GEMiner: Mining Social and Programming Behaviors to Identify Experts in Github

Wenkai Mo, Beijun Shen, Yuming He, Hao Zhong
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, 200240, China
bjshen@sjtu.edu.cn

ABSTRACT

Hosting over 10 million repositories, GitHub becomes the largest open source community in the world. Besides sharing code, Github is also a social network, in which developers can follow others or keep track of their interested projects. Considering the multi-roles of Github, integrating heterogenous data of each developer to identify experts is a challenging task. In this paper, we propose GEMiner, a novel approach to identify experts for some specific programming languages in Github. Different from previous approaches, GEMiner analyzes the social behaviors and programming behaviors of a developer to determine the expertise of the developer. When modeling social behaviors of developers, to integrate heterogenous social networks in Github, GEMiner implements a Multi-Sources PageRank algorithm. Also, GEMiner analyzes the behaviors of developers when they are programming (e.g., their commit activities and their preferred programming languages) to model programming behaviors of them. Based on our expertise models and our extracted programming languages data, GEMiner can then identify experts for some specific programming languages in Github. We conducted experiments on a real data set, and our results show that GEMiner identifies experts with 60% accuracy higher than the state-of-the-art algorithms.

Categories and Subject Descriptors

H.2.9 [**Software Engineering**]: Management

Keywords

Github, Social Network, Experts Identification.

1. INTRODUCTION

Distributed version control systems have played an important role in modern software engineering. One of such popular systems is Git, supported by Github[1], the largest open source community in the world. As of 2015, it is reported that there have been over 9 million developers and over 21.1 million repositories on Github[2]. There are a lot of excellent projects ranging from the Linux kernel to famous Web application frameworks such as Ruby on Rails, thanks to the effort of developers on Github. In addition to its role of hosting code, Github is also a social network. Like any other social networks like Twitter, developers can follow other developers. Also, developers can watch their interested projects (also known as repositories in Github), to keep track of them.

With the increasing of open source projects and registered developers, it becomes difficult to seek experts for solving problems in various tasks, especially for those new developers. Therefore, recommendation systems play an important role to enhance the experience of developers. To recommend experts, identifying them is a prerequisite. Compared with StackOverflow[3], another social network targeting at developers, Github does not implement a tagging system for experts [1]. As a result, identifying experts in specific domains (*e.g.*, C# and C++), on Github is challenging.

Most existing approaches focus on analyzing programming behaviors [2] or social influence [3] of users. Our approach is different in the following aspects:

- Our approach identifies both experts and high-quality projects, since intuitively, high-quality projects are likely developed by experts, and experts are often willing to contribute to high-quality projects.
- When identifying experts, our approach considers both the programming behaviors and the social influence of developers.
- Instead of general experts, our approach identifies experts for some specific programming languages. As a developer cannot be an expert in all domains, it is more practical to identify domain experts.

In this paper, we propose a novel approach, called GEMiner, to mine domain experts on Github. Specifically, GEMiner consists of two major steps:

- **Network-based Expertise Modeling.** In this step, GEMiner treats Github as a social network and analyzes the social structures of each developer. Different from other social networks such as Facebook,

[1]https://github.com/

[2]https://en.wikipedia.org/wiki/GitHub
[3]http://stackoverflow.com/

Internetware '15, November 06 2015, Wuhan, China
© 2015 ACM. ISBN 978-1-4503-3641-3/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2875913.2875924

there are three kinds of social networks in Github, the following-followed network, the watching network and the collaboration network. To integrate information in the three networks, we propose a Multi-Sources PageRank algorithm, an extended PageRank algorithm. At the end of this step, each repository and developer will be assigned to a separate PageRank. The higher a value is, the better a developer or a project is.

- **Programming Behavior-based Expertise Modeling.** In this step, GEMiner treats Github as a distributed version control system. In particular, GEMiner analyzes how often a developer commits, and the programming language of each commit.

Our results show that GEMiner identifies experts with 60% higher precisions than other state-of-the-art algorithms. This paper makes the following contributions: (1) Our approach focuses on identifying experts for specific programming languages on Github. (2) Our approach utilizes information extrated from social networks and programming behaviors of each Github developer, and models the expertise of developers from the two aspects. (3) We propose a Multi-Sources PageRank algorithm to integrate heterogenous social networks on Github.

The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 formally defines the experts identification problem. Section 4 presents our approach. Section 5 we conduct experiments in real world data set to validate our method. We conclude the paper and discuss future work in Section 6.

2. RELATED WORK

Experts identification in social networks and software repositories is a hot research topic. As Github is a combination of social networks and software repositories, our approach is related to these two lines of research:

2.1 Network-based Expert Identification

Along with the popularity of social networks, one of hot research topics in social network analysis is identifying famous persons or experts, which is also a prerequisite to construct the recommendation systems. The basic idea to identify famous persons is to analyze the social influence of users in the social network [4]. The core of a social network is the social topological graph, defined as $G = (V, E)$, in which V is the vertex set and E is the edge set. In traditional graph analysis, weak ties [5] and edge betweenness [6] are two common criteria to measure the importance of edges, while node-based centralities such as degree and PageRank [7] are defined to measure the importance of a node. Based on these basic criteria, many approaches have been proposed, which can be categorized into (1) static social influence analysis, and (2) dynamic social influence analysis.

