



Article

A Multi-Factor Approach for Selection of Developers to Fix Bugs in a Program

Shikai Guo ^{1,2,3,4} , Shifei Chen ⁵, Siwen Wang ¹, Decheng Zhang ¹, Yaqing Liu ^{1,*}, Chen Guo ², Hui Li ^{1,3}  and Tingting Li ⁴

¹ The College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China

² The School of Marine Electrical Engineering, Dalian Maritime University, Dalian 116026, China

³ Collaborative Innovation Center for Transport Studies of Dalian Maritime University, Dalian 116026, China

⁴ Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China

⁵ Sichuan University-Pittsburgh Institute, Chengdu 610207, China

* Correspondence: liuyaqing@dmlu.edu.cn

Received: 26 June 2019; Accepted: 9 August 2019; Published: 13 August 2019



Abstract: In a software tracking system, the bug assignment problem refers to the activities that developers perform during software maintenance to fix bugs. As many bugs are submitted on a daily basis, the number of developers required is quite large, and it therefore becomes difficult to assign the right developers to resolve issues with specific bugs. Inappropriate dispatches results in delayed processing of bug reports. In this paper, we propose an algorithm called ABC-DR to solve the bug assignment problem. The ABC-DR algorithm is a two-part composite approach that includes analysis between bug reports (i.e., B-based analysis) and analysis between developers and bug reports (i.e., D-based analysis). For analysis between bug reports, we use the multi-label k-nearest neighbor (ML-KNN) algorithm to find similar bug reports when compared with the new bug reports, and the developers who analyze similar bug reports recommend developers for the new bug report. For analysis between developers and bug reports, we find developer rankings similar to the new bug report by calculating the relevance scores between developers and similar bug reports. We use the artificial bee colony (ABC) algorithm to calculate the weight of each part. We evaluated the proposed algorithms on three datasets—GCC, Mozilla, and NetBeans—comparing ABC-DR with DevRec, DREX, and Bugzie. The experimental results show that the proposed ABC-DR algorithm achieves the highest improvement of 51.2% and 53.56% over DevRec for recall@5 and recall@10 in the NetBeans dataset.

Keywords: developer recommendation; composite method; ML-KNN algorithm; relevance score

1. Introduction

In software development systems, unrepaired bug reports account for about 70% of open source bug tracking systems [1,2]. Developers spend 50–80% of their time looking for, understanding, and fixing bugs [3,4] and also, the number of software bug fixes has become quite large. For example, approximately 200 bug reports were submitted to the Eclipse bug tracking system around the release date [5], and Debian submitted nearly 150 reports each day [6]. The bug tracking system can help developers manage bug reports and fix bugs, however, assigning the right developer to a specific bug report is still a huge challenge.

In traditional bug repositories, bug reports are manually sorted by some professionals, which requires labor costs. Some methods for automatic bug classification have been proposed to reduce labor costs and improve efficiency, such as semi-supervised text categorization methods for bug classification.

To measure the accuracy of the current state of the arts, precision@k, recall@k and f-measure@k are the frequently-used evaluation methods [7–11]. Top-k prediction accuracy is the percentage of bugs with supplementary patches whose ground truth methods are ranked in the top-k positions in the returned ranked lists of change locations (i.e., methods). This utilizes existing tagged bug reports and unlabeled bug reports, combined with Naive Bayes classifier and expectation maximization (EM) algorithm, to train the classifier using weighted recommendation lists. This results in improved performance by applying multiple developer weights on the training classifier [12]. However, when unlabeled data exist on a smaller scale, the EM algorithm may lose classification accuracy [13–15]. To solve the problem of poor classification and inaccuracy, many researchers have proposed a bug classification method based on machine learning. For example, Zhang et al. proposed a method called KSAP to solve the bug report allocation problem using KNN search and heterogeneous proximity functions. This method improved the efficiency of bug allocation by using historical bug reports and heterogeneous networks of a bug repository. When a new bug report is submitted to the bug repository, KSAP first finds a historical bug report similar to the new bug report through the KNN algorithm, and then ranks the developers based on the heterogeneous proximity of these historical bug reports based on the contribution size [16]. In the developer-based ranking algorithm, a developer ranking algorithm based on a topic model and the developer relationship with automatic bug classification was proposed by Zhang et al. [7]. This method uses the Latent Dirichlet Allocation (LDA) topic model to extract topics from historical bug reports that are similar to the new bug report. Based on similar topical features, it discovers the interest and expertise of these developers in dealing with bug reports, and then fixes the given bug report by analyzing the most appropriate developers. Currently, these methods only consider the contribution of developers to historical bug reports, but not the analysis of the specific features within the bug report.

In this paper, we propose the ABC-DR algorithm to fix bug reports and recommend a suitable list of developers to resolve a new bug report. The ABC-DR algorithm consists of two parts: analysis between bug reports (i.e., the B-based analysis) and analysis between developers and bug reports (i.e., the D-based analysis). In the B-based analysis, we first use the KNN algorithm to search for historical bug reports similar to the new bug report and then use the multi-label classification methods to find developers with historically similar bug reports. A score output is then obtained for potential developers reporting new bug reports based on this historical developer information. In the D-based analysis, we analyze the correlation between historical bug reports and developers. We use terms, topics, products, and components to characterize bug reports. The relevance scores for four features are obtained by analyzing the correlation between each feature and developer. Combining these two analyses, we obtain the ABC-DR method. We use the artificial bee colony algorithm to solve the weights for each part of the parameters [17,18]. According to the recall@k, a list of developer recommendations is obtained. Finally, we evaluate the four datasets to verify the effectiveness of our proposed ABC-DR method.

The main contributions of this paper are as follows:

- We propose a novel approach named ABC-DR that considers and integrates multiple factors to recommend suitable developers for bug fixing.
- We propose that the weight of multi-factors changes after each new solution is generated, which could increase the search range. This method can improve the accuracy and efficiency of the parameter training process.
- We evaluated our ABC-DR approach on four open bug repositories (Mozilla, GCC, and NetBeans) and further validated the effectiveness of our proposed method by comparing it with DevRec, Bugzie, and DREX.

The structure of this paper is as follows: We introduce the related work and motivation in Section 2. In Section 3, we describe in detail the ABC algorithm. Section 4 shows our experimental setup, evaluation methods, and experimental results. In Section 5, we summarize the study and introduce the future direction of this work.

2. Related Work and Motivation

2.1. Related Work

In this section, we discuss methods for recommendation of developers for bug fixes. We then briefly introduce the DREX, DevRec, and Bugzie algorithms mentioned in this paper.

