Advances in Complex Systems

RANKING DEVELOPER CANDIDATES BY SOCIAL LINKS

QI XUAN
Department of Automation, Zhejiang University of Technology
Hangzhou 310023, China
Department of Computer Science, University of California, Davis
Davis, CA 95616-8562, USA
crestxq@hotmail.com

CHENBO FU
Department of Automation, Zhejiang University of Technology
Hangzhou 310023, China
cbfu@zjut.edu.cn

LI YU
Department of Automation, Zhejiang University of Technology
Hangzhou 310023, China
lyu@zjut.edu.cn

Received (received date)
Revised (revised date)

In Open Source Software (OSS) projects, participants initially communicate with others and then may become developers if they are deemed worthy by the community. Recent studies indicate that the abundance of established social links of a participant is the strongest predictor to his/her promotion. Having reliable rankings of the candidates is key to recruiting and maintaining a successful operation of an OSS project. This paper adopts degree-based, PageRank, and Hits ranking algorithms to rank developer candidates in OSS projects based on their social links. We construct several types of social networks based on the communications between the participants in Apache OSS projects, then train and test the ranking algorithms in these networks. We find that, for all the ranking algorithms under study, the rankings of emergent developers in temporal networks are higher than those in cumulative ones, indicating that the more recent communications of a developer in a project are more important to predict his/her first commit in the project. By comparison, the simple degree-based and the PageRank ranking algorithms in temporal undirected weighted networks behave better than the others in identifying emergent developers based on four performance indicators, and are thus recommended to be applied in the future.

Keywords: Ranking algorithm; social network; temporal network; Open Source Software; email communication.

1. Introduction

In recent decades, we witness the emergence of Open Source Software (OSS) projects [1] whose success partially relies on the contributions of numerous volunteer developers with different motivations [2–4], such as enjoyment, skill improvement, reputation building so as to attract potential employers, etc. Generally, users contribute to projects by sharing experiences [5, 6], reporting bugs [7, 8], submitting patches [9, 10], and becoming developers, with the ability to modify and commit source code themselves. OSS projects rely on the community to evaluate these activities and further encourage the outstanding ones to accept more responsible roles [11], i.e., from users to developers, from developers to managers, and so on. The openness of these projects means that the traces of social communications and technical actions are available publicly [12], and can be used to find, recruit, and retain highly responsible developers with good programming skills, which is strongly correlated with the health and success of OSS projects [13].

In works on software engineering performance and productivity, researchers have tracked social network and commit activities of developers to identify the most relevant ones for the outcome of interest, e.g., quality of code, productivity, etc. Zhang *et al.* [14] proposed a series of network measurements to identify the core developers of each release in ArgoUML. They found that the results are better on bipartite networks, including the information of both developer cooperation and email topics, than on the traditional unipartite network, only considering developer information. Wu *et al.* [15] introduced a work motivation model for OSS developers by analyzing the data of 148 OSS participants, and found that the developers' feelings of satisfaction and their intentions to continue with OSS development was influenced by both helping behavior and economic incentives. Capiluppi *et al.* [16] proposed a measure similar to the h-index used in academic contexts, which captures both the number of projects a developer contributed to and the number of commits he/she made in each of them, in order to objectively compare the contributions of OSS developers. Saul *et al.* [17] designed a recommendation system by a random-walk approach in order to recommend related interesting functions for developers based on certain functions they found. Besides, Wu *et al.* [18] proposed a developer recommendation algorithm based on k-nearest-neighbor search and expertise ranking with various frequency and network metrics to recommend capable developers for certain bugs. In our previous work [12], we used time-series methods to identify synchronous pairwise collaborations between developers, and defined code growth and effort based on the added and deleted lines of code. We found that, during the synchronous collaboration, the project size grows faster even though the overall coding effort slows down.

However, similar algorithms can rarely be found in the literature to rank normal users, named developer candidates in the rest of the paper, by their pre-commit activities, such as communications, bug reports, and patch submissions, although there are migration models for the relationships between these quantities and the time

needed to become a developer [13]. Communication is important in OSS projects. Herbsleb and Grinter [19] found that a lack of communication between software developers introduces coordination problems. We found that commits and communication may accelerate, rather than slow down, each other in OSS projects by comparing the real time-series of these activities with simulated ones [1]. Note that, in many OSS projects, bug reports and patch submissions are always associated to emails [9], thus are also partially reflected in communication activities. Recently, Gharehyazie *et al.* [10] stated: "*strong working code leads to trust in the participant's ability to develop within the context of the project; strong social skills signify to the team that the participant can be trusted to work within the project's team setting; and generally, the more trustworthy a participant, the more likely it is that he may eventually achieve developer status*". By comparison, they found that the abundance of established social links, rather than bug reports or patch submissions, of a participant is the strongest predictor for his/her promotion to a developer. Here, we argue that such statistical models can provide insights for the evolution of OSS projects macroscopically, but have limited abilities to tell which candidates are more important than the others in real time. Ranking algorithms [20, 21], on the other hand, can help managers to identify highly responsible and skillful candidates in real time and then promote their roles. Furthermore, these algorithms can also help the candidates themselves to recognize their shortages by comparing with others and encourage them to seek further improvements. Based on social networks, such algorithms have been successfully used in many areas, e.g., to measure the influence of bloggers [22], reveal the roles of employees in enterprises [23], find important authors [24] in academia and so on.

The findings of Gharehyazie *et al.* [10] inspired us to rank developer candidates just by their social links, to investigate how good the rankings of emergent developers could be, which, to the best of our knowledge, has not been studied yet. We think that developer candidates tend to interact frequently with others by exchanging messages in the same OSS project before they make direct code contribution to it. Note that we can certainly integrate other non-social information into the framework to further improve the rankings of the emergent developers, which however is not the focus of this paper. In the present work, we collected 31 OSS projects from the *Apache Software Foundation* [1], and use those four with the largest number of developers to establish social networks, based on which we train and test ranking algorithms. Different from other works on ranking individuals in cumulative networks [25–28], here we also design ranking algorithms in temporal networks [29–32]. The reason is apparent: OSS projects are always highly dynamic [12], i.e., developers join and quit very frequently. As a result, there may be many users that have been inactive for a long time, but their historical communication records are still preserved in the database. These inactive users may compete rankings with the current active users in cumulative networks, however, most of such trivial competition can be filtered out in temporal networks. Our main contributions thus include:

- We adopt degree-based, PageRank, and Hits algorithms to rank developer candidates in OSS projects for the first time based on the communications between them. We find that the ranking results of emergent developers obtained by these algorithms are surprisingly good compared to the rankings obtained by the random algorithm [a]. This indicates that the candidates communicating more are indeed more likely to make direct code contributions to the projects in the future.
- We establish temporal social networks by introducing a Gaussian time-window function, and find that all the ranking algorithms under study behave better in temporal networks than in cumulative ones. This validates the intuition that more recent communications of a developer in a project are more important to predict his/her first commit in this project.
- Besides the well-known *receiver operating characteristic* (ROC) curve [33] and the corresponding *area under ROC curve* (AUC) [34], we additionally utilize the percentage of the best rankings, and the risk of missing developers, to compare the performances of different ranking algorithms on different types of networks.