(1) **Static social influence analysis** focuses on analyzing a static snapshot of a social networks. Static social influence analysis methods can be further categorized to: (a) Network Analysis-Based; (b) Ontology-Based; and (c) Learning-Based. (a) Network Analysis-Based methods [3, 8, 9] focus on using some metrics to measure the importance of nodes or edges in social networks. Zhang *et al.* [3] tested a set of network-based ranking algorithms, including PageRank [7] and HITS [10], on the large size social network in order to identify users with high expertise and they found that the PageRank-based expert finding algorithm is better than other network-based algorithms in the online community setting. Wang *et al.* [8] modified the PageRank algorithm to evaluate user's authority. They explored three different expert ranking strategies that combine document-based relevance and authority. (b) Ontology-Based methods [11, 12, 13] are based on some existing ontologies, a description of the social network schema, such as FOAF[4]. Li *et al.* [12] developed a Find-XpRT project for finding experts via rules and taxonomies. (c) Learning-Based [14, 15] approaches aim at applying machine learning techniques to learn some knowledge from training data, and then analyzing the influence in future data. Tang *et al.* [14] proposed a Topical Factor Graph (TFG) model to formalize the topic-level social influence analysis into a unified graphical model, and presented Topical Affinity Propagation (TAP) for model learning. All these methods mentioned above can only adopt to a single social network. However, there are three kinds of social networks in Github, to integrating information in these networks, we proposed a Multi-Sources PageRank algorithm, a kind of Network Analysis-Based approaches to obtaining the social influence of each user or repository.

(2) **Dynamic social influence analysis** considers the time dimension of a social network. Holme *et al.* [15] proposed a generative model to balance the effects of selection and influence. Crandall *et al.* [16] proposed techniques for identifying and modeling the interactions between social influence and selection, using data from online communities. Scripps *et al.* [17] proposed a matrix alignment framework that incorporates the temporal information to learn the weight of different attributes for establishing relationships between developers. Our approach also considers the time factor to evaluate the expertise of user. However, it is not in networks analysis, but in programming behaviors analysis.

2.2 Behavior-based Expert Identification

Github is not only a social network, but also a distributed version control system. Developers commit their code when creating or maintaining their projects; therefore, behaviors, such as how many times a developer commits to a project, are also important criteria to identify experts. Identifying experts or high-quality projects has been investigated by several researchers [18, 19]. For high-quality projects identification, Oskar *et al.* [2] analyzed the quality of projects in Github and gave some interesting conclusions. For example, projects owned by companies and other organizations are in general more popular than projects owned by individual developers. For experts identification, Minto and Murphy [20] proposed the Emergent Expertise Locator (EEL) approach that identifies experts based on the recently edited or selected files. John and Gail [21] presented an empirical evaluation of two approaches to determining implementation expertise from the data in source and bug projects. When determining experts for fixing bugs, they consider two factors such as bug reports and bug networks. However, the networks in bug reports is less complex than networks in Github. Elben and Matthew [22] identified experts for a specific piece of code based on a code base and its fix history. Their approach models the expertise of a developer by a function of that developer modifying the code (e.g. writing a function) rather than using the code (e.g.

[4]http://www.foaf-project.org/

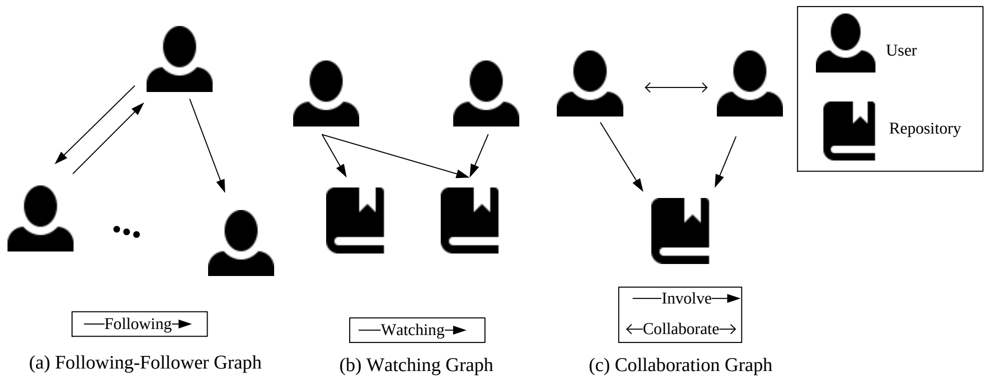


Figure 1: Social Network Schemas.

calling a function). When identifying experts by analyzing programming behaviors, our approach borrows their idea [22].

On the other hand, there are many researches about experts identification in another popular software social network, StackOverflow. However, user behaviors in StackOverflow and those in Github have huge differences, where the former focuses on asking and answering questions, while the latter focuses on programming. A kind of methods is to measure the users' expertise by non-textual information such as their reputations and so on [23, 24, 25]. Jeon *et al.* [25] proposed a method to model problem difficulty and expertise in StackOverflow, where they select features such as the activity of users and so on to create model to measure the expertise of users. A different kind of methods is based on the textual data of answers and questions users posted [26]. Riahi *et al.* used a topic model [26], User Persona Model (UPM), to model each question based on its content similar to other probabilistic topic models and then used these learned topics to represent persona distribution of user profile. Experts can be finally identified from these profiles.

