Chinese Journal of Electronics
Vol.25, No.6, Nov. 2016

# BAHA: A Novel Approach to Automatic Bug Report Assignment with Topic Modeling and Heterogeneous Network Analysis*

ZHANG Wen[1], WANG Song[2] and WANG Qing[3]

(1. *School of Economics and Management, Beijing University of Chemical Technology, Beijing 100019, China*)

(2. *Department of Electrical and Computer Engineering at University of Waterloo, Ontario N2L 3G1, Canada*)

(3. *Laboratory of Internet Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*)

**Abstract — We propose an approach called Bug report assignment with topic modeling and heterogeneous network analysis (BAHA) to automatically assign bug reports to developers. Existing studies adopt social network analysis to characterize the collaboration of developers. The networks used in these studies are all homogenous. In real practice of bug resolution, different developers collaborate on different bug reports that makes the homogenous network unable to capture this information. We use heterogeneous network to describe the relations between reporters, bug reports and developers to characterize developers' collaboration. Experiments on Eclipse JDT project show that BAHA outperforms the state of art methods on automatic bug report assignment.**

**Key words — Bug report assignment, Topic model, Heterogeneous network, Bug report tracking.**

## I. Introduction

Open bug repository, such as Bugzilla**, JIRA*** and GNATS****, has been widely adopted in software development and maintenance for software quality improvement[1,2]. With hundreds of bug reports coming to the open bug repository each day, it is hard to keep track of all the developers and their expertise within a large project[3,4]. For instance, about 200 bugs are filed to Eclipse bug repository per day near its release dates, and for Debian project, this number is about 150[5]. Moreover, about two person-hours per day have to be spent on bug triage in Eclipse project and nearly 25% of Eclipse bug reports are reassigned due to inaccurate manual bug assignment and, it takes about 40 days to assign a new bug report to an appropriate developer in Eclipse project[6].

To reduce the workload of bug report assignment, researchers propose several methods based on text analysis of the bug report[7,8]. They usually assume that the textual contents of bug reports are of high quality since it is the solely clues for the classifier to learn the correspondences between bug reports and developers. However, this assumption is often not the case in real practice[9−11]. For instance, Bettenburg *et al.*[9] reported that bug reporters often provide inadequate even incorrect information that seriously slow down the process of bug report assignment. Moreover, the textual contents of bug reports often contain large amount of source code fragments, stack traces, hyper links even messy code that cannot be properly processed by the state of art Natural language processing (NLP) techniques.

To address the problem, on the one hand, researchers attempt to adopt more advanced NLP techniques to deal with the textual contents[5,12]. On the other hand, researchers are intending to utilize the collaboration of developers to enhance bug report assignment[13,14]. An available space that needs more work is that currently, all the networks used to model developer collaboration in bug repositories are homogenous[15]. That is, all the nodes in the network are of the same type as developers and all the links in the networks denote the same relation as co-commenting same bug reports. However, in real practice

*Manuscript Received Nov. 20, 2014; Accepted Nov. 19, 2015. This work is supported by the National Natural Science Foundation of China (No.71101138, No.61379046, No.61432001), the Beijing Natural Science Fund (No.4122087), the Fundamental Research Fund for the Central Universities in BUCT.

**http://www.bugzilla.org/*

***http://www.atlassian.com/software/jira/overview*

****http://www.gnu.org/software/gnats/*

 DOI:10.1049/cje.2016.08.012

of bug resolution, this is not the case. In fact, the collaboration of developers are not directly interacted as that in paper citation network and email network[16] but with indirect relation because, a developer submit a bug report and then other developers comment on the bug report. That is to say, the developers in bug repositories manipulate bug reports directly rather than interact with each other directly as they behave in social media like Facebook and Twitter.

Taking Fig.1 for an example, where $D_1$, $D_2$, $D_3$ and $D_4$ are 4 developers and $B_1$, $B_2$, $B_3$ and $B_4$ are 4 bug reports, we can see that in heterogenous network, $D_4$ has more participation in bug resolution than $D_1$ but, in homogenous network, the places of $D_1$ and $D_4$ are symmetric that means $D_1$ and $D_4$ have same importance. If the homogenous network is adopted in bug report assignment, then $D_1$ and $D_4$ should be at the same place. However, if both $D_1$ and $D_4$ are candidate developers for a new bug report, $D_4$ is obviously more proffered than $D_1$ because $D_4$ is more versatile.

