Hindawi
Complexity
Volume 2020, Article ID 7264396, 12 pages
https://doi.org/10.1155/2020/7264396

Research Article

BurstBiRank: Co-Ranking Developers and Projects in GitHub with Complex Network Structures and Bursty Interactions

Dengcheng Yan,[1] **Zhen Shao**,[2] **Yiwen Zhang**,[2] **and Bin Qi**[2]

[1]*Institutes of Physical Science and Information Technology, Anhui University, Hefei 230601, China*
[2]*School of Computer Science and Technology, Anhui University, Hefei 230601, China*

Correspondence should be addressed to Yiwen Zhang; zhangyiwen@ahu.edu.cn

Received 31 July 2020; Accepted 4 December 2020; Published 16 December 2020

Academic Editor: Shi Cheng

With the wide adoption of social collaborative coding, more and more developers participate and collaborate on platforms such as GitHub through rich social and technical relationships, forming a large-scale complex technical system. Like the functionalities of critical nodes in other complex systems, influential developers and projects usually play an important role in driving this technical system to more optimized states with higher efficiency for software development, which makes it a meaningful research direction on identifying influential developers and projects in social collaborative coding platforms. However, traditional ranking methods seldom take into account the continuous interactions and the driving forces of human dynamics. In this paper, we combine the bursty interactions and the bipartite network structure between developers and projects and propose the BurstBiRank model. Firstly, the burstiness between each pair of developers and projects is calculated. Secondly, a weighted developer-project bipartite network is constructed using the burstiness as weight. Finally, an iterative score diffusion process is applied to this bipartite network and a final ranking score is obtained at the stationary state. The real-world case study on GitHub demonstrates the effectiveness of our proposed BurstBiRank and the outperformance of traditional ranking methods.

1. Introduction

Social collaborative coding is now a popular paradigm among software developers, and collaborations of developers from all over the world can be easily conducted with the social and technical functionalities provided by such kind of platforms like GitHub. For example, in GitHub, developers can follow each other to form a social network, keep track of the updates of a project by the star and watch functionalities, contribute codes by the commit and pull request functionalities, or participate in the discussions of new features design or bug fix by the issue functionality. Rich social and technical functionalities connect developers and projects to form a large-scale complex technical system. It is known that critical nodes usually play important role in operation management and optimization of complex systems. The same goes for complex technical systems such as GitHub, which is usually driven by influential developers and projects to more optimized states with higher efficiency for software development. For example, in addition to direct collaboration, developers always seek popular developers and projects for improving coding ability and technical selection, which in turn makes collaborations more efficient. Thus, identifying influential developers and projects is of great significance for the improvement of developer's ability and the prosperity of open source community and also has important applications in service recommendations [1, 2] and quality of service prediction [3–6].

Existing work on influence analysis in open source software community often simply employs basic properties [7], network structural metrics [8], or traditional unipartite graph ranking model [9–12]. The influence of developers

and projects, two major and tightly coupled components of open source software community, is usually evaluated separately although many new graph ranking methods for complex network structures like bipartite network [13] have been proposed. On the other hand, abundant activities of developers are not utilized effectively while our previous study [14] indicates the statistical characteristics of developers' behavior is useful for distinguishing elite and common developers. Figure 1 shows a comparison of contributions between an elite developer Taylor Otwell and a common developer Franz Liedke, which shows different statistical characteristics of their behavior.

In this paper, we aim at mutually identifying influential developers and projects in GitHub by adopting a combination of the burstiness behavior of developers and the bipartite network topology of developer-project interactions. The contributions of this paper are listed as follows:

(1) We propose a burstiness-weighted bipartite network model to incorporate bursty behaviors between developers and projects into network topology.

(2) We combine the diffusion-based ranking method BiRank and the burstiness-weighted bipartite network and propose a new ranking method called BurstBiRank for mutually identifying influential developers and projects in GitHub.

(3) We apply the proposed model to a real-world GitHub dataset, showing that burstiness can correctly measure developers' attention to projects and our model outperforms baseline models.

The remainder of the paper is organized as follows. Section 2 introduces the related works on graph ranking methods and human dynamics. The details of our proposed BurstBiRank method are illustrated in Section 3. Then, the experiment results and discussions are given in Section 4. Finally, we briefly summarize our work and explain future directions in Section 5.

2. Related Work

Influential node identification has been a hot topic in network science research for decades, and many graph ranking methods have been proposed from different views of network structures or information diffusion mechanisms on various kinds of complex networks [15–18]. PageRank [19] and HITS [20] are the most popular ones. PageRank [19] is a random walk-based ranking method and uses the probability a random surfer appears on a web page as the influence score of the web page, while HITS [20] distinguishes authority and hub features of a web page and ranking a web page with both authority score and hub score.

Many graph ranking methods are based on PageRank and HITS. Considering individual's preference, Haveliwala et al. [21] proposed a personalized PageRank algorithm, and a personalized vector was introduced for expressing individual's preference for certain topics, novelty, and sensitivity of individuals' generated contents. Inspired by the discrete-time Markov process interpretation of PageRank, Liu et al. [22] proposed BrowseRank based on continuous-time Markov processes and used user behavior data to rank the importance of pages. In order to overcome parameter tuning of PageRank which is caused by dangling nodes in the network, Lu et al. [23] introduced a ground node connecting to all other nodes and proposed LeaderRank. Then, Li et al. [24] extended LeaderRank to weighted network.

In addition to ranking on unipartite network, recent research studies also extend graph ranking methods of unipartite network to bipartite network. In contrast to random walk-based graph ranking methods, He et al. [13] proposed BiRank, an optimization based ranking method for bipartite network. Xu et al. [25] applied singular value decomposition to bipartite network and proposed SVDRank and SVDARank. Morone et al. [26] extended the k-core decomposition method to the bipartite network and pointed out that in the ecological symbiosis network, the extinction of the maximum k-core node would make the ecosystem reach the critical point of collapse.

The rapid development of graph ranking models also promotes the research in influence analysis for open source software community. Xuan et al. [9] modeled the communications between developers in Apache as networks and analyzed developers' influence using degree, PageRank, and HITS. Joblin [8] et al. classified developers into core and peripheral with several network metrics. From the view of software projects, Inoue et al. [11] constructed a component graph with use relations for ranking software components. Pan et al. [27] constructed a multilayer complex network by extracting structural information from Java software systems and proposed a weighted PageRank algorithm for ranking classes or packages.

