EARec: Leveraging Expertise and Authority for Pull-Request Reviewer Recommendation in GitHub

Haochao Ying[1], Liang Chen[2], Tingting Liang[1], Jian Wu[1]
[1] Zhejiang University, Hangzhou, China
{haochaoying, liangtt, wujian2000}@zju.edu.cn
[2] RMIT University, Melbourne, Australia
liang.chen@rmit.edu.au

ABSTRACT

Pull-Request (PR) is a primary way of code contribution from developers to improve quality of software projects in GitHub. For a popular GitHub project, tens of PR are submitted daily, while only a small number of developers, i.e core developers, have the grant to judge whether to merge these changes into the main branches or not. Due to the time-consumption of PR review and the diversity of PR aspects, it is becoming a big challenge for core developers to quickly discover the useful PR. Currently, recommending appropriate reviewers (developers) for incoming PR to quickly collect meaningful comments, is treated as an effective and crowdsourced way to help core developers to make decisions and thus accelerate project development. In this paper, we propose a reviewer recommendation approach (EARec) which simultaneously considers developer expertise and authority. Specifically, we first construct a graph of incoming PR and possible reviewers, and then take advantage of text similarity of PR and social relations of reviewers to find the appropriate reviewers. The experimental analysis on MSR Mining Challenge Dataset[1] provides good evaluation for our approach in terms of precision and recall.

CCS Concepts

•**Software and its engineering** → **Programming teams;** *Software version control;* •**Information systems** → *Recommender systems;*

Keywords

Pull-Request, Reviewer Recommendation, Social Network, Random Walk with Restart

1. INTRODUCTION

GitHub, the largest web-based source code host in open source, provides a convenient service for developers to collaborate with each another [8]. The pull-based software development model is supported by GitHub to make collaboration more flexibly and efficiently with comparison to traditional collaborative methods such as patch submission and acceptance via mailing lists and issue tracking systems [15].

In GitHub, almost half of the collaborative projects use PR exclusively or complementary to the shared repository model [4]. According to the analysis on Top-90 popular projects, it could be observed that almost 45% of PR are finally merged in the main branch to improve the qualify of projects. Moreover, it is common in popular project to receive tens of PR daily covering nearly 60% of code commits from contributors [14]. Thus, PR is considered as a primary method of code contributions from developers to improve quality and extend capabilities of software projects.

However, only core developers can merge PR in this mechanism while anyone can create a PR after cloning the repository. As thousands of developers utilize this mechanism to convey their opinions, it increases a lot of stress on core developers to decide whether to integrate PR into the main branch, especially in large projects. Figure 1 shows the number of PR and reviewers in five popular projects in GitHub. From Figure 1(a), we can observe that large projects generate thousands of PR to facilitate software improvement and maintenance. For example, there are more than 12,000 PR in the *homebrew* project. Therefore, how to deal with the volume of incoming PR effectively is becoming a big challenge in large projects [4].

In order to solve above problem, one way to tackle it is to accelerate the PR review process which is highly related to software evolution and maintenance. Unfortunately, the discussion among reviewers is time-consuming and reviewer assignment is now organized manually in GitHub. According to the results of analysis, PR requires 17 days to close on average in our dataset. What's worse, some relevant reviewers may not notice the corresponding PR timely, since GitHub sends the notification to every follower without considering their interests. Currently, recommending appropriate reviewers for incoming PR to quickly collect meaningful comments, has been treated as an effective and crowdsourcing way to help core developers to make decision.

However, it is not appropriate to directly employ traditional recommendation algorithm to solve this problem. Compared with the traditional scenarios, the main differences of PR reviewer recommendation problem are two-fold:

- **Data Sparsity**. According to the statistics in [3], 95% of PR are discussed by less than 4 reviewers. From Figure 1(b), it could be observed that there are more than

[1]http://ghtorrent.org/msr14.html

CSI-SE'16, May 16 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897659.2897660

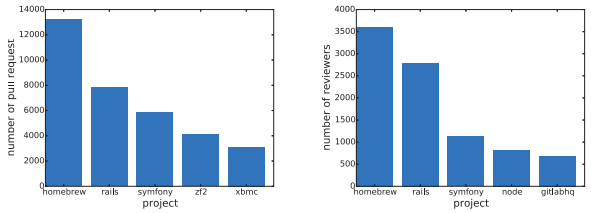


Figure 1: the number of PR (a) and reviewers (b) in five popular projects in GitHub

2,500 reviewers in some popular projects. Thus, the PR-Reviewer matrix is very sparse, which poses challenge to the traditional recommendation algorithm.