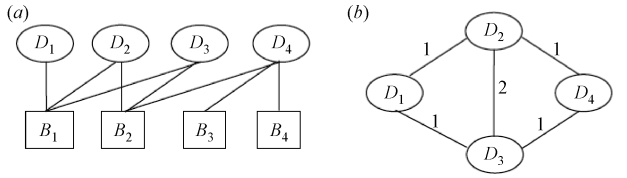


Fig. 1. Heterogenous and homogenous networks for describing developer collaboration. (*a*) Heterogeneous network; (*b*) Homogenous network

Following this line of thought, we propose a novel approach called BAHA (Bug report assignment with topic modeling and heterogeneous network analysis) to automatically assign bugs to developers. Firstly, we use LDA topic model[17,18] to analyze textual contents of bug reports. Secondly, the heterogeneous network[19] is adopted to capture structural information inherent in the bug reports. Finally, we combine the outcome of analysis of textual contents and heterogenous network to assign bug reports to developers. To evaluate the performance of BAHA, we conduct extensive experiments using the bug reports of Eclipse JDT project. Experimental results show that BAHA can improve the recall by 19.19% at most compared to the state of art methods for bug report assignment.

The rest of this paper is organized as follows. Section II proposes BAHA for bug report assignment. Section III conducts the experiments to examine the performances of BAHA and the state of art methods in bug report assignment. Section IV discusses the experimental results. Section V concludes the paper.

## II. BAHA-The Proposed Approach

### 1. The framework of the approach

Fig.2 shows the proposed bug triage approach called BAHA. The dotted line shows the training process (above) and the solid line (below) shows the process of handling a new bug report. Firstly, we use the textual contents of historically-resolved bug reports to train LDA topic model. Secondly, based on the trained LDA topic model and the structural information extracted from the historically-resolved bug reports, we train the RankClass model. Thirdly, when a new bug report is incoming, the trained LDA model and RankClass model will be used to calculate its topic distribution. Finally, the topic distribution of the bug report and the developers' expertise scores on the topics are combined to decide the candidate developers for the new bug report. Overall, BAHA includes four steps as follows.

1) Extract text contents of bug reports and calculate topic distribution of the bug reports using LDA model.
2) Build heterogeneous network and train the RankClass model initialized by the output of LDA model.
3) Calculate the topic distribution of bug reports by combining the outputs of RankClass model and LDA model.
4) Combine the topic distribution of bug reports and expertise scores to produce the candidate developer list for new bug reports.

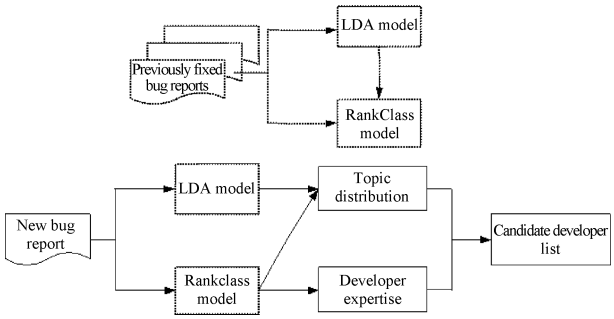


Fig. 2. The overall structure of BAHA

### 2. LDA topic modeling for bug reports

Following Anvik *et al.*[8], we use the bug reports whose status are RESOLVED, VERIFIED or CLOSED and their resolution status are FIXED to train the LDA model. We extract the textual contents including on-line summaries and full text descriptions from the bug reports in training set. Tokenization, stop words and non-alphabetic token elimination[20] are conducted to preprocess the textual contents of bug reports.

An open source LDA tool called GibbsLDA++[21] with Gibbs sampling is used to infer the the topic-document distribution matrix $\boldsymbol{\theta}_k (1 \leq k \leq K)$ and topic-word distribution matrix $\boldsymbol{\varphi}_m (1 \leq m \leq M$. The default values from GibbsLDA++ as $\alpha = 50/K$ and $\beta = 0.01$

are used as the predefined parameters. To the best of our knowledge, there is no known method of estimating a priori number of topics in LDA which can best represent the textual contents. Following Blei *et al.*[17], we use perplexity measure to estimate the LDA model for a range of $K$ topics and chose the best one. The lower perplexity of the distributions of words given topics and documents, the better is the quality of LDA modeling. In our experiments on the Eclipse JDT bug reports, we find that the number of topics $K = 10$ is the best to model the topics of the bug reports.

**3. Training RankClass model**

The input of RankClass algorithm includes the heterogeneous network extracted from bug reports and the prior knowledge of topics given the bug reports, *i.e.*, $\{P(x|T(x), K)^0\}_{k=1}^K$, that makes some bug reports "labeled" with the prior topics. The heterogeneous network of BAHA contains 3 types of objects as reporters, bug reports and developers, *i.e.*, $T = 3$ in Section II.3.

The prior knowledge of topics given the bug reports for RankClass algorithm is derived by LDA topic modeling. We use the topic with the highest probability to label the bug report. We admit that this handling of labeling topics to bug reports directly is a little arbitrary since, in some cases, the differences of probabilities of topics given bug reports are not always very obvious. However, Ji *et al.*[22] shows that RankClass is theoretically robust and even if the quality of initial "labeled" bug reports are not very high. This is also the reason we adopt RankClass algorithm combined with LDA to consider textual contents and structural information of bug reports simultaneously.

