




Predicting the Fixer of Software Bugs via a Collaborative Multiplex Network: Two Case Studies

Jinxiao Huang and Yutao Ma ^(✉) 

School of Computer Science, Wuhan University,
Wuhan 430072, People's Republic of China
ytma@whu.edu.cn

Abstract. Bug triaging is an essential activity of defect repair, which is closely related to the cost of software maintenance. Researchers have proposed automatic bug triaging approaches to recommend bug fixers more efficiently and accurately. In addition to text features, most of the previous studies focused on single-layer bug tossing (or reassignment) graphs, but they ignored the multiplex (or multi-layer) network characteristics of human cooperative behavior. In this study, we build a collaborative multiplex network composed of a tossing graph and an e-mail communication graph in the bug triaging process. By integrating the idea of network embedding and multiplex network measures, we propose a new strategy of random walks. Moreover, we present a bug fixer prediction model that takes structure and text features as inputs. Experimental results on two large-scale open-source projects show that the proposed method outperforms the selected baseline approaches in terms of commonly-used evaluation metrics.

Keywords: Collaborative multiplex network · Network embedding ·
Bug fixer prediction · Structure and text features

1 Introduction

As a necessary activity of software development, defect repair (also known as bug fixing) plays a vital role in software quality assurance. Defect repair usually uses bug tracking systems (such as Bugzilla¹ and JIRA²) to manage software bug information and assists developers in fixing reported bugs. Bug triaging is an essential part of defect repair [1], and its primary goal is to go through a list of bug reports and assign bugs which have been confirmed to appropriate developers to fix. Since a large number of duplicate, invalid, or unreproduced bugs are regularly reported to bug tracking systems, accurate, automated bug triaging becomes an urgent problem for open-source software development and maintenance. Ideally, a confirmed bug can be assigned directly to the right developer to fix immediately. However, bug triaging is a time-consuming process

¹ <https://www.bugzilla.org/>.

² <https://www.atlassian.com/software/jira>.

in practice. For example, a previous study on the Eclipse project³ showed that it took 40 days on average in the first assignment process, and then it took 100 days or more time in the reassignment process (or called tossing process) [2].

In recent years, many researchers have studied automated bug triaging approaches. In general, these existing methods can be categorized into three main classes: (1) machine learning-based approach [3–6], where classification models are trained based on bug features, such as title, comment, and description; (2) graph (or network)-based approach [2], in which prediction models are built with tossing graph attributes to reduce the tossing path length; and (3) hybrid approach which combines machine learning-based and graph-based approaches. Besides the prediction of the ultimate fixer, some researchers argue that non-fixer developers (also known as tossers) also play a crucial role in the whole bug fixing process. Their contributions include but are not limited to reproducing bugs, commenting bugs, and modifying bug status. This type of developers was called “bug resolution catalyst” by Mani *et al.* [7]. In the bug triaging process, different types of developers work together and collaborate to find the right fixer via deepening the understanding of given bugs.

Social network analysis (SNA) has been widely recognized as a commonly-used tool to analyze the interactions between individuals from network-structured populations in real life and cyberspace. Many previous studies of bug triaging were conducted based on bug tossing graphs, a specific type of social networks, where each node represents a developer and each edge represents an action of bug reassignment (or called tossing). However, most of the previous studies focused on a single-layer tossing graph which has only a single type of relationships between nodes, and they ignored the multiplex (or multi-layer) network characteristics of human cooperative behavior. For example, when different developers collaborate to develop a project, they usually communicate by e-mail or chat by instant messengers (e.g., WhatsApp and WeChat), thus leading to a multiplex network that possesses several layers, each of which represents one type of relationships between nodes.

To leverage more interaction information between developers to predict the right fixer, in this study, we propose a new concept of *collaborative multiplex network*, which is a network composed of different developers and development activities. Different types of activities are represented as different layers of the network to describe various relationships between developers. Because hundreds or thousands of developers are usually involved in a large open-source software project, there has been an increasing concern for the ability of SNA in processing the unprecedented growth of activity data. To facilitate efficient analysis of a complex collaborative multiplex network, we utilize network representation learning (also known as network embedding) to understand the composition of different layers of the network, as well as to analyze the network from different perspectives. Also, we analyze the text information of bug reports, including *summary*, *status*, *history*, and some predefined fields. As a result, the combination of structural information and text information enables us to predict fixers more accurately. In brief, the main contributions of this study include:

³ <https://www.eclipse.org/>.

1. We construct a collaborative multiplex network composed of two everyday developer activities (i.e., bug tossing and e-mail communication) in the bug triaging process. Inspired by the idea of network embedding, we also propose a new random walk strategy designed based on multiplex network measures and network embeddings.
2. By combining the structural information of the collaborative multiplex network and text information extracted by the latent Dirichlet allocation (LDA) model [8], we propose a prediction model to recommend appropriate fixers for target bug reports.
3. We conducted an empirical study on the datasets collected from Eclipse and Gnome⁴, and the experimental results indicated that the proposed model outperformed the selected baseline methods.

The remainder of this paper is organized as follows. Section 2 introduces the related work; Sect. 3 presents the background and preliminaries of this work to facilitate understanding the concept of collaborative multiplex networks in bug triaging; Sects. 4, 5, 6 and 7 introduce the proposed prediction model, experimental setups, empirical results, threats to validity, respectively; finally, Sect. 8 concludes this work.

2 Related Work

2.1 Bug Fixer Prediction

Generally speaking, the studies of bug fixer prediction can be divided into two main types, namely text content-based approach and developer relationship-based approach, according to the feature information they used.

Text Content-Based Approach. The existing approaches based on text information usually use machine learning algorithms to predict the ultimate fixer of a given bug. The text information of a bug report mainly includes *title*, *description*, and *comments*. Cubranic had previously extracted keywords from the fields of title and description as training features and used a Naïve Bayes (NB) classification model (or called classifier) to achieve 30% accuracy [9]. Anvik *et al.* [10] then improved the data processing and classification algorithm by Cubranic's work. They filtered out all the records labeled as *invalid*, *wontfix* and *worksform*, deduplicated bug reports, and removed the bug reports processed by those inactive and non-participating developers. Their prediction model achieved an accuracy of 57% for the Eclipse dataset and 64% for the Firefox⁵ dataset.

With the rapid development of natural language processing techniques, the topic extraction of text information has also attracted the attention of researchers. For example, Xie *et al.* [11] used the Stanford topic modeling toolbox⁶ to group bug reports on the same topic. For new bug reports that have been confirmed, they obtained bug groups (or clusters) according to the model they built and then recommended top-*k* appropriate fixers based on developer experience and skills learned from historical data of fixed bugs. Naguib *et al.* [12] used the LDA model to divide bug reports into

⁴ <https://www.gnome.org/>.