The rest of the paper is organized as follows. In Section 2, the data sets, as well as some definitions of different types of networks, are briefly introduced; three algorithms, including degree-based, PageRank, and Hits algorithms, are then applied in different types of networks. In Section 3, the performances of different ranking algorithms on various types of networks are compared to each other. The paper is concluded in Section 4.

2. Methodology

Based on a data set of 31 Apache OSS projects collected in March 2012, in this paper we study the following four projects: *Ant*, *Axis2_java*, *Derby*, and *Lucene*. We chose those four projects because they exhibit the largest number of developers for whom their first email communication activities occurred before their first commit activities. The basic properties of these four projects are briefly listed in Table 1, where T_0 is the initial time of a project when first email message or commit activity was recorded, and N_U, N_D, N_M, and N_F represent the number of *users* (including developers), the number of *developers*, the number of email *messages* between all the users, and the number of *files* committed by the developers, respectively. For each project, the email messages are obtained from the online developer mailing list. Note that email messages may be posted to a mailing list automatically in an OSS community to inform others when some work is done. To exclude such trivial messages, we just consider the response emails [1]. Moreover, we also use a semi-automatic approach [b] to solve the problem of multiple aliases [35].

[a] For the random algorithm, we mean ranking the candidates randomly.

[b] At first, we automatically crawl messages, and extract all the headers to produce a list of {name, email} identifiers. Once this is done, we use a clustering algorithm to measure the similarity

Table 1. Several basic properties of the four OSS projects.

Project	T_0	N_U	N_D	N_M	N_F
Ant	2000/01/13	1402	44	17284	11620
Axis2_java	2001/01/30	3738	72	30740	129978
Derby	2004/08/10	1118	35	49353	6563
Lunece	2001/09/11	2102	41	42968	6674

2.1. *Data sets*

The paper mainly focuses on ranking developer candidates based on their social status, i.e., for each project, we emphasize the differences between emergent developers and the other candidates with respect to their local properties in the social network. Therefore, the developers with the first commit activity occurring earlier than the first communication activity are considered as *abnormal points* and thus are excluded in this study. There are in total 6, 8, 1, and 4 such abnormal points in the projects *Ant*, *Axis2_java*, *Derby*, and *Lucene*, respectively. The emergence of abnormal points may be due to fact that the candidates had already known developers or managers in the project before they joined, e.g., they had contributed to other projects or had worked for the same company.

At each time when a candidate turned to be a developer, we construct different types of social networks by the email messages before that time, and call the candidate *emergent developer* at that time. Then, in such a social network, there are three groups of nodes, including an emergent developer, existent developers, and a large number of other candidates. For example, we have four cumulative social networks in *Derby*, when the first four developers emerged one after another, as shown in Fig. 1, where the three groups of nodes are denoted by filled circles, filled triangles, and empty circles, respectively. On each network, we rank the emergent developer and the other candidates, to see whether the emergent developer indeed exhibited outstanding social status at that time.

We find that the time intervals between successive emergences of developers are not uniformly distributed, i.e., there are a few number of extremely large time gaps, which is partially determined by the non-uniform time distributions of new versions released or old developers retiring. Meanwhile, the growth rates of the total messages in *Ant*, *Axis2_java*, and *Derby* decrease as the networks evolve, although the numbers of nodes increase all the time. This phenomenon suggests that, in these projects, there are always large numbers of users becoming less and less active and finally completely withdrawing from the projects.

between every pair of identifiers. This could occur if either the names, or the emails, or both are similar. Those sufficiently similar identifiers are grouped into the same cluster. Once clusters are formed, then, they are manually post-processed.