3. PROBLEM DEFINITION

Supposed that we denote all the developers in Github as U and all the projects as R, we use $F \subseteq U \times U$ to denote the following and followed relations between developers, as shown in Figure 1 (a); $W \subseteq U \times R$ to denote the watching graphs, as shown in Figure 1 (b); and $C \subseteq U \times (R \cup U)$ to denote the collaboration graphs, as shown in Figure 1 (c). Consequently, a developer $u \in U$ is denoted as (F_u, W_u, C_u), where F_u, W_u, and C_u are three subgraphs that u involves. Our goal is to find experts for specific programming languages. Therefore, to evaluate expertise e of a developer u for a programming language p, our method needs to find a function $f(u, p) = e$, where $e \in \mathbb{R}$. Based on this function, an expert for a programming language can can be defined as:

Definition 1. For a developer u, a programming language p, and the skill of the developer $f(u, p)$, an *expert* for p is a developer, where $f(u, p)$ is greater than a threshold α.

Figure 2 is the overview of GEMiner. Our goal is to identify experts for programming languages, and high-quality projects in Github at the same time. Because Github is not only a social network for developers, but also a version control system, our approach, GEMiner, considers the multi-roles of the Github. At the first step, GEMiner analyzes the social network structures of each developer. Because each developer involves in three kinds of social graphs, we proposed a Multi-Sources PageRank algorithm to merge the information in these three graphs together and then obtain PageRanks for each developer and project in these graphs. The higher the PageRank is, more expertise this developer has or better the project is. At the second step, GEMiner analyzes the programming behaviors (commits) of each developer. When analyzing a commit, GEMiner identifies which programming language is used in committed files based on the file name extensions. The mapping from file extensions to programming languages is constructed manually. At the end of this step, GEMiner also returns a score for each developer in each programming

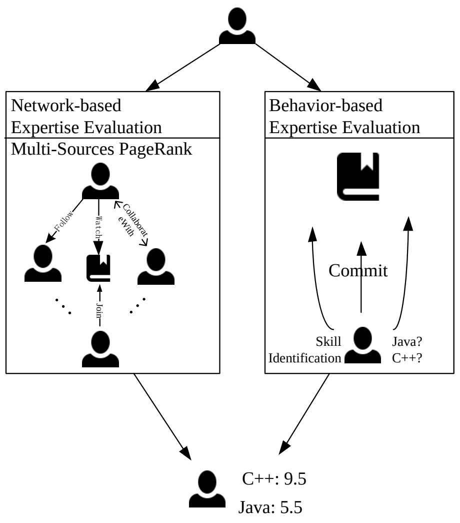


Figure 2: The Overview of GEMiner.

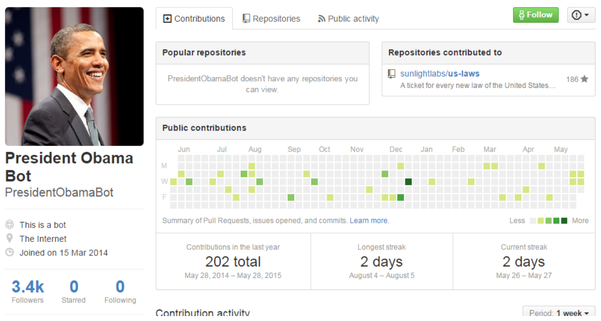


Figure 3: An Example of Famous Persons but not an Expert.

language. Finally, GEMiner combines these results obtained from Multi-Sources PageRank algorithm and those from programming behaviors analysis, and then returns the expertise of each developer in each programming language and the quality of each project in Github.

4. MINING EXPERTS IN GITHUB

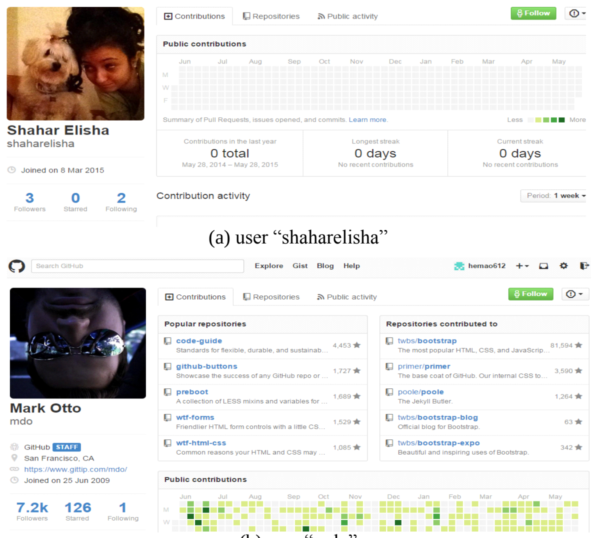
To seek experts for programming languages in Github, we consider two kinds of behaviors of each developer, which are social behaviors and programming behaviors. To model the social behaviors of each developer, we propose a Multi-Sources PageRank algorithm that combines the heterogeneous behaviors of each developer and each project, in the following-followed graph, the watching graph and the collaboration graph. Then, to model programming behaviors, we consider the number of lines of code committed by each developer. On the other hand, to identify the programming language from each commit, we manually construct a mapping from the extensions of file names to specific programming languages. Finally, GEMiner combines the results that are obtained from both models.