Although network structure plays an important role in identifying influential nodes in online social network, based on previous human dynamics study, Yan et al. [14] found that bursty behavior is a good indicator for distinguishing influential developers from common ones. Human dynamics studies the statistical characteristics of spatial or temporal behaviors of human beings and the potential laws behind it. Goh et al. [28] proposed the burstiness metric to measure to which extent the behavior deviates from periodic behavior.

3. Method

In this section, we will present a novel bipartite network ranking framework incorporating burstiness interactions, called BurstBiRank, for mutually identifying influential developers and projects in GitHub. First, we will introduce the definition of burstiness, which plays an important role in our proposed method for measuring how much attention a developer pays on a project. Then, a thorough description is given about the definition and construction of the burstiness-weighted developer-project bipartite network, and a diffusion-based ranking process is applied on this bipartite network. Finally, the overall algorithm is proposed and its time complexity is analyzed. The notations we will use throughout the article are summarized in Table 1.

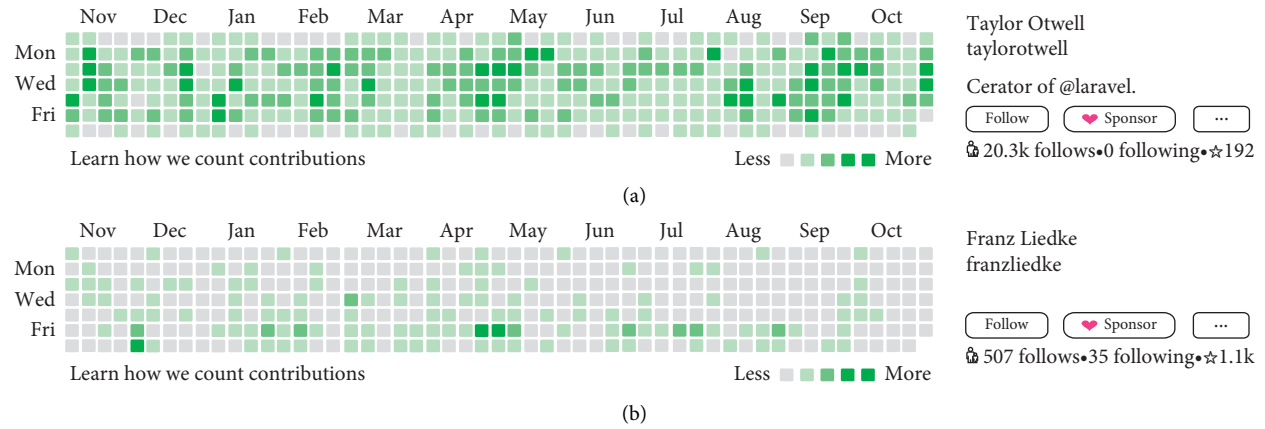

Figure 1: A comparison of two developer behaviors.

Table 1: Notations and explanations.

Symbol	Explanation
u_i	Ranking score of developer i
p_j	Ranking score of project j
u_i^0	Query vectors for developer i
p_j^0	Query vectors for project j
d_i^u, D^u	d_i^u degree of developer i D^u diagonal matrix with $D_{ii}^u = d_i^u$
d_j^p, D^p	d_j^p degree of project j D^p diagonal matrix with $D_{jj}^p = d_j^p$
B_{ij}	Burstiness of interactions between developer i and project j
w_{ij}, W	Weight matrix of burstiness-weighted bipartite network; w_{ij} is its element
α, β	Infection rate and recovery rate in SIR simulation, respectively
γ, λ	Hyperparameters for balancing diffusion scores and prior beliefs
S	Symmetric normalization of weight matrix W

3.1. Burstiness. In many real-world or online systems, people's activity is often intermittent, displaying intense activity during a short period followed by a long period of reduced activity or even no activity. For example, you may spend a total afternoon searching YouTube for videos about husky when you are free, but then seldom visit YouTube in weekdays. This pattern of human behavior and the laws behind it have been studied extensively in the field of human dynamics, and Goh et al. [28] proposed the burstiness metric to measure to which extent the behavior deviates from periodic behavior, which is defined as

$$B = \frac{\sigma_\tau - m_\tau}{\sigma_\tau + m_\tau}, \tag{1}$$

where $\sigma_\tau$and m_τ represent the standard deviation and the mean of the time interval series of human activities, respectively. It can be concluded from the definition: (1) the value of B ranges from -1 to 1; (2) $B > 0$ indicates the behavior is bursty, and the larger B is, the stronger the burstiness is; (3) $B < 0$ indicates cyclical trend, and the smaller B is, the stronger the periodicity is.

3.2. Burstiness-Weighted Developer-Project Bipartite Network

Definition 1. Burstiness-weighted developer-project bipartite network: a burstiness-weighted developer-project bipartite network is a weighted bipartite network $G = (U \cup P, E)$, where U and P denote two disjoint sets of nodes, that is, set of developers and set of projects, respectively, and E represents edges between developers and projects. The burstiness-weighted developer-project bipartite network can be described by a bipartite weight matrix W ($\in R^{|U|\times|P|}$) with elements w_{ij} ($1i \le |\mathrm{U}|$, $1j \le |\mathrm{P}|$) indicating tie strength between developer i and project j, which is a function of the burstiness of interactions between developer i and project j, that is,

$$w_{ij} = f\left(B_{ij}\right). \tag{2}$$

Figure 2 shows a sample burstiness-weighted developer-project bipartite network. Multiple interactions between each pair of developer and project are grouped and burstiness is calculated first. Then, the bipartite weight matrix W is constructed using equation (2). Essentially, the function in equation (2) can be any form, linear or nonlinear. According to the characteristics of burstiness, to ensure edge weights are positive and cyclical interactions have larger weight, we choose a linear form of function f shown in equation (3) for simplicity.

$$w_{ij} = f\left(B_{ij}\right) = -B_{ij} + 1. \tag{3}$$

3.3. BiRank. Score diffusion is a general idea behind many popular ranking methods as PageRank [19] and BiRank [13], which employ an iterative process of diffusing score to neighbors until the stationary state. The final scores at stationary state are regarded as the ranking scores. The process of score diffusion can be formulized as equations (4) and (5), and scores of developers and projects are updated in turns.