- **Information Limitation**. Through the observation of PR, we find that the semantic information is very limited and even unvalued sometimes. For example, the title (*'0install 2.8 #42405'*) and description (*'Another attempt, after the previous formula had several issues. Hopefully this is better...'*) cannot convert into any useful information[2]. Therefore, we can not only use the semantic information to recommend reviewers.

In this paper, we propose EARec, an approach leveraging reviewer expertise and authority for PR reviewer recommendation to help core developers automatically find appropriate reviewers. The intuition behind this is that recommended reviewers are not only expertise on the content of incoming PR, but also has more authority than other reviewers. Expertise can be modeled by LSI model [1], while authority is explored by graph-based propagation. Finally, in order to evaluate EARec, we conducted our experiments based on MSR Mining Dataset. In particular, we extract 9 projects which have received over 1500 PR. The results show that our approach performs better than other two representative methods in most case.

The major contributions can be summarized as follows:

1. We consider developer expertise and authority for reviewer recommendation of incoming PR by leveraging textual semantic of PR and social relationships of contributors.

2. We propose an approach based on Random Walk with Restart to rank the candidate reviewers for incoming PR.

3. We conduct experiments on the MSR Mining Challenge Dataset to evaluate the effectiveness of our proposed method.

The remainder of this paper is organized as follows. Section 2 describes background and related work. Section 3 presents the overview and detailed techniques of EARec. Section 4 gives the experimental results and discussion. Finally, Section 5 draws conclusion.

2. BACKGROUND AND RELATED WORK

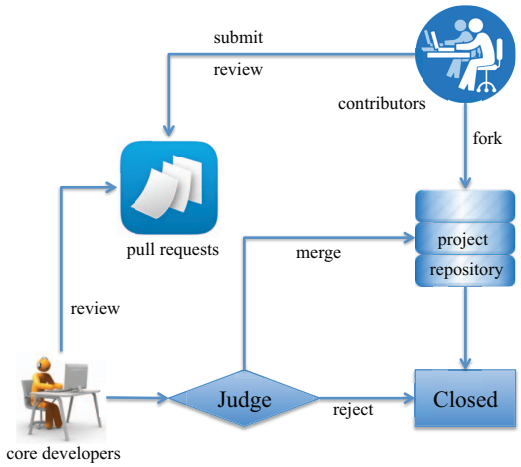


Figure 2: An overview of PR mechanism

2.1 Pull-Request Mechanism

The overview of PR mechanism is presented in Figure 2. First of all, core developers create a project repository consisting of source codes and software documents in GitHub for social coding. Any potential contributor can fork this repository and push changes (e.g implement new features or fix bugs) in his personal cloned repository without requiring access be granted to the source repository. When these changes are ready to be submitted to the source repository, the contributor create a PR for review. There are two types of review comments: code review and discussion [5]. The former usually means the reviewer comments on specific sections of code to pinpoint the potential improvements, while the latter generates comments on the overall contents regarding the suitability of the PR. Note that any GitHub user can act as a reviewer to participate in review process.

Consequently, the contributor may update PR with new commits under the reviewers' suggestions and then the reviewers will discuss repeatedly. Finally, the core developers merge the PR to the source repository if it is satisfied. Otherwise, the PR may be rejected by reason of redundant, uninteresting or duplicate in the review process. Thus, the review process is an important part of distributed software development.

However, popular projects receive so many PR daily that core developers have a lot of stress to decide whether merge them or not. Although any developers can use the @mention tool to find corresponding developers for reviewing incoming PR in Github, the review time would be longer if these developers do not catch this PR timely. Therefore, an automatic PR reviewers recommendation tool would accelerate the review process and increase the efficiency.

2.2 Related Work

The prevalence of distributed version control systems (GitHub, etc) poses the opportunity to utilize pull-based development such as code reviewing systems and integrated issue trackers. Gousios et al. [3] mentioned PR model offers fast turnaround, increased opportunities for community engagement and decreased time to incorporate contributions. However, the evaluation of PR is a complex and challenging pro-

[2]https://github.com/Homebrew/homebrew/pull/42405

cess. Recently, some publications have begun to investigate which factors influence PR acceptance and latency in GitHub. For PR acceptance, Gousios et al. [3] showed that the decision to merge a PR is mainly influenced by whether it modifies recently modified code. Tsay et al. [11, 12] considered the strength of social connection between the submitter and integrators is the crucial factor of PR acceptance. And for PR latency, Gousios et al. argued in [3, 4] that the time to merge is affected by the developers previous track record, the size of project and its test coverage and the projects openness to external contributions. Yu et al. [13] proposed the hypothesis that previously identified social and technical factors influence PR latency in expected ways.