⁵ <https://www.mozilla.org/>.

⁶ <https://nlp.stanford.edu/software/tmt>.

different topics. According to the topics and history of each bug report, they created an activity description file (i.e., a profile) for each developer to describe the role and expertise associated with the developer. Eventually, the proposed model recommended the most appropriate fixer by matching developer expertise with extracted bug topics. Xia *et al.* [13] proposed a specialized topic model-based bug triaging approach, which considered the topic distribution of a new bug report when assigning topics to words of the bug report. They evaluated the method on five software projects and demonstrated that it was better than the selected baselines.

Along with the popularity of deep learning, some new approaches based on deep neural networks have also been proposed in recent years. Lee *et al.* [14] applied deep learning-based automatic bug triager to a few industrial software projects. In particular, they built an automatic bug triager using convolutional neural networks (CNNs) and word embedding. The results obtained from both industrial and open-source projects revealed the benefits of the proposed approach. Xi *et al.* [15] presented a sequence to sequence model named SeqTriage. Since the model took into account fixed bugs which developers report themselves and the tossing sequence information, it outperformed the selected baseline methods. Besides, Mani *et al.* [16] considered the data noisy in the description of bug reports, which consists of unstructured text, code snippets, and stack trace making. By using DBRNN-A (short for an attention-based deep bidirectional recurrent neural network model) that learned syntactic and semantic features from long-word sequences in an unsupervised manner, they proposed a bug report representation algorithm to address the problem mentioned above.

Developer Relationship-Based Approach. Developer relationship-based approaches need to construct developer collaboration networks, which are sometimes referred to as tossing graphs. Jeong *et al.* [17] proposed a tossing graph model based on a Markov chain and the bug tossing process among developers. The experimental result indicated that the reassignment rate of new defects was reduced by about 72% when taking the structural features of the graph into account. Wu *et al.* [18] proposed a method based on the k -nearest neighbor (KNN) algorithm by ranking developer expertise. After constructing a developer collaboration network, the method ranks developers in terms of network measures, such as degree, out-degree, frequency, PageRank, betweenness, and closeness. Their empirical results showed that two network measures, out-degree and frequency, were more effective in predicting appropriate fixers. Besides, Zhang *et al.* [19] proposed an approach named KSAP using KNN search and heterogeneous proximity. KASP recommends possible fixers for a given bug report via automatically building a heterogeneous network of a bug repository and extracting meta-paths of developer collaborations in the network.

2.2 Network Embedding

In recent years, unsupervised learning-based network node representation (i.e., network embedding) methods have been proposed and proven to be more effective in capturing latent structural features than standard network metrics [20]. For single-layer networks, inspired by the idea of Word2Vec, each network node can be regarded as a word, and the sequence of walks on different nodes can be regarded as a sentence. Nodes of such

a network are then assigned to low-dimensional representations in a similar way to utilizing the skip-gram model [21] in Word2Vec.

Until now, researchers have proposed a few network embedding methods that embed nodes of a (single-layer) network to vectors and other low-dimensional representations while preserving the network structure. DeepWalk [22] is the first approach applying deep learning techniques which have achieved success in natural language processing to network analysis. It obtains the context sequence of a node by random walks and then optimizes the probability of neighbors around the node. Finally, DeepWalk outputs embedding results by using the skip-gram model. However, this method does not have any limitation on the direction and length of the search of neighbors, thus resulting in relatively high computational complexity.

To overcome the above shortcoming of DeepWalk, Node2Vec [23] sets two parameters p and q to control the width and breadth of random walks. Similarly, the large-scale information network embedding (LINE) method [24] extends the random walk strategy to the first-order and second-order similarity-induced weighted random walk, which can meet the embedding requirements of large-scale networks. Besides, Wang *et al.* [25] proposed a structural deep network embedding method, called SDNE, which exploits the first-order and second-order proximity jointly to preserve the local network structure and global network structure. For more information about other network embedding approaches, please refer to the survey made by Cui *et al.* [20]. Unfortunately, most of the existing methods of network embedding focus on single-layer network and do not consider multiplex networks.

Collaborative human activities such as distributed collaborative writing with wikis and software development, which usually take place on multiple layers rather than on a single one, can be better modeled by multiplex networks [26]. Multiplex networks, whose edges indicate various types of interactions belonging to different layers, represent a significant advance of network science in describing real-world networked systems [27]. Liu *et al.* [28] defined three Node2Vec-based random walk strategies on multiplex networks, including network aggregation, result aggregation, and layer co-analysis. Inspired by a specific random walk model, Guo *et al.* [29] investigated a new navigation strategy on multiplex networks. Significant analytical expressions were derived for the mean first-passage time and the average time to reach a node on these networks, using the spectral graph theory and stochastic matrix theory. Considering the information of multi-type relations, Zhang *et al.* [30] combined different types of relations while maintaining their distinctive properties by using a high-dimensional common embedding and a lower-dimensional additional embedding for each type of relation. Matsuno *et al.* [31] proposed an embedding method for multiplex networks named MELL, which was able to capture and characterize each layer's connectivity. In particular, this method exploited the overall structure effectively and was also capable of embedding both directed and undirected multiplex networks.

3 Preliminaries to This Study

3.1 Bug Tossing Graph and E-mail Communication Graph

Bug Report and Tossing Graph. Bug reports are one of the most valuable assets in the bug tracking system of an open-source software project. Each reported bug report is a semi-structured document that contains all the information about a software bug, which can be used to help developers find and fix the bug. Figure 1 illustrates an example of a bug report whose ID is 1532⁷ in the Eclipse project.

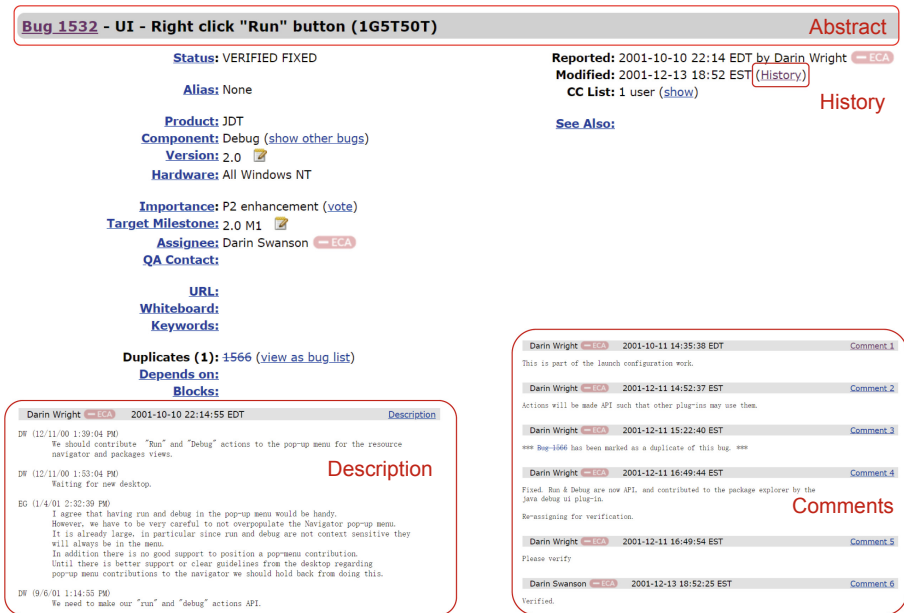


Fig. 1. An example of a bug report with ID 1532 in the Eclipse project.