4.1 Multi-Sources PageRank

PageRank [7] is an algorithm that ranks websites in the Google search engine. It is a kind of link analysis algorithm, which assigns a numerical weight to each element of a hyperlinked set of documents. The higher the PageRank of a website is, the higher quantity of this page. In a graph, the PageRank of page is defined recursively and depends on the number and PageRank metric of all pages that link to it. Therefore, computing PageRank is an iterative process. Given a graph $G = (V, E)$, where V is the vertex set of the graph and E is the edge set. Initially, the algorithm assigns each PageRank as $1/|V|, v \in V$, and then updates the PageRank of each vertex by Equation 1 iteratively until it converges. In Equation 1, $L(u)$ denote the outdegree of u.

$$PR(v) = \sum_{(u,v)\in E} \frac{PR(u)}{L(u)} \qquad (1)$$

The equation has two problems. (1) If the graph is not strongly connected, all PageRanks will be zero. (2) If there exists a trap node in the graph, which does not point to any other nodes except itself, PageRanks of all other nodes will decline to zero and thus the ranking is meaningless.



(a) user "shaharelisha"

(b) user "mdo"

Figure 4: An Example of Friends of Experts.

To overcome the two problems, in this paper, we use a variant PageRank algorithm [27] as shown in Equation 2, to calculate PageRanks. In Equation 2, d is a damping factor, and usually is set to 0.85. It plays as a random walk, which means the current node can either jump to an adjacent node or an arbitrary node in the graph. With the random walk, traps and isolated nodes problem are solved.

$$PR(v) = \frac{1-d}{|V|} + d * \sum_{(u,v)\in E} \frac{PR(u)}{L(u)} \qquad (2)$$

Following-Followed Graph. The following-followed graph (a sample is shown in Figure 1(a)) is a very common graph in social networks. Based on this type of graphs, the PageRank algorithm is very effective to find famous persons in social networks, because famous persons usually have numerous followers. Based on Equation 2, their PageRanks shall be higher than other persons. However, in Github, we cannot fully determine experts on PageRanks, for two exceptions:

(1) Famous persons may not be experts. Figure 3 shows an American official account used to manage the law of America. Although it is not related to any programming work, there are numerous persons following this account, and thus its PageRank in following-followed graph is very high. This is not a programming expert by our definition.

(2) Friends of experts. In Figure 4, a developer *shaharelisha* may be a closed friend of another developer *mdo* in the real world, so *mdo* follows *shaharelisha*. Although *shaharelisha* has a high PageRank (because *mdo* is an expert and he follows *shaharelisha*), from the profile of *shaharelisha* in Github, she is not a programming expert by our definition.

Therefore, in the two situations, the PageRank algorithm alone cannot fully identify experts. To identify whether a person is an expert or a project is high-quality, the other two graphs, the watching graph W and the collaboration graph C, shall be considered, which is the reason why we need the Multi-Sources PageRank algorithm.

Watching Graph. In Github, developers can watch their interested projects. If we construct watching graphs with direct edges from developers to projects (Figure 1(b) shows such an example), all developers do not have indegrees and all projects do not have outdegrees in this graph. Therefore, we revise Equation 1 to Equation 3, where $L_W(u)$ is the number of projects watched by developer u.

$$PR(r) = \sum_{(u,r)\in E_W} \frac{PR(u)}{L_W(u)} \tag{3}$$

Collaboration Graph. Collaboration graphs (Figure 1(c) shows such an example) have two kinds of edges such as developer-to-developer edges and developer-to-project edges. If two developers collaborated to develop a project, there is a developer-to-developer edge between them. If a developer made contributions to a project, there is a developer-to-project edge between them.

Considering watching graphs, many projects have already been assigned PageRanks. However, some projects which are not watched by any developers do not have the values. To ensure each project has its PageRank, we revise Equation 1 to Equation 4. In Equation 4, $r(U)$ denotes all developers who participate in project r, and $L_C(u)$ dentes the number of projects that u participates in.

$$PR(r) \leftarrow PR(r) + \sum_{u\in r(U)} \frac{PR_u}{L_C(u)} \tag{4}$$

At the same time, the PageRank of each developer shall be updated to avoid the pre-mentioned problem, which is that a developer may not be an expert despite of a high PageRank. Based on our observations, if a developer collaborated with experts many times, the developer may also be an expert, and if a developer made contributions to many high-quality projects, the developer is likely to be an expert. Both situations can be detected by analyzing the collaboration graph. If a developer has a high PageRank in the following-followed graph, but seldom made contributions to projects or collaborated with others, he can be lack of connections in the collaboration graph. As a result, the PageRank of the developer in the collaboration graph can be low.

$$PR(u_1) \leftarrow PR(u_1) + \sum_{(u_1,u_2)\in E_C} PR(u_2) * \frac{|u_1(R) \cap u_2(R)|}{L_C(u_2)} \tag{5}$$

$$PR(u) \leftarrow PR(u) + \sum_{r\in u(R)} \frac{PR(r)}{L_C(r)} \tag{6}$$

We define Equations 5 and 6 to address the above two situations. As shown in Equations 5, if a developer u_1 collaborated with another developer u_2, the PageRank of u_2 will transfer to u_1. In Equation 5, $L_C(u_2)$ is the number of collaborators of u_2 and $u(R)$ are all the projects that the developer u involved. As shown in Equation 6, if a developer u has made contributions to a project r, the PageRank of r will transfer to u. $L_C(r)$ is the number of contributors of project r. After updating the PageRank of each developer, we use Equation 7 to normalize all PageRanks of developers and projects respectively.

