ICIC Express Letters
Volume **10**, Number **12**, December **2016**
 ISSN 1881-803X
pp. **2837–2844**

A NEW METHOD OF IDENTIFYING INFLUENTIAL NODES IN COMPLEX SOFTWARE NETWORK BASED ON LEADERRANK

Guoyan Huang[1,2], Jing Liu[1,2,*], Xiaojuan Chen[1,2]
and Jiadong Ren[1,2]

[1]College of Information Science and Engineering
Yanshan University
[2]The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province
No. 438, West Hebei Ave., Qinhuangdao 066004, P. R. China
{ hgy; jdren }@ysu.edu.cn; *Corresponding author: thestudyone@sina.com

Received March 2016; accepted June 2016

Abstract. *Identifying influential nodes is crucial for understanding and improving the stability and robustness of complex software network. This paper presents a new method based on LeaderRank information to identify top-k influential nodes in directed-weighted complex software network. Firstly, the function and relationship of function calling or dependency are mapped to a Directed function dependency network (DN). Secondly, a so-called stem node connected with every other node by a bidirectional link is introduced into the original network. Thirdly, an algorithm Constructing DWN (C-DWN) is used to transform a DN into a Directed-Weighted function dependency network (DWN). Finally, the Identifying Influential Node in Software Network (IIN-SN) algorithm is proposed to identify influential nodes. Experimental results show that the proposed method is effective for identifying influential nodes with the final score ranking following power-laws. The top-k most influential nodes are the same and the final score (NS) of each node is a little larger during the software evolution. What is more, we present that some factors such as in-degree of nodes and influence of connected nodes are related to the influence of nodes.*
Keywords: Software network, Influential nodes, Weighted network, LeaderRank

1. **Introduction.** In recent years, software network research has received much attention in many fields, including degree distribution, average path length, preferential attachment, and clustering coefficient. In particular, evaluating the influence of nodes is a critical research task in software networks. Many mechanisms such as cascading and spreading are highly affected by a tiny fraction of influential nodes [1-3]. If problems occur in these nodes, software industry will suffer a great loss. So, identifying influential nodes is of great theoretical and practical significance in software networks.

Many empirical studies have been greatly helpful for understanding the software systems. They have uncovered that software networks, extracted from various software systems, follow power-law degree distributions, represent small-world properties, exhibit community phenomena, and show some other complex behavior characteristics [4]. Various centrality measures have been proposed over the years to rank the nodes of a graph according to their topological importance [5]. Based on these works, research on influential nodes identification is emerging and has attracted wide-spread attention. Freeman [6] adopted betweenness to measure the importance of nodes, and pointed out that nodes with larger betweenness may be more important than others in software network. However, it is time-consuming to calculate the betweenness of each node. Ugander et al. [7] found that the number of connected subgraphs between neighbor nodes has higher influence on the importance of nodes than absolute number of neighbor nodes. Chen et al. [8] proposed a local centrality measure as a trade off between the low-relevant degree centrality and other time-consuming measures. However, it did not take nearest neighbors into

consideration when identifying important nodes. Kitsak and Havlin [9] believed that the position of a node in global network has a great impact on its importance and adopted k-shell decomposition analysis to rank node importance, and the results outperform degree and betweenness. Recently, Lv et al. [10] proposed the LeaderRank algorithm to identify influential spreaders in directed and unweighted social networks. On the one hand, the LeaderRank algorithm outperforms PageRank in identifying users who lead to quick and wide spreading of useful items. On the other hand, the LeaderRank algorithm performs high robustness to intentional attacks. These advantages make LeaderRank a good method for ranking users as well as other ranking tasks.

As [11] discussed, scale-free networks guarantee the robustness of the global software systems when a few random functions fail. However, failure of some key nodes with high influence in software networks will be fatal fragility for functionality of the system [12]. To address the issue, this paper proposes a new method to identify influential nodes of spreading failure in directed-weighted software network. We introduce a so-called stem node connected with every other node by a bidirectional link. That is to say the method is relevant to global structure. Nodes with more in-degrees get more weights from the stem node to dig out influential nodes. Designers or developers should pay more attention to nodes with high ability of spreading failures to improve the robustness of complex software network.

The rest of this paper is organized as follows. In Section 2 some definitions are proposed to describe the problem. Section 3 is a detailed description of our approach to identify influential nodes. The experimental results on two open-source software are shown in Section 4. Finally, the conclusions of the paper are presented in Section 5.