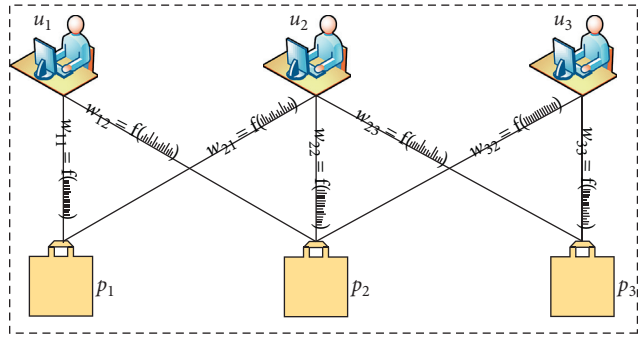

Figure 2: Burstiness-weighted developer-project bipartite network. The edge weight is function of burstiness which can be calculated using interaction sequences.

$$u_i = \sum_{j=1}^{|P|} w_{ij} p_j, \tag{4}$$

$$p_j = \sum_{i=1}^{|U|} w_{ij} u_i. \tag{5}$$

To ensure the convergence and stability, BiRank adopts symmetric normalization.

$$\begin{aligned} u_i &= \sum_{j=1}^{|P|} \frac{w_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}} p_j, \\ p_j &= \sum_{i=1}^{|U|} \frac{w_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}} u_i. \end{aligned} \tag{6}$$

BiRank also adopts a query vector in the ranking model to utilize prior beliefs on rankings of nodes as shown in equations (7) and (8). Prior beliefs and diffusion scores are balanced with hyperparameters γ and λ for developers and projects, respectively.

$$u_i = \gamma \sum_{j=1}^{|P|} \frac{w_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}} p_j + (1-\gamma) u_i^0, \tag{7}$$

$$p_j = \lambda \sum_{i=1}^{|U|} \frac{w_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}} u_i + (1-\lambda) p_j^0. \tag{8}$$

Finally, the equivalent matrix form of BiRank [7] can be obtained as equations (9) and (10)

$$u = \gamma S p + (1-\gamma) u^0, \tag{9}$$

$$p = \lambda S^T u + (1-\lambda) p^0, \tag{10}$$

where S is the symmetric normalization of weight matrix W.

$$S = \left(D^u\right)^{(-1/2)} W \left(D^p\right)^{(-1/2)}. \tag{11}$$

3.4. Overall Algorithm. Combing Sections 3.1, 3.2, and 3.3, we finally propose the BurstBiRank, and the overall algorithm is shown in Algorithm 1.

3.5. Time Complexity Analysis. The time complexity of BurstBiRank consists of two parts. The first part is the calculation of time intervals, burstiness, and edge weights, so the time complexity is $O(N_I + 2 * N_G)$, where N_I is the number of interactions and N_G is the number of developer-project groups. The second part is the iterative process of BurstBiRank algorithm, and the time complexity of equations (9) and (10) is $O(|U| \cdot |P|)$. However, most real-world networks are usually very sparse and only nonzero elements (which correspond to existing edges) should be stored and computed regarding matrix multiplication of $S^T u$ and Sp. Thus, the time complexity of the second part is $O(c|E|)$, where c is the number of iterations and $|E|$ is the number of edges. The overall time complexity of BurstBiRank is $O(N_I + 2 * N_G + c|E|)$.

4. Experiment

In this section, the performance of BurstBiRank is evaluated against real-world GitHub dataset [29]. All experiments are run on a Windows 10 PC with a corei7-4790 3.6 GHz CPU and 16 GB memory.

4.1. Datasets. GHTorrent dataset is an offline mirror of data offered through the GitHub REST API and a subset of it about PHP development community is used in this experiment. GHTorrent dataset as of November 1, 2018, is selected and preprocessed as follows: (1) commit interactions between developers and PHP projects are selected; (2) commit date is extracted from commit timestamp; (3) multiple commit interaction records of the same date are merged as one record; (4) developers who have equal or less than 10 records are excluded; (5) follow relationship between developers and watch interactions between developers and projects are extracted. The statistics of the dataset after preprocessing are shown in Table 2.

Input:
Developer-project interaction set (DP); query vectors u^0, p^0; and hyperparameters γ, λ
Output:
Ranking vectors u, p;
(1) Group developer-project interactions by developer and project;
(2) **for** developer-project interactions group in all groups **do**
(3) Sort developer-project interactions by commit time in descending order;
(4) Calculate time intervals between successive records;
(5) Calculate burstiness B_{ij};
(6) Calculate edge weight w_{ij} according to equation (3);
(7) **end for**
(8) Construct weight matrix W;
(9) Symmetrically normalize W according to equation (11);
(10) Randomly initialize u and p;
(11) **while** Stopping criteria are not met **do**
(12) Update u and p in turn according equations (9) and (10);
(13) **end while**
(14) **return** u and p

Algorithm 1: BurstBiRank algorithm.

Table 2: The statistics of the dataset.

Data	Count
Developers	129649
Projects	171296
Follows	41881
Watchers	27154
Developer-project commit interactions	8327263

4.2. Evaluation Metrics. Correlation analysis and SIR (susceptible-infected-removed) simulation are usually adopted for evaluation of graph ranking methods.

In correlation analysis, ranking results are compared with the ground truth using correlation coefficients. Kendall's tau [30] is one of such correlation coefficients and compares ranking orders instead of exact ranking scores or ground truth values. The definition of Kendall's tau is shown in the following equation:

$$\tau(X, Y) = \frac{2(C - D)}{n(n-1)}. \tag{12}$$

X and Y are two different lists with length n, which are usually the predicted ranking list and the ground truth ranking list. C and D are the numbers of concordant and discordant pairs between X and Y, respectively. Let $x_i, x_j \in X$and $y_i, y_j \in Y$; if $\text{sign}(x_i - x_j)\text{sign}(y_i - y_j) > 0$, then (x_i, y_i) and (x_j, y_j) are called a concordant pair, and if $\text{sign}(x_i - x_j)\text{sign}(y_i - y_j) < 0$, then (x_i, y_i) and (x_j, y_j) are called a discordant pair. In case of $\text{sign}(x_i - x_j)\text{sign}(y_i - y_j) = 0$, the pair is neither concordant nor discordant.