Many related methods have been proposed for appropriate developers recommendation for bug reports. Xuan et al. provided a ranking method for the developer community, which integrates collaborative rankings and functional rankings [8]. Collaborative rankings leverage developer collaboration because developers can comment on the same bugs. Functional rankings leverage the expertise of the developers to measure their ability to handle specific bugs. Terminology is the sole feature used for characterizing bug reports when measuring the link between bug reports and developers. Based on developer rankings, social networking technology is a good way to capture additional features for implementing automatic bug triage [19]. Xuan et al. modeled a developer prioritization in the bug reports of Eclipse and Mozilla bug repositories based on a social network technique [20]. On this basis, Yang et al. built a different social network, named Multi-Developer Network (MDN). MDN can help verify the ability of developers to fix a given bug report based on the comments received and the submissions sent, not just considering the comment activity, but also the commit message for the changes in the source code files [9]. The MDN developer network is used to rank developer candidates. The method extracts developer candidates using two aspects. On the one hand, it extracts developers from previous bug reports that are similar to the new bug report, whereas, on the other hand, it extracts developers from the bug report that the developer has processed and that which has the same components as the new bug. Based on the research, Yu et al. analyzed the social relationship between contributors and reviewers, and built a new social network to recommend potential developers based on the comment network (CN) of each project. The network recommends potential developers by capturing the common interest of social activities between developers. The reviewer is recommended by mining the strength of the previous social contact between the pull requester and the potential reviewer [21]. However, the link between bug reports is not considered in their approach.

Bug classification problem: Bug classification means that a bug report can only be assigned to a specific class tag, that is, to a developer. This serves to be a single tag classification problem. For recommendation of developer issues for bug reports, a bug report can be resolved by multiple developers, and a developer can also participate in multiple bug report resolutions. Hence, this can be seen as a multi-label classification problem [10,22,23]. In the field of software engineering research, labels have been widely used, and many methods of label classification are also widely used in many algorithms and fields [24–29]. Many bug classification algorithms have emerged. Shokripour et al. proposed a time-based automatic bug classification method. The method featured time metadata as a weighted term. The frequency of terms in documents and corpora is solved using tf-idf technique [30]. Xia et al. used improved LDA topic model for automatic bug classification. They proposed an incremental learning method called TopicMiner, which assigns appropriate bug reports based on the topic distribution of the bug report and the affinity relationship between the bug fixer and the topic [11]. Distasi et al. enhanced the distribution of bug reports by enhancing developer analysis and ranking, applying learning-ranking techniques, and ranking the appropriate developers for each bug report. They enriched the research's remediation activities by using information extracted from messages submitted by developers and project API descriptions that interact with code developers [31]. John et al. used the the Creation Assistant for Easy Assignment (CASEA) tool to assign developers to bug reports, leveraging the knowledge of project-related members. The recommended developers using CASEA mainly include four parts: data collection, data preparation, recommender training and recommender evaluation. The experimental results show that even if user manager has limited knowledge, the developer can be accurately recommended for bug reports [32]. For the classifier-based research, Jonsson et al. proposed an integrated learner method combining several classifiers for

automatic bug classification. When the individual classifiers have different classification results, the integrated classifier can overcome the shortcomings of single classifiers with inaccurate results, and achieve better classification performance. It combines different types of classifiers (such as naive Bayes classifier, SVM classifier, KNN classifier, decision tree classifier, etc.) with the stacked generalization (SG) classifier to achieve the final bug report classification through a smooth linear model [33]. These references only consider the relationship between developers and bug reports, and classify bug reports based on their interests and strengths, without considering the characteristics of the bug report itself. The characteristics of the bug report itself will also have a certain impact on the bug classification.

Our comparison algorithm Bugzie, in this paper, is a bug classification algorithm based on fuzzy set and cache-based approach [34]. For new bug reports, fuzzy set theory is used to arrange the fixed capabilities of developers and technical terms using the relationship between the developers and technical terms, that is, if a developer has a higher fixed relevance to a technical term, we assume that the developer has more expertise in solving bug reports related to that term and thus, will be ranked higher. The algorithm uses terms to characterize bug reports, considering the relationship between developers and terminology. The DevRec method is a composite method recommended by developers to solve the parameter weights of each part using greedy methods.

In contrast, DREX searches for similar bug reports using KNN, in turn, ranking developer expertise based on participation records found in similar bug reports, through frequency and social network metrics (such as Indegree, Outdegree, Degree, PageRank, Betweenness, and Closeness) to assess the developer's ability to handle the bug reports [35]. The algorithm includes two aspects: First, the KNN algorithm is used to find a bug set similar to the new bug report. Second, it includes sorting the developer's professional capabilities according to the developers' fix information in a similar bug set. For bug assignment issues [36–38], it attempts to find one or a series of developers who can fix bug reports. Our proposed algorithm recommends a group of potential developers who are interested in bug reports and also have professional knowledge.

We divided the types of methods used in these studies into two categories (i.e., social network metrics and machine learning algorithms). Then, we define the criteria (number of labels, projects, consider factors and methods) for summarizing the comparisons. The overview of the related work on bug triaging is shown in Table 1.

Table 1. An overview of the related work on bug triaging.

Category	Numbers of Labels	Projects	Consider Factors	Methods	Ref.
Social network metrics	Multi-label	Eclipse, Firefox, Thunderbird	Developers' relationship, summary, description	Collaborative rankings Functional rankings	[8]
	Multi-label	RALIC	Stakeholders' relationship and stakeholder roles	Betweenness centrality, Closeness centrality, PageRank, Degree centrality, Prioritised stakeholder roles, Prioritised stakeholders	[19]
	Multi-label	Eclipse, Mozilla, Jboss	Developers' relationship, summary, description, component	Leadership network, social network	[9,20]
	Multi-label	Github	Contributors and reviewers, summary, description	Comment network	[21]
	Multi-label	AspectJ, BIRT, Eclipse, JDT, SWT, Tomcat	Summary, description	Learning-to-rank technique, feature extraction	[31]
	Multi-label	Mozilla, Firefox	Summary, description, developers' relationship	Indegree, Outdegree, Degree, PageRank, Betweenness, ML-KNN and Closeness	[35]
Machine learning algorithms	Multi-label	GCC, OpenOffice, Mozilla, NetBeans, Eclipse	Summary, description, product, component	ML-KNN, Estimate weights approach	[10]
	Multi-label	Eclipse, NetBeans, ArgoUML	First commit, Last commit, developers, reported bugs, fixed bugs	Tf-idf	[30]
	Multi-label	GCC, OpenOffice, Mozilla, NetBeans, Eclipse	Summary, description, product, component, developers	LDA	[11]
	Multi-label	Project's issue tracking system (ITS)	Developers' relationship	Creation Assistant for Easy Assignment (CASEA) tool	[32]
	Multi-label	Telecom1, Telecom2, Telecom3, Telecom4	Title, heading, feature development, document updates, and release management	Integrated learner method	[33]
	Multi-label	Firefox, Eclipse, Apache, NetBeans, FreeDesktop, GCC, Jazz	Summary, description, developers	Fuzzy set and cache-based approach	[34]
	Multi-label	Eclipse	Summary, description, developers	Text categorization	[36–38]

2.2. Motivations

Since many bug reports are submitted to the bug tracking system everyday [39–41], recommending the right developers the very first time they encounter a bug report will save a lot of money and improve the quality of the software as well. Therefore, bug assignment problem has become a key issue for many researchers. In this paper, we use the ABC-DR algorithm to solve the problem of assigning the appropriate developer for a specific bug report by preventing the bug classifier from specifying an inappropriate developer. The ABC-DR algorithm is introduced in detail in Section 3.

3. Method

In this section, we introduce the overall framework of our proposed ABC-DR method and provide the specific algorithms in detail.

