Hindawi
Security and Communication Networks
Volume 2018, Article ID 1525186, 10 pages
https://doi.org/10.1155/2018/1525186

*Research Article*

# Analysis on Influential Functions in the Weighted Software Network

**Haitao He,[1,2,3] Chun Shan ,[4] Xiangmin Tian ,[1,2,3] Yalei Wei,[1,2,3] and Guoyan Huang[1,2,3]**

[1]*College of Information Science and Engineering, Yanshan University, Qinhuangdao, Hebei 066004, China*
[2]*The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao, Hebei 066004, China*
[3]*The Key Laboratory for Software Engineering of Hebei Province, Qinhuangdao, Hebei 066004, China*
[4]*Beijing Key Laboratory of Software Security Engineering Technique, Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing 100081, China*

Correspondence should be addressed to Chun Shan; sherryshan@bit.edu.cn

Received 24 October 2017; Revised 5 January 2018; Accepted 4 February 2018; Published 1 April 2018

Academic Editor: Zheng Yan



Identifying influential nodes is important for software in terms of understanding the design patterns and controlling the development and the maintenance process. However, there are no efficient methods to discover them so far. Based on the invoking dependency relationships between the nodes, this paper proposes a novel approach to define the node importance for mining the influential software nodes. First, according to the multiple execution information, we construct a weighted software network (WSN) to denote the software execution dependency structure. Second, considering the invoking times and outdegree about software nodes, we improve the method PageRank and put forward the targeted algorithm FunctionRank to evaluate the node importance (NI) in weighted software network. It has higher influence when the node has lager value of NI. Finally, comparing the NI of nodes, we can obtain the most influential nodes in the software network. In addition, the experimental results show that the proposed approach has good performance in identifying the influential nodes.

## 1. Introduction

Measuring accurately the importance of the node in the software networks is the premise to improve the security and robustness of software [1, 2]. Moreover, with the development of the software, measuring the importance of nodes in the network has practical significance for defending and protecting the influential nodes in the software network [3], if these nodes are suffered by deliberate attacks, maybe cascading failure occurs [4, 5]. Accordingly, how to mine the potential characteristics of software to control the evolution process of the software structure has become a hot spot for researching [6–9].

Many researchers introduced the idea of complex networks to the field of software structure and abstracted software to a network from different granularity point [10]. With the network structure, many potential characteristics can be discovered directly. Ma et al. [11] abstracted interaction relationship between packages into a software network, and they defined functions in package as nodes and dependencies among functions as edges. Wang et al. [12] proposed an approach to study the evolution of special software kernel components, which adopted the theory of complex networks. They also proposed a generic method to find major structural changes that happened during the evolution of software systems. Li et al. [13] proposed a modular attachment mechanism of software network evolution. Their approach treated object-oriented software system as a modular network, which was more realistic. A new definition of asymmetric probabilities was given to acquire links in directed networks when new nodes attached to the existing network. With the directed network, both of the "scale-free" and "small-world" properties were verified to be present in the software network. In [14], David proposed a method to simplify the complexity of the software network. With the method, some valuable characteristics in the network could be obtained easily. From the researches above, the complex network was proved to be applicative in the software engineering and it brought

us a new perspective to research the software structure. However, these methods of modeling the software mentioned above were based on the static structure of the source code. The execution characteristic during the software running process was neglected in these methods. For the software, most of the characteristics are exhibited during the execution process.

The characteristics of the software execution can help us to understand the software better. It is obvious that the node is an important part of the network and it has enormous influence on the stability, reliability, and robustness of the network [15]. For a software network, the software function plays a critical role in the stability and robustness of the software during the execution process. In the structure of the software, the functions carry most of the feature characteristics and topology information and they can affect each other. In most cases, the fault of a function is not only caused by itself but also infected by the other functions. Recently, the importance of the node in the network was defined from different aspects. Bhattacharya et al. [2] defined a measure to evaluate relative importance of the nodes in software by assigning a numerical weight to each node of software graph. By the value of the betweenness and the clustering coefficient, Zhang et al. [16] measured the importance of each node to analyze the influence of each node to the entire network. According to the propagation field of the classes, Li et al. [17] put forward an indicator to measure the importance classes in the software network at class level. Based on the value of the indegree and the outdegree of each node, Wang and Lü [18] proposed a method to mine the influential nodes. With the method, they proved that the fault appeared with a large probability in those nodes with large degree value. In the researches above, the node was proved to play a key role to analyze the network. However, the node was regarded as an individual unit, as well as the relationship between the node and the entire network was ignored. In practical application, the network should be considered as a whole, in which the nodes can interact with each other.

Considering the above-mentioned shortcomings, the dependency relationship between the function nodes, and the absence of efficient analysis methods, we construct the WSN to show the software structure according to the information of multiple execution. Based on the dependency relationship between the function nodes, we present a targeted method FunctionRank to evaluate the importance of the software nodes. With the analysis result of each node, we rank the influence of each node to mine the top-$k$ nodes. These function nodes have played an important part in ensuring software reliability and stability. So they should be paid more attention in the process of software updating and software maintenance.

The primary contributions of this paper can be summarized as follows:

(i) A novel method is proposed to construct weighted software network (WSN). So we make the understanding and recognition of software structure more accurate.