$$PR(x) = \frac{PR(x)}{\sum_{i\in X} PR(i)} \tag{7}$$

Algorithm 1 Multi-Source PageRank

```
Input:
    The following-follower graph $F$;
    The watching graph $W$;
    The collaboration graph $C$;
Output:
    PageRanks of all developers $PR_u$;
    PageRanks of all projects $PR_r$
 1: $n_u = numberOfDevelopers(C)$
 2: $n_r = numberOfProjects(C)$
 3: $PR_u = PageRank(F)$
 4: for $i = 1$ to $n_r$ do
 5:     $PR_{r_i} = \sum_{(u,r_i)\in W} \frac{PR_u}{L_W(u)} + \sum_{(u,r_i)\in C} \frac{PR_u}{L_C(u)}$
 6: end for
 7: for $i = 1$ to $n_u$ do
 8:     $PR_{u_i} += \sum_{(u_i,u_j)\in C} PR_{u_j} * \frac{|u_i(R)\cap u_j(R)|}{L_C(u_j)}$
 9:     $PR_{u_i} += \sum_{r_k\in u(R)} \frac{PR_{r_k}}{L_C(r_k)}$
10: end for
11: $PR_u \leftarrow normalize(PR_u)$
12: $PR_r \leftarrow normalize(PR_r)$
13: return $PR_u, PR_r$;
```

Algorithm 2 Obtaining Expertise from Programming Behaviors

```
Input:
    Developer $u$
    Project $r$
    The time factor function $g(t)$; Programming language $p$.
Output:
    Contributions made by $u$ to project $r$ with $p$.
 1: $S(u,p,r) = 0.0$
 2: for $b$ in $r$ do
 3:     if $I(b,p)$ then
 4:         $S(u,b) = 0.0$
 5:         for $i = 0$ to $numberOfVersion(b) - 1$ do
 6:             $S(u,b) += \frac{g(t)*Contribution(u,b_i,b_{i+1})}{Lines(b)}$
 7:         end for
 8:         $S(u,p,r) += S(u,b)$
 9:     end if
10: end for
11: return $S(u,p,r)$;
```

Algorithm 1 illustrates the Multi-Sources PageRank algorithm proposed in this paper. Lines 1 and 2 count the number of developers and projects in our data set respectively. Line 3 runs the PageRank algorithm that is implemented in NetworkX[5] to calculate the original rank for each developer in the following-followed graph. Then, Lines 4 to 6 calculate the PageRank of each project based on the watching graph, the collaboration graph, and original ranks from the following-followed graph. Lines 7 to 10 update the PageRank of each developer to avoid fake experts by combining developer collaboration (line 8) and the developer involved projects line (9). Finally, it normalizes PageRanks of all developers (line 11) and all projects (line 12), respectively.

4.2 Programming Behaviors

[5]https://networkx.github.io/documentation/latest/index.html

Different from traditional social networks, as a version control system, developers on Github make contributions to projects (*e.g.*, committing code). As these programming behaviors reflect whether a developer is an expert or not, in this section, we leverage programming behaviors to further determine experts.

We use the lines of code that a developer created or modified in a project to determine the contribution of the developer. First, we introduce some notations. As Github is a version control system, a code block b may go through several modifications; namely, $b = (b_0, b_1, ...b_k)$, where $b_0 = \varnothing$. We use $\Delta(b_i, b_j)$ to denote the lines of code that are modified from version b_i to b_j, and we define $Lines(b)$ as $\sum_{m=0}^{k-1} \Delta(b_m, b_{m+1})$. For a developer u, we use $Contribution\ (u, b_i, b_j)$ to denote lines of code that the developer u contributes from version b_i to b_j. The total contributions that the developer u made to block b are defined as Equation 8. In Equation 8, $g(t)$ is a time factor, which is the time difference between that commit and now. If a developer did not make contributions to the project for a long time, the expertise of the developer to this project will decrease. We use Poisson distribution (Equation 9) to measure this factor.

$$S(u, b) = \frac{\sum_{i=0}^{k-1} g(t) * Contribution(u, b_i, b_{i+1})}{Lines(b)} \tag{8}$$

$$g(t) = \frac{\lambda^t}{t!} e^{-\lambda} \tag{9}$$

On the other hand, when computing contributions of a code block, our approach identifies the programming language of each block, based on our predefined 45 mapping relations from the the extension of file to the programming language (*e.g.*, *.py* is mapped to *python*). Furthermore, we define Equation 10 to calculate the skill of a developer u on a specific language p in a project r, where $I(b, p)$ is an indicator function, returning 1 if the block is written in language p.

$$S_p(u, p, r) = \sum_{b \in r} I(b, p) * S(u, b) \tag{10}$$

Algorithm 2 illustrates how our approach computes the contribution of a developer to a project, as far as a specific programming language is concerned. As a project may be implemented in multiple languages, Line 3 checks whether a block is written in a programming language p. If it is, Lines 5 to 7 calculate the contribution accumulatively.

4.3 Expertise of Programming Languages

After obtaining the expertise from the two aspects such as PageRank and programming behavior, we use Equation 11 to obtain the final ranks. In Equation 11, $PR(u)$ and $PR(r)$ are the PageRanks of u and r, respectively. The two ranks are obtained by the Multi-Sources PageRank algorithm, and $u(R)$ denotes all the projects that the developer u involved. This Equation balances the social behaviors and programming behaviors. If a developer is active in the social network, but commits only several lines of code, the total score of this developer will not be quite high. Besides, the more projects the developer participated, the higher the score is, according to the accumulation. On the other hand, if a developer commits frequently, but is not followed by other developers, the score of this developer will not be high either. We consider these developers made many low quality commits and thus others are not interested in them or their projects.

$$S(u, p) = PR(u) * \sum_{r \in u(R)} S_p(u, p, r) * PR(r) \tag{11}$$

5. EXPERIMENTS

In this section, we conduct experiments to validate the performance of GEMiner. Firstly, we present our data set and then analyze the results of experiments.