3.1. Overview

Based on the above motivations, we propose the ABC-DR method to solve the bug assignment problem of recommending an appropriate developer for a specific bug report. Our ABC-DR method consists of two parts as follows: The first part is based on the analysis between bug reports (i.e., B-based analysis). We recommend potential developers for a new bug report through developers with historical bug reports that are similar to the new bug report. The second part is based on the analysis between developers and the bug reports (i.e., D-based analysis). Based on the developer's experience in solving bug reports, we find the right developer ranking for a given bug report. The framework of ABC-DR is shown in Figure 1.

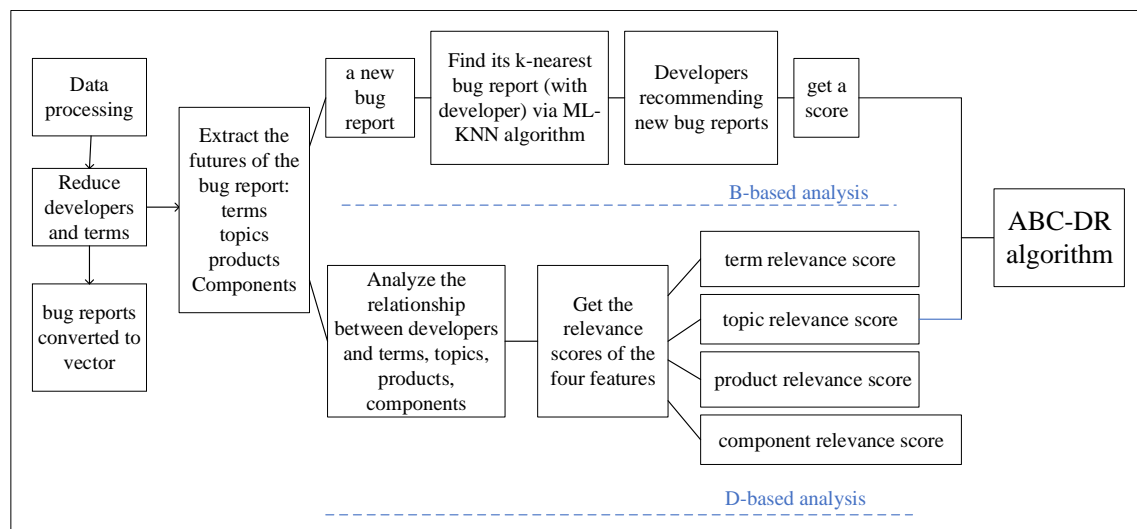


Figure 1. Overall framework of the ABC-DR method.

3.1.1. B-Based Analysis Score

In B-based analysis, for a new bug report B_{new} , we use ML-KNN algorithm to predict its resolvers (i.e., developers who have experience with bug resolution) and output a score for each potential resolver [42].

First, we find the k-nearest neighbors for B_{new} by the k-nearest neighbor algorithm. Developer recommendations are provided based on the developers of these k bug reports. Second, we use the Euclidean distance formula (see Equation (1)) to calculate the distance between the bug reports (i.e., measure the similarity between bugs) and find k bug reports with the smallest distance from the new bug report B_{new} .

$$\rho = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

We use the following features to characterize a bug report:

1. **Terms:** It is a set of preprocessed data, a collection of bug report descriptions that appear in the summary and description of the bug report. Data preprocessing involves removing words that are not meaningful and reducing the word to the root form, i.e., stemming. Each stem is a feature. Its value is determined by the frequency with which the word appears in the bug report.
2. **Topics:** We use the LDA model for topic analysis, simplifying the reports into a set of topics, and determining the probability that the reports belongs to the topics [43,44]. Each topic is a feature, and the feature value is the probability that the topic belongs to a bug report.
3. **Products:** This refers to the affected product in the bug report. Each product is a binary feature [10]. There are only two possible values. If the product is affected by a bug, the eigenvalue is 1, else the eigenvalue is 0.
4. **Components:** This refers to the component affected by the bug in the bug report, either affected or not affected. If the component is affected by a bug, the eigenvalue is 1, else the eigenvalue is 0.

Third, we use the multi-label classification method to infer the resolvers for the new bug B_1 based on the resolvers of its k-nearest neighbors. We treat each developer as a class label, each bug report as a data element, and a bug report with resolvers as a training data element. In this way, the step of providing the resolvers for bug reports is reduced to a multi-label classification problem [45]: given a data element (i.e., a new bug report) and predicting its label (i.e., the resolvers).

The ML-KNN method outputs the relative likelihood of the label being assigned to a data point [46]. After applying this method, we assign a score to each potential developer di , indicating the possibility that di is a resolver for the new bug report bi , represented by $B_{score_{bi}}(di)$.

3.1.2. D-Based Analysis Score

For D-based analysis, we simulate the extent to which developers are associated with previously resolved historical bug reports, where the degree of relevance refers to the experience, interests, and expertise of the developers who have resolved bug reports in the past.

We measure the degree of association between the developers and the bug reports by measuring the distance between the four characteristics of the bug report. We call this the relevance score. We refer to the distance between developers and terms, topics, products, and components as term relevance scores, topic relevance scores, product relevance scores, and component relevance scores, respectively.

- **Term Relevance Score:** For a bug report bi and developer di , the term relevance score is calculated as follows:

$$terms_{bi}(di) = 1 - \prod_{t \in bi} (1 - \frac{n_{di,t}}{n_t + n_{di} - n_{di,t}}) \quad (2)$$

where t refers to the terms in bi . n_{di} refers to the number of bug reports that developer di has solved. n_t refers to the number of reports with the term t appearing. $n_{di,t}$ refers to the number of bug reports containing the term t in the bug report bi resolved by developer di .

- **Topic Relevance Score:** For a bug report bi and developer di , the topic relevance score is calculated as follows:

$$topics_{bi}(di) = 1 - \prod_{t \in bi} (1 - \frac{\sum_{v \in T_d} v[t]}{\sum_{v' [t]} v'[t]} \times bi[t]) \quad (3)$$

Considering a set of topic vector sets T corresponding to all the bug reports, T_{di} indicates the topic vector associated with developer di or the bug report addressed by developer di . When t refers to the topics in bi , for a topic vector v , $v[t]$ is the probability that the relevant bug report belongs to topic t . $bi[t]$ is the probability that a new bug report bi belongs to the topic t .

- Product and Component Relevance Score: For a bug report bi and developer di , the product relevance score is calculated as follows:

$$product_{bi}(di) = \frac{\sum_{bi \in B_{di}} bi[p_{bi}]}{\sum_{bi' \in B_{di}} bi'[p_{bi}]} \quad (4)$$

and the component relevance score is calculated as follows:

$$component_{bi}(di) = \frac{\sum_{bi \in B_{di}} bi[c_{bi}]}{\sum_{bi' \in B_{di}} bi'[c_{bi}]} \quad (5)$$

The product and component defined here are affected by the developers. When we calculate the relevance score, each bug report has only one product and one component. For a bug report collection B , B_{di} represents the developer di who has previously participated in the resolution of the bug report bi .

For a product p , $bi[p] = 1$ means that the product p is affected by the bug report bi , while $bi[p] = 0$ means that the product p has not been affected by the bug report bi . p_{bi} is defined as the value of the product p field of bug report bi .