For the complexity of measuring influence, the ground truth for correlation analysis uses simply the degree of developer-developer following network or developer-project watching network. As we know, degree is local centrality metric which can only roughly measure node's influence from a local view while influence of a node in network mainly relates to its ability of spreading information to the whole network. Generally, a node with higher influence will spread information to more nodes in a network. Thus, we adopt the SIR model [31], a classical epidemic model, in our experiment to evaluate the performance of our proposed ranking method. As shown in Figure 3, in the SIR model, nodes in a network have three statuses, that is, susceptible (S), infected (I), and removed (R). Initially, a node is selected as infected node and the others are susceptible nodes. Then, an iterative transmission process is applied. At each iteration, infected nodes infect one of its susceptible neighbors with the probability α, and infected nodes recover to removed status with the probability β. The iterative transmission process stops when there are no infected nodes in the network. The final number of recovered nodes can be regarded as the influence of the node which is initially selected as infected.

4.3. Baseline Methods. To show the effectiveness of our proposed BurstBiRank, we compare it with several baseline ranking methods. In addition to burstiness-weighted developer-project bipartite network, two other developer-project bipartite networks are also constructed with unweighted edge (UW) and commit number-weighted edge (CN), and all baseline methods are evaluated on these two bipartite networks with corresponding suffix such as PageRank-UW. The hyperparameters of BurstBiRank γ and λ are both set to 0.85, and the query vectors u_0 and p_0 are set to the degrees of developers and projects over total number of nodes of each type, respectively, which can be calculated using equations (13) and (14).

$$u_i^0 = \frac{d_i^u}{|U|}, \tag{13}$$

$$p_j^0 = \frac{d_j^p}{|P|}. \tag{14}$$

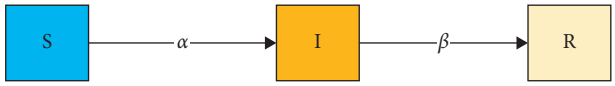

Figure 3: SIR model (source: figure is adapted from Pastor-Satorras et al. [32]).

BiRank [13] employs a diffusion-based way of utilizing mutual reinforcement between different types of nodes in bipartite network for mutually ranking two different types of nodes and adopts a normalization strategy in the iterative process. The hyperparameters are both set to 0.85 in our experiment.

SVDRank and SVDARank [25] apply singular value decomposition to bipartite network and select the first eigenvector as the final ranking vector. The difference between them is that SVDRank runs on the original bipartite network while SVDARank introduces two ground nodes to the original bipartite network before applying singular value decomposition in order to solve the problem of dangling nodes.

PageRank [19] regards ranking in network as a score diffusion process and ranks nodes by iteratively diffusing scores on the network. In our experiment, we directly apply PageRank on the developer-project bipartite networks ignoring types of nodes. The hyperparameter is set to 0.85 in our experiment.

4.4. Results

4.4.1. Correlation Analysis. In the experiment of correlation analysis, the number of followers of a developer in developer-developer following network is chosen as the ground truth for the rankings of developers, and the number of watchers of a project in developer-project watching network is chosen as the ground truth for the rankings of projects. Kendall's tau is calculated for developers and projects separately and is shown in Table 3.

From the results of correlation analysis in Table 3, we have the following observations:

(1) Our proposed BurstBiRank outperforms all baseline methods in identifying Top-20, Top-50, and Top-100 influential developers and projects except for Top-100 developers. This indicates the effectiveness of employing burstiness as the weight between developers and projects for measuring the influence of both developers and projects instead of ignoring edge weight or simply using commit number as edge weight. This result not only agrees with previous study [14] but also conforms to practical intuitions that developers pay more attention on important projects with continuous and regular work on them.

(2) For random walk-based ranking methods (i.e., PageRank and BiRank), higher performance can be obtained with unweighted edges than commit number-weighted edges for identifying high influential (i.e., Top-20) developers and projects, while it is just the opposite for decomposition-based ranking method (i.e., SVDRank and SVDARank). It indicates that how to measure the edge weight is important to model continuous interactions between developers and projects for influence analysis. This is also a key motivation why we model the edge weight as a function of burstiness, and future improvement can be applied on the form of the function.

(3) Almost all baseline methods have negative correlation results which indicate that the ranking orders by these methods are negatively correlated with the ground truth ranking orders, that is, influential developers or projects are usually ranked after less influential ones by these methods, while our proposed method always shows good positive correlation with the ground truth ranking orders, indicating good stability of our method.

4.4.2. SIR Simulation. In this section, the SIR model [31] is adopted to evaluate the performances of our proposed BurstBiRank and the baseline methods by comparing the ability of information spreading of Top-k developers and projects ranked by each method. In the experiment, BurstBiRank is compared with each baseline method separately. Because two comparing ranking methods usually rank a common group of nodes which will show equal effect in SIR simulation, for each pair of comparison, Top-50 developers (projects) ranked by each method are selected and only those developers (projects) not ranked by both methods are set as initial infected nodes. In each iteration t of SIR simulation, infected nodes randomly select one of their susceptible neighbors and infect it with probability $\alpha = 0.5$, and infected nodes recover to removed status with probability β, which is the reciprocal of the average of all node degrees. The accumulative number of infected nodes N_I is recorded for each iteration. Iteration stops when there is no infected nodes. To avoid the randomness of SIR simulation, 10 experiments are conducted for each pair of methods and the results are averaged as the final result. The final results are shown in Figures 4–7, and several significant observations are found:

(1) BurstBiRank outperforms all baseline methods which indicates the effectiveness of burstiness in identifying influential developers and projects. This finding will inspire developers to work continuously and regularly to obtain high influence in open source software community.

(2) The difference of performance between BurstBiRank and PageRank is larger than that between BurstBiRank and BiRank/SVDRank/SVDARank. As we know, PageRank is designed for unipartite network while BiRank, SVDRank, and SVDARank are special ranking methods for bipartite network which distinguish types of nodes and employ the mutual reinforcement between different types of nodes during ranking. This means it is better to

Table 3: Correlation analysis.

