence gathered from bug reports for bugs assignment. The results are summarized in Table 1; Table 2, and Table 3.

For 5 open source project datasets, we have performed a theoretical comparison of our results with their published results. Since, results in existing approaches vary on different metrics such as accuracy, precision, recall, f-measure, mean reciprocal rank (MRR), and mean average precision (MAP). The MAP a single-figure measure can be used independently unlike precision and recall which are interpreted together and only considers high-rank values. In this paper, a comparison of performance is computed on all these metrics including MAP making it is feasible to compare proposed approach results with other approaches (with one or more metrics, whenever available).

Although the pair-wise comparison between the proposed approach and other approaches is not possible for all these approaches since the results do not provide average results for all bug reports chosen for the projects. The following performance metrics were used in this evaluation:

[6] https://github.com

Table 1
Result analysis of computation of bug ranks @k accuracy.

	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8	Top 9	Top 10
Precision	92.14%	91.32%	91.12%	88.25%	82.61%	79.59%	80.17%	80.13%	78.59%	77.13%
Recall	75.68%	74.98%	74.13%	73.12%	73.17%	74.98%	77.25%	81.77%	80.11%	82.69%
f-measure	82.12%	83.34%	79.14%	79.74%	76.41%	79.81%	76.12%	79.66%	81.91%	81.16%

Table 2a
Result of analysis of comparison with Badashian et al. (Sajedi-Badashian and Stroulia, 2020)

Projects	Mean reciprocal rank (MRR)		Mean average precision (MAP)	
	Badashian et al. (Sajedi-Badashian and Stroulia, 2020)	Proposed	Badashian et al. (Sajedi-Badashian and Stroulia, 2020)	Proposed
Project 1[7]	0.64	0.66	61.23	63.56
Project 2[8]	0.61	0.64	58.70	61.04
Project 3[9]	0.65	0.65	62.48	61.11
Project 4[10]	0.60	0.63	56.94	56.95
Project 5[11]	0.63	0.63	60.92	61.46
Project 6[12]	0.58	0.60	55.21	56.56

Table 2b
Result of analysis of comparison with other approaches.

Existing Approach	Dataset	Performance
Bhattacharya et al., 2010 (Bhattacharya and Neamtiu, 2010)	Eclipse, Mozilla	Accuracy 77.43% (Eclipse), 77.87% (Mozilla)
(Tamrawi et al., 2011; Tamrawi et al., 2011; Tamrawi et al., 2011)	Eclipse	Accuracy 92.99%
Anvik et al., 2011 (Anvik and Murphy, 2011)	Eclipse, Firefox	Precision 60%, recall 3% (Eclipse), Precision 51%, recall 24% (Firefox)
Xuan et. Al.,2012 (Xuan et al., 2012)	Eclipse, Mozilla	Accuracy 53.10% (Eclipse), 56.98% (Mozilla)
(Shokripour et al., 2013; Shokripour et al., 2013; Shokripour et al., 2013)	JDT-Debug, Firefox	Accuracy 89.41% (JDT-Debug), 59.76% (Firefox)
(Wang et al., 2014; Wang et al., 2014; Wang et al., 2014)	Eclipse, Mozilla	Accuracy 84.45% (Eclipse), 55.56% (Mozilla)
Xuan et. al., 2015 (Xuan et al., 2015)	Eclipse, Mozilla	Accuracy 60.40% (Eclipse), 46.46% (Mozilla)
Badashian et. al., 2015 (Badashian et al., 2015)	20 GitHub projects, 7144 bug reports	Accuracy 89.43% (GitHub projects)
(Jonsson et al., 2016; Zhang et al., 2016)	Industry	Accuracy 89%
(Zhang et al., 2016)	Eclipse, Mozilla, Ant, TomCat6	MRP 0.28 (Eclipse), 0.28 (Mozilla), 0.35 (Ant), 0.35 (TomCat6) MAP 56.42 (Eclipse), 44.49 (Mozilla), 36.48 (Ant), 36.54 (TomCat6)
Badashian et. al., 2020 (Sajedi-Badashian and Stroulia, 2020)	13 GitHub Projects	MRR 0.59 and MAP 56.83, Accuracy 80.40%
Guo et al. (Guo et al., 2020; Guo et al., 2020)	Mozilla	Accuracy 92.99%*
Proposed Approach	Mozilla, Eclipse, NetBeans, Jira, OpenStack	Mean Accuracy 93.11%, Accuracy 92.35% (Mozilla), 92.98% (Eclipse), 92.98% (NetBeans), 93.13% (Jira), 94.21% (OpenStack). Average precision 92.05%, Average Recall 89.21%, Average f-measure 85.09%.