Following Ji *et al.*[22], we use the top 0.5% of the training bug reports with largest differences (calculated by the difference of the largest probability and the second largest probability) to initialize the RankClass algorithm. The output of RankClass algorithm are $\{P(x|T(x), k)\}_{k=1}^K$ and $\{P(k|x, T(x))\}_{k=1}^K$.

**4. Adjust the topic distribution of bug reports**

Since reporters tend to submit bug reports of same components, it is reasonable to use the topic distribution of bug reports derived from heterogenous network analysis by RankClass model, *i.e.*, $\{P(k|x, T(x))\}_{k=1}^K$ with $T(x) =$ "*bugreport*", to adjust the topic distribution of the bug report derived from LDA modeling in Section IV.2. That is, the final topic distribution of a bug report is defined as Eq.(1):

$$\theta_{bug} = r \times \theta_{\text{LDA},bug} + (1 - r) \times \theta_{HN,sub} \tag{1}$$

Here, $\theta_{bug}$ is the final topic distribution of bugs. $\theta_{\text{LDA},bug}$ is the topic distribution derived from LDA modeling, *i.e*, the topic-document matrix $\boldsymbol{\theta}_k (1 \leq k \leq K)$. $\theta_{HN,sub}$ is the topic distribution of bug reports on reporters derived from RankClass algorithm, *i.e.*, $\{P(k|x, T(x))\}_{k=1}^K$ with $T(x) =$ "*reporter*". The importance of $\theta_{\text{LDA},bug}$ and $\theta_{HN,sub}$ is determined by the weight $r$, which ranges from 0 to 1. If the quality of textual contents of bug reports are relatively high, then the value of $r$ should be larger, and vice versa.

**5. Generate candidate developer list**

The final step of BAHA is to calculate the probability of each developer with respect to a bug report in resolving the bug report, which can be denoted as a conditional probability $P(dev|bug)$. Each bug report has multiple topics and each developer has an expertise score on each topic. Thus, $P(dev|bug)$ can be calculated in Eq.(2):

$$P(dev|bug) = \sum_{topic} P(topic|bug) \times P(dev|topic) \tag{2}$$

Since we have a topic distribution of the bug denoted as $\theta_{bug}$ and expertise scores of the developer denoted as $\{P(x|T(x), k)\}_{k=1}^K$ ($T(x) =$ "*developer*"), Eq.(2) can also be written as the inner product of the two vectors as Eq.(3):

$$P(dev|bug) = \theta_{bug} \cdot \{P(x|T(x), k)\}_{k=1}^K \tag{3}$$

For each bug, we rank all the developers according to the conditional probability $P(dev|bug)$. Then we chose the top $Q$ developers to form a candidate developer list. Before BAHA assigns bug reports to developers, we make a little adjustment in candidate developer list because, in open source environment, developers can quit at any time. Thus, it is necessary to make sure that all the developers we assign are active. Following Bhattacharya *et al.*[23], we delete the developers who were inactive for more than 100 days from the time of new bug report in the list.

## III. Experiments

**1. The dataset**

We collect all the effective bug reports from Eclipse JDT project***** from 2002 to 2009, which have 18,674 bugs, 3,441 developers, 2,712 reporters and 128,058 comments in sum. Table 1 shows the distribution of comments on the developers. We see that the distribution is skewed, which means that although a large number of developers made comments for the project in history, only a small number of developers contribute to the project continuously. Only 83 developers made more than 100 comments over the 8 years. 2,933 developers commented on less than 10 bug reports and we regard them as unreliable for bug report assignment.

We also observe from the dataset that only 170 of the total 2,712 reporters in Eclipse JDT project reported more than 10 bug reports and most reporters (more than 87%) submitted only one bug report. About 70% of all

***** *http://www.eclipse.org/jdt/*

the bug reports were fixed within three months and 90% of all the bug reports were fixed within one year. The reasons for bug fixing time longer than one year are mostly related with the overall changes of the project, such as migration of bug repository and version upgrades of the software (bug reports #12430 and #12533 are typical instances). Most bug reports have 2 to 6 comments. 88.7% of the bug reports have less than 10 comments, which implies that most bug reports are easy to fix, and only about 2% of the bug reports have more than 20 comments.

**Table 1. The distribution of comments on developers**

| # of Comments | # of Developers |
|---|---|
| ≥100 | 83 |
| ≤100&≥80 | 54 |
| ≤80&≥60 | 58 |
| ≤60&≥40 | 62 |
| ≤40&≥20 | 85 |
| ≤20&≥10 | 106 |
| ≤10 | 2,933 |

We use one year as an interval to classify the whole dataset from the year 2002 to 2009 into 8 folds since about 90% of the bug reports are fixed within one year. In each fold of data, the bug reports fixed from January to September are used as training set and bug reports fixed from October to December are used as test set. For each bug report, we only retain the developers among those who contributed top 90% of the comments shown in Table 1. Meanwhile, the bug report which is submitted by minor developers(those submitted less than 10% bug reports) is also removed from the dataset.