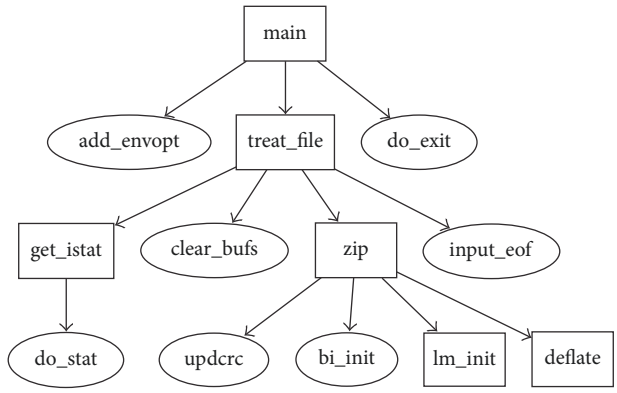



Figure 1: The software network.

(ii) A measurement node importance (NI) is put forward to evaluate the importance of each node in the network.

(iii) The IC (independent cascade) model as an attack model is used to evaluate the influential functions for software system.

(iv) The proposed algorithm is an effective method for security measurements of cybernetwork and provides basis for software security and reliability improvement.

The rest of this paper is organized as follows. The construction process of the weighted software network (WSN) is described in Section 2. The node importance of each function node is given definition in Section 3. Then, in Section 4, the method FunctionRank is given to mine the most influential nodes. In Section 5, the performances of the proposed algorithm are showed by experiments. Finally, conclusions and future works of the paper are presented in Section 6.

## 2. Definitions of Weighted Software Network

Complex networks are suitable to show the invoking relationships between the software functions. Based on the information of the multiple execution processes, we define the software execution dependency structure with a directed-weighted network.

*2.1. Software Network.* In this section, according to the multiple execution information, we define a software network to demonstrate the software execution dependency structure. Figure 1 shows a real example of software network.

Where each node represents a software function and each edge is the invoking relationship between the functions. In the software network, most of the characteristics can be exhibited during the software execution process.

*2.2. Weighted Software Network.* Next, in order to guarantee the completeness of the experimental data and make the understanding and recognition of software structure more accurate, we define a weighted software network. Compared with the software network, we consider invoking times

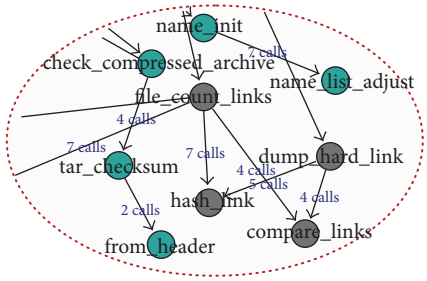



Figure 2: Weighted software network.

between the software functions in multiple execution processes as the weight of each edge. The weighted software network is suitable to demonstrate the complex invoking relationships between the software functions. The definition of weighted software network (WSN) is given as follows:

$$\mathrm{WSN} = \{\mathrm{Node}, \mathrm{Edge}, W\}, \quad \mathrm{Node}, \mathrm{Edge} \in \mathrm{software}. \tag{1}$$

Figure 2 shows a weighted software network, where Node is a software functions set and Edge is an invoking relationship set between the software functions. $W_e$ that stands for the weight of edge $e$ is calculated by the following formula:

$$W_e = \sum_j w_e, \quad e \in \mathrm{Edge}, \tag{2}$$

where $j$ is the times of the trials with different experiment cases. $w_e$ is a value of 1 or 0. If the edge $e$ of one calling relationship appears in an execution trace, no matter how many times of it, let $w_e$ be 1; otherwise it is 0.

Figure 3 presents a simple process of the WSN established. As shown in Figure 3(a), $si$ $(1 \le i \le 5)$ is a function invoking trace in one-time execution of the software. The trace $si$ contains a series of function calling relationships which can reflect the software execution process. Figure 3(b) shows a structure of WSN, in which the node and the edge of the network are defined as the function and the calling relationship between the functions appearing in $s1$~s5 in Figure 3(a), the weight of edges represents the number of each calling process executed in the 5 times' execution, and the times of a calling relationship in some execution processes were ignored.

Based on multiple execution information under the different experimental cases of software, we guarantee the completeness of the experimental data. Function nodes which have appeared during the software multiple execution processes are considered as a set of nodes of the network structure, calling relationship between the software functions is considered as a set of edges, the weight of the edge, we consider the weight $c$ to stand for the edge appearing $c$ times in the $N$ execution traces of the software, and we ignore the times appearing in an execution trace $si$. In this way, WSN is built.

## 3. Node Importance

According to the complex invoking relationships for software system, we show the most common topology structures of the weighted software network in Figure 4 to explain the importance of the function node.

*Definition 1* (IN (indegree nodes)). For a node $vi$, IN is a set of functions which call node $vi$ directly. The IN of node $vi$ is gotten by only one call step.

As shown in Figure 4(a), $\mathrm{IN}(A) = \{B, C\}$. The influence of node $vi$ is based on IN ($vi$) which call v$i$ directly.

*Definition 2* (ON (outdegree nodes)). For a node $vi$, ON is a set of functions which are called by node $vi$ directly. The number of ON($vi$) is $vi$'s outdegree, CO.

As shown in Figure 4(a), $\mathrm{ON}(A) = \{B, C, D\}$ and $\mathrm{CO}(A) = 3$.

*Definition 3* (TN (terminal nodes)). The nodes that have no outdegree and have no contribution to the influence of other nodes are defined as terminal nodes.

As shown in Figure 4(b), $C$ is a terminal node.

*Definition 4* (LTN (loop terminal nodes)). The nodes that only have an outlink to their own are defined as loop terminal nodes.

As shown in Figure 4(c), $C$ only has an outlink to its own. So $C$ is a loop terminal node.

