International Journal of Innovative
Computing, Information and Control
Volume 11, Number 2, April 2015

ICIC International ©2015 ISSN 1349-4198
pp. 719–731

MEASURING THE IMPORTANCE OF FUNCTIONS IN SOFTWARE EXECUTION NETWORK BASED ON COMPLEX NETWORK

Haitao He[1,2], Jinxiang Wang[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, Hebei Ave., Qinhuangdao 066004, P. R. China
{ haitao; jdren }@ysu.edu.cn; *Corresponding author: jinxiangwang0522@163.com

Received May 2014; revised September 2014

Abstract. *Accurately measuring the importance of nodes in network to improve software stability and robustness has important significance. Researchers generally define function as a node, relationship of function calls as an edge. Times of function calls can accurately reflect tightness between functions, however, it is being neglected. To consider tightness between functions when software executing, this article proposes a novel method for measuring the importance of functions in software network. We map relations of function calls to software execution directed-weighted networks, and define node measurement score (NMS) and relative node measurement score (RNMS) for measuring the importance of functions. Node_Score algorithm is put forward for accurately calculating NMS and RNMS. In order to guide developer to provide more targeted protection and defence while they develop next version, we predict the importance of nodes in various versions, and process of upgrading of software is tracked and distribution of RNMS in each version is analyzed. Experimental results demonstrate that Node_Score algorithm can accurately measure the importance of each node in software network.*
Keywords: Complex network, Measuring, Function calls, Important nodes

1. **Introduction.** With growing of complexity of computer software system, how to guarantee stable and secure execution of software has become more and more meaningful. Many more important nodes exist in software network, if these nodes are suffered by deliberate attacks, it maybe occur cascading failure [1,2]. Therefore, accurately measuring nodes' importance to improve software stability and robustness is significance.

Mapping software structure to software complex network from different angles and granularity has become an effective analyzing method. Valverde and Sole [3] studied network of software first. Software network was constructed by classes and interaction relationship of classes, class was abstracted as node, and interaction relationship between classes was abstracted as edge. Myers [4] adopted directed network to represent structure of software system and analyzed characteristics of software execution network. Cai and Yin [5] described processes of software execution as an evolving directed topological graph as well as an evolving software mirror graph. Ma et al. [6] abstracted interaction relationship between packages into a software network, and they defined functions in package as nodes and dependencies among functions as edges. However, times of each component called in process of software execution are not same, and tightness between components of software cannot accurately reflect if times of component called are neglected.

Freeman [7] utilized betweenness to measure the importance of node, and nodes with larger betweenness are considered as more important in network. Callaw et al. [8] used node degree to measure the importance of nodes and considered larger degree of node as

key node. Wang [9] proposed degree metric to measure the importance of node in network, namely, if node has large in-degree, node is suggested that it undertakes important task in network. Holme et al. [10] found that removals by the recalculated betweenness and degrees centralities are often more harmful. These metrics strategies of node importance do not take into account global network. Radicchi [11] developed a diffusion algorithm based on recursive technique and tennis contact network was analyzed to rank professional players. Kitsak et al. [12] believed that nodes importance is related to its position in global network, and they adopted k-shell decomposition analysis to obtain ranking index of node importance, and results are more accurate than degree and betweenness. Masuda and Kori [13] extended Laplacian-based centrality and adopted the idea of PageRank to introduce global connectivity between all pairs of nodes with certain strength. Ugander et al. [14] found decisive factor which decides degree of importance of nodes which is not absolute number of neighbor nodes but the number of connected subgraphs between neighbor nodes. Bhattacharya et al. [15] defined a measure called NodeRank that assigns a numerical weight to each node in a graph, to measure relative importance of that node in software. Chen et al. [16] proposed a local ranking algorithm ClusterRank, which takes into account not only the number of neighbors and neighbors' influences, but also clustering coefficient. Study of Xiao [17] showed that incomplete global information has different impacts to an intentional attack in different circumstances, while local information-based attacks can be actually highly efficient. Abdel-Rahim et al. [18] modified Survivable System Analysis (SSA) that can be used to assess security and survivability of critical infrastructure networks to analyze security and survivability of critical infrastructure networks. Because software is composed by logical units – functions and has the relationship of dependent between functions. These previous studies are mainly social network, power network and traffic network. So far these studies, as we know, just only degree, NodeRank method can be used to measure software network nodes' importance. So constructing software execution directed-weighted networks based on granularity of function and considering times functions called is more significative.

Due to failure of function may spread to other functions and lead to functions fault through functions calls in software network, considering tightness between functions in global software executing, in this paper, a novel method is proposed for measuring importance of functions in software network. We define node measurement score (NMS) and relative node measurement score (RNMS) to measure quantitatively functions' importance. To calculate NMS and RNMS, a novel Node_Score algorithm is put forward based on recursive technique. We map the relations of function call in the process of software gzip, tar, cflow executing to software execution directed-weighted network, and analyze RNMS distribution of each version.

The remaining paper is organized as follows. Section 2 gives process of constructing software execution directed-weighted networks. Dentitions and algorithm are given in Section 3. Experiments in Section 4 show performances of algorithm and its applications. Section 5 concludes the paper.