2. Problem Statement and Preliminaries.

We use complex software networks to represent software systems according to the calling relationship between functions. Specifically, all functions are modeled as nodes. When a node v_i calls another node v_j, the edge $v_i \rightarrow v_j$ is added to the networks. Ultimately, the edges in the final network represent all calls of functions that appear in the executive process. The directed function dependency network can be described as $\mathrm{DN} = \{\mathrm{V}, \mathrm{E}, \mathrm{M}\}$. V is a set of nodes in the network, like $\{v_1, v_2, \ldots, v_i, \ldots\}$. When function v_i calls function v_j, there would be a directed edge $< v_i, v_j >$ from the node v_i pointing to the node v_j. E is a set of directed edges, $\{< v_1, v_2 >, < v_2, v_3 >, \ldots, < v_i, v_j >, \ldots\}$. M is an adjacency matrix storing the adjacency relationship among all pairs of nodes.

In software networks, there are two types of node degrees: out-degree and in-degree. Out-degree and in-degree represent the number of edges that start from the node v_i and end at the node v_j, respectively. The out-degree of node v_i indicates the number of nodes called by node v_i, whereas the in-degree of node v_j indicates the number of nodes that call the node v_j. We use k_{in} and k_{out} to represent in-degree and out-degree of nodes, respectively.

We consider a network of DN, nodes represent the software functions and the edges represent the invoking relationship between the functions during the execution process. We introduce a stem node which connects to every nodes through bidirectional links (see Figure 1 for an illustration). Then, the network becomes strongly connected and consists of $\mathrm{V}+1$ nodes and $\mathrm{E}+2\mathrm{V}$ edges and the weight is defined as the following.

In this paper, we introduce a weighted mechanism according to the adjacency relationship among all pairs of nodes and the in-degree of normal nodes. (1) $w_{ij} = 1$ if normal node v_i points to normal v_j and 0 otherwise; (2) for any normal node v_i and the stem node s, $w_{si} = k_{in} + 1$ and $w_{is} = 1$. For example, $w_{ab} = 1$, $w_{as} = 1$, $w_{ae} = 0$. After determining the weight of every edge, the score from node v_j to node v_i is proportional to the weight w_{ji}.

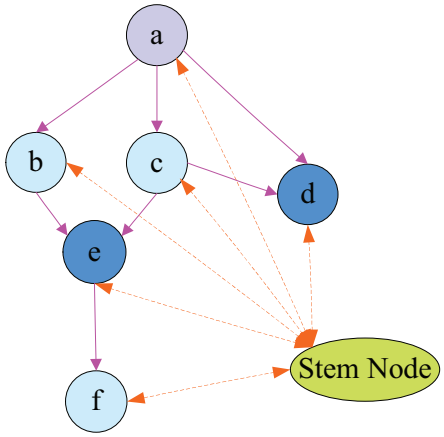


FIGURE 1. An illustration of the stem node

The directed-weighted function dependency network can be described as DWN = {V, E, W}. V is also a set of nodes in the network. E is a set of directed edges, too. W is an adjacency matrix storing the weight of adjacency relationship.

Definition 2.1. *RIS (Receive Influence Set). For a node v_i, RIS is a set of nodes, such as node v_j which calls node v_i directly, namely v_j could receive information from v_i and thus v_i will receive scores from v_j.*

$$RIS(v_i) = \{v_j \mid v_j \to v_i\}, \quad v_i, v_j \in \mathrm{V} \tag{1}$$

Definition 2.2. *RIW (Receive Influence Weight). For a node i, RIW is weight sum that gains the ability of influence. Obviously, the k_{out} and the weight of directly connected edges are important local indicators for a function to receive influence, and the k_{in} is important local indicator for a function to spread influence.*

$$RIW(i) = \sum_{l=1}^{N+1} w_{il} \tag{2}$$

Definition 2.3. *PNS (Present Node Score). PNS is defined to measure the influence of the node v_i at present. The PNS is given as follows:*

$$S_v(t+1) = \sum \frac{1}{RIW(u)} S_u(t) + \frac{w_{sv}}{RIW(s)} S_s(t), \quad u \in RIS(v) \tag{3}$$

Obviously, in-degree is an important local indicator for a function's influence in spreading. $S_v(t)$ is the score of node v at the tth step. Node u calls node v directly. The node s represents the stem node. Nodes with different in-degrees get different scores from the called nodes and the stem node. The NSs of all nodes are equal to the corresponding PNSs when PNS of each normal node remains the same or nearly the same to PNS of before time. The NS is used to quantify influence of nodes. Finally, we rank the NS of these nodes and the nodes with larger final scores are considered to be more influential in spreading.