After preprocessing, the number of bug reports in training set and test set as well as the number of developers and reporters of all folds of experimental data are shown in Table 2. We see that there are 13,526 bug reports in the training and test set. Different bug reports have different number of developers ranging from 1 to 17. In the training set, the average number of developers in a bug report is 3. Therefore, we vary the number of recommended developers $Q$ from 1 to 6 to gauge the performances.

**Table 2. Basic information of data**

| | Training set | Test set | Developer | Reporter |
|---|---|---|---|---|
| 2002 | 2594 | 512 | 66 | 52 |
| 2003 | 1661 | 457 | 58 | 52 |
| 2004 | 1589 | 463 | 67 | 46 |
| 2005 | 1882 | 308 | 76 | 43 |
| 2006 | 1274 | 218 | 50 | 25 |
| 2007 | 987 | 166 | 39 | 22 |
| 2008 | 773 | 108 | 27 | 17 |
| 2009 | 459 | 75 | 42 | 11 |

### 2. Experimental setup

ML-KNN is used to transfer the bug triage problem to multi-labeled classification where each bug report is regarded as a data point and the developers who contribute to bug report resolution are regarded as its labels. This intuition is from Anviks proposal to transfer bug fixer recommendation to a typical classification problem[8]. The parameters of ML-KNN are $K$ and $s$, where $K_m$ is the number of nearest neighbors and $s$ is used for probability smoothing. Following Zhang and Zhou[24], we set $s$ as 1 and tune the parameter $K_m$ as 15 as suggested by Wu *et al.*[14].

For DREX, its parameters includes $K_r$ and $Q$ where $K_r$ is the number of neighbours with similar textual contents as that of new bug report, and $Q$ is the number of recommended developer for the new bug report (the same meaning as that in Section II.4). We set $K_r$ as 20 and use the network metric as degree to measure developer importance in homogeneous network ranking as suggested by Wu *et al.*[14]. For DRETOM, it has three parameters: the number of topics $T_d$, $\lambda$ to trade off developers interest and expertise, and the number of developers $Q$ for recommendation. Following Xie *et al.*[5], we set $T_d$ as 200 and $\lambda$ as 0.2 for optimal performance. For Stacked generalization (SG), we follow Jonsson *et al.*[25] to use Bayes Net, Naive Bayes, SVM, KNN and Decistion Tree as the level-0 classifier and logistic regression as the level-1 classifier to weight the level-0 predictions for ensemble learning. For Location based assignment (LBA), we follow Shokripour *et al.*[26] to check out developers' code changes from the Eclipse CVS repository and developers' activity histories from the Eclipse bug tracking system (Bugzilla) to match new bug report with developers for bug report assignment. Here, only nouns are used to calculate the similarities of developers' histories and new bug reports.

$$Precision = \frac{\#\ of\ correctly\ recommended\ developers}{\#\ of\ all\ the\ recommended\ developers} \tag{4}$$

$$Recall = \frac{\#\ of\ correctly\ recommended\ developers}{\#\ of\ developers\ participate\ in\ the\ bug} \tag{5}$$

Traditionally, precision and recall are two common measurements to evaluate the performance of recommendation system. Precision measures the percentage of correctly recommended developers in all the recommended developers (see Eq.(4)). Recall measures the percentage of correctly recommended developers in all the developers who actually commented on the bug report (see Eq.(5)). However, in bug report assignment, we hold that recall is a better measurement than precision[5] because, different bug reports have different number of developers who commented on the bug report. Supposing we have two bug reports, one have 2 developers and another have 5 developers. If we recommend 5 developers to the two bug reports, then the best precision we can derive for the one bug report is 40% and for the another bug report is 100%. Thus, it is unreasonable to compare the precisions of bug reports having different number of developers. However, recall is acceptable because hundreds of developers in an

open source project and it is useful to exclude most irrelevant developers with the tolerance of a few mistakes. Therefore, we calculate the recall on each of the 8 folds to measure the performances of BAHA.

**3. Results**

We conduct a series of experiments to tune the parameter $r$ in Eq.(1) from 0 to 1 with an interval of 0.1 with varying $Q$ from 1 to 6. We see that when $r$ is increasing from 0.1 to 0.7, the performance of BAHA is also increased gradually. That is to say, to add more and more information derived from LDA topics will augment the performance of BAHA. However, when $r$ becomes larger than 0.7, the performance of BAHA decreases dramatically. This outcome shows that to further lower the structural information of BAHA will hurt its performance in bug report assignment. With this analysis, we set $r$ as 0.7 in the experiments.