Method	Top-20		Top-50		Top-100	
	Project	Developer	Project	Developer	Project	Developer
PageRank-UW	0.1618	−0.0526	0.0211	0.0378	0.1190	0.0244
PageRank-CN	−0.0686	−0.1297	−0.0462	0.0299	0.1232	0.0272
SVDRank-UW	−0.2480	−0.0897	−0.3631	0.0197	0.0105	0.0773
SVDRank-CN	0.0831	0.1221	0.0173	0.0092	0.0428	0.0321
SVDARank-UW	−0.2000	0.0011	−0.3669	0.0200	0.0121	0.0700
SVDARank-CN	0.1024	−0.0263	0.0305	0.0121	0.0503	0.0435
BiRank-UW	0.0390	−0.0263	0.0296	0.0395	0.1195	0.0071
BiRank-CN	−0.0474	−0.2054	0.0207	0.0201	0.0314	0.0534
BurstBiRank	**0.1633**	**0.1604**	**0.0315**	**0.1221**	**0.1257**	**0.0843**

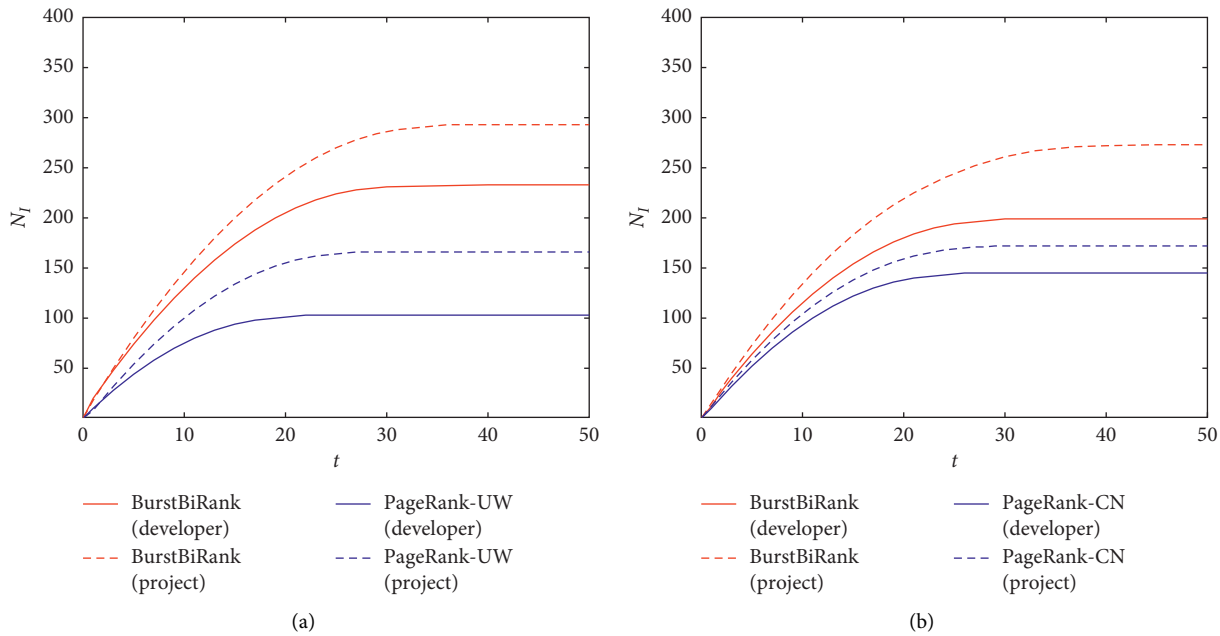

Figure 4: Performance comparison between BurstBiRank and PageRank. (a) BurstBiRank vs. PageRank-UW. (b) BurstBiRank vs. PageRank-CN.

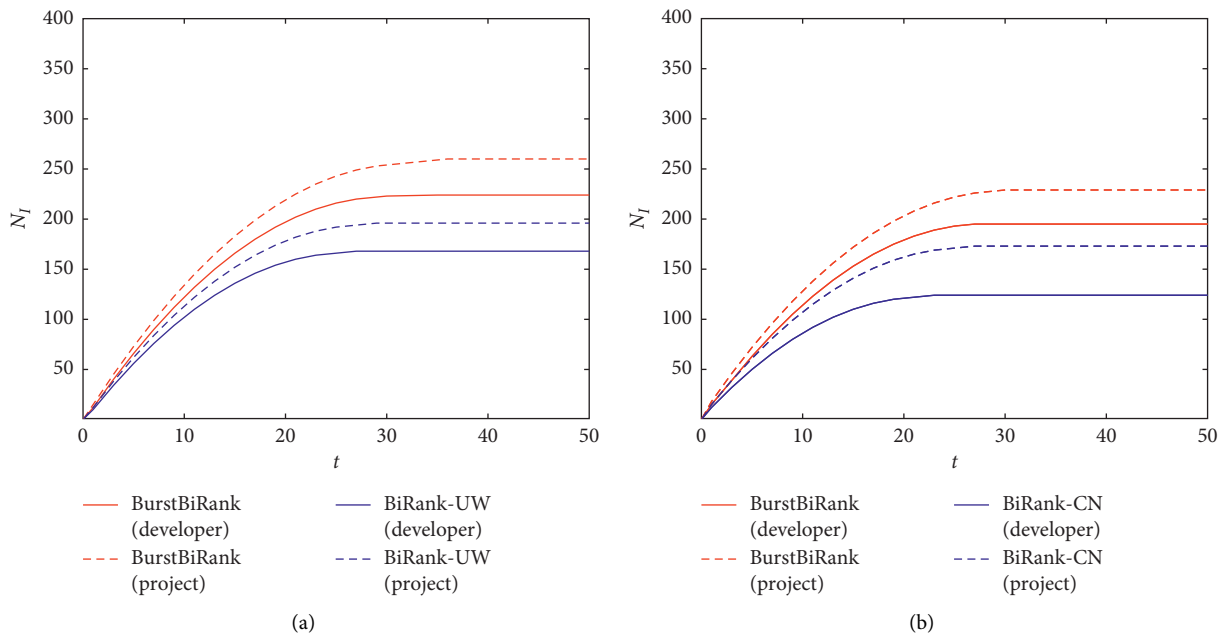

Figure 5: Performance comparison between BurstBiRank and BiRank. (a) BurstBiRank vs. BiRank-UW. (b) BurstBiRank vs. BiRank-CN.