3. **Identifying Influential Nodes Based on LeaderRank.** We propose a new method based on LeaderRank information to identity influential nodes. This method is derived from two different perspectives: the caller's ability to accumulate failure and the callee's ability to spread failure. The influence of nodes is closely related to other nodes in network. Suppose there is a calling relationship between u and v (u calls v). If v contains a failure, then it may spread to node u. Similarly, node u may spread the failure to other nodes if they call u. According to the calling or dependency relationships among the nodes in software network, we can get a network of DN. In this section, we provide an algorithm

Constructing DWN (C-DWN) that is used to get the weight of each directed edge and then the DWN is constructed. Then we put forward an algorithm Identifying Influential Node in Software Network (IIN-SN) to mine the top-k influential nodes in directed-weighted function dependency network.

The stem node is related to the global topology structure of software network. The weights of the edges express interacted influence of the nodes. We define weights according to the adjacency relationship among all pairs of nodes and the in-degree of nodes. Then according to the adjacency relationship among all normal pairs of nodes, weights of original edges are given. Nodes with more degrees get more weights from the stem node to dig out influential nodes. Finally for the edges of starting from the stem node, weights are given. Algorithm C-DWN is detailed in Algorithm 1.

Algorithm 1 C-DWN: constructing a network of DWN

Input: Directed function dependency Network (DN)
Output: Directed-Weighted function dependency Network (DWN)
1: **for** (each $v_i \in$ V)
2: **if** ($< v_{si}, v_{ej} > \in$ E *and* $v_i = v_{ej}$), then inDegreeList(v_i).add(v_{ej})
3: **end** // inDegreeList(v_i).size is equal to in-degree of node v_i.
4: **for** (*each value* $m(i,j)$ *in* M)
5: $w(i,j) = m(i,j)$ //$w(i,j) \in$ V
6: **end**
7: **for** (*new value in* W)
8: $w(i,s) = 1$ and $w(s,i) = inDegreeList(v_i).size + 1$
9: **end**
10: V and E are not changed in DWN
11: **return** DWN

To measure the influence of spreading failure, a novel algorithm IIN-SN is put forward based on recursive technique. Primary principle of algorithm is to initialize the same value to each normal node. All initial values of PNS are given. In the process of recursive calculation, normal nodes and the stem node allocate their PNS to other nodes according to correlation of nodes, so that, each node obtains corresponding node score. With spreading of node measurement score, normal nodes and the stem node get PNS, and PNS of each node is constantly updated with constant spreading. The result of PNS of the previous iteration is multiplied by the matrix P again and continues to get new value of PNS, and it achieves the goal of influence obtaining and spreading. When PNS of each normal node is equal or similar to PNS of before time, the recursion terminates.

Algorithm 2 IIN-SN: calculate the score of each node

Input: directed-weighted adjacency matrix W
Output: final Node Score vector NS
1: **for** $i = 1 : |\mathrm{V} + 1|$ // $|\mathrm{V} + 1|$ is the number of rows in W.
2: $RIW(i) = sum\ w(i,j)$
3: **end**
4: **for** each $p(i,j) \in$ P
5: $p(i,j) = w(i,j)/RIW(i)$
6: **end**
7: initialize PNS
8: **do** $\{ONS = PNS\ and\ PNS* = \mathrm{P}\}$ **while** $(|PNS - ONS| < temp)$
9: NS = PNS
10: **return** NS

In IIN-SN Algorithm 2, ONS is used to store the value of PNS, and temp is a tiny convergent threshold. The process starts with the initialization of PNS where the score of the normal node is 1 and the stem node is 0, and the process will soon converge to a unique steady state. Finally, we rank the NS of these normal nodes and mine the top-k influential nodes in the software network.

4. **Experimental Analysis.** In order to verify the effectiveness of the method for identifying influential nodes, and to research the changes of influential nodes that occur during the evolution of software systems, in this section, two open-source software Cflow and Tar will be tested for empirical analysis, which are available from the open source software library: Http://sourceforge.net.