2. Constructing Software Execution Directed-Weighted Networks.

In order to accurately reflect properties of software topological structure and accurately measure important degree of node in software network, in this paper, we propose a model to construct software execution directed-weighted network. The simplified overview of framework is shown in Figure 1.

Under Linux environment, process of constructing software execution directed-weighted adjacency matrix and network visualization is described in detail as follows.

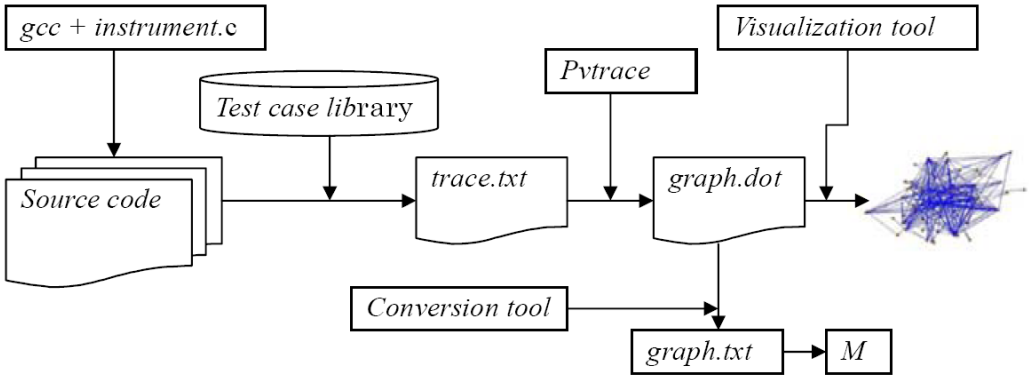

FIGURE 1. Framework of constructing software directed-weighted adjacency matrix and network visualization

1) To generate *trace.txt* file which records addresses of functions, we use test source (*instrument.c*) to track relationship of function calls when software executing.

2) Using tool of *Pvtrace* to analyze *trace.txt* file and generate document of *graph.dot*, which record relationship of function calls.

3) In order to display visualization of calls relationship between functions, we use visual tools (*Gephi*).

4) To obtain *graph.txt*, we develop a tool to convert *graph.dot* to *graph.txt*. Tool named as *Conversion tool* which used *java* to realize development.

5) We analyze document of *graph.txt* and construct software execution directed-weighted adjacency matrix M. Formula as shown in (1), if times of function V_i calls function V_j are n, weight is $w_{ij} = n$, else $w_{ij} = 0$, directed-weighted networks $G(V, E, W)$ is built.

Directed-weighted adjacency matrix:

$$M = \begin{bmatrix} w_{11} & \cdots & w_{1j} & \cdots & w_{1N} \\ \vdots & \ddots & & & \vdots \\ w_{i1} & & w_{ij} & & w_{iN} \\ \vdots & & & \ddots & \vdots \\ w_{N1} & \cdots & w_{Nj} & \cdots & w_{NN} \end{bmatrix} \tag{1}$$

N is the number of nodes in network, and w_{ij} means times of node i calls node j.

Example 2.1. *We use the framework of constructing software execution directed-weighted network to construct directed-weighted network, for example, as shown in Figure 2.*

Utilizing *Conversion tool* realizes process of converting *graph.dot* to *graph.txt*. The *graph.txt* represents information contents of Figure 2, as shown in Table 1. i is starting point of edge, j is terminal, and w_{ij} is the times i calls j. Directed-weighted adjacency matrix which is shown as Table 2 shows information of Figure 2.

3. **Method of Measuring the Importance of Nodes.** This paper creates directed-weighted adjacency matrix M according to framework of constructing software execution directed-weighted network. Node measurement score (NMS), relative measurement score (RNMS), critical node (CN) and key node (KN) are defined. We track the process of software upgrading and analyze RNMS of nodes in various versions, then predict the importance of nodes when software is updated.

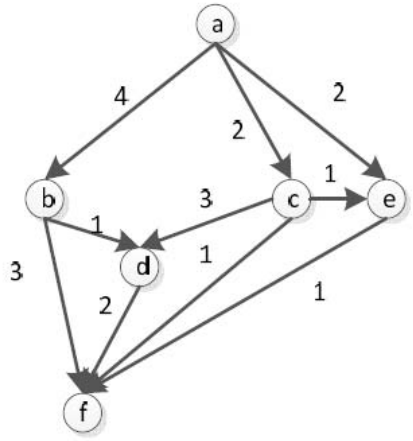

FIGURE 2. Directed-weighted network

TABLE 1. Graph.txt

i	j	w_{ij}
1	2	4
1	3	2
1	5	2
2	4	1
2	6	3
3	4	3
3	5	1
3	6	1
4	6	2
5	6	1

TABLE 2. Directed-weighted adjacency matrix

	a	b	c	d	e	f
a	0	4	2	0	2	0
b	0	0	0	1	0	3
c	0	0	0	3	1	1
d	0	0	0	0	0	2
e	0	0	0	0	0	1
f	0	0	0	0	0	0

3.1. **Definition.** NMS and RNMS are defined to measure the importance of function in software execution network. NMS is calculated by Formula (2) and RNMS by Formula (3).