With the above predefined parameters, Table 3 ("avg." abbreviates for "average" and "imp." abbreviates for "improvement") shows the comparative results of BAHA, ML-KNN, DREX, DRETOM, SG and LBA in assigning bug reports to different number of developers (*i.e.*, $Q$ is varying from 1 to 6). We can see that when $Q$ is smaller than 3, the performances of BAHA are worse than that of ML-KNN, DREX, SG and LBA. In the extreme case that $Q$ is set as 1, the performance of ML-KNN, DREX and LBA are much better than that of BAHA. We explain this outcome that DREX with degree as importance measure of data objects in homogenous network is equivalent to use the number of comments that developers made for bug reports. That is, the more bug reports a developer commented on in history, the higher priority he or she will be given among the recommended developers. Similarly, ML-KNN produces larger probabilities in recommendation for developers who have a larger number comments than other developers, when using multi-label classification for bug report assignment. Thus, the developer who has the largest number comments and the largest number of code changes is recommended for bug report assignment by ML-KNN, DREX and LBA on the condition that all the other requisites are the same. It is naturally that the developer with largest number of comments and code commits for historical bug reports would comment on a new bug report in the future.

However, when we recommend more than 4 developers for bug report assignment (*i.e.*, $Q > 4$), BAHA outperforms ML-KNN and DREX significantly. In comparison of BAHA and ML-KNN, the best case appears in the year 2008 of recommending 6 developers for bug report assignment with the increase of 35.54% (0.7048/0.5200). The worst case appears in the year 2005 of recommending 6 developers for bug report assignment with the increase of 1.05% (0.4091/0.3901). In comparison of BAHA and DREX, the best case appears in the year 2008 of recommending 6 developers with the increase of 17.19%(70.48/60.14). The worst case appears in the year 2005 of recommending 6 developers with the increase of 0.06%(40.91/40.65). We explain the outcome that when $Q$ is small, the developer with expertise best matching the topics of the bug report may be not the one who actually commented on the bug report because the developer is probably not active at that time. Nevertheless, when we recommend more developers, i.e., to relax the activeness of developers, the matching of developer expertise and topics of bug report become more and more important in bug report assignment. This point is also validated with the results from DRETOM. We see that when $Q$ is larger than 4, DRETOM performs comparably with DREX in most folds, even a little better than DREX in from the year 2005 to the year 2009.

## IV. Discussion

**1. Effect of $r$**

In Eq.(1), we introduce the parameter $r$ to trade off proportion of the topic information from the LDA topic modeling and the topic distribution of reporters from RankClass algorithm. Obviously, $r = 0$ indicates that we ignore that the LDA topic distribution and only use the topic information of reporters derived from RankClass algorithm. $r = 1$ indicates that we only use the topic distribution by LDA modeling and ignore the topic information of bug reporters given by heterogenous network analysis. $r$ is set between 0 and 1 indicates that the best choice is to combine both of the two topic information at the same time. For instance, if we set $r$ as 0.5, then the two sources of topic information are equally important. In the experiment, we find that $r = 0.7$ is the best choice, which shows that for best performances of BAHA, we should use both sources of topic information. However, the topic distribution of bug reports from LDA modeling is more important than the topic information from heterogenous network analysis by RankClass algorithm. With this outcome, we can infer that most bug reports of Eclipse JDT project have high-quality textual contents.

In Table 3, the best recall at Top 6 is 60.29% and the worst recall is 54.76% (not considering the extreme cases as $r = 0$ and $r = 1$), which is not a large difference in performances of BAHA. It seems that the parameter $r$ does not affect experimental results very much. After manual checking our experiments, we find two reasons explained for this outcome. The first reason is that the goal we introduce $r$ in the experiments is to trade off the two sources of topic information and make the topic distribution close to the "true" distribution that a bug report should be. However, if the two sources of topic information are similar, then the parameter $r$ actually dose not make an great effect.