As shown in Fig. 1, the *abstract* field present the primary information (or attributes) of the bug directly. For example, *product* and *component*, which are two necessary pre-defined attributes in this field, describe the primary and secondary categories, respectively, of a bug. The *history* field is a table which details the bug repair process. Figure 2 displays the modified history of the bug. There are five columns in the table, namely *Who*, *When*, *What*, *Removed*, and *Added*. The values of *Removed* and *Added* in a row represent a change to the value of *What*. According to modification records of the *assignee* field, we can extract bug tossing paths of resolved bug reports to form a tossing graph. Besides, a bug report provides a detailed description of the bug and developer comments in the natural language form to help find appropriate fixers.

⁷ https://bugs.eclipse.org/bugs/show_bug.cgi?id=1532.

Back to bug-1532

Who	When	What	Removed	Added
jeffmcaffer	2001-10-11 00:09:16 EDT	Assignee	Darin_Swanson	Darin_Wright
darin.eclipse	2001-10-11 14:35:38 EDT	Status	NEW	ASSIGNED
darin.eclipse	2001-10-12 22:48:16 EDT	Priority	P1	P2
darin.eclipse	2001-12-11 14:52:46 EST	Target Milestone	---	2.0 M1
darin.eclipse	2001-12-11 15:22:40 EST	CC		Darin_Swanson
darin.eclipse	2001-12-11 16:49:44 EST	Assignee	Darin_Wright	Darin_Swanson
		Status	ASSIGNED	NEW
darin.eclipse	2001-12-11 16:49:54 EST	Status	NEW	RESOLVED
		Resolution	---	FIXED
Darin_Swanson	2001-12-13 18:52:25 EST	Status	RESOLVED	VERIFIED

Fig. 2. The modified history of the bug 1532.

E-mail Communication Graph. Software developers always communicate with each other by e-mail to develop open-source software projects. Figure 3 illustrates an e-mail “Using ACTF on Linux”⁸ in the mailing list of the accessibility tools framework⁹ (ACTF), which is a subproject of the Eclipse Project. Developers can follow any e-mail in a mailing list after subscribing to the mailing list. As shown in Fig. 3, the fields *From* and *Date* record the sender and time of the e-mail, respectively. The *Follow-ups* field at the bottom of the figure indicates the developer who responded to this e-mail (i.e., Kentarou Fukuda). After entering the link in the *Follow-ups* field, we can see more information about the follow-up e-mail. After analyzing records in the archive of the ACTF project, we can construct an e-mail communication graph by extracting possible e-mail communication paths, each of which contains different developers with e-mail addresses corresponding to *From* and *Follow-Ups*.

[actf-dev] Using ACTF on Linux

- From: Yutaka HARA <yhara@xxxxxxxx>
- Date: Wed, 7 Dec 2016 14:22:25 +0900
- Delivered-to: actf-dev@eclipse.org
- User-agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Thunderbird/45.5.1

Basic Information

Hello,

I'm interested in running ACTF on Linux. Is there any plan to support Linux?

If I would try porting an ACTF-based application (say, actf.examples.htmlchecker), what will be the most difficult part?

Thank you,
Yutaka HARA <yhara@xxxxxxxx>

- Follow-Ups:
 - [Re: \[actf-dev\] Using ACTF on Linux](#)
 - From: Kentarou Fukuda

Follow-Ups

- Prev by Date: [\[actf-dev\] \[aer\] Monthly Problem Digest for Accessibility Tools Framework](#)
- Next by Date: [Re: \[actf-dev\] Using ACTF on Linux](#)
- Previous by thread: [\[actf-dev\] \[aer\] Monthly Problem Digest for Accessibility Tools Framework](#)
- Next by thread: [Re: \[actf-dev\] Using ACTF on Linux](#)
- Index(es):
 - [Date](#)
 - [Thread](#)

Fig. 3. An example of an e-mail in the mailing list of the ACTF project.

⁸ <https://www.eclipse.org/lists/actf-dev/msg00477.html>.

⁹ <https://www.eclipse.org/actf>.

3.2 Multiplex Network Measures

First of all, we introduce some notations of multiplex networks. A multiplex network system (MNS) consists of N nodes and M ($M \geq 2$) layers, each of which is a network that has the same nodes V but different edges [27]. The adjacency matrix of layer α of the MNS system is defined as $A^{[\alpha]} = \{a_{ij}^{[\alpha]}\} (\alpha \in [1, M])$. If there is an edge between nodes i and j on layer α , $a_{ij}^{[\alpha]} = 1$, otherwise 0. Then, the whole system can be expressed as $MNS = \{A^{[1]}, \dots, A^{[M]}\}$. If we take the weight of each edge into consideration, the weighted adjacency matrix of layer α is defined as $W^{[\alpha]} = \{w_{ij}^{[\alpha]}\}$, and the whole system can be represented as $MNS = \{W^{[1]}, \dots, W^{[M]}\}$. The definitions of other multiplex network measures are given in Eqs. (1)–(6).