Definition 3.1. *Node Measurement Score*

$$NMS(i) = (1-d) + d\left(NMS(j) \cdot \sum_{j=1}^{N} C(j)\right) \tag{2}$$

In Formula (2), $C(j) = \frac{w_{ij}}{\sum\limits_{j \neq i} w_{ij}}$.

Definition 3.2. *Relative Node Measurement Score*

$$RNMS(i) = \frac{NMS(i)}{\sum_{j=1}^{N} NMS(j)} \tag{3}$$

Definition 3.3. *Critical Node (**CN**) is node which RNMS equals to median of RNMS of all nodes in global network.*

The median in statistics can avoid extreme data and represent data overall moderate conditions.

Definition 3.4. *Key Node (**KN**) is a node which RNMS is greater than RNMS of CN in software network.*

NMS(i) is measurement score of node i. $C(j)$ equals to dividing times node i calls node j by total number node i called all nodes in network. w_{ij} means times node i calls node j. Many functions in software do not call other functions, for example, in cflow1.0 software, function *register_output*, *parse_rc*, *clear_active*. These functions are named *Dead Ends*. When we construct node measurement score spreading matrix, the *Dead Ends* can lead to denominator zero, so we assume these functions calls one of them one time. RNMS(i) can highlight relative importance of nodes in software network. In order to quantify key nodes, we define KN that RNMS is greater than CN in network. Decay factor is denoted as d with same meaning in PageRank algorithm and its values are ranged from 0 to 1.

Example 3.1. *NMS of each node in Figure 2 is as follows.*

$$NMS(a) = 1 - d.$$

$$NMS(b) = (1-d) + d\left(\frac{1}{2} \cdot NMS(a)\right).$$

$$NMS(c) = (1-d) + d\left(\frac{1}{4} \cdot NMS(a)\right).$$

$$NMS(d) = (1-d) + d\left(\frac{1}{4} \cdot NMS(b) + \frac{3}{4} \cdot NMS(c)\right).$$

$$NMS(e) = (1-d) + d\left(\frac{1}{4} \cdot NMS(a) + \frac{1}{5} \cdot NMS(c)\right).$$

$$NMS(f) = (1-d) + d\left(\frac{3}{4} \cdot NMS(b) + \frac{1}{4} \cdot NMS(c) + NMS(d) + NMS(e)\right).$$

3.2. **Measuring quantitatively the importance of nodes.** The importance of nodes is closely related to other nodes in global network. To calculate node relative measurement score values (RNMS), a novel Node_Score algorithm is put forward based on recursive technique. Primary principle of algorithm is to assign a random value which is ranged from 0 to 1 to each node. With spreading of node measurement score, each node gets NMS, but temporary NMS is constantly updated with constantly spread. When node of NMS is equal or similar before and after recursion, recursion terminates. In process of recursive calculation, each node allocates its current NMS to other nodes according to tightness of nodes, so that, each node obtains corresponding to NMS. For highlighting relative importance of nodes in software executing network, we normalize node measurement score (NMS) to obtain relative node measurement score (RNMS). Node_Score algorithm realizes calculation of RNMS.

Algorithm. *Node_Score*

Input: directed-weighted adjacency matrix M

Output: RNMS

1) for $i = 1 : N$
   if sum$(P(:, i)) = 0$ // if these functions which do not call other functions
   then sum$(P(:, i)) = 1$
   end
   end
2) for $i = 1 : N$
   for $j = 1 : N$
   $P = M * (w_{ij}/\text{sum}(P(i,:)))$ // construct NMS spreading matrix
   end
   end
3) initialize NMS_{init} // assign a random NMS which is ranged from 0 to 1 to each node
4) Define matrix; $A = d * P' + (1 - d) * e * e'/N$ // NMS computational formula
5) NMS = A*NMS_{init}
6) where (|A-NMS| $\geq$temp)
   NMS_{init} = NMS;
   NMS = A*NMS_{init}
   NS = NMS;
   end
7) RNMS(i) = NS(i)/sum(NS)

In Node_Score algorithm, P' is transpose matrix of NMS matrix, e is a column vector, and whose elements are all 1, *temp* is a tiny convergence threshold. Table 3 lists results of important node sorting in Figure 2 under ranking strategy of In-degree, Node-Rank and Node_Score respectively. Node-Rank algorithm calculates node NR value and Node_Score algorithm calculates RNMS. In this article, we set parameters of convergence threshold $temp = 0.001$ and decay factor $d = 0.85$.

