Who Will be Interested in? A Contributor Recommendation Approach for Open Source Projects

Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, and Huaimin Wang
College of Computer Science
National University of Defense Technology
Changsha, Hunan, China
{zhangxunhui,taowang2005,yingang,hmwang}@nudt.edu.cn, delpiero710@126.com

***Abstract*—The crowds' continuous participation and contribution are the key factors for the success of open source projects. However, among the massive competitors, it is difficult for a project to attract enough contributors by just passively waiting for enthusiasts to join in. Instead, it should actively seek gifted developers. Most of the current studies mainly focus on recommending experts inside a repository for some specific development tasks. In this paper, we propose a novel approach *ConRec* to recommend potential contributors across the entire open source community for given projects. It leverages the developers' historical activities in projects to analyze their technical interests and technical connections with others. Thereafter, it combines collaborative filtering algorithm with text matching algorithm to recommend proper developers. We conducted extensive experiments on 5,995 open source projects and 2,938,620 developers in GitHub. The results show that the proposed algorithm can recommend contributors to open source projects with the best performance of 63% in accuracy, and solve the cold start problem as well.**

Keywords-Contributor Recommendation; Collaborative Filtering; Text Matching; GitHub

A. Introduction

Open source software (OSS) has become increasingly popular in software development. Quite different from the traditional software development, OSS is driven by massive crowds including developers, users, managers and so on. These stakeholders involve in OSS by interests, and most of them have their own full-time job and can only spend spare time on OSS. They can join in or withdraw from it at any time. Nevertheless, OSS has achieved great success at creating high-quality software like Linux, MySQL, Spark and so on, and is viewed as "eating the software world" by "the Future of Open Source Survey" [1].

The behavior of open source itself does not necessarily result in a project's success, but the continuous participation and active contribution from the crowds do play a crucial role. In GitHub, there are more than 48 million open source projects. However, over 2 million projects are not forked or watched by anyone after releasing, and 15.1% of them were not updated for more than a year and finally failed. Even for those projects which used to be successful will languish without continuous contribution of developers.

Therefore, finding and attracting the right developers to participate is quite crucial for OSS. On the one hand, proper developers can provide necessary technical expertise that a project needs; On the other hand, suitable projects can inspire developers to participate in and to contribute their innovations continuously. However, there is a massive amount of competitive OSS, and developers are often limited by their time and energy to browse and choose from all the related projects. An automatic approach to bridge the gap between developers and projects and match them means much for both developers and projects.

Several studies have explored the act of recommending developers to open source projects. Nguyen et al. [2] and Ma et al. [3] proposed approaches to fix the recommendation issues by analyzing the implementation history and expertise of developers. Thongtanunam et al. [4] and Yu et al. [5] focused on the automatic pull-request reviewer recommendation problem by analyzing the development history and social connections of individuals in the software community. Most of these studies mainly focus on specific software tasks and limit the recommended candidates to the core developers of the projects. By contrast, the current study is rooted in the granularity of open source repositories, and our output allows recommendations of external experts throughout the community.

This study developed a general algorithm called *ConRec* for recommending suitable contributors to open source projects based on a collaborative filtering algorithm and text matching based method. We performed extensive experiments to compare the performances of different algorithms on various types of projects. The test data included approximately six thousand projects and over three million candidates in GitHub. *ConRec* performed well in all types of projects while solving the "cold start" problem. The main contributions of this study are as follows:

- We design a commit network to measure the collaboration connections between developers. Based on this network,

DOI reference number: 10.18293/SEKE2017-067

we develop a weighted collaborative filtering (*WCF*) algorithm to recommend.
- We design a text matching based recommendation algorithm based on the text information of projects. It can solve the cold start problem effectively for projects which have only few pre-existing developers before recommendation.
- We combine the above two algorithms together and design a hybrid approach called *ConRec*. We conduct extensive experiments on over 5,990 projects and about three million developers in GitHub, and achieves recommendation accuracy of about 63%.