Degree of node i on layer α : the number of links of node i to other nodes on layer α . This metric represents a developer’s activity level in a certain way of collaboration. Thus, the degree of node i in a multiplex network is a vector $\mathbf{k}_i = (k_i^{[1]}, \dots, k_i^{[M]})$.

$$k_i^{[\alpha]} = \sum_j a_{ij}^{[\alpha]}. \tag{1}$$

Overlapping degree of node i : the sum of the degree of node i on all the layers of a multiplex network. This metric represents a developer’s activity level in the whole multiplex network.

$$o_i = \sum_\alpha k_i^{[\alpha]}. \tag{2}$$

Strength of node i on layer α : the weighted degree of node i on layer α . Thus, the strength of node i can be represented as a vector $\mathbf{s}_i = (s_i^{[1]}, \dots, s_i^{[M]})$.

$$s_i^{[\alpha]} = \sum_j w_{ij}^{[\alpha]}. \tag{3}$$

Entropy of multiplex degree: the distribution of the degrees of node i among various layers. This metric represents the uniformity of the degree distribution of a node across different layers.

$$H_i = - \sum_{\alpha=1}^M \frac{k_i^{[\alpha]}}{o_i} \ln \left(\frac{k_i^{[\alpha]}}{o_i} \right). \tag{4}$$

The first (clustering) coefficient for node i : the ratio of the number of 2-triangles with a vertex at node i to the number of 1-triads centered at node i . Due to the popular saying, “the friend of your friend is my friend,” clustering coefficient is an essential network metric to measure the tendency of nodes to form triangles. In this study, a 2-triangle is defined as a triangle whose edges belong to two different layers, and a 1-triad centered in i is also defined as a triangle composed of nodes j , i , and k , in which both edge e_{ji} and edge e_{ik} are on the same layer.

$$C_{i,1} = \frac{\sum_{\alpha} \sum_{\alpha' \neq \alpha} \sum_{j \neq i, m \neq i} (a_{ij}^{[\alpha]} a_{jm}^{[\alpha']} a_{mi}^{[\alpha]})}{(M-1) \sum_{\alpha} k_i^{[\alpha]} (k_i^{[\alpha]} - 1)}. \quad (5)$$

Interdependence of node i : the multiplex contribution of node i to the reachability of each unit of a multiplex network. Reachability is also a crucial characteristic in graph theory and network science. In Eq. (6), σ_{ij} is the total number of shortest paths between nodes i and j in the multiplex network, and φ_{ij} is the number of shortest paths between the two nodes which make use of links on two or more layers.

$$\lambda_i = \sum_{j \neq i} \frac{\varphi_{ij}}{\sigma_{ij}}. \quad (6)$$

4 Bug Fixer Prediction Approach

4.1 Overall Framework

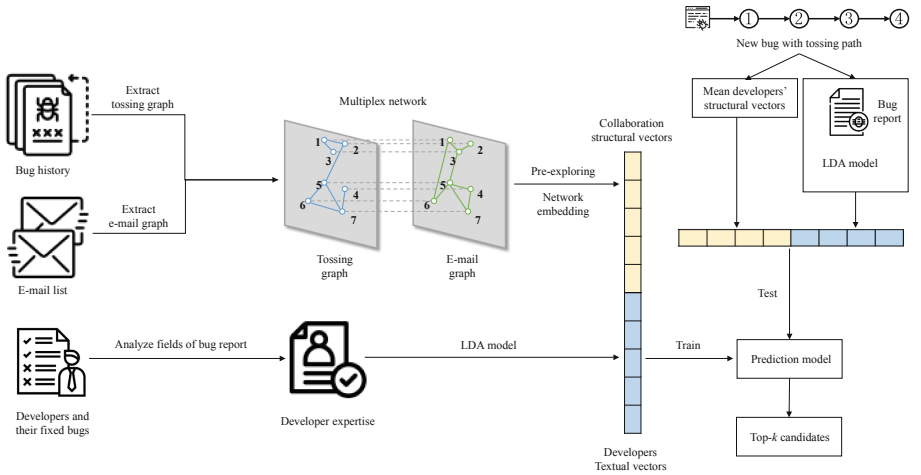


Fig. 4. The overall framework of our method.

The overall framework of our bug fixer prediction approach consists of three parts (see Fig. 4). The first part is *structural information processing*, which has three steps: (1) constructing a collaborative multiplex network; (2) calculating the measures of the multiplex network; and (3) using a new random walk strategy to obtain the structural information of all developers. The second part is *text information processing*, which has two steps: (1) collecting and preprocessing bug reports; and (2) obtaining text vectors of all bugs and developers by the LDA model. The third part is *prediction model*, which takes the structure features of developers and text features of bug reports as inputs and outputs possible k bug fixers. More details of the three parts, please refer to the following subsections.

4.2 Structural Information Processing

Building a Collaborative Multiplex Network. According to the definition of multiplex networks, we take the Eclipse project as an example to introduce the construction of a collaborative multiplex network (see Fig. 5). First, we removed duplicated and null records when building the tossing graph and e-mail communication graph. Second, we preserved the common nodes (i.e., the same developers) in the two graphs and removed other nodes and their corresponding edges. We then summed up the number of edges between each pair of nodes as the weight of the edge between the nodes. For example, in Fig. 5, the weight of the edge between *Denis* and *Olivia* is two. Third, we generated the collaborative multiplex network after mapping each developer in the two graphs into a unified node ID.

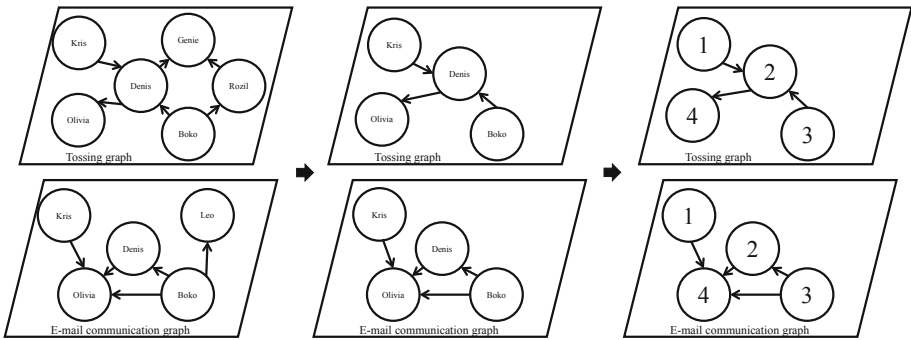


Fig. 5. An example of the construction of a collaborative multiplex network.

Multiplex Network Embedding. Most of the random walk methods were designed for single-layer networks. For random walks across layers in a multiplex network, the way of crossing layers is often random or based on degree importance, possibly leading to the loss of valuable information within the multiplex network. Therefore, we propose a new random walk strategy for collaborative multiplex networks. The proposed scheme takes into account five primary multiplex network measures, i.e., overlapping degree, node strength, clustering coefficient, interdependence, and multiplex degree entropy, in the process of random walks. More specifically, we can obtain a score by weighting these measures, which is used to choose the next-step node to generate a sequence of nodes when sampling node sequences in layer α of the multiplex network.