Table 3 illustrates that the importance of node pairs d, e and b, c cannot be distinguished if we only consider in-degree of node but ignore its called times and other nodes connected with it. Node-Rank algorithm does not consider called times of node. Then, we cannot distinguish the importance between node b and node c. Based on directed-weighted network Node_Score algorithm calculates NMS of each node in Figure 2. RNMS(a) = 0.0478, RNMS(b) = 0.0866, RNMS(c) = 0.0672, RNMS(d) = 0.1485, RNMS(e) = 0.0890, RNMS(f) = 0.5609. NMS of node f is the highest. It shows that node f is most important in network. Figure 2 can also illustrate the importance of node f, namely nodes b, c, d, e, f directly call it. In-degree of node d is same as e, node d is called by node c for 3 times and node d is called by node e for only 1 time. It means that relative measurement

TABLE 3. Node importance ranking

In-degree	*Node*	*NR*	*Node*	*RNMS*	*Node*
4	*f*	*0.5530*	*f*	*0.5609*	*f*
2	*d*	*0.1439*	*d*	*0.1485*	*d*
2	*e*	*0.1111*	*e*	*0.0890*	*e*
1	*c*	*0.0724*	*c*	*0.0866*	*b*
1	*b*	*0.0724*	*b*	*0.0672*	*c*
0	*a*	*0.0472*	*a*	*0.0478*	*a*

score of node d is greater than one of node e. Because times that node b calls node a is different from that of node c, so NRMS of these nodes are different. Accordingly, Node_Score algorithm considers connection between nodes and also their called times. It can accurately measure the importance of each node in software execution network.

4. **Experiment and Analysis.**

4.1. **Measuring the importance of nodes in software network.**

In our experiment, we choose software in open source software library (http://sourceforge.net/) and download their five versions of program package. There are software gzip (for Linux system file compression), tar (for Linux file decompression) and cflow (for static analysis of C language code) respectively. To obtain relation of function calls when each versions executing, we track process of software executing and map relationship of function calls to software execution directed-weighted networks. Figure 3 shows relationship of function calls in cflow-1.4 software.

As shown in Figure 3, software execution directed-weighted network covers information of relationship of function calls and function calls times when software executing. We build directed-weighted adjacency matrix based on software execution directed-weighted network, which is a good job for using Node_Score algorithm to measure the importance of node in software network.

On the basis of software execution directed-weighted network, we utilize three kinds of ranking strategy of In-degree, Node-Rank and Node_Score respectively. Table 4, Table 5 and Table 6 list top ten important functions after accomplishing a task. When computing RNMS of functions, we assume convergence threshold $temp = 0.001$, decay factor $d =$

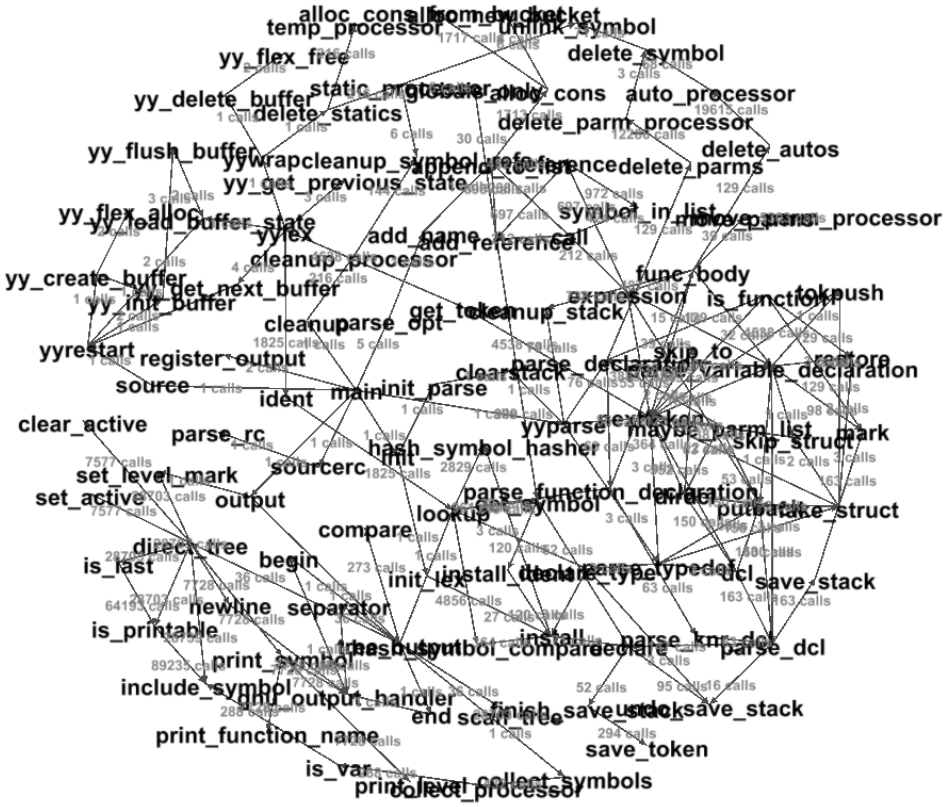

FIGURE 3. Part of relationship of function calls of cflow-1.4 software

0.85. Table 4 shows the detail ranking result of the importance of functions when software gzip-1.5 executing.

Node_Score algorithm can accurately measure relative importance of functions in software gzip execution directed-weighted networks. In our study, we find an important function *pqdownheap* in process of software executing. The function *pqdownheap* combined repeatedly with at least two frequent nodes to build Huffman tree. Nevertheless, previous ranking strategy cannot discover this function. Node_Score algorithm also can improve accuracy. For example, function *read_buffer* in global network is more important than function *write_buffer*, (RNMS(*read_buffer*)>RNMS (*write_buffer*)), but former measurement strategy cannot distinguish the importance of these two functions. In fact, functions *read_buffer* and *write_buffer* respectively undertake important task to read and write cache, what is more, functions *gzip*, *deflate*, *fill_window*, *Im_init* and *updoc* indirectly call function *read_buffer* 1 time, function *file_read* directly calls function *read_buffer* 2 times. The function *read_buffer* is called by function *gzip*, *flush_outbuf* and *write_buf* 1 time. Figure 4 shows that part of function calls of package gzip 1.5 executing.