**Table 3. Recalls of BAHA, ML-KNN, DREX and DRETOM on the 8 folds.**

| | $Q$ | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | avg. | imp. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BAHA | **Top1** | 15.79% | 17.66% | 12.85% | 10.98% | 14.64% | 20.63% | 10.88% | 18.67% | 15.26% | - |
| | **Top2** | 26.70% | 29.84% | 30.04% | 17.51% | 34.11% | 31.63% | 34.48% | 41.33% | 30.70% | - |
| | **Top3** | 38.70% | 38.17% | 38.58% | 24.79% | 47.05% | 42.34% | 48.90% | 50.56% | 41.14% | - |
| | **Top4** | 46.41% | 45.38% | 47.13% | 30.24% | 54.90% | 56.01% | 57.55% | 58.00% | 49.45% | - |
| | **Top5** | 53.25% | 52.11% | 53.80% | 35.84% | 62.39% | 59.98% | 65.91% | 67.33% | 56.32% | - |
| | **Top6** | 61.80% | 58.67% | 59.29% | 40.91% | 68.70% | 62.79% | 70.48% | 69.56% | 61.53% | - |
| ML-KNN | **Top1** | 21.15% | 20.12% | 18.43% | 15.43% | 26.17% | 24.06% | 14.15% | 19.43% | 19.87% | -30.17% |
| | **Top2** | 35.11% | 30.14% | 30.21% | 25.83% | 37.01% | 38.23% | 33.10% | 35.13% | 33.10% | -7.78% |
| | **Top3** | 41.06% | 38.12% | 37.53% | 31.01% | 45.03% | 45.10% | 39.86% | 44.11% | 40.22% | 2.21% |
| | **Top4** | 43.32% | 44.45% | 43.58% | 36.46% | 51.12% | 51.23% | 46.82% | 51.02% | 46.00% | 6.98% |
| | **Top5** | 47.01% | 44.10% | 44.97% | 38.17% | 55.46% | 53.17% | 50.02% | 56.16% | 48.63% | 13.66% |
| | **Top6** | 46.10% | 45.18% | 47.32% | 39.01% | 56.03% | 53.09% | 52.00% | 59.01% | 49.71% | 19.19% |
| DREX | **Top1** | 21.42% | 18.93% | 18.10% | 15.53% | 24.56% | 23.01% | 14.38% | 18.89% | 19.35% | -26.80% |
| | **Top2** | 33.95% | 30.29% | 29.09% | 24.23% | 36.42% | 38.24% | 29.03% | 31.44% | 31.58% | -2.87% |
| | **Top3** | 42.85% | 39.14% | 37.67% | 29.79% | 47.75% | 46.33% | 44.46% | 43.67% | 41.46% | -0.78% |
| | **Top4** | 47.27% | 45.89% | 43.13% | 33.18% | 55.40% | 50.34% | 50.43% | 51.56% | 47.15% | 4.66% |
| | **Top5** | 50.51% | 49.55% | 49.65% | 36.05% | 58.85% | 55.36% | 56.79% | 58.44% | 51.90% | 7.86% |
| | **Top6** | 52.99% | 51.36% | 53.31% | 40.65% | 62.75% | 56.52% | 60.14% | 61.67% | 54.92% | 10.73% |
| DRETOM | **Top1** | 12.22% | 6.12% | 14.83% | 8.18% | 11.44% | 17.72% | 19.81% | 19.67% | 13.75% | 9.92% |
| | **Top2** | 20.33% | 20.94% | 24.61% | 14.85% | 30.34% | 30.17% | 30.68% | 32.44% | 25.55% | 16.81% |
| | **Top3** | 27.33% | 30.89% | 31.94% | 22.59% | 44.68% | 37.17% | 37.45% | 52.22% | 35.53% | 13.62% |
| | **Top4** | 38.43% | 37.50% | 40.02% | 28.92% | 52.40% | 49.34% | 46.90% | 58.89% | 44.05% | 10.92% |
| | **Top5** | 42.77% | 43.87% | 46.88% | 39.91% | 59.85% | 56.27% | 57.64% | 62.67% | 51.23% | 9.04% |
| | **Top6** | 52.08% | 51.87% | 52.02% | 43.69% | 68.28% | 62.44% | 58.07% | 68.00% | 55.56% | 9.70% |
| SG | **Top1** | 14.54% | 16.35% | 15.65% | 13.36% | 14.87% | 22.41% | 17.68% | 18.23% | 16.64% | -0.90% |
| | **Top2** | 22.37% | 30.42% | 31.34% | 17.21% | 30.21% | 31.53% | 34.28% | 36.67% | 29.25% | 4.73% |
| | **Top3** | 35.64% | 34.78% | 36.58% | 25.90% | 43.12% | 38.34% | 43.19% | 44.84% | 37.80% | 8.11% |
| | **Top4** | 41.16% | 42.76% | 43.23% | 29.62% | 46.81% | 49.35% | 49.86% | 51.67% | 44.31% | 10.40% |
| | **Top5** | 47.06% | 49.93% | 48.58% | 36.63% | 57.02% | 55.46% | 55.19% | 60.81% | 51.34% | 8.86% |
| | **Top6** | 51.75% | 52.62% | 52.57% | 38.73% | 60.26% | 57.19% | 61.34% | 62.47% | 54.62% | 11.23% |
| LBA | **Top1** | 15.89% | 16.32% | 17.96% | 15.13% | 17.28% | 21.33% | 17.76% | 18.65% | 17.54% | -14.92% |
| | **Top2** | 21.43% | 28.38% | 29.85% | 19.30% | 29.96% | 29.73% | 31.35% | 37.87% | 28.48% | 7.23% |
| | **Top3** | 34.03% | 32.32% | 36.78% | 24.24% | 41.26% | 39.94% | 43.07% | 48.95% | 37.57% | 8.66% |
| | **Top4** | 41.34% | 42.89% | 44.70% | 26.93% | 48.36% | 51.14% | 51.39% | 52.13% | 44.86% | 9.29% |
| | **Top5** | 48.55% | 44.93% | 47.97% | 31.04% | 57.08% | 53.93% | 60.51% | 60.59% | 50.58% | 10.21% |
| | **Top6** | 51.47% | 52.56% | 52.28% | 37.65% | 61.87% | 58.15% | 61.09% | 62.37% | 54.68% | 11.13% |