Similarly, for a component c , $bi[c] = 1$ means that the component c is affected by bug report bi , while $bi[c] = 0$ means that the component c has not been affected by the bug report bi . c_{bi} is defined as the value of the component c field of bug report bi .

- D-based Score

We define the term relevance score, topic relevance score, component relevance score, and product relevance score in the previous section. These scores are combined to obtain the score of the D-based analysis.

For a bug report bi and developer di , the D-based score is calculated as follows:

$$Dscore_{bi} = l_1 \times terms_{bi}(di) + l_2 \times topics_{bi}(di) + l_3 \times product_{bi}(di) + l_4 \times component_{bi}(di) \quad (6)$$

3.2. ABC-DR Score

As ABC-DR algorithm combines B-based analysis with D-based analysis, the final ABC-DR score is defined as a linear combination of two-part scores.

For a bug report bi and developer di , the ABC-DR score is calculated as follows:

$$ABC - DR_{bi}(di) = \alpha_1 \times Bscore_{bi}(di) + \alpha_2 \times Dscore_{bi}(di) \quad (7)$$

We expand the formula of $Dscore_{bi}$ and sort it out to the following formula:

$$ABC - DR_{bi}(di) = \gamma_1 \times Bscore_{bi}(di) + \gamma_2 \times terms_{bi}(di) + \gamma_3 \times topics_{bi}(di) + \gamma_4 \times product_{bi}(di) + \gamma_5 \times component_{bi}(di) \quad (8)$$

$\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5 \in [0, 1]$ represents the weight value from the contribution of different scores. We use the artificial bee colony algorithm to solve the values of the five parameters $\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5$. The Algorithm 1 is shown as follows.

Line 1 indicates the initialization of NP/2 honey sources, and the honey source is generated using a random method, where NP is the total number of bees. Lines 5–9 represent the hired bees stage, in which hired bees traverse all NP/2 honey sources. For each honey source in the traversal process, we randomly select one from the remaining honey sources, and then use the formula in Line 7 to generate a new honey source between the two. We compare the new honey source with the previous one. If it is improved, the old one is discarded, otherwise we keep the original honey source and

increase the number of unimproved times of the honey source by one. After the previous phase is completed, we calculate a probability for each solution by the fitness value of the solution (i.e., honey source), and the calculation formula is show in Line 9 . The stages of observing the bees are shown in Lines 10–14. We traverse each observation bee in turn. Each observation bee performs a roulette selection method using the calculated selection probability matrix to select a honey source from all honey sources (i.e., honey source k). The honey source selected by the observed bee is updated, and the last stage of the greedy selection operation is repeated for the honey source. At this stage, if a honey source has a higher fitness value, it is more likely to be selected. Lines 15–19 are the stages of scouting bee, scouting bee traversing all the honey source, finding and selecting the source whose number of unimproved times is greater than the threshold L . These honey sources are discarded and a new honey source is randomly generated and replaced with the original ones, recalculating the fitness values and associated probabilities of these honey sources. On Lines 20–22, we record the optimal solution for each iteration.

Our proposed ABC-DR method uses the artificial bee colony algorithm to solve the weight parameters. In the training phase, we select 10% of the bug data in the dataset as the training data. The developers of these bug reports are known and they judge the advantages and disadvantages of the parameters based on the recall@ k of the 10% training data. In the forecasting phase, for a new bug, these five scores are calculated separately, combined with the trained parameters, and then the appropriate developers are selected according to the combined scores starting from large to small.

4. Experiments

4.1. Experimental Setup

In this study, we collected three datasets from different software development communities: GCC, Mozilla, and NetBeans. A bug report contains many fields, such as reporters, file, creation time, modification time, bug version, platform, and comment list. In this study, we collected five pieces of information from the bug report field: bug summary, bug description, bug-affected products, bug-affected components, and developers involved in the bug resolution process (i.e., bug resolvers). For small bug report datasets, we removed the terms that appear fewer than 10 times, such as GCC. For large-scale bug report collections, we removed the terms that appear fewer than 20 times, such as Mozilla and NetBeans [10]. For each bug report in these three datasets, we removed developers who have fewer than 10 occurrences. As these developers are not active during the bug resolution activity, there is no impact on the task of recommending developers to resolve the new bug reports. Table 2 shows the statistical results of the experimental data for the GCC, Mozilla, and NetBeans datasets.

Table 2. Statistical results of average workload and fixing time in our dataset.

Project	The Number of Bug Reports	The Number of Assignees	Average Workload Per Assignee (Number)	Average Fixing Time Per Bug (Days)	Period
NetBeans	19,249	265	72.6	442.5	11 February 1999–31 December 2014
Mozilla	15,501	1022	15.2	244.9	17 March 1999–31 December 2014
GCC	13,301	256	52	292.1	03 August 1999–01 December 2014

In terms of bug resolution, we compared our proposed ABC-DR algorithm to three baseline algorithms DevRec, DREX, and Bugzie.

Algorithm 1 ABC-DR algorithm to train parameters.**Input:**

B, D, NP, L, MC.
 B: All bug report data collection
 D: Bug report collection for training
 NP: Total number of bees
 L: Maximum unimproved
 MC: The maximum number of iterations

Output:

a, b, c, d, e // Output of parameters

1. Initialize $NP/2$ initial solutions, each solution is a set of parameter distributions, and the size is a random value.
2. cycle = 1;
3. while cycle < MC do:
4. $I, l = 0$;
5. Hire bee stage:
6. The update is performed for each solution, and the fitness value is calculated, and the fitness value is used as the basis for the selection probability of the next stage.
- 7.

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$$

8. Compare the recall@k of the new solution with the original solution and select the solution with high recall rate. Update unimproved
9. Calculate the probability that the next phase will be selected.

$$P_i = \frac{fit(x_i)}{\sum_{n=1}^{NP/2} fit(x_n)}$$

10. Observe the bee stage:
11. While $i < NP/2$ do:
12. Update the solution according to the probability selection, compare the recall@k, and choose a better solution.
13. $I=i+1$.
14. End while.
15. Scouting Bee Stage:
16. For all i from 1 to $NP/2$ do:
17. Determine the number of unupdated solutions. If the upper limit L is exceeded, the solution is discarded and a new solution is generated instead.
18. End if.
19. End for.
20. Record the optimal solution.
21. cycle = cycle + 1.
22. End while

For each bug report analysis, we extracted its bug ID, bug summary and description information, bug product, bug component, and bug developer [10]. We extracted terms and topics from the summary and description information after removing the stop words. We extracted the bug reports from the bug repository according to its creation time and then divided it into 11 non-overlapping frames (or windows) of the same size. The verification process was as follows: First, we used the bug reports in Frame 0 for training and we tested the bug reports in the first frame. Second, we trained the bug reports in Frames 0 and 1, and used the same method to test the bug reports in Frame 2, and so on [34,47]. In the final fold, we used the bug reports in Frames 0–9 for training and tested the bug reports in Frame 10. In the training data, we have the features that characterize the bug report and the set of resolvers for each bug report. We used these data to train ABC-DR, DevRec, Bugzie, and DREX.

In the test data, for each bug report, we used the bug reports' features to predict the set of resolvers. For this, we used the resolvers recorded in the bug repository as the basis.