*Definition 5* (OD (output degree)). The weight sum of each edge for a node $vi$ to its outdegree nodes, OD ($vi$), is named as output degree of the node $vi$.

In Figure 4(a), the weight of each edge for $A$ to ON ($A$) is 2, 2, and 5, respectively. OD ($A$) is the sum of these weights, namely, $A$'s output degree.

*Definition 6* (WC (weighted contribution)). The ratio of the weight for node $vi$ to node $vj$ and $vi$'s output degree, WC ($vj$), is the weighted contribution of $vi$ to $vj$.

In Figure 4(a), the weight of $A$ to $B$ is 2. The weighted contribution of $A$ to $B$ is given as follows:

$$\mathrm{WC}(B) = \frac{2}{\mathrm{OD}(A)} = \frac{2}{2+2+5} = \frac{2}{9}. \tag{3}$$

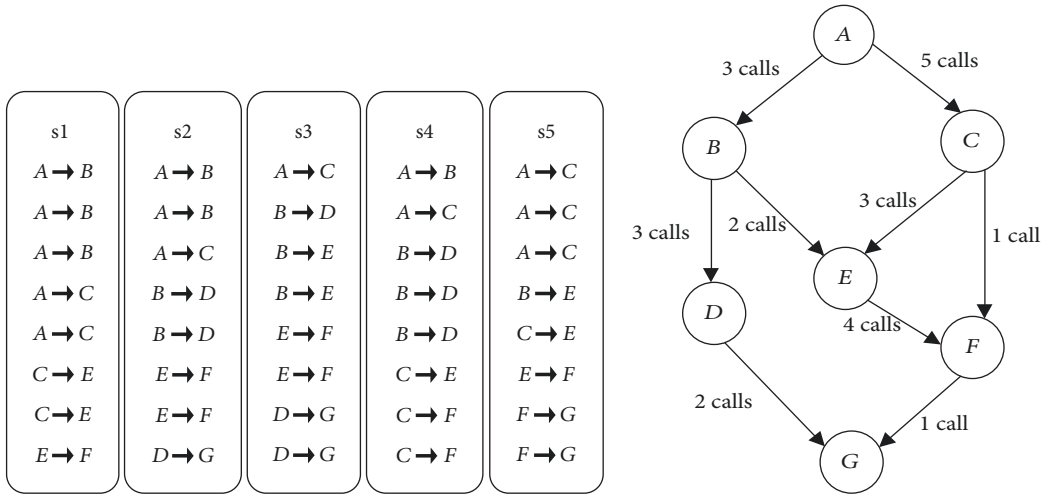
Based on the above definitions, the node importance (NI) of node $vj$ is given as follows:

$$\mathrm{NI}\left(V_j\right) = \left[\alpha + (1-\alpha) \sum_{\mathrm{IN}(Vj)} \mathrm{WC}\left(V_j\right) * \mathrm{NI}\left(V_i\right)\right] \cdot \left(\mathrm{CO}\left(V_j\right) + 1\right), \tag{4}$$

where $\alpha$ is the certain probability of calling a random node for LTN, and the probability of invoking each node is the same. It is set as 0.15 with experimental verification.

## 4. Important Nodes Mining

In this section, we first provide an algorithm outdegree nodes to get the outdegree node list of all nodes, according to the outdegree nodes of each node in the software network, and then we provide another algorithm FunctionRank




(a) The 5 times' execution traces (b) The structure of WSN

Figure 3: The constructing process of WSN.

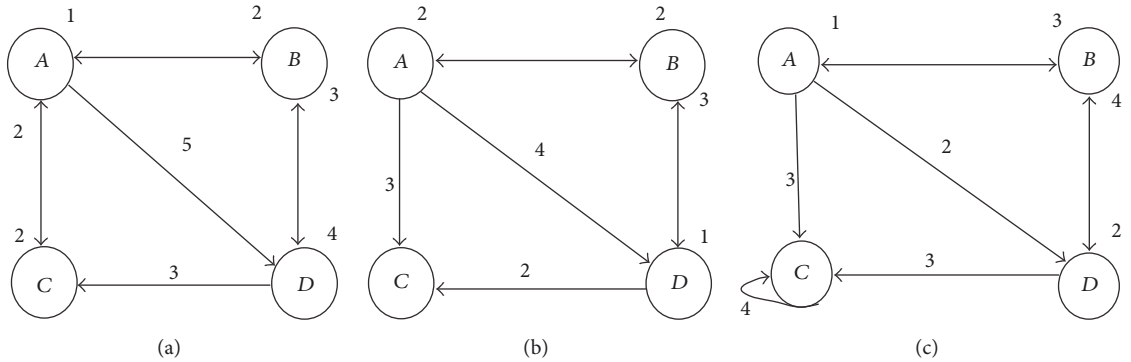



Figure 4: Three kinds of topology structures.

```
Input: node set V, edge set E
Output: childStr //the out-degree node list of all nodes
(01) for (each vi ∈ V) {
(02)  for (⟨vsi, vej⟩ ∈ E) {
(03)   if (vi = vsi)
(04)    childStr += "" + vej;
(05)  }
(06)  print (vi + childStr);
(07) }
```

Algorithm 1: Outdegree nodes.

to calculate NI of each node. In the method Function-Rank, we evaluate the importance of nodes iteratively (see Algorithms 1 and 2).

As shown in algorithm outdegree nodes, for each node in set $V$ we traverse the edges in set $E$ in line (1) and line (2). We define the nodes of an edge as start node *vsi* and end node *vej*, respectively. In line (3) to line (4) we add the end node *vej* of an edge to the childStr of node *vi*, when node *vi* equals the start node *vsi* of the edge. Finally, we print the childStr of *vi* in line (6).