As showed in [4], for many projects, especially the bigger or more popular project, the volume of incoming contributions is too big, which leads the time consuming process of PR review. Thus it is essential to build a reviewer recommendation system to make the review process more efficient. To the best of our knowledge, there exist few publications about PR assignment. Yu et al. [15] conducted a directed social network among reviewers and contributors called Comment Network and a recommendation approach was proposed based on it. This method is quite different with our proposed EARec, although we construct similar network. First, we employ different method, LSI model, for text similarity. Second, our network is constructed to measure reviewer authority, which has two types of nodes (incoming PR and reviewers) and is undirected. Third, Random Walk with Restart algorithm is utilized to balance reviewer expertise and authority in our approach.

3. METHODOLOGY

In this section, the overview as well as the detailed techniques of our reviewer recommendation algorithm EARec will be presented.

3.1 An overview of EARec

Inspired by the idea in [6], EARec automatically learns candidate reviewers' expertise and authority, and then recommend highly relevant reviewers for incoming PR. Graph construction and propagation over the graph mainly compose our approach EARec.

To model reviewer expertise and authority, we first construct a graph. As showing in Figure 3, there are two types of nodes as well as edges in the graph. $N+1$ nodes represent N candidate reviewers and one incoming PR. To achieve expertise objective, LSI model [1] is employed to the incoming and historical PR of candidate reviewers. Specifically, textual sematic information of PR extracts from its title and description, so that the edge between a reviewer and the incoming PR can be built and measured by the topic model-based similarity. Obviously, the strong connection means the reviewer has strong skill in the specific area (e.g. have reviewed many PR similar to the incoming). The edge between reviewer and reviewer signifies the number of PR which they have commented together in history. Thus, in the next step of propagation process, authority information can be involved. To better integrate the expertise and authority of each candidate reviewer, we propose a Random Walk with Restart (RWR) model [9]. In this model, there is some probability to jump back to the start node at each step. Through this, we can balance the developer expertise and authority effectively. Finally, until converging or achieving the maximum step, each reviewer obtains a utility score on behalf of the relevance with incoming PR. The set of reviewers with the highest scores is recommended to review the incoming PR.

Next, we explain how to construct graph and propagate over the graph in detail.

3.2 Graph Construction

Let $\mathcal{R} = \{r_1, r_2, ..., r_N\}$ denotes the candidate reviewers set, and let iPR denotes the incoming PR to be recommended reviewers. Then the graph can be denoted as $G = (V, E, W)$. The nodes set is $V = \mathcal{R} \cup iPR$, and the edge set is $E = \{e_{ij} | 0 < i, j \leq (N+1)\}$, where e_{ij} denotes the edge between node v_i and v_j. W denotes the weight set, where w_{ij} indicates the weight of edge e_{ij}. The edges and the value assignment of W would be established in following two subsections.

Reviewer-Reviewer Relationship. Consider the review comments provided by the developers, we assume there exists an edge between reviewer r_i and r_j if and only if the two reviewers have together commented at least one PR. The edge weight w_{ij} is decided by the number of PR reviewer r_i and r_j have commented together. The intuition behind is: 1) if a reviewer has many relationships with others, he would be regarded as an experienced reviewer; 2) the edge weight between two reviewers indicates the degree of similarity between their interest. From Figure 4, we can observe that the connections among reviewers have 'long tail' distribution. A small part of reviewers (probably core developers) join in a lot of PR.

PR-Reviewer Relationship. Suppose the set of PR that have been commented by reviewer r_i is denoted by $pr_i = \{pr_i^1, ..., pr_i^j, ..., pr_i^{m_{pr_i}}\}$, where m_{pr_i} is the number of PR commented by reviewer r_i. Then the edge weight between the incoming PR iPR and the reviewer r_i can be estimated according to the similarity between PR and the set of PR pr_i.

Latent Semantic Indexing. In order to obtain the similarity between the commented PR pr_i offered by reviewer r_i and an incoming PR iPR, we apply Latent Semantic Indexing (LSI) [1] model which has been widely utilized for document similarity computation. LSI tries to map queries and documents into a space with latent semantic dimensions. Thus there would exists high cosine similarity of a query and a document with conceptually similar words though they only share few words. Here the description and title of each PR is considered as one document.