The second reason is that we eliminate those reporters who submitted a small number of bug reports and the number of this kind of reporters is very large shown in Table 1. In Eq.(1), the bug reports submitted by the eliminated reporters do not have the source of topic information from RankClass algorithm, which means that the topic distribution from LDA modeling dominates the topic information of bug report. For this reason, parameter $r$ does not make an effect on the experimental results. Therefore, with the two reasons that $r$ doesn't affect the results and the outcome that on average, BAHA improves recall of bug report assignment by more than 5% with $Q > 3$, we can regard that $r$ makes a significant impact on experimental results.

**2. Good results or not?**

Many researches treat bug report assignment as a typical classification problem where each bug report regarded as a pattern and each developer is regarded a class label[7,8]. The bug report assignment is then to label the patterns by predicting developers for the bug reports. BAHA is proposed to recommend developers in considering expertise matching of developers and bug reports. This point is very similar to Xie *et al.*'s DRETOM [5]. However, they are completely different in that BAHA adopts heterogenous network analysis to derive the expertise of developers (*i.e.* $\{P(x|T(x),k)\}_{k=1}^{K}$ ($T(x)$ = "*developer*") and trade off the topic information with LDA modeling of textual contents of bug reports.

With this consideration, we paretically emphasize the results of BAHA and DRETOM. It can be seen from Table 3 that in all the years from 2002 to 2009, BAHA outperforms DRETOM. Although in some years such as 2005 and 2006, DRETOM performs very closely with BAHA. In other years then this two years, DRETOM performs far behind BAHA. Thus, we can draw that BAHA performs not only more accurately but also more robust than DRETOM. We attribute this outcome to the structural information derived from the heterogeneous network analysis.

When $Q$ is set as 6, the average recall of BAHA is

only about 60%, that still remains a large space to improve. After our manual checking, we find 2 reasons can be explained for this outcome. The first reason is that, in an open source project like Eclipse JDT, developers frequently join and leave the project from time to time. If a developer was absent from the project from October to December and we use the bug ports in this duration as the test set, then that means the experimental results will be hurt if the leaving developer is recommended by whatever means. Conversely, if a developer joined in the project during October to December, we can not recommend this developer because he or or she does not occur in the training set. By this time, we can not predict when the developers would join or leave the project. Thus, we can not exclude this kind of mistakes. The second reason is that developers of their own technical expertise occasionally commented on the bug reports those are not within their expertise (with randomness by our observation). Currently, BAHA has not yet taken these random events into account. We will address this problem in the future. Although there are some mistaken recommendations, we hold that the experimental results are acceptable in current stage. At least, BAHA has already successfully predicted the majority of actual developers in resolving bug reports participants and excluded hundreds of developers who definitely would not participate in the resolution given a bug report, resulting a great cost reduction of possible reassignment.

## V. Conclusion and Future Work

This paper proposes a new approach called BAHA to recommend developers who are the potential participants and contributors of the collaborative bug report resolution activities. Firstly, we extract the textual contents from bug reports and obtain topic information of the bug reports with respect to the textual contents using LDA modeling. Secondly, we use the reporter-bug-developer heterogeneous network to capture the structural information of bug reports. Using RankClass algorithm, we derive the topic information of bug reports with respect to reporters and the expertise of developers. Thirdly, we combine the two sources of topic information of bug reports and match the topics of bug reports with developers' expertise. Experimental results on Eclipse JDT project demonstrate the superiority of BAHA over other state of art methods. This outcome demonstrates that the information derived from heterogeneous network analysis can complement the textual information of bug reports.

### References


[1] W. He, R. Zhao and Q. Zhu, "Integrating evolutionary testing with reinforcement learning for automated test generation of object-oriented software", *Chinese Journal of Electronics*, Vol.24, No.1, pp.38–45, 2015.

[2] E.S. Raymond, *The Cathedral and the Bazaar*, O'Reilly & Associates, Inc., Cambridge, Massachusetts, USA, 1999.

[3] C.R. Reis, R.P. de Mattos Fortes, R. Pontin and M. Fortes, "An overview of the software engineering process and tools in the Mozilla project", *Proc. of the Open Source Software Development Workshop*, pp.155–175, 2002.

[4] T. Bao, S. Liu and X. Wang, "Research on trustworthiness evaluation method for domain software based on actual evidence", *Chinese Journal of Electronics*, Vol.20, No.2, pp.195–199, 2011.

[5] X. Xie, W. Zhang, Y. Yang and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution", *Proc. of the 8th International Conference on Predictive Models in Software Engineering*, Lund, Sweden, pp.19–28, 2012.

[6] G. Jeong, S. Kim and T. Zimmermann, "Improving bug triage with bug tossing graphs", *Proc. of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp.111–120, 2009.