The score of nodes reflects the influential degree of nodes in global network. Figure 2 and Figure 3 show the score ranking of nodes of the four versions under investigation. As mentioned previously, the Cflow and Tar are studied separately in our research. Here, Figure 2 and Figure 3 include the results of the Cflow and Tar, respectively. Note that the vertical axis is the score of nodes and the horizontal axis indicates the ranking of the nodes under investigation.

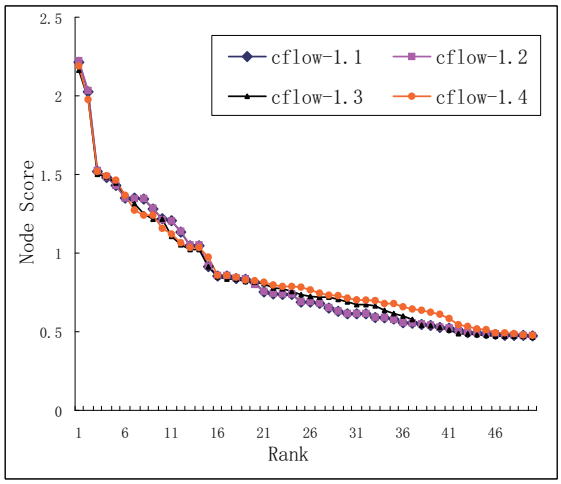


FIGURE 2. NS rank of *Cflow*

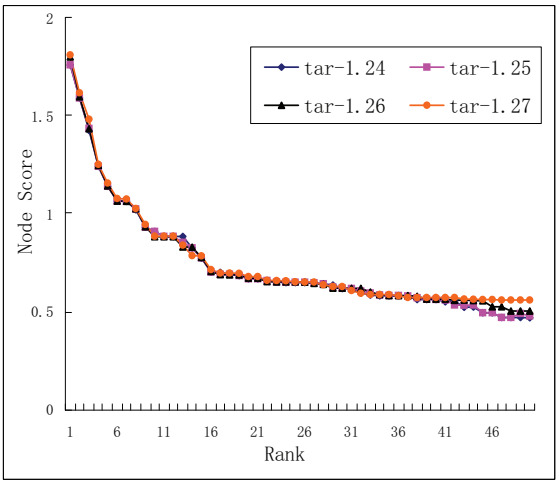
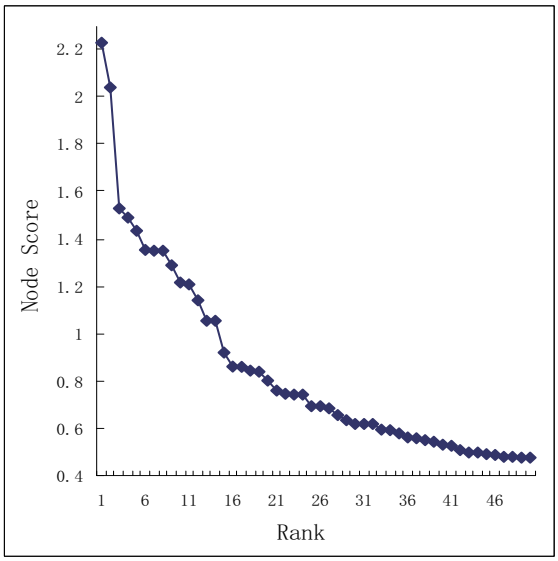


FIGURE 3. NS rank of *Tar*

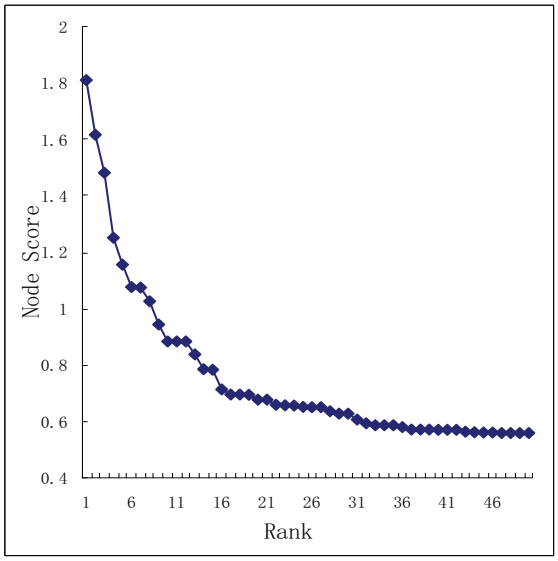
We find that, for the Cflow and Tar, the versions have similar variation tendency and each of new versions has a slightly higher one during the software evolution. It indicates that the versions do not change too much considering the system's stability and the structure of the two software systems is a little more complex during the software evolution.