We evaluate the importance of each node in the network by an iterative process, as shown in Algorithm 2. In line (1), we initialise NI(v*i*) as importance of nodes and $\alpha$ as the certain probability to call a random node, respectively. Line (2) to (19) is the iterative process to compute the importance coming from outdegree of the current node and other nodes which call the current node. The computational formula of node importance (NI) is given in line (18); it has higher influence when the node has lager value of NI. Ultimately, the importance for a node (NI) is obtained when error of current NI value and previous NI value is less than a given threshold for all nodes.

With the measuring results obtained from Algorithm 2, we choose the top-$k$ nodes as the influential nodes for the software network. In Algorithm 3, we illustrate the process of top-$k$ nodes (*KN*).

In Algorithm 3, we initialise list as the measurement list for all the nodes in line (1). Lines (2) to (4) are a looping

```
Input: node vi, childStr (vi)
Output: the NI of node vi //evaluate the importance of nodes
Process:
(01) Initialize NI(vi) = 1f, α = 0.15f
(02) if (childStr[vi] != null) {
(03) outdegree = childStr.size ();
(04) for (each vj ∈ childStr [vi]) {
(05)  if (vj is equal vi) {
(06)   outdegree - -;
(07)  }else{
(08)   weighMap.put (vj,weight (vi → vj));
(09)  weigh += weight (vi → vj);
(10)  }
(11) }
(12) for (each vj ∈ childStr [vi]) {
(13)  if (vj is not equal vi) {
(14)   tempNI (vj) += NI (vi) ∗ weighMap.get (vj)/weigh;
(15)   }
(16) }
(17) }
(18) tempNI (vi) = (α + (1 − α) ∗ tempNI (vi)) ∗ (outdegree (vi) + 1);
(19) NI (vi) = tempNI (vi);
```

Algorithm 2: FunctionRank.

```
Input: node set N, NI of each node
Output: the top - k influential nodes
Process:

(01) Initialize list //store the importance of nodes
(02) for (each node v ∈ N)
(03)  list.add (NI(v));
(04) end for
(05) Collections.sort (list);
(06) Collections.reverse (list);
(07) print list.get (k)
```

Algorithm 3: Top-$k$ nodes ($KN$).

process to store the NI value for each node. The sorting process is given in line (5) and line (6), the top-$k$ nodes are chosen from the list in line (7).

## 5. Experimental Analysis

A series of experiments were conducted to compare the performance of the proposed algorithm (named as FunctionRank) with different parameter values. They were implemented in JDK1.6.0 and executed on a PC with 3.30 GHz CPU and 5 GB memory.

*5.1. Experimental Datasets.* Firstly, several dynamic software datasets are used to evaluate the performance of the algorithms. The classical software is obtained from the open-source community. These software programs are coded in C or C++, including program software tar and cflow.

In the experiment, we chose different versions of tar and cflow, respectively, for experiment. tar is a decompression software for Linux, and cflow is an analysis tool for C program to extract the relationship of function calls (download from the open-source software library: Https://sourceforge.net).

*5.2. Evaluation on the FunctionRank.* We run the algorithm on each version of tar and cflow. By the algorithm FunctionRank, we calculate the *NI* of each function node. Here we mine top-10 nodes in each version about software tar and cflow. It is shown in Tables 1 and 2, respectively.

As it is shown in Table 1, for versions tar-1.21 and tar-1.23, the *NI* of the top-10 are almost the same. The reason is that the difference between the three versions only reflects the number of function calls. In other words, there is no change of the component function of these two versions. In the latest three versions, developers changed the logical contents of some functions or insert new functions into the software to enrich the features of software; on the other hand, the software was simplified or some features were removed to improve the robustness, which results in the ranking variation. For example, in the prior versions node _gnu_flush_read ranked 2nd or 3rd but it ranked 7th and 8th in versions tar-1.25, tar-1.27, and tar-1.28. Table 2 shows the top-10 influential functions of software cflow in different versions. The ranking of some functions in each version of cflow varies but with little range. For example, function print_symbol's ranking ranges from 1 to 2. So we can make a prediction that it may still be more influential than most others in the next new version. Meanwhile, there is no function alloc_cons for the latest versions cflow-1.3 and cflow-1.4 results in the ranking variation. In other words, there is change of the component function of these two versions.

Table 1: Top-10 influential nodes for each version of software tar.

| Function name | V1.21 Rank/value | V1.23 Rank/value | V1.25 Rank/value | V1.27 Rank/value | V1.28 Rank/value |
|---|---|---|---|---|---|
| dump_file0 | 1/3.020 | 1/2.808 | 1/2.804 | 1/2.604 | 1/2.603 |
| flush_archive | 2/2.245 | 5/1.969 | 4/1.949 | 4/1.943 | 4/1.939 |
| _gnu_flush_read | 3/2.212 | 2/2.130 | 7/1.593 | 7/1.592 | 8/1.591 |
| update_archive | 4/2.139 | 4/2.108 | 2/2.270 | 2/2.270 | 2/2.246 |
| to_chars | 5/2.124 | 3/2.124 | 3/2.127 | 3/2.124 | 3/2.124 |
| _gnu_flush_write | 6/1.865 | 6/1.738 | 12/1.152 | 13/1.151 | 14/1.149 |
| start_header | 7/1.672 | 7/1.673 | 6/1.698 | 6/1.674 | 7/1.674 |
| _open_archive | 8/1.589 | 8/1.581 | 8/1.578 | 8/1.578 | 6/1.803 |
| dump_regular_file | 9/1.460 | 9/1.468 | 10/1.305 | 9/1.478 | 9/1.478 |
| find_next_block | 10/1.317 | 13/1.102 | 13/1.105 | 15/1.100 | 15/1.097 |