5.1 Data Set and Evaluation Metrics

To evaluate the performance of GEMiner, we use the data dump in [28]. In our data set, there are totally 142,535 developers, corresponding to 1,075,194 projects. Our task is to determine the expertise of developers and the quantity of projects, and we consider it as an information retrieval task. Therefore, we use the evaluation metrics in information retrieval to evaluate the performance of GEMiner. As GEMiner can identify experts for 40 programming languages, we rank the developers and projects respectively by their scores in all domains. After that, we ask four students whose major is computer science to evaluate the top-20 experts and projects and give scores to our results (like or dislike). Specifically, they will investigate profiles of the corresponding persons in Github, from the following-followed information, the quality of repositories they have contributed, and the activity of public contributions to determine whether these persons are experts of not. For example, in Figure 4, *mdo* is an expert while *shaharelisha* is not. As for projects, they will investigate the number of commits, stars, etc. and then give a judgement whether they are high-quality. Based on their scores, we use Equation 12 to define the precision of our approach, where A denotes the number of likes, and B is the number of total results (the number is 20 in our experiments).

$$Precision = \frac{|A|}{|B|} \tag{12}$$

5.2 Experiments

In total, we conduct three experiments: (1) Network-based Experts Identification without Programming Languages; (2) Experts Identification with Programming Languages; (3) Domain Experts Identification Compared with Authoritative Sets; and (4) High-Quality Repositories Identification.

5.2.1 *Network-based Experts Identification without Programming Languages*

In this experiment, we compare the results of our Multi-Sources PageRank algorithm, the original PageRank based on the random walk as described by Equation 2, and the HITS algorithm [10] for their effectiveness. Our Multi-Sources PageRank algorithm analyzes the following-followed graph, the watching graph and the collaboration graph, while the PageRank algorithm and the HITS algorithm analyze only a single graph. We use the two compared algorithms to analyze only the following-followed graph. The reason why we do not apply them to the watching

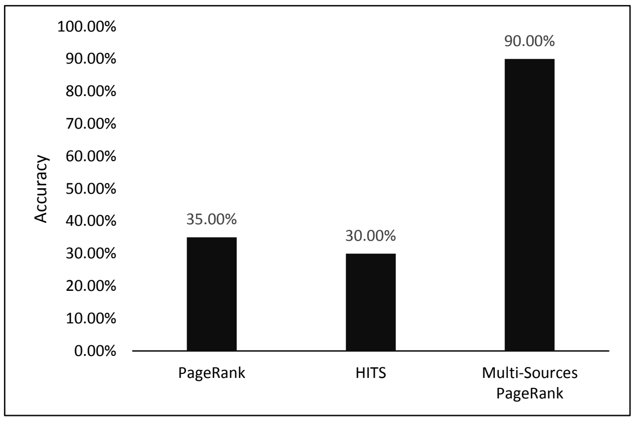


Figure 5: The Results of Network-based Expert Identification without Domains.

graph and the collaboration graph is that in these two graphs, there are two kinds of nodes, developers and projects, but the nodes in following-followed graph only consist of developers, which better fits the settings of the two algorithms. To conduct this experiment, we select the top-20 developers from the three algorithms, and ask three students to manually give a like or dislike score to each result.

Figure 5 shows the results. In Figure 5, the horizontal axis donates the three algorithms, and the vertical axis denotes their precisions. From the results, the performance of our Multi-Sources PageRank is the best, at 90.00%, followed by the PageRank algorithm, and then the HITS algorithm. We investigate the two situations mentioned in Section 4.1, and we find that both the PageRank and HITS algorithms do not deal with the two situations effectively. Some identified persons are popular developers or friends of real experts, but not real experts. In total, our Multi-Sources PageRank algorithm correctly identifies 18 experts, while the PageRank and HITS algorithms identify 7 and 6 experts, respectively. Interestingly, we find that 5 experts appear are identified by all the three algorithms, which indicates that our Multi-Source PageRank algorithm may filter fake experts and preserve real ones. We investigate those experts that are identified by the other algorithms and are not identified by our Multi-Sources PageRank algorithm. We find that these developers are all not real experts, since they do not join any high-quality projects. In the contrast, real experts have high PageRanks, and they join many high-quality projects. In summary, our results show that our Multi-Sources PageRank detects more experts while filtering fake experts.

5.2.2 *Experts Identification with Programming Languages*

In this experiment, we compare the results of the three algorithms for their effectiveness in identifying experts for some specific programming languages. When our Multi-Sources PageRank algorithm analyzes developer behaviors, it obtains the expertise of each developer in each programming language. The final expertise of each developer in each language is calculated by Equation 11. To compare with PageRank and HITS, we multiply the scores that are obtained from programming behaviors and those scores that are obtained from PageRank and HITS. We evaluate top-20 developers of all the 40 programming languages.

Figure 6 illustrates the results of this experiment. In Figure 6(a), we select the three programming languages such as JavaScript, CSS and HTML; in Figure 6(b), we select the three programming languages such as C++, Python and C; Figure 6(c) illustrates the average results in all the 40 programming languages. According to these figures, our algorithm achieves the highest accuracy in domain experts identification, at 91.59% in average, around 60% higher than that of PageRank and HITS.