We also find that, slope of curve changes greatly at the beginning, and then decreases gradually. For further analysis of these characteristics, we specifically research a version of the Cflow and Tar, respectively in Figure 4. We show the score ranking of nodes for tar-1.27 and cflow-1.2, respectively. Interestingly, the distributions roughly follow power-laws. A few nodes tend to have high influence, but very low influence for most of the nodes. Scale-free distribution is a common property in complex networks. For example, the cumulative in-degree distribution also obeys the power law.

The influential functions working well are vital to software systems. In Table 1 and Table 2, we present the top rank of the nodes for different versions of the software of Tar and Cflow. From Table 1 and Table 2, we find that nodes of high score and the rank of nodes are almost unchanged in the two softwares. The two functions to_ chars and nexttoken have much higher NS than other functions in each of the versions, so we can predict that the two functions have also high influence in update versions in the future, respectively. We also find that the values of most functions increase gradually, which



(a) NS rank of tar-1.27



(b) NS rank of cflow-1.2

FIGURE 4. NS rank of the Tar version and the Cflow version

TABLE 1. Node ranking for each version of Tar

	tar-1.24		tar-1.25		tar-1.26		tar-1.27	
function name	rank	value	rank	value	rank	value	rank	value
to_ chars	1	1.7588	1	1.7588	1	1.7994	1	1.8099
to_ octal	2	1.5896	2	1.5896	2	1.5965	2	1.6163
assign_ string	3	1.4246	3	1.4346	3	1.4340	3	1.4817
tar_ copy_ str	4	1.2408	4	1.2408	4	1.2464	4	1.2516
flush_ archive	5	1.1397	5	1.1397	5	1.1423	5	1.1561
find_ next_ block	6	1.0646	6	1.0646	6	1.0641	6	1.0769
set_ next_ block_ after	7	1.0634	7	1.0634	7	1.0626	7	1.0748
tar_ stat_ destroy	8	1.0203	8	1.0233	8	1.0250	8	1.0265
yy_ flush_ write	9	0.9300	9	0.9300	15	0.7760	14	0.9443
tar_ stat_ close	11	0.9090	10	0.9094	13	0.8300	15	0.8842

TABLE 2. Node ranking for each version of Cflow

	cflow-1.1		cflow-1.2		cflow-1.3		cflow-1.4	
function name	rank	value	rank	value	rank	value	rank	value
nexttoken	1	2.2190	1	2.2280	1	2.1688	1	2.1948
gnu_ output_ handler	2	2.0306	2	2.0387	2	1.9843	2	1.9821
print_ symbol	3	1.5230	3	1.5290	4	1.4883	4	1.4960
putback	4	1.4847	4	1.4907	5	1.4508	5	1.4682
lookup	5	1.4342	5	1.4346	6	1.3720	6	1.3719
hash_ symbol_ hasher	6	1.3536	7	1.3509	9	1.2202	8	1.2202
hash_ symbol_ compare	7	1.3536	8	1.3509	10	1.2202	9	1.2459
yy_ load_ buffer_ state	8	1.3484	6	1.3539	7	1.3180	7	1.2771
include_ symbol	9	1.2843	9	1.2894	8	1.2550	10	1.1619
linked_ list_ append	11	1.2112	10	1.2160	3	1.5060	3	1.5240

means that the invoking relationships are more complex and the influences are much higher during the software evolution.

As mentioned previously, the cumulative in-degree distribution also obeys the power law. So we compare the top-10 functions in our method with In-Degree using tar-1.25 and cflow-1.4 in Table 3 to find the internal contact between them. We found that, for most of nodes in each version, the larger in-degree is, the larger NS is, and vice versa. Usually, the nodes with large in-degree represent simple, fundamental functions in software systems, which are frequently reused and not necessary to be dependent on other classes. And the nodes have significant external responsibility and high influence. So the influence of nodes has approximately positive correlation with the in-degree.