Table 2: Top-10 influential nodes for each version of software cflow.

| Function name | cflow-1.0 Rank/value | cflow-1.1 Rank/value | cflow-1.2 Rank/value | cflow-1.3 Rank/value | cflow-1.4 Rank/val |
|---|---|---|---|---|---|
| print_symbol | 1/4.195 | 1/4.195 | 1/4.195 | 1/4.196 | 2/4.104 |
| yylex | 2/3.822 | 2/3.822 | 2/3.822 | 2/3.823 | 3/4.074 |
| nexttoken | 3/2.985 | 3/2.985 | 3/2.985 | 3/2.987 | 4/3.334 |
| parse_variable_declaration | 4/2.309 | 4/2.309 | 4/2.309 | 5/2.310 | 10/2.310 |
| parse_dcl | 5/2.200 | 5/2.200 | 5/2.200 | 7/2.200 | 11/2.200 |
| gnu_output_handler | 6/2.068 | 6/2.068 | 6/2.068 | 8/2.068 | 12/2.011 |
| yyrestart | 7/2.042 | 7/2.042 | 7/2.042 | 9/2.050 | 8/2.593 |
| alloc_cons | 8/2.037 | 9/42.010 | 8/2.010 | — | — |
| tree_output | 9/1.985 | 10/1.985 | 9/1.985 | 10/1.993 | 5/3.101 |
| lookup | 10/1.906 | 11/1.892 | 11/1.880 | 11/1.831 | 14/1.822 |

In addition, the number of nodes which have high *NI* is rather small in each version. These high value nodes have taken a great part in ensuring software reliability and stability. It means that there are little functions that should be paid more attention in software updating and software maintenance. We calculate the count for different range of *NI* values. The results of software tar and cflow are shown in Figures 5 and 6, respectively.

As we can see in Figure 5, most of nodes are ordinary functions. We would not pay more attention to them. Meanwhile, a handful of nodes that have high *NI* should be paid more attention. They play important roles in the process of software updating and software maintenance. For cflow, the number of nodes in each scope is shown in Figure 6. It has the same characteristic with tar. The number of nodes with high *NI* is much less than that of low *NI*. By paying more attention to these influential nodes in future versions, we can improve software reliability and stability. Thereby we can greatly reduce the amount of work and improve work efficiency.

At the same time, *NI* of the same ranking nodes within different versions has slight wave, as shown in Figures 7 and 8.

As it is shown in Figure 7, the NI distribution of software tar is similar extremely in the six versions. With the increasing of node ranking, the NI of each node shows a decrease trend.

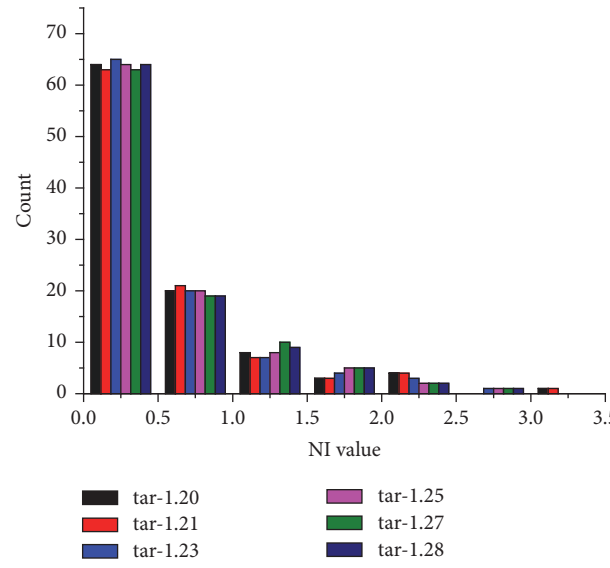



Figure 5: Number of nodes in value scope of tar.

As the lower rank, the value shows a trend of increase. The higher NI ranges from 0.7 to 3.0; most nodes' values are around 0.4. The development of versions follows the same

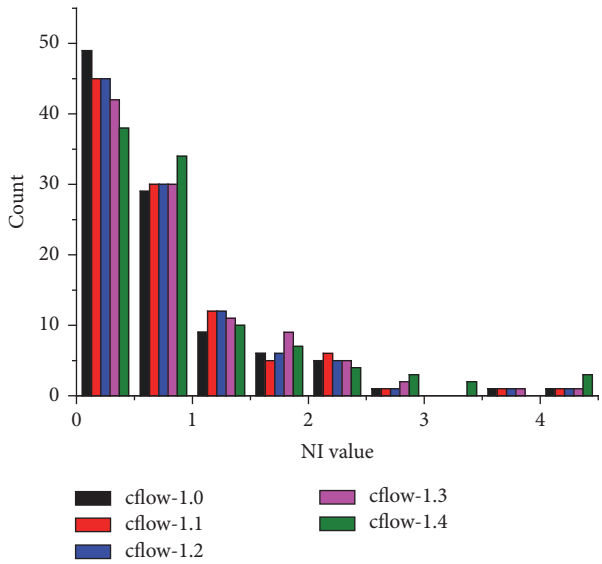



Figure 6: Number of nodes in value scope of cflow.