Our proposed ABC-DR method uses the LDA topic model. For LDA parameter settings, we set the maximum number of iterations to 500 and the hyperparameters α and β to $50/T$ and 0.01, respectively, where T is the number of topics. By default, we set the number of topics T to 5% of the number of different terms (i.e., words) in the training data. We used jGibbsLDA as the LDA implementation [10]. For our proposed ABC-DR method, by default, we set the number of neighbors to 15. For Bugzie, there are two parameters: the developer cache size and the number of descriptive terms. We used 100% developer cache size and set the number of descriptive terms to 10. This parameter setting has been proven to attain the best performance. We set the number of neighbors for DREX to 15.

The goal of our proposed ABC-DR approach is to resolve bug reports recommendation problem with the highest quality while ensuring the most appropriate developer assignment. To ensure that our approach meets this goal, our assessment should answer three main questions as follows:

1. How much improvement has our proposed ABC-DR method obtained when compared to DevRec, Bugzie, and DREX?
2. How does our proposed ABC-DR method compare to B-based components and D-based components?
3. How does the contribution of the five parts of the ABC-DR method affect the performance of the ABC method?

In this work, we used recall@5, recall@10, precision@5, and precision@10 to evaluate our proposed ABC-DR method alongside DevRec, Bugzie, and DREX [47]. recall@k and precision@k are defined as follows:

Let b be a collection of bug reports containing k bug reports. d is represented as a developer collection, and each bug report bi corresponds to developers di . $AD(di, bi)$ represents the actual developers di involved in the bug report bi solution. $RD(di, bi)$ is the recommended developers di for a given bug report bi through our algorithm.

The recall@k and precision@k for the k bug reports are given by [48]:

$$\text{recall@k} = \frac{1}{k} \sum_{i=1}^k \frac{|AD(di, bi) \cap RD(di, bi)|}{|AD(di, bi)|}$$

$$\text{precision@k} = \frac{1}{k} \sum_{i=1}^k \frac{|AD(di, bi) \cap RD(di, bi)|}{|RD(di, bi)|}$$

4.2. Experiments Results

4.2.1. Our Proposed ABC-DR Method's Performance

We compared our proposed ABC-DR method with DevRec DREX, and Bugzie. Apart from directly presenting the results, we evaluated the percentage of improvement between ABC-DR and the other approaches. In Tables 3 and 4, we can see the recall@k and precision@k of the ABC algorithm on four projects' datasets. It shows the comparison between our proposed ABC-DR method and the DevRec algorithm, DREX algorithm, and Bugzie algorithm at recall@5, recall@10, precision@5, and precision@10.

In Tables 3 and 4, we can see that our proposed ABC-DR method has better performance than DevRec. The ABC-DR algorithm has increased by 9.425% and 6.8025% over the DevRec algorithm on average recall@5, and our proposed ABC-DR method is better than DREX and Bugzie in the recall@5 and the recall@10. The average recall@5 increased by 26.2825% and 81.545%, and average recall@10 increased by 8.1375% and 20.89%. In the NetBeans dataset, our proposed ABC-DR algorithm has

achieved the highest improvement of 153.89% and 68.64% over Bugzie for recall@5 and recall@10. The developer recommendation problem discussed in this paper is to make accurate developer recommendations based on the bug classification, which is an extension of the bug classification problem. Therefore, the performance of the Bugzie algorithm is poor compared to other methods. In Table 4, we can see that our proposed ABC-DR algorithm has an improvement over DREX and Bugzie at precision@5 and precision@10. In the NetBeans dataset, the ABC-DR algorithm has the greatest improvement compared to the Bugzie, and the precision@5 is 76.58%. In the Mozilla dataset, the ABC-DR algorithm achieves the greatest improvement of 71.32% and 20.93% over DREX for precision@5 and precision@10.

Table 3. recall@5 and recall@10 of our proposed ABC-DR and DevRec, DREX, and Bugzie. It also shows the improvement of ABC-DR over DevRec (Imp.De), the improvement of ABC-DR over DREX (Imp.DR), and the improvement of ABC-DR over Bugzie (Imp.Bu).

project		ABC-DR	DevRec	Imp.De	DREX	Imp.DR	Bugzie	Imp.Bu
GCC	recall@5(%)	41.3	39.8	3.76	41.1	0.72	38.1	8.39
	recall@10(%)	51.1	49.8	2.61	51.7	−1.16	57.6	−11.28
Mozilla	recall@5(%)	39.6	34.7	14.12	23.8	66.38	23.4	69.23
	recall@10(%)	40.3	38.1	5.77	32.3	24.76	38.4	4.94
NetBeans	recall@5(%)	51.2	49.3	3.85	41.9	22.19	26.3	94.67
	recall@10(%)	53.6	51.6	3.87	51	5.09	44.2	21.26
Aver	recall@5(%)	43.625	40.725	9.425	25.385	26.2825	26.125	81.545
	recall@5(%)	48.35	45.4	6.8025	45.4	8.1375	42.225	20.89

Table 4. precision@5 and precision@10 of our proposed ABC-DR and DevRec, DREX, and Bugzie. It also shows the improvement of ABC-DR over DevRec (Imp.De), the improvement of ABC-DR over DREX (Imp.DR), and the improvement of ABC-DR over Bugzie (Imp.Bu).

Project		ABC-DR	DevRec	Imp.De	DREX	Imp.DR	Bugzie	Imp.Bu
GCC	precision@5(%)	24.9	24	3.75	24.4	2.04	22.6	10.17
	precision@10(%)	15.4	15	2.66	15.7	−1.91	17.3	−10.98
Mozilla	precision@5(%)	23.3	21.9	6.39	13.6	71.32	15.3	52.28
	precision@10(%)	11.9	10.4	14.42	9.84	20.93	12.7	−6.29
NetBeans	precision@5(%)	27.9	27	3.33	23.2	20.25	15.8	76.58
	precision@10(%)	14.9	14.4	3.47	14.4	3.47	13.2	12.87
Aver	precision@5(%)	25.55	24.275	5.33	20.85	27.7925	16.25	67.5
	precision@10(%)	14.3	13.225	8.7625	13.635	6.305	13.175	13.3725

4.2.2. Performance Comparison between Our Proposed ABC-DR Method and B-Based and D-Based Components

Our proposed ABC-DR method consists of two parts (B-based component analysis and D-based component analysis), and we examined the performance of each component. We needed to see if the combined performance of the two components has improved when compared to the performances of the individual components. The experiments were done for the GCC, Mozilla, and NetBeans datasets, and recall@5, precision@5, recall@10, and precision@10 were used as evaluation criteria.

In Table 5, it can be noted that the ABC-DR method has better performance when compared to the individual components for the GCC, Mozilla, and NetBeans datasets. For the GCC dataset, the results of ABC-DR method are 41.30%, 24.90%, 51.10%, and 15.40%. It can be seen in Table 5 that the ABC-DR method shows the best performance and the B-based component approach performs better than the

D-based component approach. For the Mozilla dataset, the results of ABC-DR method are 39.60%, 23.30%, 40.30%, and 11.90%, which have significantly improved compared to the B-based component method and the D-based component method. The ABC-DR method performs best compared to the B-based component and the D-based component methods. For the NetBeans dataset, the results of ABC-DR method are 51.20%, 27.90%, 53.60%, and 14.90%; the results of B-based component method are 37.10%, 20.70%, 45.90%, and 12.90%; and the results of D-based component method are 33.10%, 18.70%, 45.30%, and 12.80%, respectively. It can be found that the improvement effect of B-based component method is more obvious than that of the D-based component method for the NetBeans dataset. The ABC-DR method shows optimal performance compared to the B-based component and D-based component.