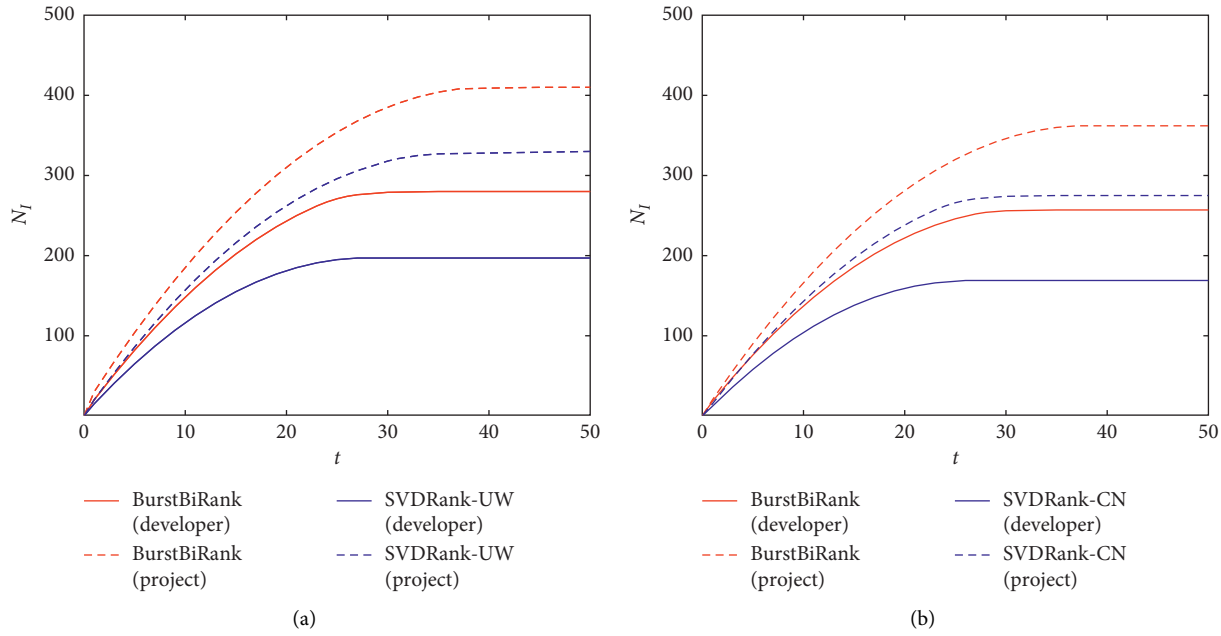

Figure 6: Performance comparison between BurstBiRank and SVDRank. (a) BurstBiRank vs. SVDRank-UW. (b) BurstBiRank vs. SVDRank-CN.

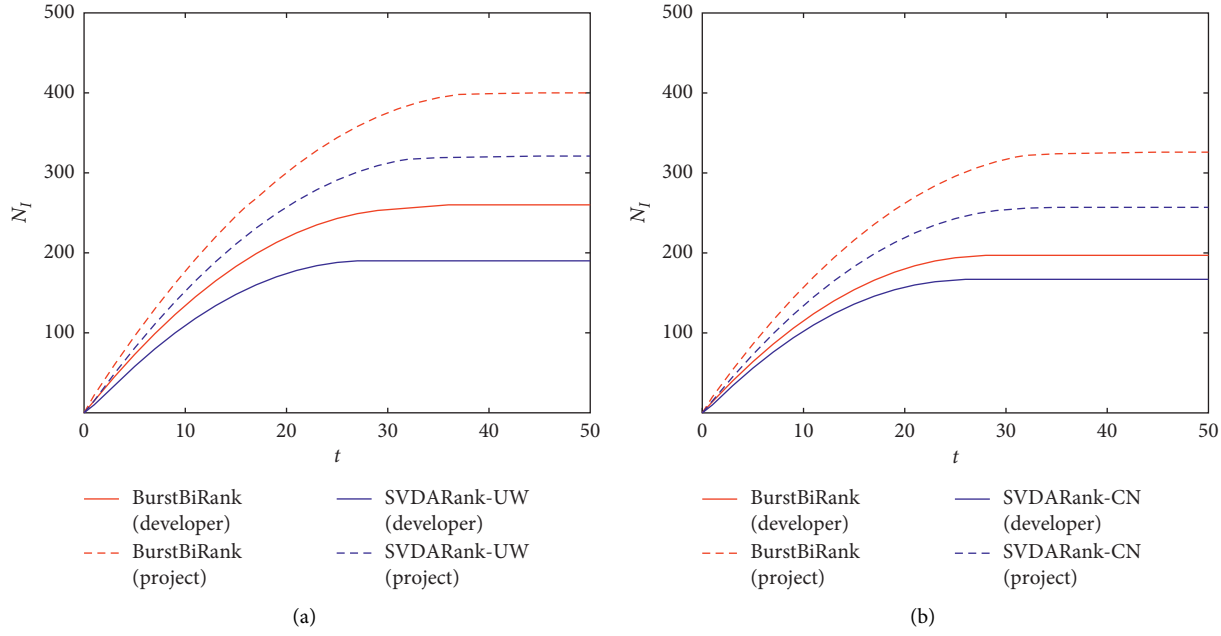

Figure 7: Performance comparison between BurstBiRank and SVDARank. (a) BurstBiRank vs. SVDARank-UW. (b) BurstBiRank vs. SVDARank-CN.

mutually co-rank developers and projects than to simply mix them up.

4.5. Case Study. In addition to correlation analysis and SIR simulation, we further do a detailed case study to show the effectiveness of our model in identifying influential developers and projects. Top-20 developers and projects ranked by BurstBiRank are shown in Tables 4 and 5, respectively, with their rankings in baseline methods.

From Table 4, we can see some major contributors to famous projects can be identified by BurstBiRank but they have lower rankings in baseline methods. For example, Marco Pivetta (GitHub ID: Ocramius), a major contributor of both ZendFramework and Doctrine ORM, is not ranked in Top-20 by BiRank-CN, SVDRank-UW, SVDRank-CN,

Table 4: Top-20 developer ranking.

Developer	Ranking								
	BurstBiRank	PageRank-UW	PageRank-CN	BiRank-UW	BiRank-CN	SVDRank-UW	SVDRank-CN	SVDARank-UW	SVDARank-CN
web-flow	1	1	1	1	1	1	1	1	1
StyleCIBot	2	4	6	4	5	2	16	2	16
QUVAUNKR	3	2	2	2	2	8	3	6	5
translatewiki	4	7	7	7	9	—	11	—	—
dereuromark	5	8	8	8	7	4	8	4	11
invalid-email-	6	3	4	3	3	—	—	20	—
weierophinney	7	6	5	6	4	9	—	—	7
fabpot	8	5	3	5	6	—	12	15	—
GrahamCampbell	9	14	16	14	—	3	7	3	4
s-nakajima	10	16	14	16	11	6	—	8	—
freekmurze	11	—	—	—	—	15	—	—	13
CTYLOQFP	12	17	19	—	—	10	9	—	9
Nyholm	13	—	—	—	—	7	—	9	—
legoktm	14	—	—	20	—	—	—	17	—
yunosh	15	10	9	10	13	—	—	—	20
FaustBrian	16	—	—	—	19	20	18	19	—
FHWXWWSE	17	—	13	—	—	—	20	—	19
Ocramius	18	19	12	18	—	—	—	—	—
taylorotwell	19	—	—	—	20	—	—	—	—
stronk7	20	12	10	15	14	17	—	—	—