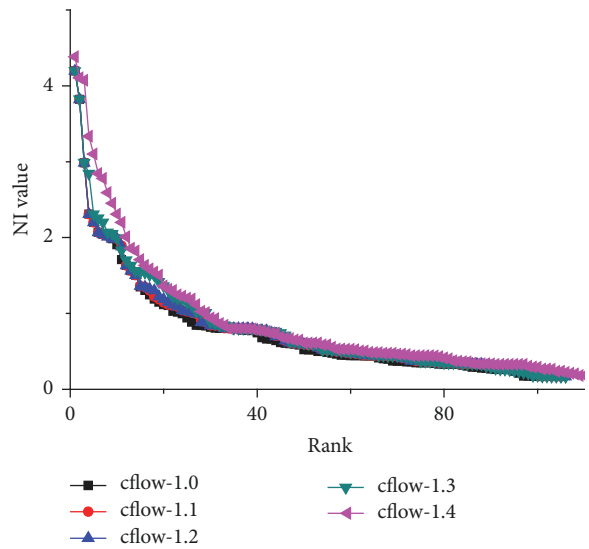



Figure 8: NI distribution of cflow.

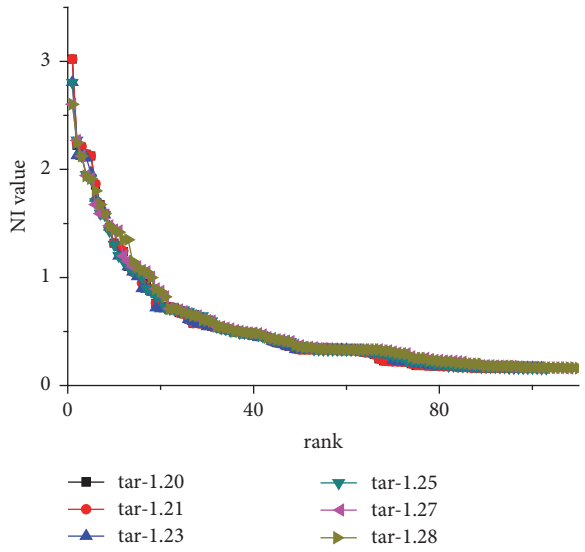



Figure 7: NI distribution of tar.

laws, the NI of a certain ranking remains stable and the NI distribution of different software versions is nearly the same. So, we can predict the future versions' trends based on this. Meanwhile, Figure 8 shows the NI distribution of software cflow, the higher NI ranges from 0.8 to 4.0, and most nodes' values are around 0.5. The curve of each version has the same tendency; namely, the NI distribution of software cflow follows the same trend.

*5.3. Performance Evaluation.* In the study of complex network, we often examine the effectiveness of a method [19, 20] through the analysis of spreading influence about top-$k$ nodes. Therefore, this paper will introduce IC (independent cascade) model. The IC model derived from the SIR (Susceptible-Infected-Recovered) model, the SIR model is a theory about virus spreading and has to be researched widely in complex networks, such as the marketing, advertising, early warning, and social stability. In software engineering, the similar algorithms were used to analyze the change impact [21] and error propagation [22].

The IC model is a probability model; when a node $v$ is activated, it will attempt to activate its inactive outdegree nodes with probability $p$ only once [23]. Whether node $v$ can activate its neighbor nodes successfully, $v$ is still active, but it has no influence later. The communication process is over when there are no influential active nodes in the network, while, in the actual execution process of software, the running fault can affect the other function running due to the invoking relationship. When running fault, all of the invoked functions would affect the normal execution of the parent function. So the faults can widely spread among the function nodes during the running process. So we take IC model as a software attack model to evaluate the effectiveness of our method. A software attack instance is shown in Figure 9.

We assume the node $a$ and node $b$ are attacked as Figure 9(a) shows, and then $a$ and $d$ will attack its inactive outdegree nodes with probability $p$ only once, where $b$ and $c$ are attacked successfully by $a$; meanwhile $e$ and $h$ are attacked successfully by $d$ in Figure 9(b), next $a$ and $d$ have no aggressivity, and the nodes attacked by a and d can attack their inactive outdegree nodes with probability $p$ in the same way. Finally, the number of attacked nodes represents the influence of original attacked nodes.

When calculating the influence of the top-$k$ important nodes obtained by different methods, we will separately run IC model about 10 times and then consider the average of active nodes as the performance evaluation of the method.

The software key entities typically account for a small proportion and only account for one point five percent to two percent in the study of class size [24]. At the same time, it is not acceptable for the cost of checking most of the key entities. So an appropriate number of key entities is needed to be selected. By ranking all functions as descending order