We also use Node_Score algorithm to evaluate the importance of functions in process of package tar-1.27 and cflow-1.4 executing. As shown in Table 5 and Table 6, we can find some important functions. Function *checkpoint_run* is called 391 times by function *_gnu_flush_read* in software tar. Function *sparse_select_optab* is called 890 times by function *Tar_sparse_init*. We can also obtain the importance of function *lookup* in software cflow which is called by functions *parse_function_declaration* (calls 40 times), *get_symb* (calls 1034 times) and *ident* (calls 1941 times) when software executing. Therefore, Node_Score

TABLE 4. Important function of gzip-1.5 package executing

In-degree	*Function Name*	*NR*	*Function Name*	*RNMS*	*Function Name*
4	*send_bits*	*0.0810*	*send_bits*	*0.0752*	*read_buffer*
2	*gzip_base_name*	*0.0699*	*bi_reverse*	*0.0737*	*write_buffer*
2	*get_stat_mtime*	*0.0634*	*updcrc*	*0.0728*	*send_bits*
2	*gen_codes*	*0.0607*	*read_buffer*	*0.0714*	*strlwr*
2	*bi_reverse*	*0.0607*	*write_buffer*	*0.0496*	*bi_reverse*
2	*updcrc*	*0.0590*	*strlwr*	*0.0485*	*file_read*
2	*init_block*	*0.0550*	*file_read*	*0.0482*	*pqdownheap*
2	*file_read*	*0.0288*	*gen_codes*	*0.0449*	*updcrc*
1	*write_buffer*	*0.0275*	*write_buf*	*0.0316*	*ct_tally*
1	*read_buffer*	*0.0266*	*open_and_stat*	*0.0316*	*write_buf*

TABLE 5. Important function of tar-1.27 package executing network

In-degree	*Function Name*	*NR*	*Function Name*	*RNMS*	*Function Name*
6	*from_header*	*0.0821*	*_gnu_flush_read*	*0.1275*	*checkpoint_run*
4	*assign_string*	*0.0745*	*represent_uintmax*	*0.0806*	*_gnu_flush_read*
3	*tar_stat_destroy*	*0.0724*	*checkpoint_run*	*0.0724*	*represent_uintmax*
3	*find_next_block*	*0.0724*	*short_read*	*0.0458*	*gnu_flush_read*
3	*set_next_block_after*	*0.0468*	*gnu_flush_read*	*0.0410*	*from_header*
3	*xheader_xattr_free*	*0.0423*	*from_header*	*0.0253*	*flush_read*
3	*sparse_member_p*	*0.0259*	*flush_read*	*0.0224*	*assign_string*
3	*chdir_do*	*0.0203*	*tar_sparse_init*	*0.0209*	*sparse_select_optab*
3	*mv_size_left*	*0.0203*	*ptr_align*	*0.0201*	*ptr_align*
3	*set_stat*	*0.0201*	*assign_string*	*0.0189*	*xattrs_print*

TABLE 6. Important function of cflow-1.4 package executing network

In-degree	*Function Name*	*NR*	*Function Name*	*RNC*	*Function Name*
13	*nexttoken*	*0.0459*	*deref_linked_list*	*0.0580*	*lookup*
8	*putback*	*0.0428*	*print_symbol*	*0.0550*	*hash_symbol_compare*
6	*linked_list_append*	*0.0391*	*static_free*	*0.0411*	*hash_symbol_hasher*
5	*lookup*	*0.0346*	*linked_list_append*	*0.0407*	*print_symbol*
5	*gnu_output_handler*	*0.0324*	*yy_load_buffer_state*	*0.0374*	*ident*
4	*mark*	*0.0318*	*gnu_output_handler*	*0.0345*	*deref_linked_list*
3	*install*	*0.0305*	*unlink_symbol*	*0.0318*	*static_free*
3	*yy_load_buffer_state*	*0.0292*	*linked_list_destroy*	*0.0299*	*gnu_output_handler*
3	*tokpush*	*0.0253*	*delete_symbol*	*0.0279*	*nexttoken*
3	*restore*	*0.0228*	*yylex*	*0.0276*	*yylex*

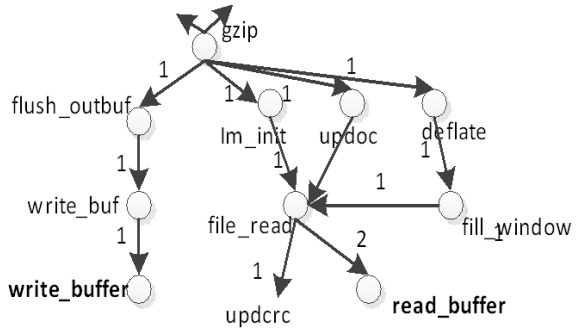

FIGURE 4. Part of function calls for gzip-1.5 package executing

algorithm can measure important node more accurately, it can be applied to improve software robustness and anti-destroying.