[7] D. Cubranic and G.C. Murphy, "Automatic bug triage using text categorization", *Proc. of the 16th International Conference on Software Engineering & Knowledge Engineering*, pp.92–97, 2004.

[8] J. Anvik, L. Hiew and G.C. Murphy, "Who should fix this bug?", *Proc. of the 28th International Conference on Software Engineering*, Shanghai, China, pp.361–370, 2006.

[9] N. Bettenburg, T. Zimmermann, R. Premraj, S. Just, A. Schröter and C. Weiss., "What makes a good bug report?", *IEEE Transactions on Software Engineering*, Vol.36, pp.618–643, 2010.

[10] P. Hooimeijer and W. Weimer, "Modeling bug report quality", *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, pp.34–43, 2007.

[11] P.J. Guo, T. Zimmermann, N. Nagappan and B. Murphy, "Not my bug and other reasons for software bug report reassignments", *Proc. of the ACM 2011 Conference on Computer Supported Cooperative Work*, pp.395–404, 2011.

[12] S.N. Ahsan, J. Ferzund and F. Wotawa, "Automatic software Bug triage system (BTS) based on latent semantic indexing and support vector machine", *Proc. of the 4th International Conference on Software Engineering Advances*, pp.216–221, 2009.

[13] J. Xuan, H. Jiang, Z. Ren and W. Zou, "Developer prioritization in bug repositories", *Proc. of the 34th International Conference on Software Engineering*, Zurich, Switzerland, pp.25–35, 2012.

[14] W. Wu, W. Zhang, Y. Yang and Q. Wang, "DREX: Developer recommendation with K-nearest-neighbor search and expertise ranking", *Proc. of the 18th Asia-Pacific Software Engineering Conference*, pp.389–396, 2011.

[15] D. Zhang and L. Gao, "Virtual network mapping through locality-aware topological potential and influence node ranking", *Chinese Journal of Electronics*, Vol.23, No.1, pp.61–64, 2014.

[16] J. Hopcroft, O. Khan, B. Kulis and B. Selman, "Tracking evolving communities in large linked networks", *Proceedings of the National Academy of Sciences*, Vol.101, pp.5249–5253, 2004.

[17] D.M. Blei, A.Y. Ng and M.I. Jordan, "Latent dirichlet allocation", *The Journal of Machine Learning Research*, Vol.3, pp.993–1022, 2003.

[18] J. Ouyang, Y. Liu, X. Li and X. Zhou, "Multi-grain sentiment/topic model based on LDA", *Acta Electronica Sinica*, Vol.43, No.9, pp.1875–1880, 2015. (In Chinese)

[19] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng and T. Wu, "RankClus: Integrating clustering with ranking for heterogeneous information network analysis", *Proc. of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp.565–576, 2009.

[20] W. Zhang, Y. Yang and Q. Wang, "Network analysis of OSS evolution: An empirical study on ArgoUML project", *Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, pp.71–80, 2011.

[21] X.H. Phan and C.T. Nguyen, "GibbsLDA++: A C/C++ Implementation of Latent Dirichlet Allocation", available at *http://gibbslda.sourceforge.net/*, 2007.

[22] M. Ji, J. Han and M. Danilevsky, "Ranking-based classification of heterogeneous information networks", *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.1298–1306, 2011.

[23] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging", *Proc. of the 26th IEEE International Conference on Software Maintenance*, pp.1–10, 2010.

[24] M. Zhang and Z. Zhou, "ML-KNN: A lazy learning approach to multi-label learning", *Pattern Recognition*, Vol.40, No.7, pp.2038–2048, 2007.

[25] L. Jonsson, M. Borg, D. Broman, K. Sandal, S. Eldh and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts", *Empirical Software Engineering*, pp.1–46, 2015.

[26] R. Shokripour, J. Anvik, Z.M. Kasirun and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation", *Proc. of 10th IEEE Working Conference on Mining Software Repositories*, pp.2–11, 2013.

**ZHANG Wen** received the Ph.D. degree in knowledge science from Japan Advanced Institute of Science and Technology in 2009. He has published more than 30 papers in the field of data mining and software engineering. Now, he is a full professor in School of Economics and Management, Beijing University of Chemical Technology. (Email: zhangwen@mail.buct.edu.cn)

**WANG Song** received the M.S. degree in computer software and theory, Chinese Academy of Sciences in 2014. Now, he is a Ph.D. candidate in the Department of Electrical and Computer Engineering at University of Waterloo, Canada. His recent research interests include software defect resolution, software defect prediction, data mining and test prioritization. (Email: wangsong@uwaterloo.ca)

**WANG Qing** received the Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences. She has published more than 150 papers in software engineering and knowledge management. Now, she is a full professor at the Institute of Software, Chinese Academy of Sciences. (Email: wq@itechs.iscas.ac.cn)