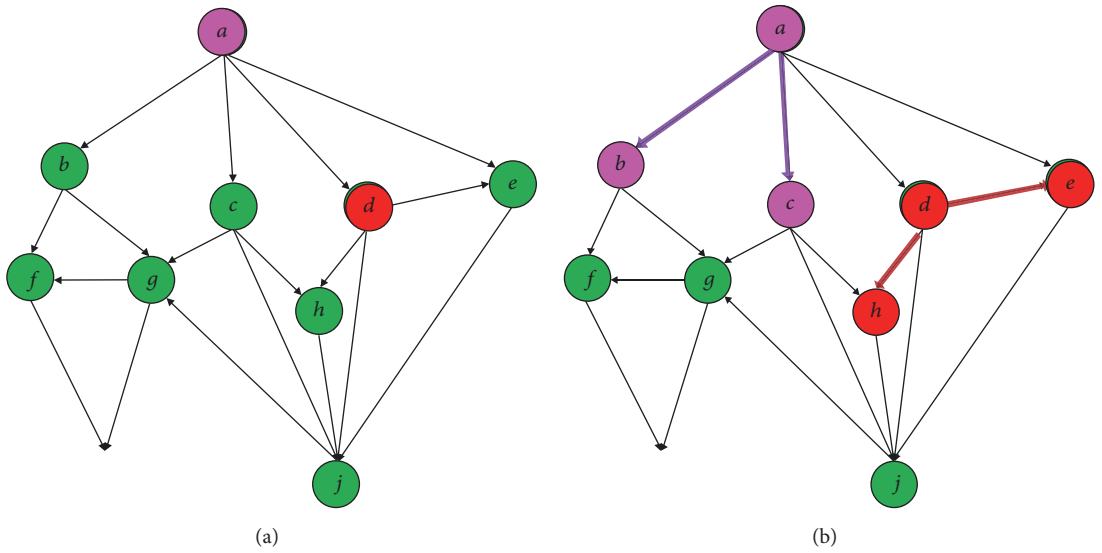



Figure 9: A software attack instance.

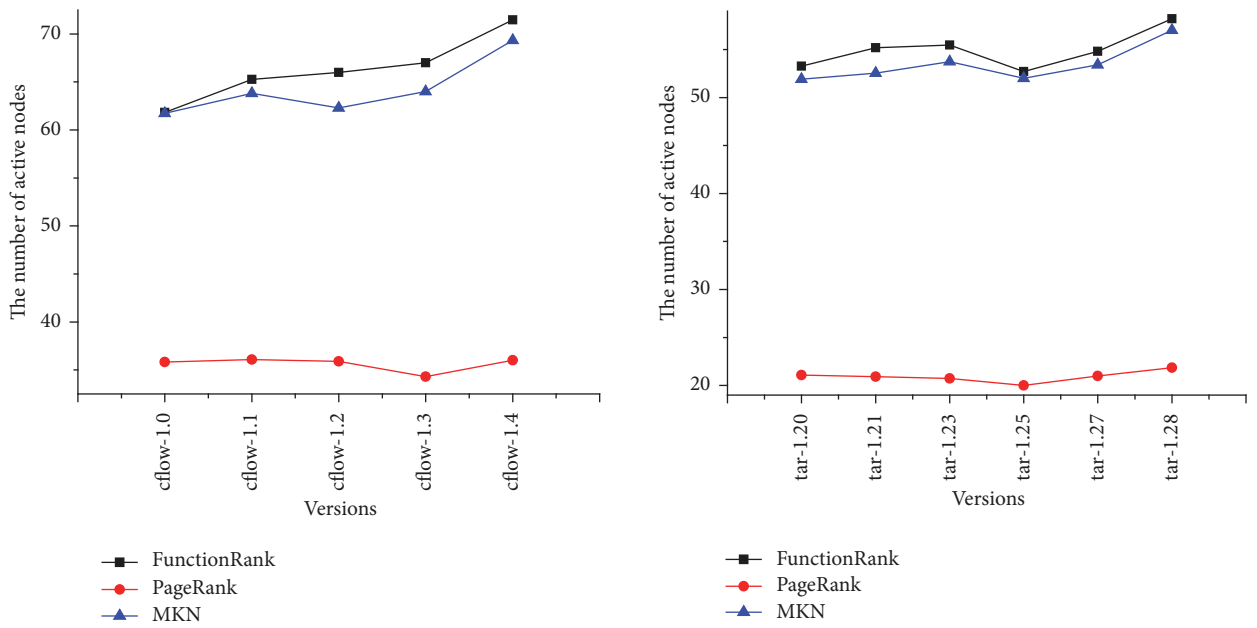



Figure 10: The curves of the number of active nodes.

according to the measurements, we chose little key functions for different systems: top 20 for tar and top 30 for cflow.

Figure 10 shows the average of active nodes for different software versions. In all the different versions of the software systems, key functions identified by NI can activate more nodes than that identified by the method PageRank and MKN [25] as Figure 10 shows. Visibly, compared with another two methods, NI is more effective in the identification of the key functions. The key functions play an important role in software system in terms of reducing the numbers of test data, detecting the vulnerabilities of software structure, and analyzing software reliability, and they should be paid more attention in the process of software updating and software maintenance. Measuring accurately the importance of the node in the software networks is the premise to improve the security and robustness of software. Moreover, with the development of the software, measuring the importance of nodes in the network has practical significance for protecting the influential nodes from deliberate attacks in the software network.

## 6. Conclusions and Future Work

In order to understand and recognize software structure better, a novel method is proposed in this paper to mine the influential nodes in weighted software network. Firstly, taking

into account the invoking times, we construct a directed-weighted network structure to make the understanding and recognization of software structure more accurate. Then, a measurement of NI is put forward to evaluate the node importance, where we provide an idea of importing PageRank and WSN to Software engineering domain. Furthermore, we also consider the outdegree value as a key parameter to the node importance. The outdegree value can reflect the complexity of the node. Finally, the algorithm named FunctionRank is presented to calculate the NI and the change trends of nodes' importance are analyzed by different software versions. In addition, the experimental results show that the proposed feasible approach has good performance in identifying the influential software nodes.

Although the approach we proposed shows some feasibilities in identifying influence nodes in complex software network, the broad validity of our approach should be demonstrated further. Our future work is using more open-source software network to evaluate the validity to improve our approach.