Suppose i and j denote the current node and the next-step node, respectively, on the same layer α . The score of node j is defined as follows:

$$score_j = (\omega_1 * o_j + \omega_2 * s_j^{[\alpha]} + \omega_3 * H_j + \omega_4 * C_{j,1} + \omega_5 * \lambda_j) * w_{ij}^{[\alpha]}, \quad (7)$$

$$s.t. \sum_{k=1}^5 \omega_k = 1$$

where $\omega_k \in [0, 1]$ is the weight of a multiplex network measure and $w_{ij}^{[\alpha]}$ represents the weight of the edge between nodes i and j . Nodes v_i are then generated according to the following distribution:

$$P(v_i = p | v_{i-1} = q) = \begin{cases} \frac{score_p}{Z} & \text{if } q \text{ has a link to } p \\ 0 & \text{otherwise} \end{cases}. \quad (8)$$

where Z is a normalizing constant. We set the same objective function as Node2Vec [23], described as follows:

$$\max_f \sum_{v \in V} [-\log \sum_{v \in V} \exp(f(u) \cdot f(v)) + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u)], \quad (9)$$

where $f : V \rightarrow \mathbb{R}^d$ is a mapping function from nodes to vection representations and $N_S(u) \subset V$ is a set of the neighbours of node u .

By employing the embedding algorithm based on the random walk strategy (see **Algorithm 1**), we can obtain the embedding result of each node in the multiplex network, which can be used as structural features of developers in this study.

Algorithm 1. The multiplex network embedding algorithm

Input: Two-layer collaborative multiplex network G , Dimensions d , Walks per node r , Walk length l , Context size k , Parameter weight vector ω

Output: Mapping function f for layer α

1: $G_s = \text{CountScore}(G, \omega)$ // Update G with each node's score calculated using Eq. (7)

2: Initialize *walks* to Empty

3: **for** *iter* = 1 **to** r **do**

4: **for all** *nodes* $u \in V$ **do**

5: *walk* = *RandomWalk*(G_s, α, u, l)

6: Append *walk* to *walks*

7: $f = \text{StochasticGradientDescent}(k, d, \text{walks})$

8: **return** f

RandomWalk(G_s, α, u, l)

1: Initialize *walk* to [u]

2: **for** *walk_iter* = 1 **to** l **do**

3: $V_{curr} = \text{GetNearestNeighbors}(\text{walk}[-1], G_s, \alpha)$ // Layer α is also a graph $(V, W^{[\alpha]})$

4: Find node s with the highest score in V_{curr}

5: Append s to *walk*

6: **return** *walk*

4.3 Text Information Processing

A previous study [32] showed that the *summary*, *product*, *component*, *description*, and *comments* fields in a bug report are significant textual descriptions of a bug. Therefore, we collect and preprocess the above fields of each fixed bug and then integrate them into a text representation (or called text information). Finally, we obtain text features of each bug after inputting the text information to the LDA model [8].

Text Preprocessing. As mentioned above, we first clean the text information of each bug report by using Gensim¹⁰. Moreover, numbers, punctuations, and special characters are removed, and stop words and stems for long texts are deleted. More details of text preprocessing, please refer to previous studies [9–13, 32].

Text Feature Extraction. Considering the success of topic models in automatic bug triaging, in this study, we also utilize the LDA model to extract text features of bug reports. The LDA model can capture themes (or topics) of each bug in the whole dataset in the form of a probability distribution, and it clusters (or groups) bugs according to the topics extracted from the text information. Besides, the topics of bugs fixed by a developer can represent the expertise or skills of the developer. Similarly, we input all bugs fixed by a developer to the LDA model, and the output of the LDA model is used to represent the professional skills of the developer.

4.4 Fixer Prediction Model

After extracting structure and text features, we train a fixer prediction model, which is essentially a multi-class classification model, for new bug reports. In theory, the fixer prediction model can be trained by any classification algorithm. For example, support vector machines (SVMs) have been used as a standard classification model and proven to be effective in automatic bug triaging in many previous studies [2, 10, 17, 32]. In this study, we utilize a multilayer perceptron (MLP), which is a typical class of feedforward artificial neural networks, to train the fixer prediction model.

5 Experimental Setups

5.1 Research Questions

In this study, we attempt to answer the following two research questions.

RQ1: Does the proposed network embedding (structure features learning) method perform better than the selected baseline approaches?

The first research question aims to test the effectiveness of the proposed random walk strategy designed based on multiplex network measures and network embeddings. Since it is hard to evaluate network embedding results obtained by different approaches directly, we evaluate the efficacy of the approaches under discussion on two binary

¹⁰ <https://radimrehurek.com/gensim>.

classification tasks, i.e., link prediction and node classification. Note that we perform these two tasks on the layer of bug tossing regarding the primary goal of this study.

RQ2: Does the proposed bug fixer prediction (multi-class classification) model outperform the selected baseline approaches?

The primary goal of the second research question is to test the effectiveness of the proposed bug fixer prediction model. We will compare the performance of different classification models when inputting the same features. In the context of this research question, a bug’s fixer is the ultimate developer who fixes the bug.

5.2 Data Collection

The experimental datasets used in this study were collected using Bugzilla, a popular bug tracking system. In the whole lifecycle of each defect, its status is ever-changing. Therefore, in this study, we selected only fixed bugs, whose statuses were “RESOLVED” or “CLOSED” and bug resolution was “FIXED,” to crawl their *abstract*, *description*, *comments*, *product*, *component*, and *history* fields. Finally, we collected 200,000 bugs (record number from 1 to 357553) and 200,000 bugs (record number from 93 to 782996) from the Eclipse project and the Gnome project, respectively.

Large-scale open-source software projects often set up mailing lists for developers to communicate with each other via e-mail. A developer can send e-mails through a mailing list to inform any developer who has subscribed to the mailing list. The projects of Eclipse and Gnome have a large number of mailing lists and e-mails. As shown in Fig. 3, each e-mail contains *From*, *Follow-Ups*, *Date*, *Follow-Ups-Date*, and other fields. We crawled 30,338 and 1,748,687 e-mail records created before August 2017 from public subprojects in the two projects, respectively.

We then constructed two collaborative multiplex networks following the procedures introduced in Subsect. 4.2. According to the definition of multiplex networks, the two-layer collaborative multiplex networks were smaller than the combination of a tossing graph and an e-mail communication graph. The statistics of the two constructed network are shown in Table 1, where TG, ECG, and LSCC denote tossing graph, e-mail communication graph, and the largest strongly connected component, respectively.

Table 1. Statistics of the experimental datasets.

Dataset	#Nodes	#Edges (TG/ECG)	Transitivity (TG/ECG)	#LSCC Nodes (TG/ECG)
Eclipse	716	6,954 (4,041/2,913)	0.067/0.054	323/594
Gnome	916	20,310 (1,280/19,030)	0.029/0.186	115/893

5.3 Baseline Approaches

We use the following four network embedding (structure features learning) methods as baseline approaches to answer RQ1.