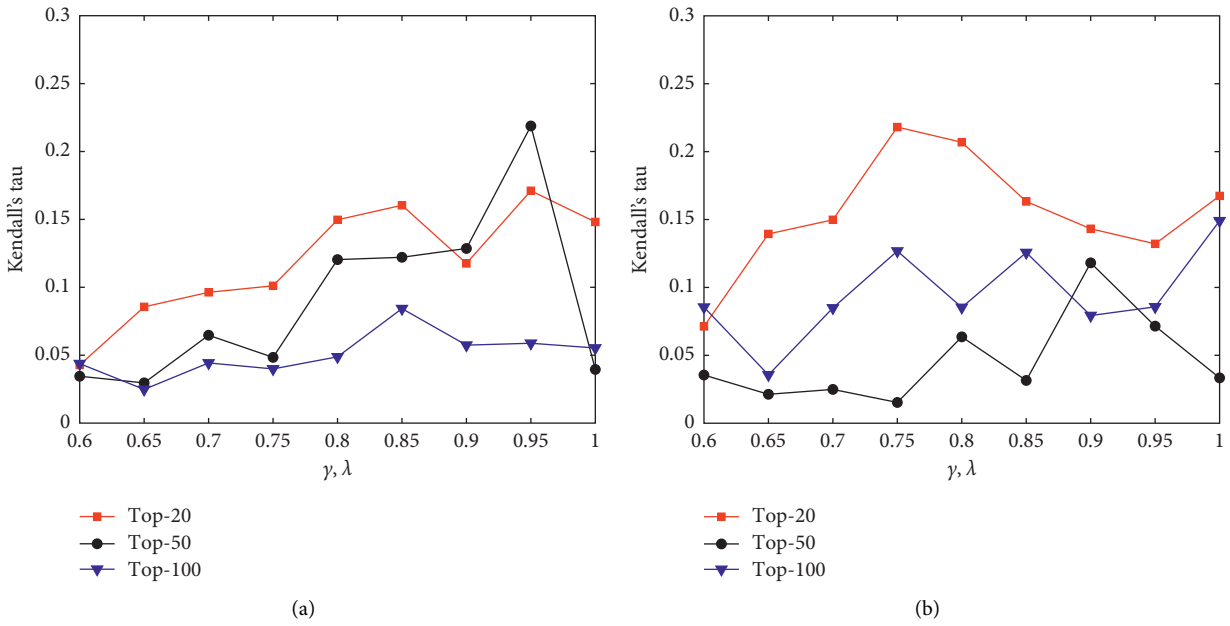

Figure 8: Parameter analysis for (a) developers' ranking performance and (b) projects' ranking performance.

SVDARank-UW, and SVDARank-CN. Taylor Otwell (GitHub ID: taylorotwell), the creator and major contributor to Laravel, is only identified by our BurstBiRank and BiRank-CN.

As for projects, famous PHP projects like Symfony (GitHub ID: symfony/symfony) and MediaWiki (GitHub ID: wikimedia/mediawiki) can also be identified as Top-20 influential projects by our method but with lower rank by baseline methods.

BurstBiRank and the baseline methods all can identify several influential developers and projects, but some less popular developers and projects are also given a high rank because the dataset is a real-world dataset and only a little filtering operations are applied to it. To sum up about correlation analysis, SIR simulation, and case study, our proposed BurstBiRank outperforms baseline methods and can identify some influential developers and projects of real-world open source software community. But further improvement should be conducted.

4.6. Parameter Analysis. In this section, we investigate how the performance varies with the hypermeters that balance the prior beliefs and diffusion scores. For simplicity, we constrain γ to be equal to λ, and Kendall's tau is adopted to indicate the model's performance. Figure 8 shows the raking performance by varying the balance parameters γ and λ from 0.6 to 1. For best ranking performance, the balance

Table 5: Top-20 project ranking.

Project	Ranking								
	BurstBiRank	PageRank-UW	PageRank-CN	BiRank-UW	BiRank-CN	SVDRank-UW	SVDRank-CN	SVDARank-UW	SVDARank-CN
magento/magento2	1	1	1	1	1	15	9	—	6
Wikia/app	2	2	2	2	2	7	3	10	2
brion/mediawiki-	3	3	6	4	8	—	—	—	—
owncloud/core	4	7	4	7	4	1	2	1	4
kaltura/server	5	6	5	6	5	—	6	18	10
oroinc/platform	6	20	19	—	15	13	—	8	—
ZimbraOS/zm-aspel	7	4	8	3	6	—	—	—	—
Automattic/jetpack	8	5	3	5	3	—	13	—	15
wikimedia/mediawi	9	10	12	—	—	19	—	—	—
svick/mediawiki	10	12	17	15	19	—	—	9	8
PrestaShop/PrestaS	11	11	10	9	7	10	11	12	11
orocrm/platform	12	—	—	—	—	—	18	—	—
joomla/joomla-cms	13	15	14	13	13	9	—	7	—
symfony/symfony	14	14	15	14	17	—	—	17	—
orocrm/crm	15	—	—	—	—	20	—	—	—
orocommerce/oroco	16	—	—	—	—	18	16	—	17
ec-europa/platform-	17	19	—	19	20	—	20	20	—
akeneo/pim-	18	—	7	—	9	—	5	—	7
ILIAS-eLearning/IL	19	—	—	—	—	—	—	—	20
cakephp/cakephp	20	—	9	—	10	2	1	3	1

parameters γ and λ are set not equal to 1, indicating the prior beliefs are useful for ranking developers and projects.

5. Conclusions

In this work, we aim at identifying influential developers and projects in open source software community. Continuous interactions between developers and projects are modeled as a burstiness-weighted bipartite network, and an iterative diffusion process is applied on it to calculate ranking scores for developers and projects. The proposed BurstBiRank is evaluated against four baseline methods on a real-world GitHub dataset. Extensive experimental analysis and case study show BurstBiRank outperforms baseline methods in both correlation analysis and SIR simulation.