In summary, the results show that the best performance is based on a combination of B-based component and D-based component.

Table 5. recall@5, recall@10, precision@5, and precision@10 of our proposed ABC-DR and B-based component (B-based) and D-based component (D-based).

Project		ABC-DR	B-Based	D-Based
GCC	recall@5(%)	41.3	36.9	25.1
	recall@10(%)	51.1	47.1	44
	precision@5(%)	24.9	21.8	15.7
	precision@10(%)	15.4	14.6	12.2
Mozilla	recall@5(%)	39.6	19.7	22.3
	recall@10(%)	40.3	27.4	30.7
	precision@5(%)	23.3	11.6	13.4
	precision@10(%)	11.9	8.3	9.3
NetBeans	recall@5(%)	51.2	37.1	33.1
	recall@10(%)	53.6	45.9	45.3
	precision@5(%)	27.9	20.7	18.7
	precision@10(%)	14.9	12.9	12.8

4.2.3. The Effect of Fifteen Weight Settings on the Performance of the ABC-DR Method

To answer this question, we randomly selected fifteen groups of parameter values and used them to determine the contribution of each part to the performance of the ABC method, i.e., fifteen groups of $\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5$. Comparing them with the parameters determined by our proposed ABC-DR algorithm, we needed to explore the effects of the different parameter values on the experimental results and whether the parameters measured by artificial bee colony algorithm are optimal. We performed the experiment for the GCC, Mozilla, and NetBeans datasets, and used the recall@5, precision@5, recall@10, and precision@10 as the evaluation standard. Tables 6–8 show our experimental results for three datasets.

In Table 6, it can be seen that the parameters measured by our proposed ABC-DR algorithm perform best when fixing bug reports for the GCC dataset. For parameters of 0.55, 0.36, 0.01, 0.06, and 0.01, the performance is better, reaching a recall@5 of 0.392 and a precision@5 of 0.236, respectively. We can find that if the proportion of the product relevance score is increased, the overall performance of the model will deteriorate for the GCC dataset. As a result, it can be seen that the product features in the report have little impact on the GCC dataset during the process of fixing the bug. In Table 7, it can be found that the parameters measured by our proposed ABC-DR algorithm are the highest recall@5 and precision@5 for the Mozilla dataset. For parameters 0.85, 0.02, 0.51, 0.1, and 0.06, the highest recall@10 and precision@10 are achieved, which are 0.419 and 0.123, respectively. On the whole,

the parameters measured by the ABC-DR algorithm have better performance on fixing bug reports. In Table 8, it can be seen that the parameters measured by our proposed ABC-DR algorithm have the highest recall@5 and precision@5 for the NetBeans dataset, and the highest recall@10 and precision@10 for parameters 0.88, 0.19, 0.87, 0.62, and 0.09, which are 0.539 and 0.153, respectively. Parameters 0.17, 0.333, 0.03, 0.69, and 0.87 have the lowest recall@5 and precision@5, 0.282 and 0.161, respectively, and have the worst performance in repairing bug reports. On the whole, the parameters we measured have better performance on fixing bug reports.

In short, the parameters calculated by artificial bee colony algorithm are the optimal parameters, and our ABC-DR algorithm has the best performance in fixing bug reports.

Table 6. Effects of different parameter settings on experimental results for the GCC dataset, and the last behavior in the table is the parameters measured by our proposed ABC-DR algorithm for the GCC dataset.

Project	GCC								
ID	Five-Part Parameter Setting					Evaluation Criterion			
	γ_1	γ_2	γ_3	γ_4	γ_5	recall@5	precision@5	recall@10	precision@10
1	0.48	0.01	0.93	0.68	0.94	0.195	0.122	0.361	0.111
2	0.91	0.3	0.29	0.05	0.61	0.287	0.179	0.436	0.134
3	0.94	0.12	0.48	0.16	0.88	0.224	0.144	0.392	0.121
4	0.87	0.66	0.23	0.52	0.64	0.339	0.207	0.47	0.144
5	0.25	0.38	0.17	0.81	0.54	0.273	0.171	0.462	0.141
6	0.81	0.71	0.64	0.35	0.33	0.374	0.226	0.502	0.154
7	0.51	0.16	0.99	0.44	0.36	0.312	0.193	0.458	0.14
8	0.68	0.34	0.54	0.54	0.44	0.337	0.206	0.469	0.144
9	0.77	0.24	0.47	0.86	0.74	0.269	0.169	0.439	0.134
10	0.55	0.36	0.01	0.06	0.01	0.392	0.236	0.507	0.156
11	0.93	0.14	0.18	0.29	0.22	0.363	0.221	0.474	0.146
12	0.12	0.25	0.95	0.15	0.92	0.157	0.117	0.361	0.112
13	0.11	0.18	0.49	0.54	0.26	0.284	0.176	0.467	0.143
14	0.36	0.09	0.85	0.38	0.73	0.194	0.127	0.381	0.118
15	0.29	0.46	0.81	0.96	0.69	0.264	0.165	0.457	0.139
	0.4	0.08	0.06	1	0.01	0.413	0.249	0.51	0.154

Table 7. Effects of different parameter settings on experimental results for the Mozilla dataset, and the last behavior in the table is the parameters measured by our proposed ABC-DR algorithm for the Mozilla dataset.

Project	Mozilla								
	Five-Part Parameter Setting					Evaluation Criterion			
ID	γ_1	γ_2	γ_3	γ_4	γ_5	recall@5	precision@5	recall@10	precision@10
1	0.93	0.54	0.32	0.46	0.08	0.337	0.193	0.437	0.129
2	0.96	0.37	0.22	0.45	0.5	0.246	0.147	0.313	0.095
3	0.69	0.18	0.22	0.51	0.46	0.248	0.148	0.312	0.094
4	0.95	0.39	0.38	0.09	0.29	0.244	0.147	0.322	0.098
5	0.88	0.09	0.55	0.37	0.59	0.196	0.121	0.272	0.083
6	0.69	0.01	0.84	0.3	0.51	0.212	0.128	0.282	0.086
7	0.91	0.51	0.15	0.73	0.6	0.267	0.159	0.331	0.1
8	0.14	0.17	0.08	0.59	0.86	0.105	0.071	0.175	0.055
9	0.89	0.08	0.1	0.92	0.78	0.218	0.131	0.283	0.085
10	0.22	0.37	0.07	0.92	0.59	0.247	0.147	0.31	0.094
11	0.85	0.02	0.51	0.1	0.06	0.294	0.17	0.419	0.123
12	0.82	0.73	0.11	0.3	0.63	0.196	0.122	0.262	0.081
13	0.48	0.41	0.22	0.42	0.09	0.351	0.202	0.441	0.131
14	0.6	0.35	0.99	0.34	0.19	0.349	0.202	0.445	0.133
15	0.3	0.01	0.6	0.39	0.78	0.107	0.073	0.177	0.056
	0.2	0.11	0.16	0.92	0.01	0.396	0.233	0.403	0.119

Table 8. Effects of different parameter settings on experimental results for the NetBeans dataset, and the last behavior in the table is the parameters measured by our proposed ABC-DR algorithm for the NetBeans dataset.