The rest of this paper is organized as follows. Section 2 reviews a few related studies. Section 3 describes the framework of *ConRec* and the detailed recommendation algorithms. Section 4 presents the design of the experiments. Section 5 discusses the experiment results. Section 6 elaborates the conclusion and describe the future plans of the study.

I. Related Work

A. Expert Identification and Prediction

In software development, the expertise of developers is one of the most important factors that influence the efficiency and quality of software development. Hence, an accurate expert identification is of immense importance in global and distributed software development. Many studies have explored expert identification and prediction from different perspectives. In references [2][3], Nguyen and Ma, et al. evaluated the expertise of developers by analyzing the usage frequency of methods and time distribution on issue fixing, respectively. Thereafter, they designed the corresponding metrics to measure the technical capabilities of developers from the aspects of usage and implementation. Gharehyazie et al. [6] leveraged the activities of participants in a mailing-list communication network and issue fixing history, as well as proposed an approach to predict the probability of becoming a developer in a project. Robbes [7] used the interactions between developers to calculate their expertise. Schuler and Zimmermann [8] measured the expertise of developers by using the frequency of usage of specific application programming interface.

Most of these studies mainly focused on the capability of developers to solve problems, thereby failing to consider the probability of participating in target projects. The current study combines the ability of developers and the association among them to find suitable committers to projects.

B. Task Assignment

The development of social coding communities has resulted in collaboration and cooperation becoming particularly common. Moreover, the rapidly increasing numbers of repositories and tasks pose difficulties for developers.

Automatic task assignment has been extensively studied. In particular, bug assignment has attracted significant research interest. Bhattacharya [9], [10], Naguib [11], Xia [12], Shokripour [13], and others introduced many methods to recommend experts to solve bugs. Yu et al. [5], [14], [15] analyzed the social coding pull-request mechanism and developed methods for pull-request recommendation. Balachandran and Thongtanunam et al. [16][4] generated methods for code reviewing in social coding communities. Kagdi [17] heuristically created a code reviewer recommendation system using the history of activities and expertise of developers.

These studies mainly focused on the fine-grained tasks of open source repositories. By contrast, the present research is concerned with finding suitable contributors for a project, which is a coarse-grained topic.

C. Collaborative Network in Social Coding Communities

Although social coding is quite different from traditional software development, the working habits of developers are quite similar. People tend to work in groups and form new groups with familiar cooperators, thereby leading to the formation of a collaborative network.

Hertel et al. [18] studied the motivation of participants in OSS, and determined that people tend to work as a team. Grewal et al. [19] explored the network embeddedness of open source projects and realized that strong embeddedness leads to success. Weiss et al. [20] revealed that developers will migrate from one community to another with other collaborators. Hahn et al. [21] used these studies as basis to indicate that prior social relations in a network of developers can assist to attract additional developers. They also determined that developers tend to join in projects with which they have strong collaborative ties [22].

In considering the existing studies, we learned that developers in OSS tend to form collaborative networks. Furthermore, cooperators tend to form groups when developing new projects. On the basis of this phenomenon, we developed the following recommendation algorithm.

II. Method

In this section, we present the framework of *ConRec* and introduce the detailed mechanism of the proposed algorithm.

A. Intuition of ConRec

The basic idea stems from the observation of the relationship between developers and projects. Hahn [22] and Madey et al. [23] determined that developers in social coding communities tend to form groups, and that these collaborative networks affect the choice of participation of developers. We extract a part of the commit network in Figure 1.

Figure 1 shows that developers who have committed to the same project tend to collaborate in other projects. The same is true real life; when developing a project, developers form groups, become familiar with one another, and gain mutual trust. Therefore, they are likely to collaborate in other projects. Figure 1 also shows that developers who focus on "cocos2d" tend to work on the same projects. This condition indicates that similar technical interests lead to similar behaviors of participation. The same is true in reality, in which developers with

Pre-existing developer: developer who has already committed to the project.