The basic idea behind BurstBiRank is measuring the tie strength between developers and projects by the burstiness of the continuous interactions between them with an intuitively reasonable assumption that more regular interactions mean stronger ties. Under our framework, burstiness can be employed into the developer-project bipartite network by any linear or nonlinear functions, but in our experiment, a linear function is adopted for simplicity. In addition to burstiness, there are other metrics in human dynamics like memory, which may reflect the tie strength between developers and projects. Attributes of developers and projects such as programming language also affect rankings of them. In future work, we will adopt more types of functions, more metrics in human dynamics, and more attributes of developers and projects in our framework.

Data Availability

The data used in this study can be accessed via https://ghtorrent.org/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (grant no. 61872002), the University Natural Science Research Project of Anhui Province (grant no. KJ2019A0037), the University Collaborative Innovation Program of Anhui Province (grant no. GXXT- 2019-013), and the Doctoral Scientific Research Foundation of Anhui University (grant no. Y040418194).

References

[1] Y. Zhang, G. Cui, S. Deng, F. Chen, Y. Wang, and Q. He, "Efficient query of quality correlation for service composition," *IEEE Transactions on Services Computing*, 2018.

[2] Y. Zhang, C. Yin, Q. Wu, Q. He, and H. Zhu, "Location-aware deep collaborative filtering for service recommendation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 99, pp. 1–12, 2019.

[3] M. Tang, Z. Zheng, G. Kang, J. Liu, Y. Yang, and T. Zhang, "Collaborative web service quality prediction via exploiting matrix factorization and network map," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 126–137, 2016.

[4] Y. Zhang, K. Wang, Q. He et al., "Covering-based web service quality prediction via neighborhood-aware matrix factorization," *IEEE Transactions on Services Computing*, p. 1, 2019.

[5] S. Wan, Y. Xia, and L. Qi, "Automated colorization of a grayscale image with seed points propagation," *IEEE Transactions on Multimedia*, vol. 22, pp. 1756–1768, 2020.

[6] Y. Zhang, J. Pan, L. Qi, and Q. He, "Privacy-preserving quality prediction for edge-based IoT services," *Future Generation Computer Systems*, vol. 114, pp. 336–348, 2021.

[7] H. Borges, A. Hora, and M. T. Valente, "Predicting the popularity of github repositories," in *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 1–10, Ciudad Real, Spain, 2016.

[8] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: an empirical study on count and network metrics," in *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 164–174, Buenos Aires, Argentina, 2017.

[9] Q. Xuan, C. Fu, and L. Yu, "Ranking developer candidates by social links," *Advances in Complex Systems*, vol. 17, no. 07n08, Article ID 1550005, 2014.

[10] Y. Hu, S. Wang, Y. Ren, and K. K. R. Choo, "User influence analysis for github developer social networks," *Expert Systems with Applications*, vol. 108, pp. 108–118, 2018.

[11] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Ranking significance of software components based on use relations," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 213–225, 2005.

[12] S. Ding, S. Qu, Y. Xi, and S. Wan, "Stimulus-driven and concept-driven analysis for image caption generation," *Neurocomputing*, vol. 398, pp. 520–530, 2020.

[13] X. He, M. Gao, M.-Y. Kan, and D. Wang, "Birank: towards ranking on bipartite graphs," *IEEE Transactions on Knowledge and Data*, vol. 29, no. 1, pp. 57–71, 2016.

[14] D.-C. Yan, Z.-W. Wei, X.-P. Han, and B.-H. Wang, "Empirical analysis on the human dynamics of blogging behavior on github," *Physica A: Statistical Mechanics and Its Applications*, vol. 465, pp. 775–781, 2017.

[15] L. Lü, D. Chen, X.-L. Ren, Q.-M. Zhang, Y.-C. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *Physics Reports*, vol. 650, pp. 1–63, 2016.

[16] S. Wan, L. Qi, X. Xu, C. Tong, and Z. Gu, "Deep learning models for real-time human activity recognition with smartphones," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 743–755, 2020.

[17] Z.-K. Zhang, C. Liu, X.-X. Zhan, X. Lu, C.-X. Zhang, and Y.-C. Zhang, "Dynamics of information diffusion and its applications on complex networks," *Physics Reports*, vol. 651, pp. 1–34, 2016.

[18] Y. Zhao, H. Li, S. Wan et al., "Knowledge-aided convolutional neural network for small organ segmentation," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 4, pp. 1363–1373, 2019.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web," Technical report, Stanford InfoLab, Stanford, CA, USA, 1999.

[20] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[21] T. H. Haveliwala, "Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 784–796, 2003.

[22] Y. Liu, B. Gao, T.-Y. Liu et al., "Browserank: letting web users vote for page importance," in *Proceedings of the31st Annual International ACM SIGIR Conference*, pp. 451–458, Singapore, 2008.

[23] L. Lu, Y.-C. Zhang, C. H. Yeung, and T. Zhou, "Leaders in social networks, the delicious case," *PloS One*, vol. 6, no. 6, 2011.

[24] Q. Li, T. Zhou, L. Lü, and D. Chen, "Identifying influential spreaders by weighted leaderrank," *Physica A: Statistical Mechanics and Its Applications*, vol. 404, pp. 47–55, 2014.

[25] S. Xu, P. Wang, and C. Zhang, "Identification of influential spreaders in bipartite networks:A singular value decomposition approach," *Physica A: Statistical Mechanics and Its Applications*, vol. 513, pp. 297–306, 2019.

[26] F. Morone, G. Del Ferraro, and H. A. Makse, "The k-core as a predictor of structural collapse in mutualistic ecosystems," *Nature Physics*, vol. 15, no. 1, pp. 95–102, 2019.

[27] W. Pan, H. Ming, C. Chang, Z. Yang, and D.-K. Kim, "Elementrank: ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, 2019.

[28] K.-I. Goh and A.-L. Barabási, "Burstiness and memory in complex systems," *EPL (Europhysics Letters)*, vol. 81, no. 4, p. 48002, 2008.

[29] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 233–236, San Francisco, CA, USA, 2013.

[30] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," *Journal of the American Statistical Association*, vol. 61, no. 314, pp. 436–439, 1966.

[31] R. Yang, B.-H. Wang, J. Ren et al., "Epidemic spreading on heterogeneous networks with identical infectivity," *Phys. Lett. A*, vol. 364, no. 3-4, pp. 189–193, 2007.

[32] R. Pastor-Satorras, C. Castellano, P. Van Mieghem, and A. Vespignani, "Epidemic processes in complex networks," *Reviews of Modern Physics*, vol. 87, no. 3, p. 925, 2015.