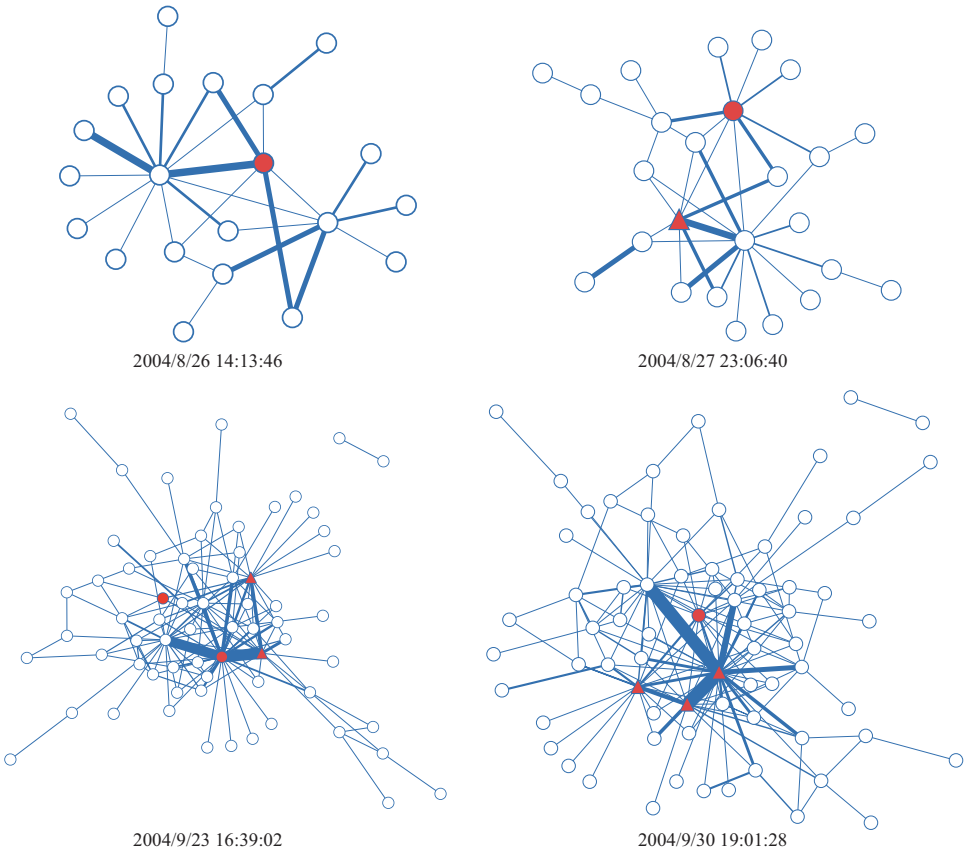

Fig. 1. The topologies of the cumulative social networks at the time when the first four developers emerged one after another in *Derby*. Here, filled circles, filled triangles, and empty circles represent the emergent developers, the existent developers, and the other candidates at that time, the width of the lines connecting two nodes is proportional to the numbers of messages between the corresponding individuals.

2.2. *Types of networks*

Suppose we need to rank candidates at time T in order to select one of them as the emergent developer, let n be the total number of users, including developers and candidates, in a project before T.

Denote by $\overrightarrow{G}_C(V, \overrightarrow{E}, \overrightarrow{W}_C, T)$ the cumulative directed weighted network at time T, where $V = \{v_1, v_2, \ldots, v_n\}$ is the node set, $\overrightarrow{E} \subseteq V \times V$ and $\overrightarrow{W}_C$ are the directed link set and the corresponding weight set, respectively, i.e., $(v_i, v_j) \in \overrightarrow{E}$ if and only if v_i sent at least one message to v_j, and $\overrightarrow{w}_{ij} \in \overrightarrow{W}_C$ is the number of messages that v_i sent to v_j, before time T. Denote by $G_C(V, E, W_C, T)$ the corresponding cumulative undirected weighted network, where $E \subseteq V \times V$ and W_C are the link

and weight sets, respectively, i.e., $(v_i, v_j) \in E$ if and only if v_i sent/received at least one message to/from v_j, and $w_{ij} \in W_C$ is the number of messages that v_i sent to and received from v_j, before time T, i.e.,

$$w_{ij} = w_{ji} = \overrightarrow{w}_{ij} + \overrightarrow{w}_{ji}. \tag{1}$$

Denote by $\overrightarrow{G}_\sigma(V, \overrightarrow{E}, \overrightarrow{W}_\sigma, T)$ the temporal directed weighted network at time T, where V, $\overrightarrow{E}$, and $\overrightarrow{W}_\sigma$ are the node set, the directed link set, and the corresponding weight set, respectively. Here, the definitions of V and $\overrightarrow{E}$ are the same for the cumulative directed weighted network and σ is the time window, or observation interval, i.e., only the links occurring in the intervals $[T - \sigma, T + \sigma]$ are considered. Generally, in a temporal directed weighted network, we consider the email messages inside, while we ignore or quickly attenuate the effect of messages outside the time window. Here, we use a Gaussian function [12] to decrease the effects of email messages outside the time window. For each directed link, i.e., $(v_i, v_j) \in \overrightarrow{E}$, we denote by $t_{ij}^1 < t_{ij}^2 < \ldots < t_{ij}^{\overrightarrow{w}_{ij}} < T$ the time at which v_i sent one of the $\overrightarrow{w}_{ij}$ messages to v_j before time T. Then, the weight of this directed link is defined as

$$\overrightarrow{u}_{ij} = \sum_{k=1}^{\overrightarrow{w}_{ij}} e^{\frac{-(t_{ij}^k - T)^2}{2\sigma^2}}. \tag{2}$$

In particular, we have $\overrightarrow{u}_{ij} = \overrightarrow{w}_{ij}$ when $\sigma \to \infty$, indicating that the temporal directed weighted network degenerates to the cumulative one in this case. Note that we use Gaussian function because it is simple and mathematically beautiful [12]; and meanwhile the resulting temporal network will be always connected and thus make the rankings more robust than those based on hard time windows [c]. The time window σ can be tuned, and we will use the training set of samples to estimate the optimal σ for each ranking algorithm.

Similarly, denote by $G_\sigma(V, E, W_\sigma, T)$ the corresponding temporal undirected weighted network, where the weight between two linked nodes v_i and v_j is calculated by

$$u_{ij} = u_{ji} = \overrightarrow{u}_{ij} + \overrightarrow{u}_{ji}. \tag{3}$$

All the unweighted networks can be obtained from the corresponding weighted ones by just ignoring the difference between the weights of links and considering all of them equal to 1. Then, for each node v_i in a directed network, denote by I_i and O_i its *incoming* and *outgoing* neighborhoods, respectively, and by N_i its *neighborhood* in the corresponding undirected network, satisfying $N_i = I_i \cup O_i$.

[c]By adopting a hard time window, we just count the number of interactions in a certain interval. In this case, we may get an unconnected temporal network (there is even no link in the network when the time window is extremely small).

2.3. *Ranking algorithms*

For a project, denote by T the time when a developer emerged and by n the total number of users in the project before that time. For the network constructed at time T, we sort the n users by a ranking algorithm. We then remove the existent developers and get a new list of rankings of the candidates including the emergent developer. Suppose there are n_d existent developers before time T in the project and the emergent developer has the same ranking like h other candidates, and these candidates including the emergent developer have the positions $\lambda, \lambda+1, \ldots, \lambda+h$ in the new sorted list. We define the ranking of the emergent developer as the mean value of these positions, i.e., $(\lambda+\lambda+h)/2$, which is further normalized to

$$R = \frac{1}{n-n_d}\left(\lambda + \frac{h}{2}\right), \tag{4}$$

satisfying $0 < R \leq 1$. Thus, for an emergent developer, a *higher* ranking is equivalent to a *smaller* value of R.

2.3.1. *Ranking by single network properties*

The social status of an individual can be characterized by its local structural properties [36] in the social network. In this paper, we select six of these properties, including the degree $k_i = |N_i|$, the incoming degree $k_i^{in} = |I_i|$, and the outgoing degree $k_i^{out} = |O_i|$ for node v_i in unweighted networks, and those in weighted networks, defined as $\varphi_i = \sum_{j\in N_i} w_{ij}$, $\varphi_i^{in} = \sum_{j\in I_i} \overrightarrow{w}_{ji}$, and $\varphi_i^{out} = \sum_{j\in O_i} \overrightarrow{w}_{ij}$, respectively, satisfying $k_i \geq \max(k_i^{in}, k_i^{out})$ and $\varphi_i = \varphi_i^{in} + \varphi_i^{out}$. In OSS projects, a high incoming degree of a user indicates that many other users are interested in his/her shared information [13], while a high outgoing degree indicates that he/she is relatively active and interested in the information shared by many others.