The collection of titles and descriptions of commented PR can be represented as a term-document matrix $A_{t \times d}$. After the transformation of the term-document matrix, LSI applies singular value decomposition (SVD) to the matrix:

$$A = TSD^T, \tag{1}$$

where T is a $t \times r$ matrix with orthonormal columns, D is a $d \times r$ matrix with orthonormal columns, and S is an $r \times r$ singular value matrix with the diagonal entries sorted in decreasing order. LSI uses a truncated SVD that keeps only the k largest singular values and the associated vectors:

$$\hat{A} = T_k S_k D_k^T, \tag{2}$$

where k is the dimensionality of the reduced space and $k \ll r$. The rows in T_k and D_k respectively are the term vectors and document vectors in latent semantic space.

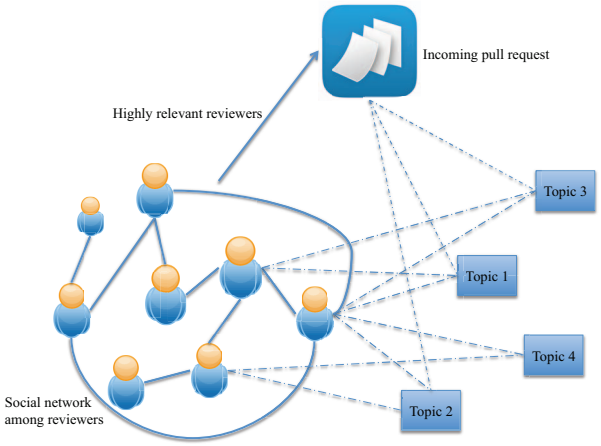


Figure 3: The graph of reviewers recommendation

Before the similarity calculation, the title and description of incoming PR iPR should be represented as a vector in the k-dimensional latent semantic space by multiplying the transpose of iPR's term vector with T_k and S_k^{-1}: $t_{\hat{iPR}} = t_{iPR}^T T_k S_k^{-1}$. Then cosine similarity between PR pr_i^j and an incoming PR iPR can be calculated as:

$$sim(pr_i^j, iPR) = \frac{t_{pr_i^j} \cdot t_{\hat{iPR}}}{|t_{pr_i^j}| \times |t_{\hat{iPR}}|}, \tag{3}$$

where $t_{pr_i^j}$ and t_{iPR} respectively denote the term vectors of the PR pr_i^j and iPR in latent semantic space.

Weight Assignment. Since it is usually for a reviewer to comment a PR many times, we take the number of comments provided by a reviewer in the same PR into consideration. The weight $w_{r_i,iPR}$ of the edge between the incoming PR and reviewer r_i can be evaluated by

$$w_{r_i,iPR} = \frac{\sum_{j=1}^{m_{pr_i}} n_{pr_i^j} sim(pr_i^j, iPR)}{\sum_{j=1}^{m_{pr_i}} n_{pr_i^j}}, \tag{4}$$

where $n_{pr_i^j}$ denotes the number of comments submitted by r_i in PR pr_i^j and m_{pr_i} is the number of PR commented by reviewer r_i. The rationale behind this assignment is that if a reviewer has commented PR which are similar in semantics with the incoming PR, he would be considered as an expert candidate to review the new PR.

3.3 Propagation Over the Graph

we present propagation approach over the constructed graph to recommend reviewers for incoming PR. We first formulate the problem and then model developer expertise and authority based on random walk with restart (RWR).

Problem Definition. Given the initial PR node iPR, the task is to propagate the initial PR information through the entire graph to predict the relevance score for each reviewer node. The relevance score capture not only the global structure of graph but also the multi-facet relationship between nodes [9]. Then according to the scores, we can recommend relevant reviewers for incoming PR. The RWR process can consider the reviewer expertise and authority simultaneously when propagating in the graph. Let $\vec{q_i}$ denotes $n \times 1$ starting vector in which the i^{th} element equal 1 and 0 for others. Let $\vec{p_i} = [p_{i,j}]$ denotes $n \times 1$ ranking vector where $p_{i,j}$ is the relevance score of node j w.r.t node i. Note that the i^{th} element represents incoming PR node iPR.