DeepWalk. DeepWalk (Perozzi *et al.* 2014) [22] assumes that the space formed by the corresponding vectors of nodes can accurately represent the actual relationship

between nodes in a network. It transforms network nodes into a linear sequence, regards the hierarchical softmax of the skip-gram model as an objective function, and finally obtains the corresponding vector of each node.

LINE. LINE (Tang *et al.* 2015) [24] is an improved network embedding method based on DeepWalk. It preserves the local and global properties of a network by taking into account the first-order proximity and second-order proximity. Moreover, this approach can be applied to different types of networks and is useful for large-scale network embedding.

Node2Vec. Inspired by the idea of maximizing the probability of the neighbors of network nodes, Node2Vec (Grover *et al.* 2016) [23] combines the depth-first search and breadth-first search. It uses return parameter p and in-out parameter q to explore the neighbors of nodes flexibly. At the same time, this approach satisfies the local and global attributes of a network, thus making random walks relatively controllable.

PMNE. PMNE (Liu *et al.* 2017) [28] proposes three patterns based on Node2Vec, which combine all the layers of a multiplex network. *Network aggregation* merges nodes and edges on multiple layers directly to form a new network. Node2Vec takes the new network as an input. *Result aggregation* embeds each layer of a multiplex network with Node2Vec and then concatenate s node vectors on different layers to obtain the final embedding result. *Layer co-analysis* adds a parameter r to Node2Vec to determine whether it crosses layers in the random walk process. The final sequence of nodes is a cross-layer sequence embedded by the skip-gram model.

As for RQ2, the proposed MLP model uses the *softmax* function as an activation function for the output layer. Because the SVM algorithm and Cosine similarity have been widely used as classification models in many previous studies, we also choose them as baseline models for comparison in this study.

5.4 Configurations of Tasks and Parameters

To answer RQ1, we used five-fold cross-validation in the link prediction and node classification tasks for bug tossing. In the link prediction task, the cosine similarity between two nodes represented by embedding vectors determines whether there is a link between the two nodes. In our experiments, the decision rule is that a cosine similarity value greater than 0.5 indicates a link. Besides, we randomly selected the same number of non-existent edges from the existing network as negative samples for training, and the negative sample size was set to five.

In the node classification task, more specifically, a multi-label classification task, embedding vectors of nodes were inputted to an SVM model [33] implemented by the tool `sklearn`¹¹. The SVM model outputs a binary classification result to determine whether the target developer corresponding to a given node is a skilled developer in fixing software defects.

The parameter weight vectors ω were set to [0,0,0.3,0.4,0.3] and [0,0,0.9,0,0.1] for Eclipse and Genome, respectively. The weights were obtained from their respective largest strongly connected components. The embedding dimension of each method was

¹¹ <https://scikit-learn.org/stable>.

set to 200. Since LINE has two embedding settings, we set each of the LINE embedding dimension to 100 and concatenated them directly to form the final embedding. Besides, the window size of random walks was set to ten, and other parameters of our method used in the experiments were set as the same as Node2Vec unless specified otherwise.

To answer RQ2, we employed five-fold incremental learning [2, 13, 32] to compare the performance of different classification models. The topic number of the LDA model was set to 200. We also used the tool sklearn to implement the MLP model with default parameter settings. Note that we optimized the log-loss function of this model using stochastic gradient descent (SGD).

5.5 Evaluation Metrics

To answer RQ1, we utilize the area under the curve (AUC) [34] as an evaluation metric, which has been used by a previous study [23]. As with previous work [13, 16] that studied RQ2, we choose $Accuracy@k$ as an evaluation metric to measure how many fixers are hit by the top k recommend developers.

6 Experimental Results

6.1 Answer to RQ1: Node Representation Capability

Table 2 shows the experimental results of the two tasks on the Eclipse and Gnome datasets. NA, RA, and CA represent network aggregation, result aggregation, and layer co-analysis, respectively, of the PMNE method. The best result is highlighted in bold.

Table 2. Results of link prediction and node classification in term of AUC (mean \pm s.d.).

Approach	Link prediction		Node classification	
	Eclipse	Gnome	Eclipse	Gnome
DeepWalk	0.639 \pm 0.007	0.482 \pm 0.009	0.720 \pm 0.123	0.536 \pm 0.022
LINE	0.708 \pm 0.011	0.581 \pm 0.009	0.720 \pm 0.000	0.510 \pm 0.000
Node2Vec	0.639 \pm 0.007	0.482 \pm 0.009	0.704 \pm 0.016	0.518 \pm 0.012
PMNE (NA)	0.644 \pm 0.011	0.630 \pm 0.008	0.794 \pm 0.019	0.519 \pm 0.012
PMNE (RA)	0.691 \pm 0.009	0.582 \pm 0.011	0.817 \pm 0.012	0.531 \pm 0.043
PMNE (CA)	0.707 \pm 0.010	0.641 \pm 0.004	0.818 \pm 0.016	0.527 \pm 0.015
Ours	0.731 \pm 0.004	0.656 \pm 0.002	0.823 \pm 0.007	0.625 \pm 0.065

As shown in Table 2, our method achieves the best results in both link prediction and node classification tasks. Because DeepWalk, LINE, and Node2Vec are used for single-layer networks, they cannot leverage useful information from the other layer in the collaborative multiplex network. Therefore, in the two tasks, the results of the three methods were, in general, worse than those of PMNE and our approach. Although the PMNE method merged two-layer embedding results, its result was still worse than that

of our approach. The main reason for the difference is that PMNE does not consider multiplex network measures when designing the random walk strategy. The results indicate that these network measures can represent the connectivity and transitivity of nodes in the whole multiplex network better. Due to limitations of space, we do not show the results of parameter sensitivity.

6.2 Answer to RQ2: Fixer Prediction Performance

In this experiment of multi-class classification, the Eclipse dataset contains 531 fixers and 98,301 bug reports, and the Gnome dataset has 603 fixers and 19,701 bug reports. We compared the prediction performance of different models in two cases: considering “S” (short for structural features of developers) and considering “S + T” (short for both structural features of developers and text features of bug reports). The main results of the two datasets using five-fold incremental learning are presented in Tables 3 and 4, respectively. Note that the value of k ranges from one to five. The best result is highlighted in bold.

Table 3. Accuracy@ k values of different models on the Eclipse dataset (mean \pm s.d.).