Project	NetBeans								
	Five-Part Parameter Setting					Evaluation Criterion			
ID	γ_1	γ_2	γ_3	γ_4	γ_5	recall@5	precision@5	recall@10	precision@10
1	0.6	0.19	0.17	0.9	0.37	0.443	0.244	0.521	0.146
2	0.77	0.32	0.35	0.58	0.31	0.442	0.245	0.529	0.149
3	0.2	0.06	0.01	0.28	0.03	0.425	0.239	0.535	0.151
4	0.01	0.57	0.18	0.98	0.73	0.332	0.187	0.449	0.129
5	0.88	0.19	0.87	0.62	0.09	0.434	0.242	0.539	0.153
6	0.11	0.31	0.11	0.01	0.38	0.289	0.167	0.375	0.111
7	0.17	0.33	0.03	0.69	0.87	0.282	0.161	0.392	0.115
8	0.76	0.6	0.31	0.13	0.1	0.427	0.239	0.533	0.151
9	0.25	0.29	0.54	0.41	0.98	0.283	0.163	0.377	0.111
10	0.77	0.76	0.35	0.17	0.14	0.419	0.235	0.527	0.149
11	0.19	0.89	0.38	0.27	0.51	0.341	0.196	0.459	0.132
12	0.75	0.41	0.37	0.21	0.31	0.4	0.226	0.496	0.141
13	0.54	0.07	0.5	0.1	0.82	0.29	0.167	0.363	0.106
14	0.37	0.19	0.96	0.84	0.44	0.432	0.238	0.512	0.143
15	0.28	0.91	0.57	0.24	0.18	0.416	0.234	0.531	0.15
	0.38	0.01	0.25	1	0.01	0.512	0.279	0.536	0.149

5. Discussion

In this paper, we propose the ABC-DR method to solve the bug assignment problem, consider the multifaceted factors of the developer, list the recommended developer rankings, and achieve the purpose of recommending more suitable developers for bug reports. We experimented using three datasets and found that our proposed ABC-DR algorithm has a higher performance than the other state-of-art methods, namely DevRec, Bugzie, and DREX. Then, we analyzed the performance of each component in our method (B-based component analysis and D-based component). We found that the combined performance of the two components has improved when compared to the performances of the individual components. Finally, we randomly generated fifteen sets of weights for each dataset. We compared the effectiveness of using these fifteen sets of weights (randomly generate) with the set of weights that was automatically estimated by ABC-DR in terms of recall-rate@5, recall-rate@10, and recall-rate@20. The experimental results show that our method could effectively improve the performance of bug triaging.

In the future, we can use some term selection approaches to select the most recognizable terminology to describe bug reports. For example, the Chi-square selection method saves time and improves efficiency. We can further improve the ABC-DR algorithm. In the B-based analysis, we can use other data mining methods instead of the KNN algorithm for prediction, such as constructing a Naive Bayesian model.

The time efficiency is one of the factors affecting the method evaluation. Our method exposes the relationship between the appropriate developer and the bug through the parameter form in the training phase. In the subsequent stages, we can predict the developers of new bugs without having to train the parameters every time. This way of working offline increases the overall time efficiency of the method.

For future improvements and practical application possibilities, we also need to consider whether the tool can automatically analyze and select the other appropriate features, such as code, time, etc., when performing different task assignments. In this way, tools can be automatically adapted and resolved when encountering different tasks.

Author Contributions: Conceptualization, S.G. and Y.L.; methodology, S.C.; software, S.W. and D.Z.; validation, C.G. and H.L.; data curation, T.L. and S.C.; writing-review and editing.

Funding: This research was supported by the National Natural Science Foundation of China (grant number. 61672122, 61602077, 61771087, 51879027, 51579024, and 71831002), Program for Innovative Research Team in University of Ministry of Education of China (No. IRT 17R13), the Fundamental Research Funds for the Central Universities (Nos. 3132019501 and 3132019502, JLU), and CERNET Innovation Project (Nos. NGII20181203 and NGII20181205).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Anvik, J.; Hiew, L.; Murphy, G.C. Who should fix this bug? In Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 20–28 May 2006; pp. 361–370.
2. Guo, S.; Chen, R.; Wei, M.; Li, H.; Liu, Y. Ensemble data reduction techniques and Multi-RSMOTE via fuzzy integral for bug report classification. *IEEE Access* **2018**, *6*, 45934–45950. [[CrossRef](#)]
3. Collofello, J.S.; Woodfield, S.N. Evaluating the effectiveness of reliability-assurance techniques. *J. Syst. Softw.* **1989**, *9*, 191–195. [[CrossRef](#)]
4. Gu, Z.; Barr, E.T.; Hamilton, D.J.; Su, Z. Has the bug really been fixed? In Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, South Africa, 1–8 May 2010; pp. 55–64.
5. Anvik, J.; Hiew, L.; Murphy, G.C. Coping with an open bug repository. In Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange, San Diego, CA, USA, 16–17 October 2005; pp. 35–39.
6. Wu, W.; Zhang, W.; Yang, Y.; Wang, Q. Time series analysis for bug number prediction. In Proceedings of the 2nd International Conference on Software Engineering and Data Mining, Chengdu, China, 23–25 June 2010; pp. 589–596.