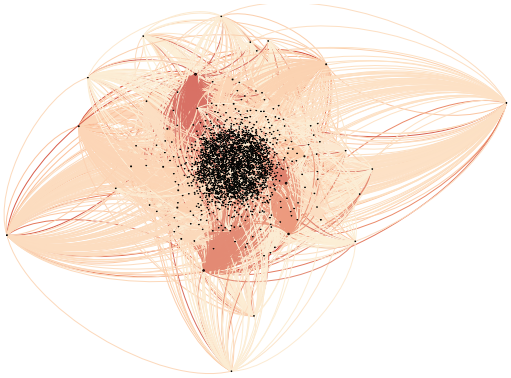


Figure 4: Reviewers connection in project *symfony*

Modeling Developer Expertise and Authority. To model developer expertise and authority, we employ random walk with restart process [10]. First, we compute adjacency matrix $W^{'}$ in the following way:

$$W^{'} = \begin{cases} \frac{w_{ij}}{\sum_j w_{ij}} & \mathrm{V}_i = iPR, \mathrm{V}_j \neq iPR \\ \frac{w_{ij}}{\sum_j w_{ij}} & \mathrm{V}_i \neq iPR, \mathrm{V}_j \neq iPR \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where the edge wieght w_{ij} is as defined in subection 3.2. Actually, the matrix $W^{'}$ is the row normalization of weight matrix W.

Developer expertise can be built by the semantic similarity and strong social links among N reviewers represent authority. Therefore, we assign higher weight to nodes that are both relevant to the incoming PR and well connected in RWR process. Specifically, suppose that a random walker starts from node i. The walker can randomly choose neighborhood as next node with the probability defined in adjacency matrix $W^{'}$ in each step. Except that, he has some probability λ to jump back to the starting node i. After multiple iterations, we will obtain the steady state probability p_{ij}, which means the probability score of each reviewer with respect to the incoming PR. This process can be formulated as follow:

$$\vec{p_i} = (1 - \lambda) W^{'} \vec{p_i} + \lambda \vec{q_i} \tag{6}$$

where λ ranges from 0 to 1, which balances the developer expertise and authority. With the increase of λ, authority place more importance.

In order to solve above RWR problem, one of the most widely used way is the iterative method. By iterating equation 6 until convergence (the L_2 norm of successive estimates of p_i is below threshold or maximum iteration step is reached), the steady probability will be computed. Then we can assign reviewers with top high probability score to review the incoming PR. Note that the probability score has taken candidate developer expertise and authority into consideration.

4. EXPERIMENTS AND EVALUATION

In this section, we conduct experiments to answer the following questions: (1) Does EARec accurately recommend appropriate reviewers? (2) What is the impact of λ in EARec? (3) Does EARec have consistent performance on different type of project? All the experiments were conducted in a machine with a 2.2 GHz Intel CPU and 16 GB RAM, running OS X Yosemite.

Experiment Setup and Data Preprocessing. To illustrate the performance of our proposed approach, we use MSR challenge dataset which is a trimmed down version of the original GHTorrent dataset [2]. There are 75930 closed PRs (34309 merged and 41621 merge failed) made to 88 base projects from 2010-08-31 to 2013-10-06. According to the time sequence, PR in each project are divided into two parts: 90% as train set and the rest as test set. The titles and descriptions can be extracted from collection of pull_requests while reviewer relation can be observed from PR comments. Note that there are two types of PR comments: code review and discussion, which store in collection of pull_request_comments and issue_comments respectively in the dataset. Our experiments consider both of them. In order to compute the text similarity validly between incoming PR and historical PR, we first remove punctuation and stop words over the titles and descriptions. Then the suffix of all rest words that have same stem is striped by the Porter stemmer [7]. After that, those PR with less than 10 words should be omitted. Moreover, some simple PR which contain less than 4 comments need not be reviewed in test set. Consequently, we won't recommend reviewers for this kind of PR. After data filtering, the preprocessed experiment dataset is presented in Table 1.

Evaluation Metric. To quantify the performance of our method over each project, we use two standard metrics: Precision and Recall, which have been widely adopted in the information retrieval community.

$$Precision = \frac{1}{m}\sum_{i=1}^{m}\frac{|Rec(i) \cap Actual(i)|}{|Rec(i)|} \tag{7}$$

$$Recall = \frac{1}{m}\sum_{i=1}^{m}\frac{|Rec(i) \cap Actual(i)|}{|Actual(i)|} \tag{8}$$

where m is the number of test PR in each project, $Rec(i)$ is the set of recommendation reviewers for ith test PR, and $Actual(i)$ is the set of actual reviewers for ith test PR. In addition, we employ traditional F1 measure for analyzing experiment results, which conveys the balance between precision and recall:

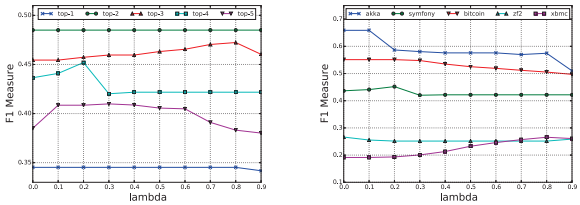
$$F1 = \frac{2Precision \times Recall}{Precision + Recall} \tag{9}$$

Performance Comparison. In order to evaluate the effectiveness of our proposed approach, we compare our method with two state-of-art methods in this section. The details of these algorithms are given below:

- ***POP***. Any GitHub user can participate in any PR by a code review or a discussion. However, only core developers have merge decision so that they usually join in the discussion to understand the changes committed by contributor(PR creator). Therefore, it is common that a few core developers review the majority of PR in some projects. In this baseline approach, the top-k active developers will be recommended to each incoming PR, which have reviewed the most number of PR in the history just like recommending popular items in traditional commodity recommendation.

- ***CN***. In this approach, the authority among developers are extracted from comment relations between contributors and reviewers in historical PR. Specifically, if the contributor has submitted PR before, this approach recommends the users who have interacted with him directly. If the contributor does not submit any PR but reviews some, this approach utilizes association rule mining to construct authority. Besides, for a newcomer, the most probable recommendation candidates consist of the most active reviewers in different kinds of communities [15].

- ***EARec***. In this approach, we exploit developers expertise and authority for reviewers assignment. In particular, our method combines semantic similarity and social network, which is proposed in Section 3.



(a) (b)

Figure 5: the impact of λ in project *symfony* (a) and in different project (b)

Prediction Accuracy. To evaluate the detailed performance of the above three reviewer recommendation approaches, we run these in each project. Note that most of PR are discussed by less than 4 people [3]. Therefore, we only presents part of our results from top-1 to top-5 recommendation in 9 projects. Empirically, we set that topic number to 10, different λ in different projects and 100 iterations for random walk with restart algorithm in our proposed approach.

Table 2 shows the results of each approach in comparison of precision and recall. As we can observe, our proposed method outperforms the other methods in most cases. Specially, the precision in project *akka* reaches 0.72 in average, which improves 31% and 44% compared to *POP* and *CN* method respectively. Moreover, with the increasing number of recommendations, the recall improves in general. This suggests that our approach captures the case where the real reviewer involving the discussion and code review was not chosen and the actual reviewer is the sub-optimal choice. For example, consider a situation where the best reviewer has a schedule (e.g vacations) or goes through high workload, it is reasonable that our approach EARec will recommend other reviewers with relevant expertise and authority.

Impact of λ. In our approach, λ is the restart probability of random walk algorithm, which balance between developer expertise and authority. To further investigate its impact, we set λ from 0 to 0.9 with step of 0.1 in the experiments.

Figure 5 shows the impact of λ in project *symfony* and 5

Table 1: Preprocessed Experiment Dataset

Project	Language	#PR		#comments		#candidates
		train	test	train	test	
rails	Ruby	3036	222	25258	1881	2235
homebrew	Ruby	3344	254	23169	2614	2523
symfony	PHP	2213	148	22129	1334	902
zf2	PHP	1153	75	8334	534	404
bitcoin	TypeScript	990	77	9797	687	254
scala	Scala	1521	133	15624	1526	142
akka	Scala	649	48	12064	779	90
xbmc	C	1610	130	16892	1475	439
node	JavaScript	1011	69	7132	751	616

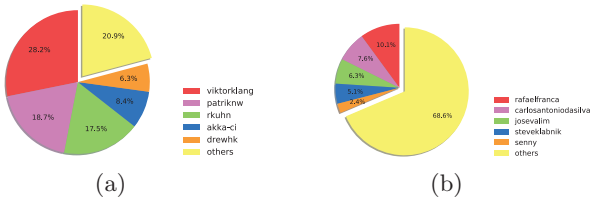
Table 2: Precision/Recall Comparison (a greater value means a better performance)