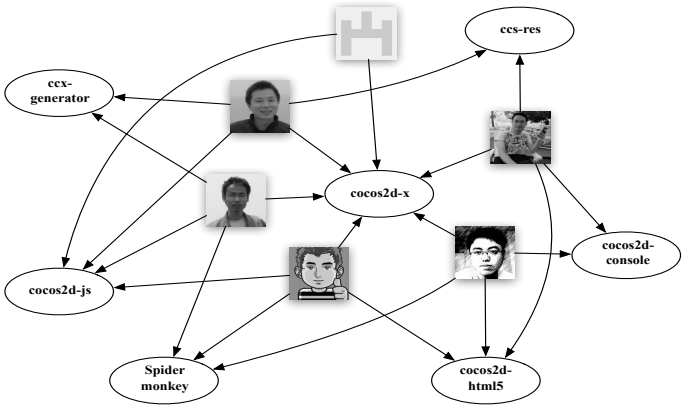

Fig. 1: Part of Commit Network

different skills or in different research fields focus on different areas. Accordingly, we developed the *ConRec* algorithm for recommending suitable contributors to open source projects.

B. Framework of ConRec

ConRec involves three steps: gathering prepared information for the recommendation algorithm, calculating relations between potential developers and target projects, and ranking and obtaining the final results. Figure 2 illustrates the framework.

1) Gather Prepared Information: The first step is to gather prepared information for the target project, including its pre-existing developers, potential developers, major programming language, and the technical terms in the project name and description.

2) Calculate Relation: The second step is to perform the recommendation task for the target project based on the prepared information. We use the *WCF* algorithm and text matching based algorithm to calculate the relation between potential developers and projects. Thereafter, we consolidate the results by merging the recommendation results of *WCF* with the text matching results.

3) Rank Results: The third step is to obtain the final recommendation results of the target project. After calculating the relation of developers, we rank the results in descending order and identify the top-k developers as the final result.

C. WCF Algorithm

This part describes the *WCF* algorithm in detail, which comprises two steps. Figure 3 shows the work flow of this algorithm.

1) Expert Selector: The first step is to select the developers who are familiar with the major programming language of the target project. If the developer lacks experience in the target programming language, then he/she may lack interest and experience difficulty committing to the target project. Therefore, we count the number of times a developer commits with the target programming language, and we consider that potential developers should commit at least four times to projects which use the target language as the major programming language. The reason for opting for this value as the threshold is discussed in research question 3.

2) WCF: The second step is to recommend developers based on the collaborative filtering algorithm. Here, we select the pre-existing developers from the prepared information gathered in the first step of *ConRec*. Thereafter, we calculate the relation between each pre-existing developer and potential developer based on their relation to other projects. Equation 1 shows the relation between a developer and a project where the first and second parameters denote the developer and project, respectively. $Commit(d,p)$ refers to the number of times developer d commits to project p, and U_p denotes the set of pre-existing developers for project p.

$$R_{dp}(d,p) = \frac{Commit(d,p)}{\sum_{i=1}^{|U_p|} Commit(U_p[i],p)} \tag{1}$$

For the developer relation, we use the Vector Space Similarity algorithm (Equation 2), where P_A refers to the projects that A have committed to.

$$R_{dd}(A,B) = \frac{\sum_{p:\{P_A \cap P_B\}} R_{dp}(A,p) * R_{dp}(B,p)}{\sqrt{\sum_{p:P_A} R_{dp}^2(A,p) * \sum_{p:P_B} R_{dp}^2(B,p)}} \tag{2}$$

After calculating the relation between the developers, we compute the relation between the developer and project according to Equation 3, where D_p denotes the pre-existing developers for project p.

$$result(A,p) = \sum_{d:D_p} R_{dp}(d,p) * R_{dd}(A,d) \tag{3}$$

Lastly, we rank the results of all potential developers for the target project in descending order, and select the top-k values as the final result.