4.2. **Applications based on measurement of the importance of node.** In our study, we track the process of different versions of software gzip, tar, cflow executing, and map those processes to software execution directed-weighted network. RNMS of each node in network is calculated. We analyze RNMS distribution of each version and compare RNMS to other versions. Results are shown in Figure 5 and Figure 6 and Figure 7.

From Figure 5, Figure 6 and Figure 7, RNMS mostly distributes in small score area and less nodes have high RNMS. We can find that RNMS distribution varies but little in process of software upgrading.

Changes of the number of key nodes in different versions of software gzip, tar and cflow are shown in Figure 8. From Figure 8, we find the phenomenon that the number of key nodes in different versions is less and the number varies within a small range. The less key nodes may be to ensure better maintain software.

Table 7 and Table 8 list RNMS of functions in software gzip and cflow. Table 7 shows that RNMS of function *write_buffer* is greater than function *write_buf*. Developer defines a new function *write_buffer* in gzip-1.3.14 version. Function *write_buffer* calls function *write_buf* and indirectly calls some functions which are called by function *write_buf*. Because RNMS of functions *read_buffer* and *pqdownheap* are bigger in every version of software network, we can predict that these functions also play more important role in update of versions in the future.

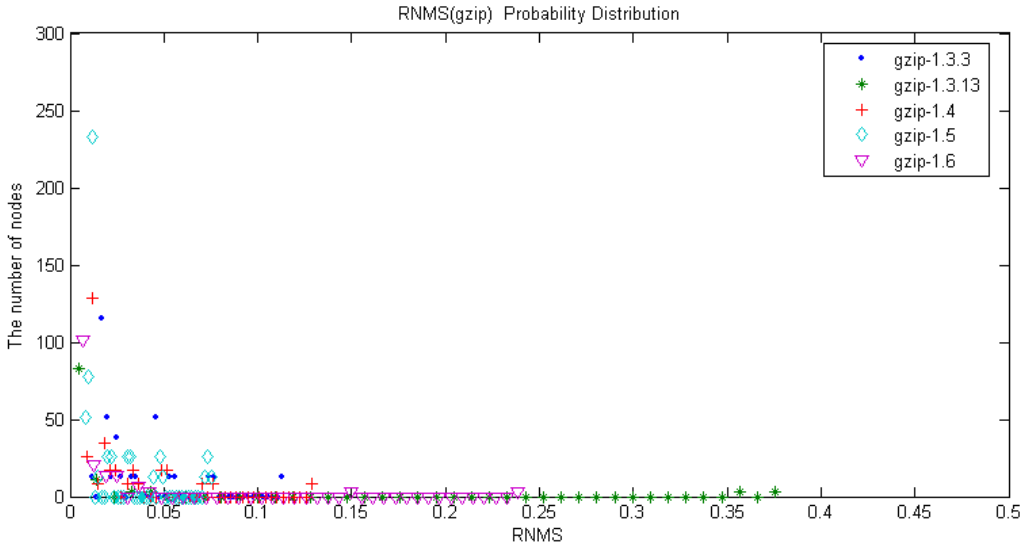

FIGURE 5. RNMS distribution of different versions of gzip

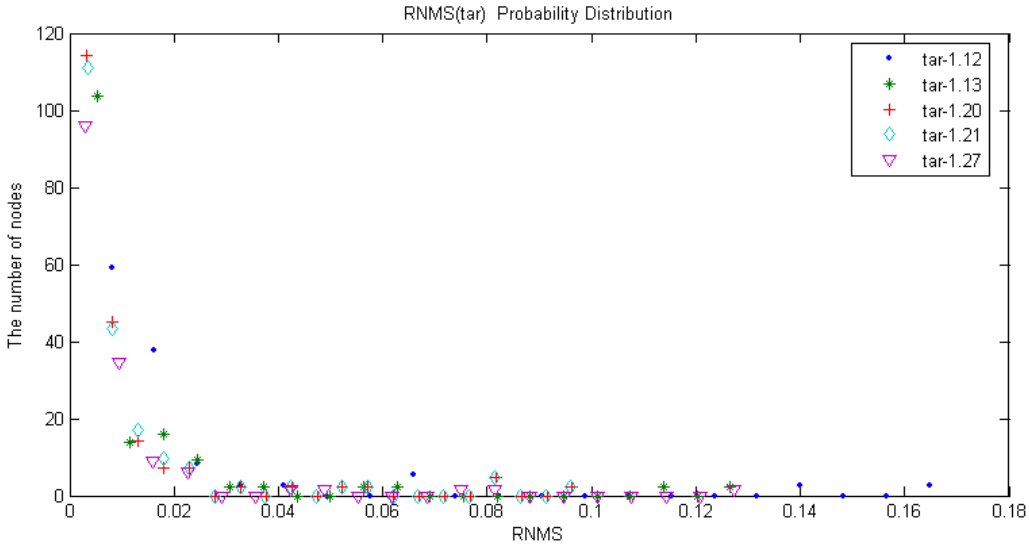

FIGURE 6. RNMS distribution of different versions of tar

TABLE 7. Function of software executing network of each gzip version importance ranking