a) **Accuracy**: is quantified as acceptability difference computed between existing and proposed approaches on datasets considered.
b) **Precision**: it refers to the fraction of relevant instances among the retrieved instances.
c) ***Recall***: it refers to the fraction of retrieved relevant instances among all relevant instances. Both precision and recall are used to understand the measure of relevance.
d) **f-measure**: it refers to the measure of a test's accuracy. It is calculated from precision and recall values. It is computed as follows: f-measure = (2*Precision*Recall)/(Precision + Recall)
e) **Mean reciprocal rank**: refers to the average of the reciprocal rank of the developer selected for allocation. Only the highest ranks are considered others are ignored.
f) **Mean average precision:** refers to the mean value of the average precision overall bug reports.
g) **Robustness:** is defined as the capability of the system to deal with erroneous inputs injected as inputs during execution to test the efficacy of results.

Furthermore, 3 research questions are investigated to validate the performance of results. Following are the research questions addressed:

RQ1. How effective is the fuzzy multi-criteria TOPSIS method?

- Table 1 presents the results of analysis of successful computation and ranking of bugs @k accuracy, where k = Top 1 to Top 10.

RQ2. How easy/difficult is it to compute workload, combine two inputs to run the hybrid BFOA and BAR algorithm in terms of precision, recall, f-score, and accuracy of results?

- Table 2(a and b) presents the analysis of results of comparison with other approaches in terms of precision, recall, f-measure, and accuracy.

RQ3. Whether the results of the presented approach can cope with errors?

- Table 3 presents the results of robustness.

It can be seen that good values of precision, recall, and f-measure on Top 1 to Top 10 list sizes are obtained with fuzzy multi-criteria TOPSIS method (refer to Table 1). Additionally, a comparative table for precision, recall, f-measure, an accuracy value of the proposed solution with selected 12 other approaches indicates that the bugs can be ranked successfully and a high triage accuracy can be achieved effectively with the proposed solution. Table 2(a) summarizes the results with 6 GitHub projects and it can be seen that the results of proposed approach are promising. Table 2(b) presents the summary of comparison results with other

Table 3
(p-values observed at different error rates injected in input to the algorithm with alpha = 0.05)

For error rate of	5%	10%	15%	20%
Bug-Rank-Error vs. bug rank accuracy	0.000011	0.00005	0.000027	0.00018

approaches. It can be seen that the proposed approach is effective with higher accuracy of mean accuracy of 93.11%. The overall system accuracy for all datasets and protocols is around 90%±2%. To answer RQ3, the erroneous inputs were injected into the algorithm with error rates of 5%, 10%, 15%, and 20% one at a time. This is done to test the efficacy of results during execution. The same experiment is executed 10 times to calculate the average to assess the performance. The *p-value* obtained is less than 0.05 with alpha as 0.05, indicating a robust solution capable of handling erroneous inputs. It can be concluded that the proposed algorithm performs effectively when compared to other studies in bug triaging.

5. Conclusion and future directions

In this paper, a novel automated fuzzy TOPSIS multi-criteria method is combined with BFOA and BAR algorithms for bug assignment. The fuzzy TOPSIS method is used to capture the intuitive judgment of triagers while making a concrete decision. The bug reports having resolution status (labels) as RESOLVED, FIXED, CLOSED, and VERIFIED are selected and pre-processed to remove all stop words and noise from the collected data. A bug priority metric is computed to generate the bug priority queue using a fuzzy multi-criteria TOPSIS method to ranking based on priority and severity of the bug in the bug report. A metric for the developer's availability and current workload is computed for final allocation using the hybrid optimization techniques of BFOA and BAR. The heuristic algorithms optimize the results to recommend adequate developers satisfying the constraint of their current workload using a relative threshold value. Experimentation results obtained on a total of 11 datasets confirm that the proposed solution can provide a trustworthy and robust solution with the harmonic mean of precision, recall, f-measure, and accuracy of 92.05%, 89.21%, 85.09%, and 93.11% respectively.

Overall, it is a novel solution that improves the bug assignment process utilizing intuitive judgment, relative workload, the distinction between active, inactive, and new developers to handle the new workload. The presented approach determines the capacity to undertake a new load and successfully determines inactive/new developers, but it does not distribute the workload evenly among available developers. We plan to address this concern as the extension of this work in the future.