D. Text Matching Based Recommendation Algorithm

This part aims to improve the recommendation result of *WCF* algorithm. When lacking pre-existing developers, *WCF* is unsuitable for recommendation. To solve the cold start problem, we extract the technical terms and match the developers that focus on the target techniques. The idea is based on reference [24], which uses text information to measure expertise. Figure 4 shows the work flow.

1) Generate Technical Terms: First, we generate the technical terms from the name and description of the target project using smart IKAnalyzer, and remove the words that are included in the stop-word dictionary.

2) Generate Term Map: Second, we identify the potential developers based on the technical terms of the target project. Developers who have participated in projects can get their related terms. Thereafter, we can calculate the relation between terms and developers based on the *TF-IDF* algorithm [25]. The equation is shown as below, where P_t represents the set of projects with term t, and T refers to the entire set of terms.

$$R_{dt}(d,t) = \sum_{p:P_t} R_{dp}(d,p) * log\frac{|\bigcup_{x:T} P_x|}{|P_t|} \tag{4}$$

Latent developer: developer that may commit to the project in the future.

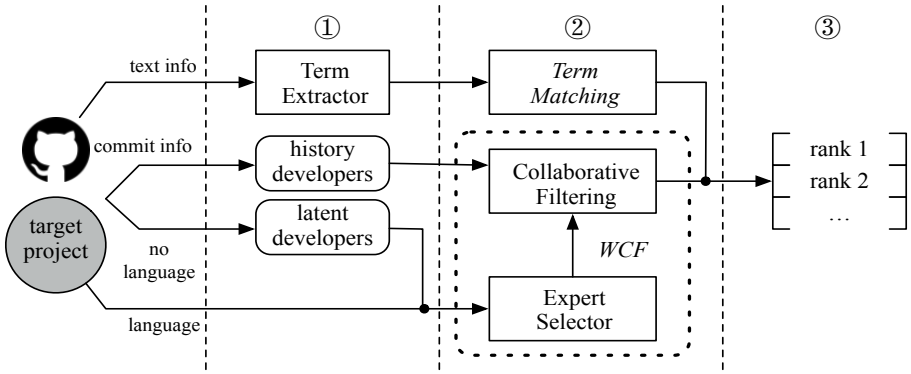

Fig. 2: Framework of *ConRec*

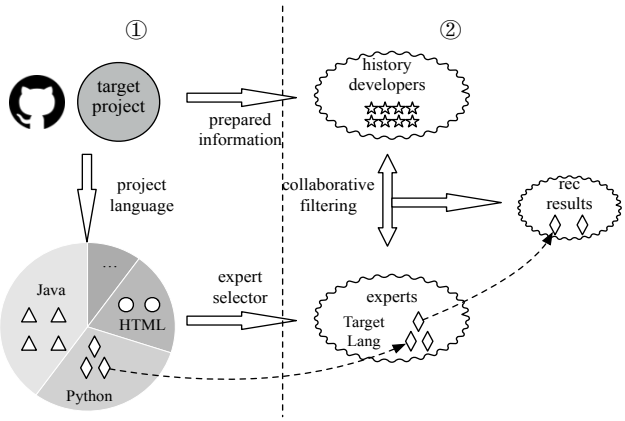

Fig. 3: Weighted Collaborative Filtering Algorithm

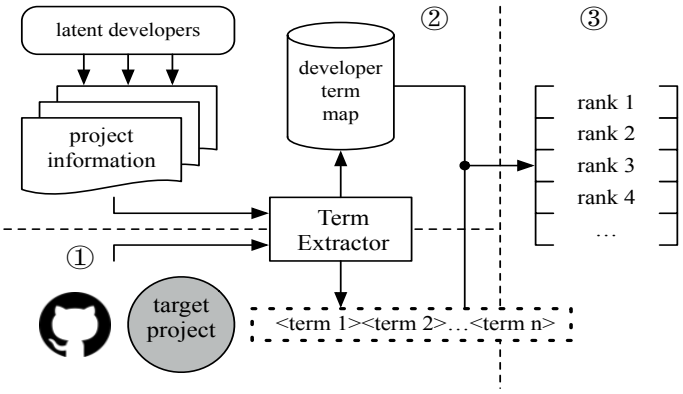

Fig. 4: Work Flow of Text Matching Algorithm

3) Rank Result: After calculating the relation between the potential developers and terms, we compute the final results using Equation 5, where T_{dp} denotes the set of terms that match developer d with project p. Thereafter, we rank the results in decreasing order. When *WCF* cannot recommend a sufficient number of developers, the text matching based algorithm serves as the supplement of the *WCF* algorithm, the recommendation result of which is used for those projects that lack of pre-existing developers.