7. Zhang, T.; Yang, G.; Lee, B.; Lua, E. A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference, Jeju, Korea, 1–4 December 2014; pp. 223–230.
8. Xuan, J.; Jiang, H.; Zhang, H.; Ren, Z. Developer recommendation on bug commenting: A ranking approach for the developer crowd. *Sci. China Inf. Sci.* **2017**, *60*, 072105. [[CrossRef](#)]
9. Yan, G.; Zhang, T.; Lee, B. Utilizing a multi-developer network-based developer recommendation algorithm to fix bugs effectively. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 24–28 March 2014; pp. 1134–1139.
10. Xia, X.; Lo, D.; Wang, X.; Zhou, B. Accurate developer recommendation for bug resolution. In Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany, 14–17 October 2013; pp. 72–81.
11. Xia, X.; Lo, D.; Ding, Y.; Al-Kofahi, J.; Nguyen, T. Improving automated bug triaging with specialized topic model. *IEEE Trans. Softw. Eng.* **2016**, *43*, 272–297. [[CrossRef](#)]
12. Jiang, H.; Nie, L.; Sun, Z.; Ren, Z.; Kong, W.; Zhang, T.; Luo, X. ROSF: Leveraging Information Retrieval and Supervised Learning for Recommending Code Snippets. *IEEE Trans. Serv. Comput.* **2019**, *12*, 34–46. [[CrossRef](#)]
13. Guo, S.; Liu, Y.; Chen, R.; Sun, X.; Wang, X. Improved SMOTE algorithm to deal with imbalanced activity classes in smart homes. *Neural Process. Lett.* **2018**. [[CrossRef](#)]
14. Zhao, H.; Zheng, J.; Xu, J.; Deng, W. Fault diagnosis method based on principal component analysis and broad learning system. *IEEE Access* **2019**, *7*, 99263–99272. [[CrossRef](#)]
15. Nigam, K.; McCallum, A.K.; Thrun, S.; Mitchell, T. Text classification from labeled and unlabeled documents using EM. *Mach. Learn.* **2000**, *39*, 103–34. [[CrossRef](#)]
16. Zhang, W.; Wang, S.; Wang, Q. KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Inf. Softw. Technol.* **2016**, *70*, 68–84. [[CrossRef](#)]
17. Deng, W.; Xu, J.; Zhao, H. An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. *IEEE Access* **2019**, *7*, 20281–20292. [[CrossRef](#)]
18. Deng, W.; Zhao, H.; Zou, L.; Li, G.; Yang, X.; Wu, D. A novel collaborative optimization algorithm in solving complex optimization problems. *Soft Comput.* **2017**, *21*, 4387–4398. [[CrossRef](#)]
19. Lim, S.L.; Quercia, D.; Finkelstein, A. StakeNet: Using social networks to analyse the stakeholders of large-scale software projects. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Cape Town, South Africa, 1–8 May 2010; pp. 295–304.
20. Xuan, J.; Jiang, H.; Ren, Z.; Zou, W. Developer prioritization in bug repositories. In Proceedings of the 2012 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2–9 June 2012; pp. 25–35.
21. Yu, Y.; Wang, H.; Yin, G.; Wang, T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Inf. Softw. Technol.* **2016**, *74*, 204–218. [[CrossRef](#)]
22. Guo, J.; Mu, Y.; Xiong, M.; Liu, Y.; Gu, J. Activity Feature Solving Based on TF-IDF for Activity Recognition in Smart Homes. *Complexity* **2019**, *2019*, 5245373. [[CrossRef](#)]
23. Su, J.; Sheng, Z.; Leung, V.C.M.; Chen, Y. Energy efficient tag identification algorithms for RFID: Survey, motivation and new design. *IEEE Wirel. Commun.* **2019**, *26*, 118–124. [[CrossRef](#)]
24. Deng, W.; Zhao, H.; Yang, X.; Xiong, J.; Sun, M.; Li, B. Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment. *Appl. Soft Comput.* **2017**, *59*, 288–302. [[CrossRef](#)]
25. Su, J.; Sheng, Z.; Xie, L.; Li, G.; Liu, A. Fast splitting-based tag identification algorithm for anti-collision in UHF RFID system. *IEEE Trans. Commun.* **2018**, *67*, 2527–2538. [[CrossRef](#)]
26. Treude, C.; Storey, M.A. How tagging helps bridge the gap between social and technical aspects in software development. In Proceedings of the 31st International Conference on Software Engineering, Auckland, New Zealand, 16–20 November 2009; pp. 12–22.
27. Lo, D.; Jiang, L.; Thung, F. Detecting similar applications with collaborative tagging. In Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM), Eindhoven, The Netherlands, 22–28 September 2012; pp. 600–603.
28. Liu, Y.; Wang, X.; Zhai, Z.; Chen, R.; Zhang, B.; Jiang, Y. Timely daily activity recognition from headmost sensor events. *ISA Trans.* **2019**. [[CrossRef](#)] [[PubMed](#)]

29. Storey, M.A.; van Deursen, A.; Cheng, L. The impact of social media on software engineering practices and tools. In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, Santa Fe, NM, USA, 7–8 November 2010; pp. 359–364.
30. Shokripour, R.; Anvik, J.; Kasirun, Z.M.; Zammani, S. A time-based approach to automatic bug report assignment. *J. Syst. Softw.* **2015**, *102*, 109–122. [CrossRef]
31. Improving Developer Profiling and Ranking to Enhance Bug Report Assignment. Available online: <https://pdfs.semanticscholar.org/4271/186b0496caf5514ee5117552421c6049cd00.pdf> (accessed on 20 June 2019).
32. Evaluating an Assistant for Creating Bug Report Assignment Recommenders. Available online: <http://ceur-ws.org/Vol-1705/04-paper.pdf> (accessed on 20 June 2019).
33. Jonsson, L.; Borg, M.; Broman, D.; Sandahl, K.; Eldh, S.; Runeson, P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empir. Softw. Eng.* **2016**, *21*, 1533–1578. [CrossRef]
34. Tamrawi, A.; Nguyen, T.; Al-Kofahi, J.M.; Nguyen, T. Fuzzy set and cache-based approach for bug triaging. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, Szeged, Hungary, 5–9 September 2011; pp. 365–375.
35. Wu, W.; Zhang, W.; Yang, Y.; Wang, Q. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference, Ho Chi Minh City, Vietnam, 5–8 December 2011; pp. 389–396.
36. Murphy, G.; Cubranic, D. Automatic bug triage using text categorization. In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, Banff, AB, Canada, 20–24 June 2004.
37. Matter, D.; Kuhn, A.; Nierstrasz, O. Assigning bug reports using a vocabulary-based expertise model of developers. In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, Canada, 16–17 May 2009; pp. 131–140.
38. Anvik, J. Automating bug report assignment. In Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 20–28 May 2006; pp. 937–940.
39. Zhao, H.; Yao, R.; Xu, L.; Yuan, Y.; Li, G.; Deng, W. Study on a novel fault damage degree identification method using high-order differential mathematical morphology gradient spectrum entropy. *Entropy* **2018**, *20*, 682. [CrossRef]
40. Guo, S.; Chen, R.; Li, H.; Zhang, T.; Liu, Y. Identify Severity Bug Report with Distribution Imbalance by CR-SMOTE and ELM. *Int. J. Softw. Eng. Knowl. Eng.* **2019**, *29*, 139–175. [CrossRef]
41. Chen, R.; Guo, S.; Wang, X.; Zhang, T. Fusion of Multi-RSMOTE with Fuzzy Integral to Classify Bug Reports with an Imbalanced Distribution. *IEEE Trans. Fuzzy Syst.* **2019**. [CrossRef]
42. Tsoumakas, G.; Katakis, I. Multi-label classification: An overview. *Int. J. Data Warehous. Min.* **2007**, *3*, 1–13. [CrossRef]
43. Blei, D.M.; Ng, A.Y.; Jordan, M.I. Latent dirichlet allocation. *J. Mach. Learn. Res.* **2003**, *3*, 993–1022.
44. Somasundaram, K.; Murphy, G.C. Automatic categorization of bug reports using latent dirichlet allocation. In Proceedings of the 5th India Software Engineering Conference, Kanpur, India, 22–25 February 2012; pp. 125–130.
45. Xia, X.; Lo, D.; Wang, X.; Zhou, B. Tag recommendation in software information sites. In Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR), San Francisco, CA, USA, 18–19 May 2013; pp. 287–296.
46. Zhang, M.L.; Zhou, Z.H. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognit.* **2007**, *40*, 2038–2048. [CrossRef]
47. Bhattacharya, P.; Neamtiu, I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; pp. 1–10.
48. Frakes, B.; Baeza-Yates, R. *Information Retrieval: Data Structures & Algorithms*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1992.