Funding

This work was supported by Operação Centro-01-0145-FEDER-000019 – C4 – Centro de Competências em Cloud Computing, co-financed by the Programa Operacional Regional do Centro (CENTRO 2020), through the Sistema de Apoio à Investigação Científica e Tecnológica – Programas Integrados de IC&DT. This work was also funded by FCT/MCTES through national funds and, when applicable, co-funded by EU funds under the project UIDB/EEA/50008/2020.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Zhang, T., T., Chen, J., Jiang, H., Luo, X., & Xia, X., 2017. In: May). Bug report enrichment with the application of automated fixer recommendation. IEEE, pp. 230–240.
Sajedi-Badashian, Ali, Stroulia, Eleni, 2020. Guidelines for evaluating bug-assignment research. Journal of Software: Evolution and Process. 32 (9). https://doi.org/10.1002/smr.v32.910.1002/smr.2250.
Tamrawi, A., Nguyen, T.T., Al-Kofahi, J.M., Nguyen, T.N., 2011. September). Fuzzy set and cache-based approach for bug triaging. In: In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of software engineering*, pp. 365–375.
Guo, Shikai, Zhang, Xinyi, Yang, Xi, Chen, Rong, Guo, Chen, Li, Hui, Li, Tingting, 2020. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. Neural Processing Letters 51 (3), 2589–2606.
Shokripour, Ramin, Anvik, John, Kasirun, Zarinah M., Zamani, Sima, 2015. A time-based approach to automatic bug report assignment. Journal of Systems and Software 102, 109–122.
Chen, S.J., Hwang, C.L., 1992. Fuzzy multiple attribute decision-making methods. Fuzzy multiple attribute decision making, 289–486.
Passino, K.M., 2002. Biomimicry of bacterial foraging for distributed optimization and control. IEEE control systems magazine 22 (3), 52–67.
Del Acebo, E., & de-la Rosa, J. L. (2008, April). Introducing bar systems: a class of swarm intelligence optimization algorithms. In *AISB 2008 Convention Communication, Interaction and Social Intelligence* (Vol. 1, p. 18).
Sajedi-Badashian, A., Stroulia, E., 2020. Investigating the information value of different sources of evidence of developers' expertise for bug assignment in open-source projects. IET Software 14 (7), 748–758.
Bhattacharya, Pamela, Neamtiu, Iulian, Shelton, Christian R., 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. J. Syst. Softw. 85 (10), 2275–2292. https://doi.org/10.1016/j.jss.2012.04.053.
M. R. Karim, "Key features recommendation to improve bug reporting," 2019, doi: 10.1109/ICSSP.2019.00010.
G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," 2014, doi: 10.1109/COMPSAC.2014.16.
V. Govindasamy, V. Akila, G. Anjanadevi, H. Deepika, and G. Sivasankari, "Data reduction for bug triage using effective prediction of reduction order techniques," 2016, doi: 10.1109/ICCPEIC.2016.7557229.
Jonsson, Leif, Borg, Markus, Broman, David, Sandahl, Kristian, Eldh, Sigrid, Runeson, Per, 2016. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empir. Softw. Eng. 21 (4), 1533–1578. https://doi.org/10.1007/s10664-015-9401-9.
Tian, Yuan, Lo, David, Xia, Xin, Sun, Chengnian, 2015. Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. 20 (5), 1354–1383. https://doi.org/10.1007/s10664-014-9331-y.
M. Sharma, P. Bedi, K. K. Chaturvedi, and V. B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation," 2012, doi: 10.1109/ISDA.2012.6416595.
S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," 2019, doi: 10.1145/3297001.3297023.
Xia, Xin, Lo, David, Ding, Ying, Al-Kofahi, Jafar M., Nguyen, Tien N., Wang, Xinyu, 2017. Improving Automated Bug Triaging with Specialized Topic Model. IEEE Trans. Softw. Eng. 43 (3), 272–297. https://doi.org/10.1109/TSE.2016.2576454.
R. Johnson and T. Zhang, "Supervised and semi-supervised text categorization using LSTM for region embeddings," 2016.
Deng, Wu, Xu, Junjie, Zhao, Huimin, 2019. An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem. IEEE Access 7, 20281–20292. https://doi.org/10.1109/ACCESS.2019.2897580.
Guo, Shikai, Liu, Yaqing, Chen, Rong, Sun, Xiao, Wang, Xiangxin, 2019. Improved SMOTE Algorithm to Deal with Imbalanced Activity Classes in Smart Homes. Neural Process. Lett. 50 (2), 1503–1526. https://doi.org/10.1007/s11063-018-9940-3.
H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," 2013, doi: 10.1109/MSR.2013.6623999.
X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," 2012, doi: 10.1145/2365324.2365329.
Zamani, Sima, Lee, Sai Peck, Shokripour, Ramin, Anvik, John, 2014. A noun-based approach to feature location using time-aware term-weighting. Inf. Softw. Technol. 56 (8), 991–1011. https://doi.org/10.1016/j.infsof.2014.03.007.
Kaur, Arvinder, Jindal, Shubhra Goyal, 2019. Text analytics based severity prediction of software bugs for apache projects. Int. J. Syst. Assur. Eng. Manag. 10 (4), 765–782. https://doi.org/10.1007/s13198-019-00807-8.
S. Gujral, G. Sharma, S. Sharma, and Diksha, "Classifying bug severity using dictionary based approach," 2015, doi: 10.1109/ABLAZE.2015.7154933.