$$result(d,p) = |T_{dp}| * \sum_{t:T_{dp}} R_{dt}(d,t) \tag{5}$$

III. Experiment

In this section, we will propose some research questions about our recommendation algorithm, and describe the experiment data set. Meanwhile, we describe the metrics that is used to validate the proposed algorithm.

A. Research Questions

We derive the following research questions to explore the relation between developers and projects based on the commit number and analyze the performance of our recommendation algorithm.

- Q1: How does *ConRec* perform compared with the two-value traditional collaborative filtering method?
- Q2: How do the algorithms differ for projects with different numbers of pre-existing developers?
- Q3: How will the number of commits affects the recommendation performance when defining the experts in given programming languages?

For Q1, we conduct experiments and compare the accuracy of different algorithms when recommending to a number of projects. For Q2, we divide the test projects into two parts. One part involves projects that have few pre-existing developers and the other involves projects with adequate pre-existing developers. Thereafter, we compare their performance. For Q3, we conduct experiments to observe the performance of *ConRec* when providing different threshold values to the expert selector part.

B. Data Set

Prior to the experiment, we generate data from the GHTorrent MySQL dump, which was released in March 2016. We firstly determine a time point. All the information prior to the time point is referred to as pre-existing information and the remaining is regarded as potential information.

http://ghtorrent.org/downloads.html
Time point: 2014-09-14.

In order to validate our recommendation algorithm, we should select well developed open source projects and determine whether our algorithm can recommend new committers to them after the time point. We select 492,590 projects, the commit numbers of which rank in the top 5% of all projects with over one committer. We consider them well-developed projects after the time point. Meanwhile, we select 450,170 projects, the number of new committers of which rank in the top 5% after the time point. Thereafter, we calculate the intersection of the projects with the top 5% commits and the projects with top 5% number of new committers after the time point. We also remove those projects created after the time point or had already been deleted or forked from others, and eventually obtain 5,995 projects. In the entire set, 888 projects have less than two committers prior to the time point; these projects are treated as ones with few pre-existing developers. The rest of the projects are deemed to have adequate pre-existing developers.

For potential developers, we remove those who have no commit experience prior to the time point; thus, from 10,132,629 users in GitHub, we arrive at 2,938,620 developers.

For the *CF* and *WCF* algorithm, we run them on the entire set of 10,132,629 developers and 28,118,416 projects.

The data set that we obtained is shown in Table I.

TABLE I: Dataset for Test

Items		Number
$Project_{CF}$ [a]		28,118,416
$Project_{test}$ [b]	$num(p.d.)$[c]≤ 1	888
	$num(p.d.) \geq 2$	5107
$Developer_{CF}$ [d]		10,132,629
$Developer_{potential}$ [e]		2,938,620

[a] $Project_{CF}$: projects used in *CF* algorithm.
[b] $Project_{test}$: projects used to test the performance.
[c] $num(p.d.)$: the number of pre-existing developers.
[d] $Developer_{CF}$: developers used in *CF* algorithm.
[e] $Developer_{potential}$: potential developers with over one commit prior to the time point.

C. Experiment Metrics

For the validation of our experiment, precision, recall, and MRR are unsuitable to evaluate the performances. For precision, we cannot ensure that the false positives (i.e., recommended developers who have not committed to the project) will not commit to the target project sometime in the future. For recall, the developers we recommend may commit to the target project later even though they have yet to commit. For MRR, committers whose commit numbers rank low are temporary, but they may surpass those leaders in the future.