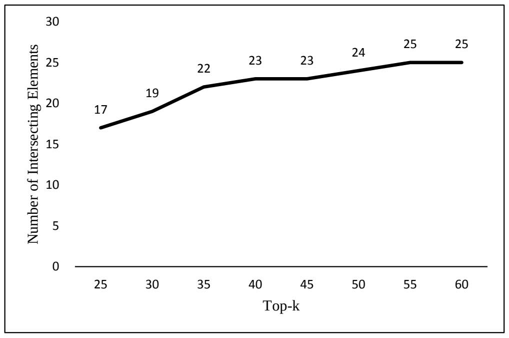


Figure 7: Experts in the Domain JavaScript.

5.2.3 *Domain Experts Identification Compared with Authoritative Sets*

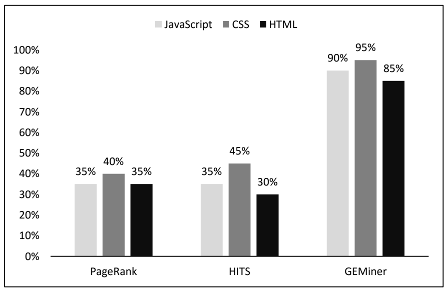
In the mainland China, CSDN[6] is the largest community for software developers. On CSDN, a recent article[7] presents a list of the most influential JavaScript developers on Github. In this experiment, we use the list to evaluate the effectiveness of our method. To conduct this experiment, we present the top-k (k>=25) experts obtaining by GEMiner, and then count the number of intersecting elements. Figure 7 illustrates the results, in which the horizontal axis donates the number of top results (k), and the values in vertical axis is the the number of intersecting elements between the authoritative set and the top-k set. According to Figure 7, when $k = 25$, the GEMiner totally seek 17 experts provided in the authoritative list. When $k = 55$, all 25 experts provided in the authoritative set are covered.

On CSDN, another recent article[8] presents a list of the most influential Web-Front developers on Github. We consider that Web-Front developers shall have expertise in domains such as HTML, CSS and JavaScript. To validate the results in this comprehensive domain, we sum up the scores that are obtained from domain HTML, CSS and JavaScript and then rank. Figure 8 shows the results. According to Figure 8, when $k = 10$, our algorithm seeks 6 experts in the authoritative list. When k is set to 30, all the 10 experts provided in the authoritative list are detected.

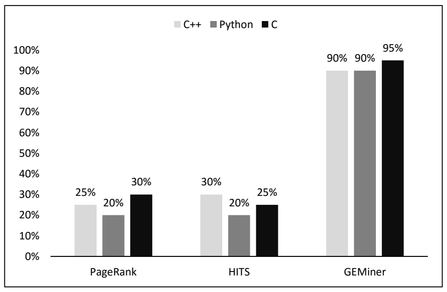
5.2.4 *High-Quality Repositories Identification*

In this experiment, we validate the performance of repositories identification of GEMiner. As GEMiner is based on the assumption that high-quality projects are likely developed by experts, and experts are often willing to contribute

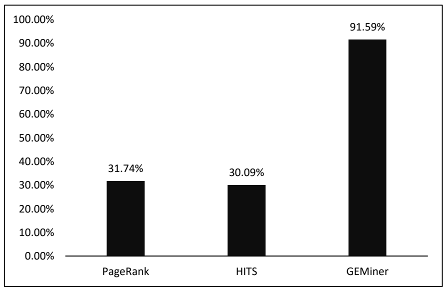
[6]http://code.csdn.net/
[7]http://www.csdn.net/article/1970-01-01/2808306
[8]http://code.csdn.net/news/2820990



(a) Domains: JavaScript, CSS, Html



(b) Domains: C++, Python, C



(c) Average in All Domains

Figure 6: The Results of Experts Identification with Domains.

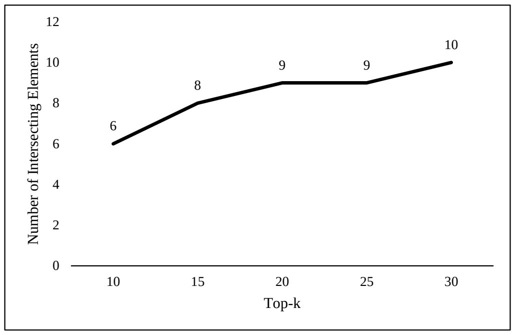


Figure 8: Experts in the Domain Web-Front.

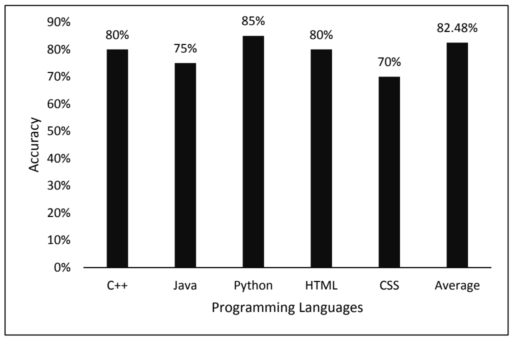


Figure 9: High-Quality Repositories Identification.