## Conflicts of Interest

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

## Acknowledgments

This work is supported by the National Key R&D Program of China (2016YFB0800700), the National Natural Science Foundation of China under Grants no. 61472341, no. 61572420, and no. 61772449, the Natural Science Foundation of Hebei Province of China under Grants no. F2015203326 and no. F2016203330, and the Advanced Program of Postdoctoral Scientific Research under Grant no. B2017003005.

## References

[1] W.-F. Pan, B. Li, Y.-T. Ma, Y.-Y. Qin, and X.-Y. Zhou, "Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks," *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1202–1213, 2010.

[2] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 419–429, IEEE, Zürich, Switzerland, June 2012.

[3] D. Chen, L. Lü, M. Shang, Y. Zhang, and T. Zhou, "Identifying influential nodes in complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 4, pp. 1777–1787, 2012.

[4] E. Zio, L. R. Golea, and G. Sansavini, "Optimizing protections against cascades in network systems: a modified binary differential evolution algorithm," *Reliability Engineering and System Safety*, vol. 103, pp. 72–83, 2012.

[5] S. M. Chen, X. Q. Zou, H. Lv, and Q. G. Xu, "Research on robustness of interdependent network for suppressing cascading failure," *Acta Physica Sinica*, vol. 63, no. 2, 2014.

[6] M. Kitsak, L. K. Gallos, S. Havlin et al., "Identification of influential spreaders in complex networks," *Nature Physics*, vol. 6, no. 11, pp. 888–893, 2010.

[7] N. Masuda and H. Kori, "Dynamics-based centrality for directed networks," *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, vol. 82, no. 5, Article ID 056107, 2010.

[8] G. Huang, B. Zhang, R. Ren, and J. Ren, "An algorithm to find critical execution paths of software based on complex network," *International Journal of Modern Physics C*, vol. 26, no. 9, Article ID 1550101, 2015.

[9] G. Huang, P. Zhang, B. Zhang, T. Yin, and J. Ren, "The optimal community detection of software based on complex networks," *International Journal of Modern Physics C*, vol. 27, no. 8, Article ID 1650085, 2016.

[10] S. M. Chen, X. Q. Zou, H. Lu, and Q. G. Xu, "Research on robustness of interdependent network for suppressing cascading failure," *Acta Physica Sinica*, 2013.

[11] J. Ma, D. Zeng, and H. Zhao, "Modeling the growth of complex software function dependency networks," *Information Systems Frontiers*, vol. 14, no. 2, pp. 301–315, 2012.

[12] L. Wang, P. Yu, Z. Wang, C. Yang, and Q. Ye, "On the evolution of Linux kernels: a complex network perspective," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 439–458, 2013.

[13] H. Li, H. Zhao, W. Cai, J.-Q. Xu, and J. Ai, "A modular attachment mechanism for software network evolution," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 9, pp. 2025–2037, 2013.

[14] F. Thung, D. Lo, M. H. Osman, and M. R. V. Chaudron, "Condensing class diagrams by analyzing design and network metrics using optimistic classification," in *Proceedings of the 22nd International Conference on Program Comprehension, ICPC '14*, pp. 110–121, ACM, Hyderabad, India, June 2014.

[15] K. Zhuang, H. Shen, and H. Zhang, "User spread influence measurement in microblog," *Multimedia Tools and Applications*, vol. 76, no. 3, pp. 3169–3185, 2017.

[16] X. Zhang, G. Zhao, T. Lv, Y. Yin, and B. Zhang, "Analysis on Key Nodes Behavior for Complex Software Network," in *Information Computing and Applications*, vol. 7473 of *Lecture Notes in Computer Science*, pp. 59–66, Springer, Berlin, Germany, 2012.

[17] D. W. Li, B. Li, P. He, and W. F. Pan, "Ranking the importance of classes via software structural analysis," in *Future Communication, Computing, Control and Management*, vol. 141 of *Lecture Notes in Electrical Engineering*, pp. 441–449, Springer, Berlin, Germany, 2012.

[18] B.-Y. Wang and J.-H. Lü, "Software networks nodes impact analysis of complex software systems," *Journal of Software*, vol. 24, no. 12, pp. 2814–2829, 2013.

[19] V. Colizza, A. Barrat, M. Barthélemy, and A. Vespignani, "The role of the airline transportation network in the prediction and predictability of global epidemics," *Proceedings of the National Acadamy of Sciences of the United States of America*, vol. 103, no. 7, pp. 2015–2020, 2006.

[20] A. Garas, P. Argyrakis, C. Rozenblat, M. Tomassini, and S. Havlin, "Worldwide spreading of economic crisis," *New Journal of Physics* , vol. 12, Article ID 113043, 2010.

[21] L. Zhang, G.-Q. Qian, and L. Li, "Software stability analysis based on change impact simulation," *Chinese Journal of Computers*, vol. 33, no. 3, pp. 440–451, 2010.

[22] W. F. Pan and B. Li, "Software quality measurement based on error propagation analysis in software networks," *Journal of Central South University (Science and Technology)*, vol. 43, no. 11, pp. 4339–4347, 2012.

[23] Y. Zhao, S. Li, and F. Jin, "Identification of influential nodes in social networks with community structure based on label propagation," *Neurocomputing*, vol. 210, pp. 34–44, 2016.

[24] A. Zaidman and S. Demeyer, "Automatic identification of key classes in a software system using webmining techniques," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 6, pp. 387–417, 2008.

[25] G. Huang, P. Zhang, Y. Li, and J. Ren, "Mining the important nodes of software based on complex networks," *ICIC Express Letters*, vol. 9, no. 12, pp. 3263–3268, 2015.