We first simply rank the candidates by these structural properties and find that the rankings of emergent developers based on degree in cumulative undirected weighted and unweighted networks are close to each other, slightly higher than those based on incoming and outgoing degrees in the corresponding cumulative directed networks. As a result, in the rest of the paper, we will only adopt degrees in undirected networks to rank the candidates, and the results will be compared with the rankings obtained by other algorithms. Note that, in temporal weighted networks, the degree of node v_i is calculated as $\varphi_i = \sum_{j\in N_i} u_{ij}$.

2.3.2. *PageRank algorithms in directed networks*

One of the most popular ranking algorithms is the PageRank algorithm which was originally designed to rank web pages. It led to the success of Google [20] and inspired many other search engines. The PageRank algorithm can be applied directly here. Denote by x_i the importance score of node v_i, i.e., x_i is nonnegative and $x_i > x_j$ indicates that node v_i is more important, and thus have higher ranking,

than node v_j. PageRank provided a scheme to calculate the importance score for each node in a network as follows. If node v_j has k_j^{out} outgoing links, one of which links to node v_i, its contribution to node v_i's score is x_j/k_j^{out}. Then, we have

$$x_i = \sum_{v_j \in I_i} \frac{x_j}{k_j^{out}}. \tag{5}$$

Denoting

$$a_{ij} = \begin{cases} \frac{1}{k_j^{out}}, & v_j \in I_i, \\ 0, & v_j \notin I_i, \end{cases} \tag{6}$$

Eq. (5) can be rewritten by the following more compact matrix form:

$$x = Ax, \tag{7}$$

with $x = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ and $A = [a_{ij}]_{n \times n}$.

In order to overcome the non-unique ranking problem introduced by the non-connectivity of the network, the elements a_{ij} in matrix A need to be replaced by

$$q_{ij} = \frac{(1-\beta)a_{ij} + \beta\delta}{\sum_{i=1}^{n}[(1-\beta)a_{ij} + \beta\delta]}, \tag{8}$$

where $\delta = 1/n$ and $\beta \in (0,1)$ is a small positive parameter. The value of β originally used by Google is reportedly 0.15 [37]. In this paper, it is set to 0.1 and similar results can be obtained by using other values. Note that this modification can also help to avoid the negative effects introduced by dangling nodes with no outgoing links. Then, Eq. (7) is replaced by

$$x = Qx, \tag{9}$$

where $Q = [q_{ij}]_{n \times n}$ is a column-stochastic matrix. Then, the ranking problem is transformed into the problem of finding an eigenvector with eigenvalue 1 for the square matrix Q.

When considering weighted networks, each directed link from node v_i to node v_j has a weight $\overrightarrow{w}_{ij}$ in a cumulative network and a weight $\overrightarrow{u}_{ij}$ in a temporal network. Taking the cumulative weighted network, we redefine the matrix A with its element now calculated by

$$a_{ij} = \begin{cases} \frac{\overrightarrow{w}_{ji}}{\sum_{v_k \in O_j} \overrightarrow{w}_{jk}}, v_j \in I_i, \\ 0, v_j \notin I_i. \end{cases} \tag{10}$$

2.3.3. *PageRank algorithms in undirected networks*

For the unweighted case, an undirected network needs to be created first from the original directed one, where two nodes are linked if there is at least one message between them. Then, the elements of matrix A are redefined as

$$a_{ij} = \begin{cases} \frac{1}{k_j}, v_j \in N_i, \\ 0, v_j \notin N_i, \end{cases} \tag{11}$$

where k_i is the degree, and N_i is the neighborhood of node v_i. For the weighted case, the elements of A are redefined as

$$a_{ij} = \begin{cases} \frac{w_{ij}}{\sum_{v_k \in N_j} w_{jk}}, v_j \in N_i, \\ 0, v_j \notin N_i, \end{cases} \tag{12}$$

where w_{ij} is the weight of the link between nodes v_i and v_j.

In reality, for those large-scale networks, Eq. (9) can be solved by the following iterative method

$$x(k+1) = \frac{Qx(k)}{\|Qx(k)\|}, \tag{13}$$

for any initial state $x(0)$, where $\|\cdot\|$ can be any vector norm, in order to decrease the complexity of the algorithm. In this paper, the 1-norm is adopted, and the iterative process is terminated when $\|x(k+1) - x(k)\| < 10^{-5}$.

2.3.4. *Hits algorithms in directed networks*

Another well-known ranking algorithm based on link information is the hits algorithm, first proposed by Kleinberg [21]. Based on this algorithm, in a directed network, each node v_i has two assigned values named as *authority* and *hub*, denoted by ϕ_i and ρ_i, respectively. Generally, a good authority represents a node that is linked to by many good hubs, and a good hub represents a node that points to many good authorities. As a result, there are two types of updates, authority and hub updates, which will be introduced for unweighted and weighted networks, respectively.

For unweighted networks, the authority ϕ_i and hub ρ_i updates are defined as

$$\phi_i(k+1) = \frac{\sum_{j \in I_i} \rho_j(k)}{\sum_{i \in V} \sum_{j \in I_i} \rho_j(k)}, \tag{14}$$

$$\rho_i(k+1) = \frac{\sum_{j \in O_i} \phi_j(k)}{\sum_{i \in V} \sum_{j \in O_i} \phi_j(k)}, \tag{15}$$

respectively. For weighted networks, they are changed to

$$\phi_i(k+1) = \frac{\sum_{j \in I_i} \overrightarrow{w}_{ji} \rho_j(k)}{\sum_{i \in V} \sum_{j \in I_i} \overrightarrow{w}_{ji} \rho_j(k)}, \tag{16}$$

$$\rho_i(k+1) = \frac{\sum_{j \in O_i} \overrightarrow{w}_{ij} \phi_j(k)}{\sum_{i \in V} \sum_{j \in O_i} \overrightarrow{w}_{ij} \phi_j(k)}, \tag{17}$$

respectively. For each case, the iterative process is initialized from $\phi(0) = 1$ and $\rho(0) = 1$, and terminated if both $\|\phi(k+1) - \phi(k)\| < 10^{-5}$ and $\|\rho(k+1) - \rho(k)\| < 10^{-5}$ are satisfied with $\phi(k) = [\phi_1(k), \phi_2(k), \ldots, \phi_n(k)]^{\mathrm{T}}$ and $\rho(k) = [\rho_1(k), \rho_2(k), \ldots, \rho_n(k)]^{\mathrm{T}}$, or the number of iterative times is larger than 10^4.