to high-quality projects, identifying high-quality repositories is an subprocess of experts identification. To evaluate the quality of a project in a specific programming language, we multiple the ratio of lines in that language in that project and its PageRank value. The results of high-quality projects identification is illustrated in Figure 9, where the horizontal axis donates the programming languages (We list six languages here), and the vertical axis denotes their top-20 accuracy. According to Figure 9, identifying Python projects achieves the highest accuracy, at 85.00%. And the average accuracy of all 40 languages is 82.48%, around 10% lower than that of experts identification. We investigate the results and find that most wrong results are from the projects created by famous persons, because when we calculate the PageRank of repositories, we accumulate the PageRank of all their contributors. However, this does not affect the results of experts identification.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose an experts identification approach, called GEMiner. To evaluate the expertise of developers, GEMiner considers both the social influence and programming behaviors of developers. To model the social influence, we propose a novel Multi-Sources PageRank algorithm, which utilizes the information of developers in three types of graphs such as the following-followed graph, the watching graph, and then collaboration graph. To model programming behaviors, GEMiner considers the historical commit information of each developer. During the process of expert identification, each developer and repository are labeled by some programming languages. Finally, based on these programming languages and expertise in each language, experts for specific programming languages can be identified.

To identify experts and high-quality projects more accurately, there are still two aspects for further improvements: (1) More Domains. Although GEMiner identifies 40 domains (programming languages), the number of identified domains is limited. Another way is to identify domains from the descriptions and readme files of projects, which consist of natural language with more meaningful information. More specific domains can be identified with more advanced techniques such as machine learning, data mining and etc. (2) More Programming Behaviors. GEMiner considers only the commit behaviors of each programmer. However, other factors (*e.g.*, the date when an account registered) can also make contribution to identify experts, and will explored in our future work.

7. ACKNOWLEDGMENTS

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

8. REFERENCES

[1] Shaowei Wang, Daniel Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec: an enhanced tag recommendation system for software information sites. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 291–300. IEEE, 2014.

[2] Oskar Jarczyk, Błażej Gruszka, Szymon Jaroszewicz, Leszek Bukowski, and Adam Wierzbicki. Github projects. quality analysis of open-source software. In *Social Informatics*, pages 80–94. Springer, 2014.

[3] Jun Zhang, Mark S Ackerman, and Lada Adamic. Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th international conference on World Wide Web*, pages 221–230. ACM, 2007.

[4] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[5] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

[6] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[8] G Alan Wang, Jian Jiao, Alan S Abrahams, Weiguo Fan, and Zhongju Zhang. Expertrank: A topic-aware expert finding algorithm for online knowledge communities. *Decision Support Systems*, 54(3):1442–1451, 2013.

[9] Albert Hupa, Krzysztof Rzadca, Adam Wierzbicki, and Anwitaman Datta. *Interdisciplinary matchmaking: Choosing collaborators by skill, acquaintance and trust*. Springer, 2010.

[10] Jon M Kleinberg. Hubs, authorities, and communities. *ACM Computing Surveys (CSUR)*, 31(4es):5, 1999.

[11] K Kalaiselvi and PS Balamurugan. An ontological approach to identify expert knowledge in academic institution. In *Current Trends in Engineering and Technology (ICCTET), 2013 International Conference on*, pages 120–122. IEEE, 2013.

[12] Jie Li, Harold Boley, Virenda Bhavsar, and Jing Mei. Expert finding for ecollaboration using foaf with ruleml rules. 2006.

[13] Gustavo Freitas, Cesar da Costa, Jorge Barbosa, Rodrigo Righi, and Abid Yamin. Expert user discovery in a spontaneous social network an approach using knowledge retrieval. In *Computational Aspects of Social Networks (CASoN), 2013 Fifth International Conference on*, pages 15–20. IEEE, 2013.

[14] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816. ACM, 2009.

[15] Petter Holme and Mark EJ Newman. Nonequilibrium phase transition in the coevolution of networks and opinions. *Physical Review E*, 74(5):056108, 2006.

[16] David Crandall, Dan Cosley, Daniel Huttenlocher, Jon Kleinberg, and Siddharth Suri. Feedback effects between similarity and social influence in online communities. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 160–168. ACM, 2008.

[17] Jerry Scripps, Pang-Ning Tan, and Abdol-Hossein Esfahanian. Measuring the effects of preprocessing decisions and network forces in dynamic network analysis. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 747–756. ACM, 2009.

[18] Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[19] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7, 2012.

[20] Audris Mockus and James D Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th international conference on software engineering*, pages 503–512. ACM, 2002.

[21] John Anvik and Gail C Murphy. Determining implementation expertise from bug reports. In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*, pages 2–2. IEEE, 2007.

[22] Elben Shira and Matthew Lease. Expert search on code repositories. 2010.

[23] Benjamin V Hanrahan, Gregorio Convertino, and Les Nelson. Modeling problem difficulty and expertise in stackoverflow. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*, pages 91–94. ACM, 2012.

[24] Nidhi Raj, Lipika Dey, and Bhakti Gaonkar. Expertise prediction for social network platforms to encourage knowledge sharing. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 1, pages 380–383. IEEE, 2011.

[25] Jiwoon Jeon, W Bruce Croft, Joon Ho Lee, and Soyeon Park. A framework to predict the quality of answers with non-textual features. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 228–235. ACM, 2006.

[26] Fatemeh Riahi, Zainab Zolaktaf, Mahdi Shafiei, and Evangelos Milios. Finding expert users in community question answering. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 791–798. ACM, 2012.

[27] Matthew Richardson and Pedro Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *NIPS*, pages 1441–1448, 2001.

[28] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.