	gzip-1.3.3	*gzip-1.3.14*	*gzip-1.4*	*gzip-1.5*	*gzip-1.6*
1	*bi_reverse*	*strcat*	*bi_reverse*	*read_buffer*	*read_buffer*
2	*pqdownheap*	*fprint_off*	*write_buffer*	*write_buffer*	*copy_block*
3	*send_bits*	*zip*	*pqdownheap*	*send_bits*	*write_buffer*
4	*build_tree*	*inflate_dynamic*	*send_bits*	*strlwr*	*pqdownheap*
5	*do_stat*	*init_block*	*clear_bufs*	*bi_reverse*	*bi_reverse*
6	*file_read*	*inflate*	*read_buffer*	*file_read*	*display_ratio*
7	*flush_block*	*inflate_block*	*build_tree*	*pqdownheap*	*file_read*
8	*write_buf*	*set_file_type*	*write_buf*	*updcrc*	*ct_tally*
9	*scan_tree*	*updcrc*	*flush_block*	*ct_tally*	*write_buf*
10	*gen_codes*	*scan_tree*	*scan_tree*	*write_buf*	*updcrc*

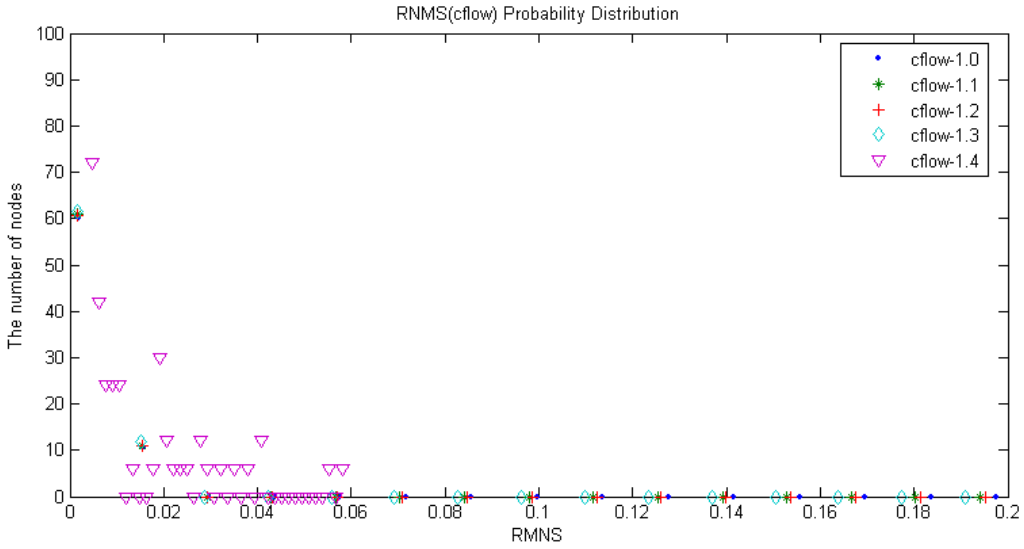

FIGURE 7. RNMS distribution of different versions of cflow

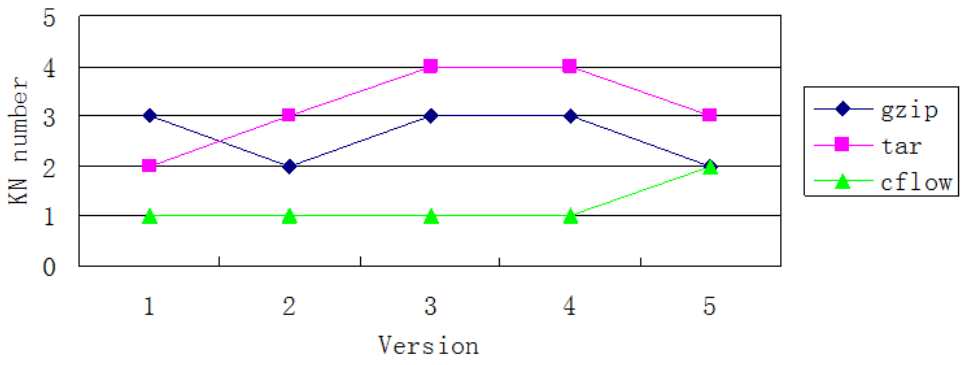

FIGURE 8. The number of key nodes in different versions

Table 8 shows that change of functions' importance is within a small range in each version. Function *scan_tree* calls itself for 36028 times in cflow-1.3, after code is optimized, in cflow-1.4, function *scan_tree* does not call itself, so the importance of function *scan_tree* in network relatively drops in cflow-1.4. RNMS of functions *lookup*, *print_symbol*, *ident*, *gnu_output_handler* and *nexttoken* is bigger in every version. Therefore, we can predict that the importance of these functions will remain in updating of versions. In order to improve software stability and robustness, developer should provide more targeted protection and defence to these important functions during the process of that they develop next version.