2.3.5. *Hits algorithms in undirected networks*

Hits algorithms can also be used in undirected networks just by simply considering each link bidirectional. In this case, the authority value and the hub value of each node are exactly the same. Still denote by x_i the importance score of a node, then, its updates in unweighted and weighted networks are defined as

$$x_i(k+1) = \frac{\sum_{j\in N_i} x_j(k)}{\sum_{i\in V}\sum_{j\in N_i} x_j(k)} \tag{18}$$

$$x_i(k+1) = \frac{\sum_{j\in N_i} w_{ij}x_j(k)}{\sum_{i\in V}\sum_{j\in N_i} w_{ij}x_j(k)}, \tag{19}$$

respectively. Similarly, the iterative process is initialized from $x(0) = 1$ and terminated if $\|x(k+1) - x(k)\| < 10^{-5}$ or the number of iterative times exceeds 10^4.

3. Results

In this part, the rankings of emergent developers obtained by different ranking algorithms including degree-based, PageRank, and Hits in different types of networks, i.e., temporal versus cumulative, directed versus undirected, and weighted versus unweighted, of the four OSS projects, i.e., *Ant*, *Axis2_java*, *Derby*, and *Lucene*, are presented and compared.

3.1. *Rankings in directed networks*

For the *directed* case, denote by $P(d,u)$, $H_A(d,u)$, and $H_H(d,u)$ the rankings of the emergent developers by using the *PageRank* algorithm, the *Hits-Authority* algorithm, and the *Hits-Hub* algorithm, respectively, in the cumulative *unweighted* networks. Denote by $P(d,w)$, $H_A(d,w)$, and $H_H(d,w)$ these rankings in the cumulative *weighted* networks, and by $P(d,o)$, $H_A(d,o)$, and $H_H(d,o)$ the corresponding *optimal* rankings in the temporal weighted networks, which may be obtained at different optimal time windows σ^o. For each project, we get the 20% earliest samples (i.e., the earliest emergent developers) [d]. They are used to determine the respective optimal window size σ^o, at which the emergent developers in the training set get their highest average ranking. Then, we use the remaining 80% samples to test all the algorithms. As a result, considering the four projects together, there are a total of 36 samples in the training set, and a total of 137 samples in the test set.

We adopt the *receiver operating characteristic* (ROC) analysis [33] to compare the rankings obtained by different algorithms. Typically, ROC captures the performance of a classifier with two classes, as the threshold for putting elements in one of the two classes is varied. For a ranking algorithm introduced here, we consider those emergent developers as positive examples, and the other candidates as

[d]We choose the earliest, but not random, samples, in order to address that these ranking algorithms have the ability to predict the *future* behaviors of programmers by using the historical data.

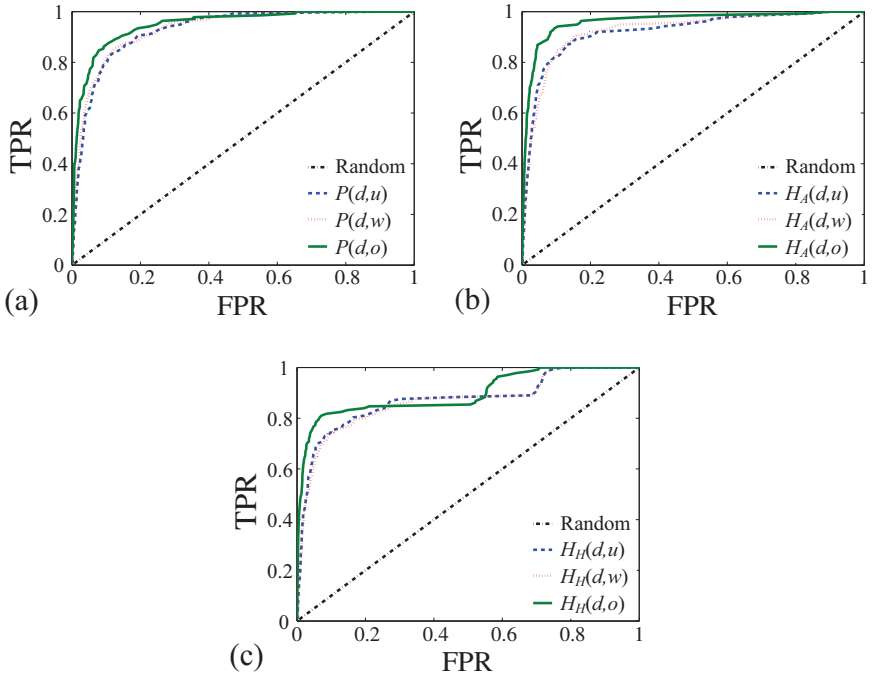

Fig. 2. ROC curves for (a) the PageRank algorithm, (b) the Hits-Authority algorithm, and (c) the Hits-Hub algorithm, in cumulative unweighted networks (dashed), cumulative weighted networks (dotted), and temporal weighted networks (solid), respectively, for the directed case.

Table 2. AUC for different algorithms in different types of directed networks. The optimal time window σ^o (day) for the algorithms in the temporal networks are also reported.

Algorithm	Unweighted	Weighted	Optimal	σ^o (day)
PageRank	0.9280	0.9311	0.9491	20
Hits-Authority	0.9219	0.9246	0.9616	77
Hits-Hub	0.8717	0.8665	0.8971	6

negative examples. We varied the threshold θ from 0 to 1, and classify a candidate as an emergent developer (predicted positive) if its normalized ranking $R \leq \theta$ and as a non-emergent developer (predicted negative) otherwise. Therefore, as θ increases, both *false positive rate* (FPR) and *true positive rate* (TPR) [e] increase, since more and more negative and positive examples are classified as emergent developers

[e] *False positive rate* (FPR) is defined as the ratio of negative examples that are predicted positive to all the negative examples; *True positive rate* (TPR) is defined as the ratio of positive examples that are predicted positive to all the positive examples.