Model	Top-1	Top-2	Top-3	Top-4	Top-5
DeepWalk	0.585 \pm 0.006	0.706 \pm 0.007	0.749 \pm 0.008	0.775 \pm 0.009	0.792 \pm 0.008
Node2Vec	0.580 \pm 0.002	0.708 \pm 0.008	0.751 \pm 0.009	0.776 \pm 0.010	0.793 \pm 0.010
LINE	0.593 \pm 0.009	0.718 \pm 0.009	0.760 \pm 0.010	0.784 \pm 0.009	0.800 \pm 0.009
PMNE (NA)	0.581 \pm 0.009	0.712 \pm 0.003	0.756 \pm 0.004	0.782 \pm 0.003	0.798 \pm 0.003
PMNE (RA)	0.561 \pm 0.014	0.689 \pm 0.007	0.729 \pm 0.008	0.751 \pm 0.009	0.766 \pm 0.008
PMNE (CA)	0.593 \pm 0.007	0.726 \pm 0.005	0.772 \pm 0.007	0.798 \pm 0.007	0.815 \pm 0.006
Ours	0.600 \pm 0.007	0.729 \pm 0.003	0.775 \pm 0.003	0.800 \pm 0.002	0.817 \pm 0.002
Cosine	0.064 \pm 0.004	0.102 \pm 0.003	0.145 \pm 0.002	0.163 \pm 0.001	0.177 \pm 0.001
SVM	0.597 \pm 0.007	0.729 \pm 0.003	0.774 \pm 0.001	0.799 \pm 0.001	0.816 \pm 0.001
MLP	0.614 \pm 0.005	0.720 \pm 0.007	0.799 \pm 0.005	0.813 \pm 0.002	0.830 \pm 0.002

In each of the two tables, the first part that ranges from the 2nd row to the 8th row represents the case of “S,” and the second part that ranges from the 9th row to the 11th row represents the case of “S + T.” The structure features extracted by each of the five approaches were inputted into an MLP classification model to predict k possible fixers. As shown in Tables 3 and 4, the proposed network embedding method outperforms the four baseline approaches in both the two datasets. In particular, for the Eclipse dataset, our method achieved a 60.0% accuracy when recommending only one developer ($k = 1$). If $k = 5$, the prediction accuracy increased to 81.7%. The results further demonstrate the effectiveness of our method that learns structure features from multiplex networks.

In the case of “S + T,” we inputted the same features set, including structure features obtained using our method and text features extracted by the LDA model, to three classification models. As shown in Tables 3 and 4, the MLP model performs

Table 4. *Accuracy@k* values of different models on the Gnome dataset (mean \pm s.d.).

Model	Top-1	Top-2	Top-3	Top-4	Top-5
DeepWalk	0.551 \pm 0.004	0.642 \pm 0.002	0.677 \pm 0.002	0.705 \pm 0.003	0.730 \pm 0.004
Node2Vec	0.552 \pm 0.005	0.641 \pm 0.002	0.675 \pm 0.003	0.705 \pm 0.005	0.726 \pm 0.006
LINE	0.555 \pm 0.004	0.635 \pm 0.002	0.670 \pm 0.002	0.699 \pm 0.003	0.723 \pm 0.004
PMNE (NA)	0.561 \pm 0.005	0.648 \pm 0.004	0.687 \pm 0.004	0.714 \pm 0.005	0.734 \pm 0.005
PMNE (RA)	0.551 \pm 0.010	0.639 \pm 0.008	0.676 \pm 0.006	0.705 \pm 0.006	0.723 \pm 0.005
PMNE (CA)	0.558 \pm 0.005	0.647 \pm 0.002	0.685 \pm 0.003	0.715 \pm 0.003	0.737 \pm 0.002
Ours	0.560 \pm 0.006	0.651 \pm 0.002	0.687 \pm 0.002	0.716 \pm 0.003	0.740 \pm 0.004
Cosine	0.047 \pm 0.005	0.087 \pm 0.003	0.136 \pm 0.002	0.143 \pm 0.002	0.155 \pm 0.002
SVM	0.562 \pm 0.004	0.647 \pm 0.002	0.685 \pm 0.003	0.713 \pm 0.003	0.735 \pm 0.004
MLP	0.586 \pm 0.007	0.683 \pm 0.004	0.731 \pm 0.005	0.756 \pm 0.004	0.775 \pm 0.003

better than the other two models. When k was equal to one, the accuracy of the MLP model was, on average, increased by 3.5% compared with that of the SVM model. Moreover, the improvement increased to 3.6% when k reached five. However, the concatenation of structure features and text features did not contribute to a significant improvement in the performance of bug fixer prediction (see the difference between “Ours” and “MLP”), which deserves further investigation on feature combination and dimension reduction.

7 Threats to Validity

Although this study achieves some useful results on the Eclipse and Genome projects, there are still potential threats to the validity of the study that may affect the results, mainly including the internal validity and external validity.

The *internal validity* concerns the explanation (or causality) of the results within the context of this study. It is hard to evaluate network embedding results directly. As with a few previous studies [22–24, 28], we evaluated different methods’ embedding results in the link prediction and node classification tasks under the assumption that better node representations contribute to higher performance in the tasks. Besides, the parameters of the baseline approaches were set as default. Therefore, computational optimization or parameter fine-tuning may change the experimental results.

The *external validity* focuses on the generalizability of the proposed network embedding and fixer prediction approaches outside the context of this study. First, due to the strict definition of multiplex networks [27], i.e., a multiplex network’s layers have different types of edges with the same nodes, the conclusions of this study do not apply to single-layer tossing graphs [17] or heterogeneous information networks [19]. Second, the performance of our methods on other large-scale software projects is yet to be tested. Therefore, their advantages over the selected baseline approaches on other software projects remain unexplored. Third, since commonly-used network embedding and classification algorithms were selected as the baseline methods in our experiments, we are unable to determine whether the proposed approaches can outperform some recently-proposed approaches, such as graph convolutional networks [35].

8 Conclusion and Future Work

Software bug triaging is a necessity in software development and maintenance. In this paper, we study the problem of automatic bug triaging from the perspective of multiplex networks rather than from single-layer tossing graphs. Due to the multi-layer network characteristics of human cooperative behavior in bug triaging, this study constructs a collaborative multiplex network composed of two layers corresponding to bug tossing and e-mail communication. We then present a new random walk strategy based on some multiplex network measures (e.g., overlapping degree, node strength, and the entropy of the multiplex degree) to generate network embeddings for nodes, which can be used as structural features of developers. Besides, we propose a bug fixer prediction model that takes structural features of developers and text features of bug reports as inputs. According to the experiments on two large-scale open-source software projects with a large number of bug reports and e-mails, we have two conclusions. On the one hand, the proposed network embedding algorithm based on the random walk strategy performed better than four standard network embedding methods in the link prediction and node classification tasks. On the other hand, the proposed prediction model outperformed the selected baseline approaches in predicting the right fixers for target bug reports.