Xuan, Jifeng, Jiang, He, Hu, Yan, Ren, Zhilei, Zou, Weiqin, Luo, Zhongxuan, Wu, Xindong, 2015. Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. 27 (1), 264–280. https://doi.org/10.1109/TKDE.6910.1109/TKDE.2014.2324590.

P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," 2010, doi: 10.1109/ICSM.2010.5609736.

K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent Dirichlet allocation," 2012, doi: 10.1145/2134254.2134276.

Deng, Wu, Zhao, Huimin, Zou, Li, Li, Guangyu, Yang, Xinhua, Wu, Daqing, 2017. A novel collaborative optimization algorithm in solving complex optimization problems. Soft Comput. 21 (15), 4387–4398. https://doi.org/10.1007/s00500-016-2071-8.

Guo, Shikai, Chen, Rong, Wei, Miaomiao, Li, Hui, Liu, Yaqing, 2018. Ensemble data reduction techniques and Multi-RSMOTE via fuzzy integral for bug report classification. IEEE Access 6, 45934–45950. https://doi.org/10.1109/ACCESS.2018.2865780.

Zhao, Huimin, Zheng, Jianjie, Xu, Junjie, Deng, Wu, 2019. Fault Diagnosis Method Based on Principal Component Analysis and Broad Learning System. IEEE Access 7, 99263–99272. https://doi.org/10.1109/Access.628763910.1109/ACCESS.2019.2929094.

Xuan, J., Jiang, H., Ren, Z., Yan, J., Luo, Z., 2017. Automatic bug triage using semi-supervised text classification. arXiv.

J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," 2012, doi: 10.1109/ICSE.2012.6227209.

M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," 2013, doi: 10.1109/ICSE.2013.6606653.

W. Wu, W. Zhang, Y. Yang, and Q. Wang, "DREX: Developer recommendation with K-nearest-neighbor search and EXpertise ranking," 2011, doi: 10.1109/APSEC.2011.15.

T. Zhang and B. Lee, "An automated bug triage approach: A concept profile and social network based developer recommendation," 2012, doi: 10.1007/978-3-642-31588-6_65.

G. Yang, T. Zhang, and B. Lee, "Utilizing a multi-developer network-based developer recommendation algorithm to fix bugs effectively," 2014, doi: 10.1145/2554850.2555008.

W. Zhang, S. Wang, Y. Yang, and Q. Wang, "Heterogeneous network analysis of developer contribution in bug repositories," 2013, doi: 10.1109/CSC.2013.23.

Chen, Liguo, Wang, Xiaobo, Liu, Chao, 2011. An approach to improving bug assignment with bug tossing graphs and bug similarities. J. Softw. 6 (3). https://doi.org/10.4304/jsw.6.3.421-427.

G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," 2009, doi: 10.1145/1595696.1595715.

Anvik, John, Murphy, Gail C., 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Transactions on Software Engineering and Methodology (TOSEM) 20 (3), 1–35.

Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S., 2013. In: May). Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. IEEE, pp. 2–11.

Wang, S., Zhang, W., & Wang, Q. (2014, September). FixerCache: Unsupervised caching active developers for diverse bug triage. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10).

Badashian, A.S., Hindle, A., Stroulia, E., 2015. In: September). Crowdsourced bug triaging. IEEE, pp. 506–510.

Zhang, Wen, Wang, Song, Wang, Qing, 2016. KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. Information and Software Technology 70, 68–84.