Table 3. AUC for different algorithms in different types of undirected networks. The optimal time window σ^o (day) for the algorithms in the temporal networks are also reported.

Algorithm	Unweighted	Weighted	Optimal	σ^o (day)
Degree	0.9342	0.9433	0.9685	4
PageRank	0.9282	0.9356	0.9731	50
Hits	0.9313	0.9355	0.9620	2

(predicted positive). For example, by adopting the random algorithm [f], emergent developers have random rankings, as a result, the fractions of positive and negative examples that are predicted positive are exactly the same, i.e., FPR equals to TPR for different values of threshold θ.

The ROC curves for different ranking algorithms in directed networks are shown in Fig. 2, where x-axis is the FPR and y-axis is the TPR. An algorithm is considered better if it has larger TPR for a given FPR. Thus, as we can see, all these algorithms based on any kind of directed networks are much better than the random algorithm as the reference. Meanwhile, in this case, all the algorithms behave better in the temporal weighted networks than in the cumulative unweighted and weighted networks. In particular, for the temporal and cumulative weighted networks, the average rankings (R values) are 5.33% versus 7.22% for the PageRank, 4.17% versus 7.87% for the Hits-Authority, and 10.55% versus 13.63% for the Hits-Authority, respectively, i.e., the ranking algorithms always have higher rankings in temporal weighted networks. We also calculate the *area under ROC curve* (AUC) [34] for these algorithms in different types of directed networks, as presented in Table 2, where we find that, for all the algorithms, the optimal rankings in temporal weighted networks indeed have relatively larger AUC, and the Hits-Authority in temporal weighted networks seems to be the best one.

3.2. *Rankings in undirected networks*

For the *undirected* case, denote by $D(u,u)$, $P(u,u)$, and $H(u,u)$ the rankings of the emergent developers by using the *Degree* based algorithm, the *PageRank* algorithm, and the *Hits* algorithm, respectively, in the the cumulative *unweighted* networks. Denote by $D(u,w)$, $P(u,w)$, and $H(u,w)$ these rankings in the cumulative *weighted* networks, and by $D(u,o)$, $P(u,o)$, and $H(u,o)$ the corresponding *optimal* rankings in the temporal weighted networks, obtained at different optimal time windows σ^o. Similarly, here, we use the 20% earliest samples to train the ranking algorithms in temporal networks to determine the respective optimal σ^o, and then use the remaining 80% samples to test all the algorithms.

[f]For the random algorithm, here we mean randomly assigning ranks to all candidates.

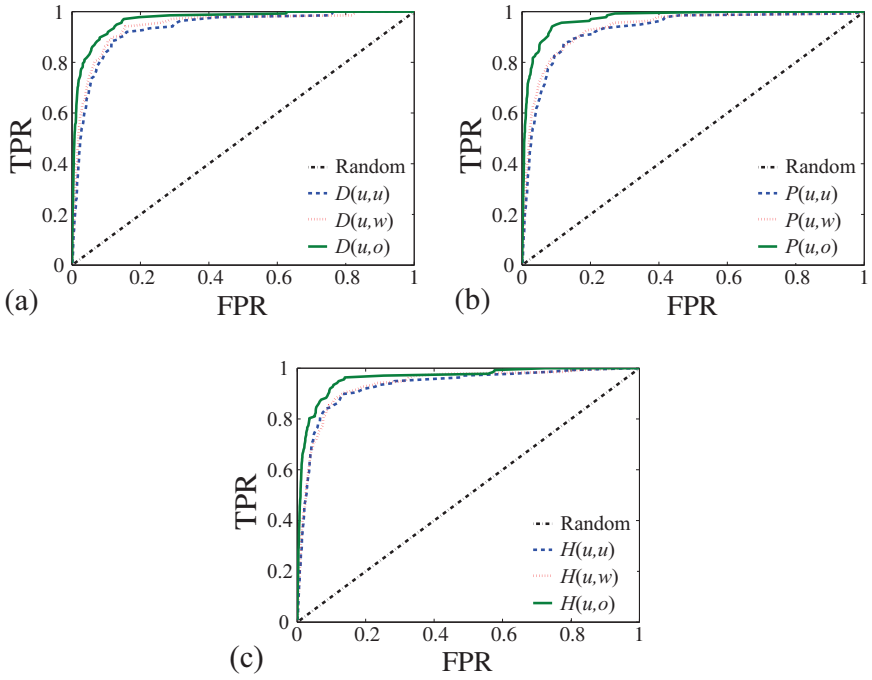

Fig. 3. ROC curves for (a) the degree-based algorithm, (b) the PageRank algorithm, and (c) the Hits algorithm, in cumulative unweighted networks (dashed), cumulative weighted networks (dotted), and temporal weighted networks (solid), respectively, for the undirected case.

In this case, the ROC curves of different algorithms in different types of networks are shown in Fig. 3; for the temporal and cumulative weighted networks, the average rankings are 3.38% versus 5.98% for the Degree, 2.91% versus 6.81% for the PageRank, and 4.07% versus 6.81% for the Hits, respectively; while the corresponding AUC for the ROC curves are shown in Table 3. The results suggest that, in the undirected case, all the three algorithms also behave better in the temporal weighted networks than in the cumulative unweighted and weighted networks, and overall, the PageRank in temporal networks behaves best.

3.3. *Percentages of the best rankings*

As we can see, in both directed and undirected cases, for each algorithm, the optimal rankings of emergent developers in temporal weighted networks are always higher (smaller R) than their rankings in cumulative unweighted and weighted networks. Then, to further compare the optimal rankings obtained by different algorithms in temporal weighted networks, we count the times that certain algorithm behaves best (get the highest rankings) over the others when the developers emerged. In particular, we consider the six ranking algorithms: $P(d,o)$, $H_A(d,o)$, and $H_H(d,o)$ in directed networks and $D(u,o)$, $P(u,o)$, and $H(u,o)$ in undirected networks. For

Table 4. The percentages of the best rankings obtained by different ranking algorithms in temporal directed and undirected weighted networks.