In addition to testing the generalizability of the proposed approach in more large-scale software projects, our future work includes two aspects. First, we will extend the collaborative multiplex network to k -layer collaborative networks where $k \geq 3$ by leveraging other available social relationships between developers such as “Follow” on GitHub¹². Second, due to the complexity of the proposed network embedding algorithm, we will investigate more efficient deep feature learning algorithms to extract structure features of nodes in multiplex networks that have greater than or equal to three layers.

Acknowledgment. This work was partially supported by the National Key Research and Development Program of China (No. 2018YFB1003801), National Science Foundation of China (Nos. 61832014, 61672387, and 61572371), and Natural Science Foundation of Hubei Province of China (No. 2018CFB511).

References

1. Aberdour, M.: Achieving quality in open source software. *IEEE Softw.* **24**(1), 58–64 (2007)
2. Bhattacharya, P., Neamtiu, I., Shelton, C.R.: Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *J. Syst. Softw.* **85**(10), 2275–2292 (2012)
3. Helming, J., Arndt, H., Hodaie, Z., et al.: Semi-automatic assignment of work items. In: *The 5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pp. 149–158. SciTePress (2010)

¹² <https://github.com/>.

4. Anvik, J., Murphy, G.C.: Reducing the effort of bug report triage: recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* **20**(3), 10:1–10:35 (2011)
5. Shokripour, R., Anvik, J., Kasirun, Z.M., et al.: Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In: The 10th Working Conference on Mining Software Repositories (MSR), pp. 2–11. IEEE (2013)
6. Jonsson, L., Borg, M., Broman, D., et al.: Automated bug assignment: ensemble-based machine learning in large scale industrial contexts. *Empirical Softw. Eng.* **21**(4), 1533–1578 (2016)
7. Mani, S., Nagar, S., Mukherjee, D., et al.: Bug resolution catalysts: identifying essential non-committers from bug repositories. In: The 10th Working Conference on Mining Software Repositories (MSR), pp. 193–202. IEEE (2013)
8. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. *J. Mach. Learn. Res.* **3**, 993–1022 (2003)
9. Murphy, G., Cubranic, D.: Automatic bug triage using text categorization. In: The 16th International Conference on Software Engineering & Knowledge Engineering (SEKE), pp. 92–97. KSI Research Inc. and Knowledge Systems Institute Graduate School (2004)
10. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: The 28th International Conference on Software Engineering (ICSE), pp. 361–370. ACM (2006)
11. Xie, X., Zhang, W., Yang, Y., et al.: Dretom: developer recommendation based on topic models for bug resolution. In: The 8th International Conference on Predictive Models in Software Engineering (PROMISE), pp. 19–28. ACM (2012)
12. Naguib, H., Narayan, N., Brügge, B., et al.: Bug report assignee recommendation using activity profiles. In: The 10th Working Conference on Mining Software Repositories (MSR), pp. 22–30. IEEE (2013)
13. Xia, X., Lo, D., Ding, Y., et al.: Improving automated bug triaging with specialized topic model. *IEEE Trans. Softw. Eng.* **43**(3), 272–297 (2017)
14. Lee, S.R., Heo, M.J., Lee, C.G., et al.: Applying deep learning based automatic bug triager to industrial projects. In: The 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 926–931. ACM (2017)
15. Xi, S., Yao, Y., Xiao, X., et al.: An effective approach for routing the bug reports to the right fixers. In: The 10th Asia-Pacific Symposium on Internetware (INTERNETWARE), p. 11. ACM (2018)
16. Mani, S., Sankaran, A., Aralikalte, R.: Deeptrriage: exploring the effectiveness of deep learning for bug triaging. In: The ACM India Joint International Conference on Data Science and Management of Data (COMAD/CODS), pp. 171–179. ACM (2019)
17. Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: The 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 111–120. ACM (2009)
18. Wu, W., Zhang, W., Yang, Y., et al.: Drex: developer recommendation with k-nearest-neighbour search and expertise ranking. In: The 18th Asia-Pacific Software Engineering Conference (APSEC), pp. 389–396. IEEE (2011)
19. Zhang, W., Wang, S., Wang, Q.: KSAP: an approach to bug report assignment using KNN search and heterogeneous proximity. *Inf. Softw. Technol.* **70**, 68–84 (2016)
20. Cui, P., Wang, X., Pei, J., et al.: A survey on network embedding. *IEEE Trans. Knowl. Data Eng.* **31**(5), 833–852 (2018)
21. Guthrie, D., Allison, B., Liu, W., et al.: A closer look at skip-gram modelling. In: The 5th International Conference on Language Resources and Evaluation (LREC), pp. 1222–1225. European Language Resources Association (2006)

22. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 701–710. ACM (2014)
23. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 855–864. ACM (2016)
24. Tang, J., Qu, M., Wang, M., et al.: LINE: large-scale information network embedding. In: The 24th International Conference on World Wide Web, pp. 1067–1077. ACM (2015)
25. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 1225–1234. ACM (2016)
26. Gómez-Gardenes, J., Reinares, I., Arenas, A., et al.: Evolution of cooperation in multiplex networks. *Sci. Rep.* **2**, 629 (2012)
27. Boccaletti, S., Bianconi, G., Criado, R., et al.: The structure and dynamics of multilayer networks. *Phys. Rep.* **544**(1), 1–122 (2014)
28. Liu, W., Chen, P.Y., Yeung, S., et al.: Principled multilayer network embedding. In: The 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 134–141. IEEE (2017)
29. Guo, Q., Cozzo, E., Zheng, Z., et al.: Levy random walks on multiplex networks. *Sci. Rep.* **6**, 37641 (2016)
30. Zhang, H., Qiu, L., Yi, L., et al.: Scalable multiplex network embedding. In: The 27th International Joint Conference on Artificial Intelligence (IJCAI), pp. 3082–3088. IJCAI Organization (2018)
31. Matsuno, R., Murata, T.: MELL: effective embedding method for multiplex networks. In: Companion of The Web Conference 2018 on The Web Conference 2018 (WWW Companion), pp. 1261–1268. ACM (2018)
32. Wu, H., Liu, H., Ma, Y.: Empirical study on developer factors affecting tossing path length of bug reports. *IET Softw.* **12**(3), 258–270 (2018)
33. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)
34. Fawcett, T.: An introduction to ROC analysis. *Pattern Recognit. Lett.* **27**(8), 861–874 (2006)
35. Ghorbani, M., Baghshah, M.S., Rabiee, H.R.: Multi-layered Graph Embedding with Graph Convolutional Networks. *Computing Research Repository (CoRR)*, arXiv: 1811.08800 (2018)