We can use the accuracy value to test the effectiveness of the algorithm. If our algorithm can find a developer who commits to the target project after the time point, then we can say that the system takes effect. The accuracy value is positively correlated with the effectiveness of the algorithm. The equation of accuracy is shown below, where $hit@k$ refers to the number of hit projects when recommending k developers and $|test|$ represents the number of all the test projects.

$$accuracy = \frac{hit@k}{|test|} \tag{6}$$

New committer: developer who commit to the project for the first time.

IV. Results

In order to answer the three research questions mentioned above, we carried out three sets of experiments.

A. Overall Performance of ConRec

To answer Q1, we compare the accuracy value of our recommendation algorithm with the traditional collaborative filtering (*CF*) algorithm using the whole set of testing projects. For the *CF* algorithm, it first constructs a two-value matrix to record the commits of potential developers. Accordingly, 1 stands for the developer who has already committed to the project and 0 stands for the opposite. Thereafter, it calculates the relationship between the potential developers and the target project, and eventually ranks the results in descending order.

Figure 5 shows the result.

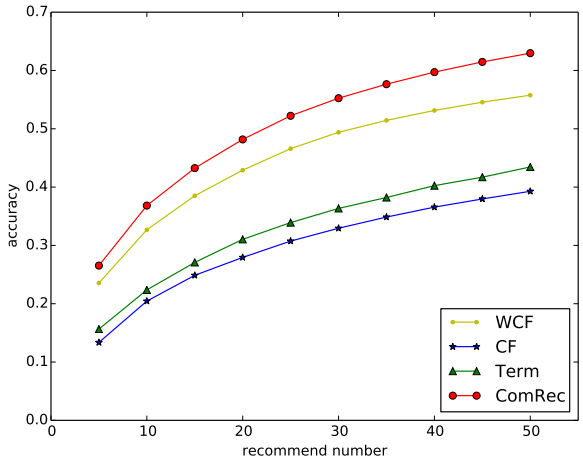

Fig. 5: Accuracy Comparison with the Entire Set of Projects

Our recommendation algorithm *ConRec* performs considerably better than the *CF* algorithm, with the accuracy of the former ranging from 26.5% to 63%. Hence, *ConRec* can meet the requirements of many projects and recommend suitable committers to them. For *CF*, the result shows that the two-value relation between committers and projects is inaccurate, that is, the algorithm cannot measure the strength of relationships.

B. Performance under Different Numbers of Pre-existing Developers

For Q2, we perform experiments and compare the performance of different algorithms in projects that have few or adequate pre-existing developers to determine whether *ConRec* can solve the cold start problem. Precisely 888 projects have few pre-existing developers. Figures 6 and 7 show the result.