$P(d,o)$	$H_A(d,o)$	$H_H(d,o)$	$D(u,o)$	$P(u,o)$	$H(u,o)$
17.52%	28.47%	33.58%	30.66%	28.47%	25.55%

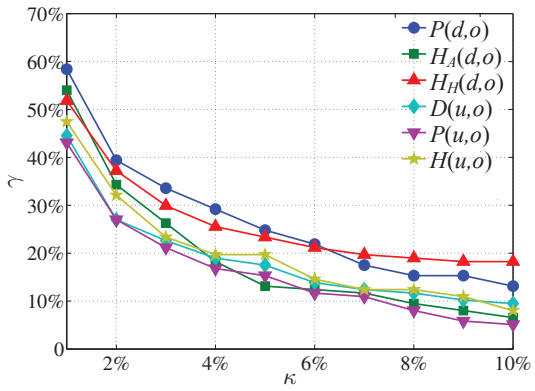

Fig. 4. The relationships between the risk of missing emergent developers, γ, and the preferential observing ratio, κ, of all the candidates by considering the four projects together and adopting different ranking algorithms in temporal weighted networks. γ is a decrease function of κ, meaning that the more candidates you observe, the lower risk you will miss emergent developers.

each test case, if an algorithm gets the highest ranking for the emergent developer over the other five, we say it has the best ranking at that time. Then, for each algorithm, we define the percentage of the best rankings as the ratio of the times it has the best ranking to the number of all the test cases.

The percentages of the best rankings for the six algorithms in temporal weighted networks are recorded in Table 4. Note that the sum of the percentages in the table may be larger than 1, since more than one algorithm may have the best ranking at the same time. We find that the Hits-Hub algorithm in directed networks, $H_H(d,o)$, has the best rankings in most test cases (33.58% of all test cases), the degree-based algorithm in undirected networks, $D(u,o)$, follows, while the PageRank algorithm in directed networks, $P(d,o)$, has the best rankings in fewest test cases (17.52% of all test cases).

3.4. *Risks of missing developers*

In many situations, provided an algorithm to rank developer candidates, the managers of the OSS projects may just be interested in the top κ candidates, and thus are concerned about the risk of missing emergent developers by this algorithm,

which is defined as

$$\gamma(\kappa) = \frac{N_C(\kappa)}{N_C} \times 100\%, \tag{20}$$

where N_C is the total number of test samples, i.e., 137 emergent developers in the four projects here, and $N_C(\kappa)$ is the number of cases where the rankings of the emergent developers are lower than κ. The risk of missing emergent developers defined by Eq. (20) then can be used as another measurement to compare the performances of different ranking algorithms, i.e., smaller risk of missing emergent developers means higher performance of the algorithm. Fig. 4 shows the missing risk γ as function of the ratio κ by considering the four projects together and adopting different ranking algorithms in temporal weighted networks are shown in . There we find that the PageRank algorithm in undirected networks, $P(u, o)$, always has the lowest risk of missing emergent developers, when the managers just observe a small number of top candidates based on their rankings.

Generally, by comparison, we find that the degree-based and the PageRank algorithms in temporal undirected weighted networks, $D(u, o)$ and $P(u, o)$, have the following advantages:

- Obtaining a relatively large true positive rate for a given false positive rate, as presented in Fig. 3 and Table 3.
- Having a relatively low risk of missing emergent developers when only observing a small number of top candidates based on their rankings, as shown in Fig. 4.

We thus recommend to apply these two algorithms based on temporal undirected weighted networks in OSS projects to assist the managers to find responsible developer candidates.

4. Conclusion

In this paper, we adopt degree-based, PageRank, and Hits ranking algorithms to rank developer candidates in four OSS projects based on different types of networks constructed by the email communication records. Interestingly, we find that all of these algorithms behave better in temporal networks than in cumulative ones, which indicates that the more recent communications of a developer in a project are more important to predict his/her first commit in the project. This finding suggests that the use of ranking algorithms in time-varying networks [29–32] has the potential to improve existing works which are based on a static network perspective [22–24, 38].

Meanwhile, we find that the degree-based and the PageRank algorithms in temporal undirected weighted networks behave well in predicting emergent developers and thus are promising candidates to be applied in OSS projects. In fact, about 80% of all emergent developers are ranked among the top 5% candidates ($R < 0.05$) by using these two algorithms, i.e., if there is a total of 1000 candidates in an OSS project, the manager only needs to observe about 50 of them with top rankings, since these important candidates will become developers in the near future with

relatively high probabilities. These ranking algorithms thus can be used to design developer recommendation systems for OSS projects, which can assist managers to find highly responsible and skillful programmers. The feedbacks of these candidates, either agree to become developers or not, can be used to improve the current ranking algorithms. Such ranking algorithms are most valuable in very large OSS projects, which is why future work should test our approach on suitable data sets from larger projects.

Our work can be further improved by considering more information about pre-commit activities of developers. For example, one can analyze sequences of emails [12] and the contents [39] in them, rather than just considering the total number of messages, thus possibly detecting more precise patterns of emergent developers; one can also measure the importance of code pieces contributed by the candidates in other communities or the submitted patches to address the quality of their work; moreover, these more detailed information can be used to characterize the similarity between the pre-commit activities of candidates and those of the existent developers, which may allow for more efficient ranking algorithms.

Acknowledgements

The authors gratefully acknowledge support from the National Natural Science Foundation of China (Grant No. 61004097, 61273212), the China Scholarship Council (CSC), and the China Postdoctoral Science Foundation (Grant No. 2014M551770).

References

[1] Xuan, Q., Gharehyazie, M., Devanbu, P. T., and Filkov, V., Measuring the effect of social communications on individual working rhythms: A case study of open source software, in *Proceedings of the 2012 IEEE International Conference on Social Informatics* (Washington, DC, USA, 2012), pp. 78–85.

[2] Ye, Y. and Kishida, K., Toward an understanding of the motivation of open source software developers, in *Proceedings of the 25th International Conference on Software Engineering* (Portland, OR, USA, 2003), pp. 419–429.

[3] Krogh, G. and Hippel, E., The promise of research on open source software, *Management Science* **52**(7) (2006) 975–983.

[4] Scacchi, W., Free/Open Source Software Development: Recent Research Results and Emerging Opportunities, in *Proceedings of the 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Dubrovnik, Croatia, 2007), pp. 459–468.

[5] Sowe, S. K., Stamelos, I., and Angelis, L., Understanding knowledge sharing activities in free/open source software projects: An empirical study,*Journal of Systems and Software* **81**(3) (2008) 431–446.

[6] Chen, X., Li, X., Clark, J. G., and Dietrich, G. B., Knowledge sharing in open source software project teams: A transactive memory system perspective, *International Journal of Information Management* **33**(3) (2013) 553–563.