Table 3. The comparison of the functions ranking

top-10 functions in tar-1.25			top-10 functions in cflow-1.4		
function name	NS	In-Degree	function name	NS	In-Degree
to_ chars	1	1	nexttoken	1	1
to_ octal	2	20	gnu_ output_ handler	2	2
assign_ string	3	2	linked_ list_ append	3	3
tar_ copy_ str	4	3	print_ symbol	4	25
flush_ archive	5	10	putback	5	2
find_ next_ block	6	3	lookup	6	3
set_ next_ block_ after	7	3	yy_ load_ buffer_ state	7	7
tar_ stat_ destroy	8	4	hash_ symbol_ hasher	8	10
yy_ flush_ write	9	20	hash_ symbol_ compare	9	12
tar_ stat_ close	10	10	include_ symbol	10	12

We also note that, although the positive correlation is clear, there are a few special points. For the version tar-1.25, there is one special function in Table 3. NS is very high but in-degree is low for the to_ octal function. We analyze the call graph of kernel version tar-1.25 and find that function to_ chars calls the function to_ octal. As a result, to_ octal receives score from to_ chars. As a result of the same reason, there exist special points such as the function print_ symbol in the version cflow-1.4, as seen in Table 3. So a node's caller is of high influence, and this node will be highly influential as well.

In conclusion, influential nodes are not only related to the in-degree but also associated with the influence of the connected nodes.

5. **Conclusions.** In this paper, a new method of identifying influential nodes in complex software network was proposed based on LeaderRank information. Not only were the adjacency relationship among all pairs of nodes considered, but also the in-degree of nodes in a weighted network. The algorithm C-DWN was presented to transform a network of DN into a DWN, and IIN-SN algorithm was used to calculate the NS of each node in the network of DWN. The experimental results on two open-source software show that our method can well identify influential nodes. This paper not only finds some helpful factors to influence of nodes but also enriches the ranking method of complex software networks. Though C-DWN and IIN-SN are already effective algorithms, extensions may lead to further improvement. For instance, the role of the stem node would be more prominent if weights are set on the times of call to each node, according to its significance or other criteria.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 6157220, No. 61472341 and the Natural Science Foundation of Hebei Province, P. R. China, under Grant No. F2013203324, No. F2014203152 and No. F2015203326. The authors are grateful to the valuable comments and suggestions of the reviewers.

REFERENCES

[1] G. Barach and M. Tuchman, Distributions of betweenness in cascades of overload failure in random regular networks, *APS Meeting Abstracts*, 2014.
[2] Y. He and X. Zhu, Statistics and developing model of Chinese skyway network, *International Journal of Modern Physics B*, pp.2595-2598, 2012.
[3] Q. Wu and X. Fu, Immunization and epidemic threshold of an SIS model in complex networks, *Physica A Statistical Mechanics*, p.444, 2016.
[4] P. Hosek and C. Cadar, Safe software updates via multi-version execution, *Proc. of the International Conference on Software Engineering*, pp.612-621, 2013.
[5] P. Bhattacharya, M. Iliofotou, I. Neamtiu et al., Graph-based analysis and prediction for software evolution, *Proc of the 34th International Conference on Software Engineering*, pp.419-429, 2012.
[6] L. C. Freeman, Centrality in social networks, *Social Networks*, vol.1, no.3, pp.215-239, 2015.
[7] J. Ugander, L. Backstrom and C. Marlow, Structural diversity in social contagion, *Proc. of the National Academy of Sciences*, vol.109, no.16, pp.5962-5966, 2012.
[8] D. Chen, L. Lv and M. S. Shang, Identifying influential nodes in complex networks, *Physica A: Statistical Mechanics and Its Applications*, vol.391, no.4, pp.1777-1787, 2012.
[9] M. Kitsak and S. Havlin, Identification of influential spreaders in complex networks, *Nature Physics*, vol.6, no.11, pp.888-893, 2010.
[10] L. Lv, Y. C. Zhang and C. H. Yeung, Leaders in social networks, the delicious case, *PloS One*, vol.6, no.6, 2011.
[11] R. Albert, H. Jeong and A. L. Barabási, Error and attack tolerance of complex networks, *Nature*, vol.406, no.6794, pp.378-382, 2000.
[12] L. Wang, P. Yu, Z. Wang and Q. Ye, On the evolution of Linux kernels, *Journal of Software*, vol.25, no.5, 2013.