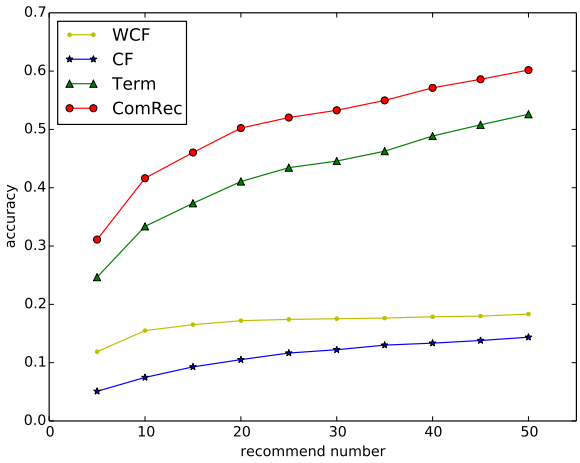

Fig. 6: Accuracy for Projects with Few Pre-existing Developers

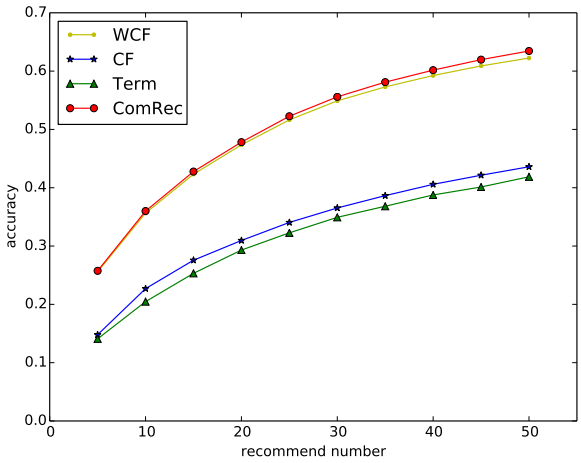

Fig. 7: Accuracy for Projects with Adequate Pre-existing Developers

Figure 6 shows that the *CF* and *WCF* algorithms do not perform satisfactorily when recommending developers to projects that have few pre-existing developers. Hence, collaborative filtering is not suitable for newly joined or unpopular projects. The text matching based algorithm shows a good performance, but the performance of *ConRec* is superior. For projects with adequate pre-existing developers (Figure 7), *ConRec* achieves the best performance, followed by *WCF* and *CF*. *ConRec* evidently performs better than *WCF*; thus, the complement of the text matching based results can facilitate the improvement of the recommendation results even for projects with adequate pre-existing developers. The *CF* algorithm performs better than the text matching based algorithm, thereby indicating that a collaborative network is more suitable than text information when recommending. This factor is our basis for using the text matching based algorithm to complete the matching result of *WCF*. The text matching based algorithm performs better than *CF* when recommending developers to an entire project set (see Figure 5). The reason is that, the number of projects that have few pre-existing developers is 888, which comprises 14.8% of all the test projects.

Therefore, the terms generated from the text information of the projects can facilitate solving the cold start problem and improve the recommendation results for projects that have few or adequate pre-existing developers.

C. Performance of ConRec in Different Threshold Values of Expert Selector

Considering about Q3, we perform a contrast experiment with different threshold values of the expert selector. Figure 8 shows the accuracy of *ConRec* when recommending 50 developers, where the x- and y-axes represent the threshold and accuracy values, respectively. We set the threshold value range from 2 to 30 with every other number.

Figure 8 shows that when the threshold value is set to 4, the accuracy performs the best. Hence, when a developer commits

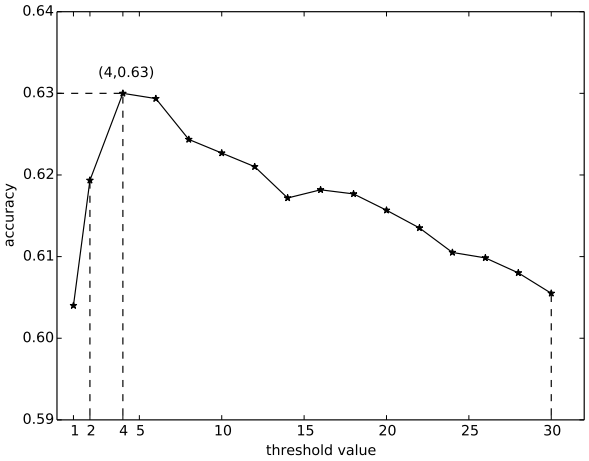

Fig. 8: Accuracy Change with Different Threshold Values

over 4 times with the target language, we can regard him/her as an expert. For threshold values below 4, the accuracy is not that good because developers who are unfamiliar with the programming language remain included. For threshold values over 4, the performance decreases with the increase of the threshold value. The reason is that *ConRec* filters accurate developers and use text matching based algorithm to complete the recommendation result.

In conclusion, the threshold value of the expert selection part influences the performance of *ConRec*. If the value is considerably small, then developers who are unfamiliar with the target programming language are not filtered. If the value is substantially large, then a few related developers are filtered, and the text matching based algorithm is initiated. However, the results are not as good as those of the collaborative filtering algorithm. Therefore, we select the threshold value to 4 in

ConRec.