Method	POP	CN	EARec	POP	CN	EARec	POP	CN	EARec
Project	akka			homebrew			symfony		
top-1	0.29/0.07	0.57/0.14	**0.90/0.24**	0.76/0.24	0.65/0.21	**0.76/0.24**	0.76/0.22	0.54/0.16	**0.76/0.22**
top-2	0.59/0.31	0.54/0.26	**0.82/0.43**	0.52/0.33	0.45/0.30	**0.52/0.33**	0.66/0.38	0.52/0.30	**0.66/0.38**
top-3	0.65/0.50	0.50/0.36	**0.65/0.50**	0.52/0.48	0.43/0.42	**0.52/0.48**	0.49/0.42	0.41/0.35	**0.51/0.44**
top-4	0.65/0.67	0.48/0.47	**0.65/0.67**	0.39/0.48	0.34/0.44	**0.46/0.56**	0.37/0.42	0.33/0.37	**0.42/0.48**
top-5	0.59/0.74	0.41/0.50	**0.59/0.74**	0.37/0.56	0.31/0.48	**0.37/0.56**	0.31/0.45	0.27/0.38	**0.35/0.50**
Avg.	0.55/0.46	0.50/0.35	**0.72/0.52**	0.51/0.42	0.44/0.37	**0.53/0.43**	0.52/0.38	0.41/0.31	**0.54/0.40**
Project	bitcoin			zf2			node		
top-1	0.55/0.12	**0.62/0.14**	0.55/0.13	0.71/0.23	0.55/0.18	**0.71/0.23**	0.68/0.22	0.47/0.15	**0.68/0.22**
top-2	0.51/0.24	**0.63/0.28**	0.55/0.25	0.43/0.27	0.30/0.20	**0.43/0.27**	0.57/0.36	0.43/0.26	**0.57/0.36**
top-3	0.48/0.33	**0.60/0.40**	0.51/0.35	0.28/0.27	0.23/0.22	**0.30/0.29**	0.38/0.36	0.35/0.30	**0.43/0.39**
top-4	0.22/0.29	0.21/0.27	**0.24/0.30**	0.22/0.29	0.21/0.27	**0.24/0.30**	0.32/0.39	0.28/0.31	**0.32/0.39**
top-5	0.55/0.65	0.47/0.52	**0.55/0.65**	0.19/0.30	0.18/0.29	**0.20/0.32**	**0.29/0.46**	0.24/0.33	0.26/0.40
Avg.	0.51/0.36	**0.57**/0.36	0.55/**0.38**	0.37/0.27	0.29/0.23	**0.38/0.28**	**0.45/0.36**	0.35/0.27	**0.45**/0.35
Project	rails			xbmc			scala		
top-1	0.38/0.11	0.27/0.08	**0.38/0.11**	0.33/0.10	0.21/0.05	**0.33/0.10**	0.18/0.04	**0.33/0.09**	0.18/0.04
top-2	0.29/0.17	0.22/0.14	**0.29/0.17**	0.25/0.14	0.17/0.09	**0.26/0.15**	0.22/0.12	**0.35/0.19**	0.22/0.12
top-3	0.22/0.18	0.21/0.19	**0.26/0.22**	**0.22/0.19**	0.19/0.14	0.21/0.18	0.30/0.26	**0.31/0.27**	0.30/0.26
top-4	0.22/0.24	0.17/0.20	**0.22/0.24**	0.19/0.21	0.19/0.19	**0.25/0.28**	0.41/0.49	0.29/0.33	**0.41/0.49**
top-5	0.18/0.25	0.15/0.22	**0.18/0.25**	0.18/0.24	0.18/0.23	**0.25/0.33**	**0.39/0.60**	0.27/0.37	0.34/0.51
Avg.	0.26/0.19	0.20/0.17	**0.27/0.20**	0.23/0.18	0.19/0.14	**0.26/0.21**	0.30/**0.30**	**0.31**/0.25	0.29/0.28

different projects. Note that the higher F1 score means better performance. From Figure 5(a), we can find the patterns from top-1 to top-5 recommendation are different. For example, the λ does not affect the precision and recall in top-1 recommendation. While the performance achieves best in top-4 when $\lambda = 0.2$. That means we must set different λ for different number of reviewer recommendations in each project. The influence of λ to top-4 recommendation of 5 projects presents in Figure 5(b). There are also different patterns, which convey the optimal value of λ in reviewer assignment for specific project is different. For instance, F1 score increases as λ changes from 0.4 to 0.8 in *xbmc* but decrease in *bitcoin*.

Impact of type of project. We explain why some projects can achieve very high precision and recall while others can't in all of three methods. To exploit this reason, project *akka* and *rails*, as two typical representative of these projects, are employed for further study.

There are two types of reviewer in PR: core and external developers. If most of comments come from a small part of core developers, it is obvious that recommending them for new PR will reach high precision and recall. Figure 6 presents the reviewer distribution in train dataset of *akka*. Top-5 reviewers who probably are core developer generate 79.1 percentage of comments in Figure 6(a). It seems that core developers control and dominate the review activity. Therefore, if we recommend these reviewers for new PR, we can achieve high performance as show in Table 2. However, *POP* approach gets low precision in top-1 recommendation. This is because the most comment developer *viktorklang* just contributes 28.2%. It is worth mentioning our approach achieve 90% precision in top-1 recommendation which is much better than other methods.