[7] Bachmann, A. and Bernstein, A., When process data quality affects the number of bugs: Correlations in software engineering datasets, in *Proceedings of the IEEE*

Working Conference on Mining Software Repositories (Cape Town, South Africa, 2010), pp. 62–71.

[8] Zanetti, M. S., Scholtes, I., Tessone, C. J., and Schweitzer, F., Categorizing bugs with social networks: A case study on four open source software communities, in *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA, 2013), pp. 1032–1041.

[9] Bird, C., Gourley, A., Devanbu, P. T., Detecting patch submission and acceptance in OSS projects, in *Proceedings of the 4th International Workshop on Mining Software Repositories* (Minneapolis, USA, 2007).

[10] Gharehyazie, M., Posnett, D., and Filkov, V., Social activities rival patch submission for prediction of developer initiation in OSS projects, in *Proceedings of the 25th International Conference on Software Maintenance* (Eindhoven, The Netherlands, 2013), pp. 340–349.

[11] Jensen, C. and Scacchi, W., Role migration and advancement processes in OSSD projects: A comparative case study, in *Proceedings of the 29th International Conference on Software Engineering* (Minneapolis, USA, 2007), pp. 364–374.

[12] Xuan, Q. and Filkov, V., Building it together: synchronous development in OSS, in *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India, 2014), pp. 222–233.

[13] Bird, C., Gourley, A., Devanbu, P. T., Swaminathan, A., and Hsu, G., Open borders? Immigration in open source projects, in *Proceedings of the 4th International Workshop on Mining Software Repositories* (Minneapolis, USA, 2007).

[14] Zhang, W., Yang, Y., and Wang, Q., An empirical study on identifying core developers using network analysis, in *Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies* (Lund, Sweden, 2012), pp. 43–48.

[15] Wu, C.-G., Gerlach, J. H., and Young, C. E., An empirical analysis of open source software developers' motivations and continuance intentions, *Information & Management* **44**(3) (2007) 253–262.

[16] Capiluppi, A., Serebrenik, A., and Youssef, A., Developing an h-index for OSS developers, in *Proceedings of the 9th International Working Conference on Mining Software Repositories* (Zurich, Switzerland, 2012), pp. 251–254.

[17] Saul, Z. M., Filkov, V., Devanbu, P. T., and Bird, C., Recommending random walks, in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Dubrovnik, Croatia, 2007), pp. 15–24.

[18] Wu, W., Zhang, W., Yang, Y., and Wang, Q., DREX: Developer recommendation with k-nearest-neighbor search and expertise ranking, in *Proceedings of the 18th Asia Pacific Software Engineering Conference* (Ho Chi Minh City, Vietnam, 2011), pp. 389–396.

[19] Herbsleb, J. and Grinter, R., Architectures, coordination, and distance: Conway's law and beyond, *IEEE Software* **16**(5) (1999) 63–70.

[20] Bryan, K. and Leise, T., The $25,000,000,000 eigenvector: The linear algebra behind google, *SIAM Review* **48**(3) (2006) 569–581.

[21] Kleinberg, J. M., Authoritative sources in a hyperlinked environment, *Journal of the ACM* **46**(5) (1999) 604–632.

[22] Bui, D. L., Nguyen, T. T., and Ha, Q. T., Measuring the influence of bloggers in their community based on the H-index family, in *Advanced Computational Methods for Knowledge Engineering*, 2014, pp. 313–324.

[23] Wang, Y., Iliofotou, M., Faloutsos, M., and Wu, B., Analyzing Communication Interaction Networks (CINs) in enterprises and inferring hierarchies, *Computer Networks*,

57(10) (2013) 2147–2158.

[24] Ding, Y., Yan, E., Frazho, A., and Caverlee, J., PageRank for ranking authors in co-citation networks, *Journal of the American Society for Information Science and Technology*, *60*(11) (2009) 2229–2243.

[25] Xuan, Q., Du, F., and Wu, T.-J., Empirical analysis of Internet telephone network: From user ID to phone, *Chaos* **19**(2) (2009) 023101.

[26] Newman, M. E. J., Forrest, S., Balthrop, J., Email networks and the spread of computer viruses, *Physical Review E* **66**(3) (2002) 035101.

[27] Bird, C., Pattison, D., D'Souza, R., Filkov, V., and Devanbu, P. T., Latent social structure in open source projects, in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Atlanta, Georgia, USA, 2008), pp. 24–35.

[28] Leskovec, J. and Horvitz, E., Planetary-scale views on a large instant-messaging network, in *Proceedings of the 17th International Conference on World Wide Web* (Beijing, China, 2008), pp. 915–924.

[29] Kempe, D., Kleinberg, J., and Kumar, A., Connectivity and inference problems for temporal networks, in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (Portland, OR, USA, 2000), pp. 504–513.

[30] Li, X. and Croft, W. B., Time-based language models, in *Proceedings of the 12th International Conference on Information and Knowledge Management* (New Orleans, LA, USA, 2003) pp. 469–475.

[31] Yu, P. S., Li, X., and Liu, B., On the temporal dimension of search, in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters* (New York, NY, USA, 2004), pp. 448–449.

[32] Jatowt, A., Kawai, Y., and Tanaka, K., Temporal ranking of search engine results, *Lecture Notes in Computer Science* **3806** (2005) 43–52.

[33] Zweig, M. H. and Campbell, G., Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. *Clinical Chemistry 39*(4) (1993) 561-577.

[34] Bradley, A. P., The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* **30**(7) (1997) 1145–1159.

[35] Bird, C., Gourley, A., Devanbu, P. T., Gertz, M., and Swaminathan, A., Mining email social networks, in *Proceedings of the 4th International Workshop on Mining Software Repositories* (Washington, DC, USA, 2006) pp. 137–143.

[36] Costa, L. D. F., Rodrigues, F. A., Travieso, G., and Boas, P. R. V., Characterization of complex networks: A survey of measurements, *Advances in Physics* **56**(1) (2007) 167–242.

[37] Langville, A. N. and Meyer, C. D., Deeper inside pageRank, *Internet Mathematics* **1**(3) (2004) 335–380.

[38] http://www.githire.com/

[39] Ehrlich, K., Lin, C. Y., and Griffiths-Fisher, V., Searching for experts in the enterprise: combining text and social network analysis, in *Proceedings of the 2007 International ACM Conference on Supporting Group Work* (Sanibel Island, Florida, USA, 2007), pp. 117–126).