V. Conclusion and Future Work

The study aims to develop a general algorithm for recommending suitable committers to open source projects based on a collaborative network. We derive three research questions and perform many experiments on 5,995 popular projects in GitHub. The results indicate the superior performance of *ConRec*. *ConRec* somewhat solves the cold start problem by combining the collaborative filtering and text matching algorithms. *ConRec* is suitable for different open source communities because it simply considers the commit information and the text information of a project, including the name, description and language. These details are common information in social coding communities.

However, it still has a few limitations.

Expert selection is relatively simple, that is, the programming skill of a developer is more related to the quantity of code than to the number of commits.

Many projects involve over one programming language. Therefore, developers may commit to a project without using the major programming language.

Many short-term developers engage in open source projects; hence, they are likely to commit to a project in a short time. Therefore, the algorithm should also consider the commit time of pre-existing developers. For example, commits from a long time ago may not be considered.

Different types of activities of developers like fork, watch and star may also be considered. Different types of developers can simultaneously participate in different numbers of projects.

For our future study, we will firstly build up a prototype system by implementing our algorithm. Thereafter, we improve the algorithm by simultaneously considering these limitations and iterating the system.

Acknowledgment

The research is supported by the National Grand R&D Plan (Grant No. 2016-YFB1000805) and National Natural Science Foundation of China (Grant No.61502512,61432020,61472430,61532004).

References

[1] M. Silic, "Dual-use open source security software in organizations–dilemma: Help or hinder?" *Computers & Security*, vol. 39, pp. 386–395, 2013.

[2] T. T. Nguyen, T. N. Nguyen, E. Duesterwald, T. Klinger, and P. Santhanam, "Inferring developer expertise through defect analysis," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 1297–1300.

[3] D. Ma, D. Schuler, T. Zimmermann, and J. Sillito, "Expert recommendation with usage expertise," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 535–538.

[4] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 141–150.

[5] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[6] M. Gharehyazie, D. Posnett, and V. Filkov, "Social activities rival patch submission for prediction of developer initiation in oss projects," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 340–349.

[7] R. Robbes and D. Röthlisberger, "Using developer interaction data to compare expertise metrics," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 297–300.

[8] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008, pp. 121–124.

[9] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[10] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2275–2292, 2012.

[11] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 22–30.

[12] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Reverse engineering (WCRE), 2013 20th working conference on*. IEEE, 2013, pp. 72–81.

[13] 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," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 2–11.

[14] Y. Yu, H. Wang, G. Yin, and C. Ling, "Reviewer recommender of pull requests in GitHub," in *ICSME*. IEEE, 2014, pp. 609–612.

[15] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 367–371.

[16] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 931–940.

[17] H. Kagdi, M. Hammad, and J. I. Maletic, "Who can help me with this source code change?" in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, 2008, pp. 157–166.

[18] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[19] R. Grewal, G. L. Lilien, and G. Mallapragada, "Location, location, location: How network embeddedness affects project success in open source systems," *Management Science*, vol. 52, no. 7, pp. 1043–1056, 2006.

[20] M. Weiss, G. Moroiu, and P. Zhao, "Evolution of open source communities," in *IFIP International Conference on Open Source Systems*. Springer, 2006, pp. 21–32.

[21] J. Hahn, J. Y. Moon, and C. Zhang, "Impact of social ties on open source project team formation," in *IFIP International Conference on Open Source Systems*. Springer, 2006, pp. 307–317.

[22] J. Hahn and Moon, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," *Information Systems Research*, vol. 19, no. 3, pp. 369–391, 2008.

[23] G. Madey, V. Freeh, and R. Tynan, "The open source software development phenomenon: An analysis based on social network theory," *AMCIS 2002 Proceedings*, p. 247, 2002.

[24] R. Venkataramani, A. Gupta, A. Asadullah, B. Muddu, and V. Bhat, "Discovery of technical expertise from open source code repositories," in *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 2013, pp. 97–98.

[25] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, 2003.