(a) (b)

Figure 6: reviewer distribution in comments in *akka* (a) and *rails* (b)

To be contrary, from Figure 6(b), we can observe that top-5 reviewers just occupy 31.4% comments of project *rails*. That means the PR review activity are well-distribution and balanced. Therefore, it is difficult relatively for every PR to assign appropriate reviewers. Although three approaches don't achieve high performance, our approach outperforms the others in this project.

5. DISCUSSION

From the results, we can observe that EARec and CN get higher performance than the baseline approach POP in most projects. This suggests that EARec and CN successfully utilize social network for reviewer recommendation. However,

they capture different factor to explain why a developer will comment a specific PR. CN considers the common interests among developers for reviewer recommendation, while EARec deems that developer expertise and authority is more important. Because EARec outperforms CN in most case, we believe that our approach depicts the problem more accurately.

Internal Validity. In our approach, λ is the restart probability to the start node. From the results, we have found that in different projects or different number of recommendations, the optimum λ is different. Therefore, the results may be impact ont the setting of λ in other projects. Moreover, many factors (e.g the developer workload and schedule) influence the decision to involve the discussion and code review of PR. I may be the case that when a reviewer receives a PR from our approach, he could be on holiday or busy doing other tasks, whereas we treat it as such.

External Validity. In this paper, the evaluation of our proposed approach is limited to nine popular open source projects. Although the number of projects to validate is enough, we cannot claim that our approach achieves satisfactory results in other projects. However, the selected projects are representative of large and popular projects, so that we believe the results derived from these projects can be generalizable to many other projects. Future studies on more types of projects, especially in smaller projects, may increase this belief.

6. CONCLUSION

In this paper, we consider developer expertise and authority and propose a novel approach for reviewer recommendation to speed up the review process and reduce the stress of core developers. In particular, our approach employ LSI model to mine the semantic similarity to model expertise, and calculate the number of PR which reviewers have commented together to represent authority. The final ranked reviewers result is computed via random walk with restart. In the experiment, our approach outperforms the other two state-of-art methods in most case.

To further explore PR mechanism, we have following plans:

- In the experience, we have present the efficiency of EARec in comparison with other two methods. However, we have no investigated the relative importance of expertise and authority respectively. We will deeply explore our approach to improve the performance.
- When computing the textual sematic similarity, we just employ the LSI model. The time factor and words segmentation effect of short comments will be taken into consideration in the furture.
- GitHub provides a lot of social information, such as the profile of reviewers and the social media (e.g fork and watch). Thus, we plan to utilize extra information to improve our approach.

7. ACKNOWLEDGEMENTS

This research was partially supported by the Natural Science Foundation of China under grant of No. 61379119, Science and Technology Program of Zhejiang Province under grant of No. 2013C01073, the Open Project of Qihoo360 under grant of No. 15-124002-002, the Fundamental Research Funds for the Central Universities.

8. REFERENCES

[1] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[2] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 233–236, 2013.

[3] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th ACM Conference on Software Engineering*, pages 345–355, 2014.

[4] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. Technical report, Delft University of Technology, Software Engineering Research Group, 2014.

[5] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th ACM Working Conference on Mining Software Repositories*, pages 92–101, 2014.

[6] X. Liu, T. Suel, and N. Memon. A robust model for paper reviewer assignment. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 25–32, 2014.

[7] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[8] M. M. Rahman and C. K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th ACM Working Conference on Mining Software Repositories*, pages 364–367, 2014.

[9] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM conference on Knowledge discovery and data mining*, pages 404–413, 2006.

[10] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining*, pages 613–622. IEEE Computer Society, 2006.

[11] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th ACM conference on Software engineering*, pages 356–366, 2014.

[12] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM Symposium on Foundations of Software Engineering*, pages 144–154, 2014.

[13] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. Wait for it: Determinants of pull request evaluation latency on GitHub. In *12th IEEE Working Conference on Mining Software Repositories*, 2015.

[14] Y. Yu, H. Wang, G. Yin, and C. X. Ling. Reviewer recommender of pull-requests in github. In *Proceedings of the IEEE Conference on Software Maintenance and Evolution*, pages 609–612, 2014.

[15] Y. Yu, H. Wang, G. Yin, and C. X. Ling. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. *21st Asia-Pacific Software Engineering Conference*, pages 335–342, 2014.