5. **Conclusions.** In order to accurately reflect relationship of function calls when software executing, this paper proposed a novel method to measure the importance of functions in software executing network. We map process of software executing to software execution directed-weighted network. Functions are defined as nodes, relationships of calls between functions are abstracted as edges, and times of function calls are set to weight. Considering tightness between functions, we build directed-weighted adjacency matrix M and define node measurement score (NMS) and node relative score (RNMS). Node_score algorithm is presented to calculate RNMS of each node in software executing network, and then important node will be found. Experimental results demonstrate

TABLE 8. Function of software executing network of each cflow version importance ranking

	cflow-1.0	*cflow-1.1*	*cflow-1.2*	*cflow-1.3*	*cflow-1.4*
1	*scan_tree*	*scan_tree*	*scan_tree*	*scan_tree*	*lookup*
2	*lookup*	*lookup*	*lookup*	*lookup*	*hash_symbol_compare*
3	*print_symbol*	*hash_symbol_compare*	*hash_symbol_compare*	*print_symbol*	*hash_symbol_hasher*
4	*nexttoken*	*print_symbol*	*print_symbol*	*hash_symbol_compare*	*print_symbol*
5	*hash_symbol_compare*	*nexttoken*	*nexttoken*	*nexttoken*	*ident*
6	*gnu_output_handler*	*hash_symbol_hasher*	*gnu_output_handler*	*gnu_output_handler*	*deref_linked_list*
7	*hash_symbol_hasher*	*gnu_output_handler*	*hash_symbol_hasher*	*hash_symbol_hasher*	*static_free*
8	*ident*	*ident*	*ident*	*ident*	*gnu_output_handler*
9	*include_symbol*	*unlink_symbol*	*unlink_symbol*	*deref_linked_list*	*nexttoken*
10	*alloc_cons_from_bucket*	*include_symbol*	*include_symbol*	*static_free*	*yylex*

that Node_Score algorithm this article proposed can accurately measure the importance of each node in software network. By calculating RNMS of each node in every version of software, we analyze distributions and changes of nodes RNMS in various versions and in order to guide developer provide more targeted protection and defence to the more important nodes, we predict the important nodes in the network in the process of version upgrade.

Acknowledgment. This work is supported by the National Natural Science Foundation of China under Grant No. 61170190, and the Natural Science Foundation of Hebei Province P. R. China under Grant No. F2012203062, No. F2013203324 and No. F20142031 52. Authors also gratefully acknowledge the helpful comments and suggestions of reviewers, which have improved the presentation.

REFERENCES

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

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

[3] S. Valverde and R. V. Sole, Hierarehieal small worlds in software architecture, *Working Paper of Santa Fe Institute, SFI/03-07-44*, 2003.

[4] C. Myers, Software systems as complex network: Structure, functions, and evolvability of software collaboration graphs, *Physical Review E*, vol.68, no.4, 2003.

[5] K. Y. Cai and B. B. Yin, Software execution processes as an evolving complex network, *Information Sciences*, vol.179, no.12, pp.1903-1928, 2009.

[6] 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.

[7] L. C. Freeman, Centrality in social network: Conceptual clarification, *Social Networks*, vol.1, no.3, pp.225-239, 1979.

[8] D. S. Callaw, M. E. J. Newman and S. H. Strogatez, Network robustness and fragility: Percolation on random graphs, *Physical Review Letters*, vol.85, no.25, pp.5468-5471, 2000.

[9] X. F. Wang, Complex network: Topology, dynamics and synchronization, *International Journal of Bifurcation and Chaos*, vol.12, no.5, pp.885-916, 2002.

[10] P. Holme, B. J. Kim, C. N. Yoon et al., Attack vulnerability of complex networks, *Physical Review E*, vol.65, no.5, 2002.

[11] F. Radicchi, Who is the best player ever? A complex network analysis of the history of professional tennis, *PLoS One*, vol.6, no.2, 2011.

[12] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley and H. A. Makse, Identification of influential spreaders in complex network, *Nat. Phys.*, vol.6, no.11, pp.888-893, 2010.

[13] N. Masuda and H. Kori, Dynamics-based centrality for directed networks, *Physical Review E – Statistical, Nonlinear, and Soft Matter Physics*, vol.82, no.5, 2010.

[14] J. Ugander, L. Backstrom, C. Marlow and J. Kleinberg, Structural diversity in social contagion, *Proc. of the National Academy of Sciences of the United States of America*, vol.109, no.16, pp.5962-5966, 2012.

[15] P. Bhattacharya, M. Iliofotou, I. Neamtiu and M. Faloutsos, Graph-based analysis and prediction for software evolution, *Proc. of International Conference on Software Engineering*, pp.419-429, 2012.

[16] D.-B. Chen, H. Gao, L. Lv and T. Zhou, Identifying influential nodes in large-scale directed networks: The role of clustering, *PLoS One*, vol.8, no.10, 2013.

[17] S. Xiao and G. Xiao, On intentional attacks and protections in complex communication networks, *Global Telecommunications Conference*, pp.1-5, 2006.

[18] A. S. Abdel-Rahim, B. M. Aly Ahmed and Y. Ochoa-Huaman, A qualitative approach to analyze the security and survivability of critical infrastructure networks, *International Journal of Engineering and Technology*, vol.6, no.1, pp.60-64, 2014.