Prediction of the Software Security Defects Based on the Complex Network

Xiao-lin ZHAO\textsuperscript{*}, Yi-man ZHANG, Jing-feng XUE, Ji ZHANG and Quan-bao CHEN

Software of School, Beijing Institute of Technology, Beijing

Keywords: Complex network, Software defects, Defects prediction, KeyNodeRank algorithm.

Abstract. The traditional software defects prediction methods just evaluate the software defects based the unweighted undirected network, which do not reflect the real complex software system. Therefore, this paper proposes a software security defects prediction method based on the complex network. This method improves the PageRank algorithm and develops a KeyNodeRank algorithm based on the complex network. In this method, the software system is divided into different classes and these classes constitute a weighted directed network. In addition, the KeyNodeRank algorithm evaluates and ranks the importance of class nodes in the global network. In order to examine the validity of the software defects prediction method, this paper carries out an experiment. It is found that this method not only is accurate to predict and locate software defects and but also is very significant for improving software quality and maintaining software.

Introduction

Nowadays, the software system is larger and the interaction between the classes of the software is more complicated, which result in a series of software development problems, such as poor stability and poor quality [1]. Therefore, it is very urgent and important to improve the software quality and find out the software defects.

In the last decades, the development of information technology and biology is rapid, so the complex network has gradually become the common focus in many fields [2]. In the complex network, the nodes are huge and the relationship between nodes is very complicated [3]. Besides, the complex network is very different from the random network and the regular network [4]. At present, the complex network is being gradually applied to the software fields, which provides a new method to study the software [5]. Many studies have shown that there is a complex network in the large software systems. In addition, there are many units with different granularity and complex relationship in the large software systems, which can be modeled into a complex network [6].

In many previous studies, the statistical and metrics characteristics of the complex network have been used to predict the software defects. Now, there are mainly three methods based on the complex network [7]. The element level is to measure the degree, average distance and aggregation coefficient of the software [8]. The class level uses the modularity, cohesion and coupling methods to evaluate the software system [9,10]. The network level measures the software defects by degree distribution, average aggregation coefficient, average distance and correlation. However, there are several limits for this three methods [11]. One hand, these methods are based on the undirected network, but the real network is a weighted network. Therefore, these methods can not truly reflect the structure of the software. On the other hand, these methods only use a single indicator without considering the correlation of different indicators, which lost the difference between nodes and the recognition accuracy of the center nodes is also decreased.

This paper focus on the weighted directed networks and builds a network model concerning the direction and weight. In addition, this paper improves the PageRank algorithm and develops the KeyNodeRank algorithm. KeyNodeRank algorithm measures the node importance combining the local attributes and global network. In addition, this paper carries out an experiment and the results show KeyNodeRank algorithm is valid and accurate to evaluate the software.
Directed and Weighted Network Model of Software

Unweighted Network Model

In object-oriented software, the basic unit of the program is a class, which contains member variables and member methods. The exchange of information between the two classes is through the reference of variables and the call of methods. Variables can be called by the methods, so variables and methods can be treated equally. The calls of methods between two classes is more, the relationship between two classes is closer and the weight of the corresponding edge is larger. The relationship between two classes is varied by the call way of the class and the weight can reflect the relationship between the two classes. However, the unweighted network model can not reflect the relationship information of two classes.

Figure 1 indicates the interaction between the members inner the class. Figure 2 shows the dependency relationship between classes. In Figure 2, C1, C2 and C3 are interconnected. If C3 has an error, it will be passed to C2 and C2 will pass the error to C1. However, it is not corresponded to the real situation. The details of classes are shown in Figure 1. For C2, if the variable c2 or the function C2F () is incorrect, it is passed to C1 through the call of function C1F1 () or C1F2 (). For C3, if the variable c3 is incorrect, the error is not passed to class C2 and C1. Obviously, the dependency between C1, C2 and C3 are incomplete. The traditional software network model often exaggerates the wrong communication between classes ignoring the details of the interaction in the class. Therefore, the traditional software network model can not measure the software system completely and effectively. Therefore, it is very important to build a model that can truly reflect the dependency of the software [12].

Definition of Weight

The directed weighted network model is defined as $G=(V,E)$. $V=(V_1, V_2, ..., V_n)$ is a set of nodes in the network and $E=\{e_1, e_2, ..., e_m\} \subseteq V \times V$ stands for the edges between the nodes in the network. $n=|V|$ is the number of the nodes and $m=|E|$ is the number of the edges in the network. Besides, $v_i \in V$, $(i=1, 2, .. n)$ indicates a node in the network, $(v_i, v_j) \in E$ represents a edge from $V_i$ to $V_j$ and $\omega(v_i,v_j)$ shows the weight of $(v_i,v_j) \in E$.

In the software network model, nodes correspond to the classes of the software system, and the edges correspond to the dependency relationship between the two classes. The dependency relationship includes inheritance, implementation and method of calling.
Figure 3 illustrates the weight definition model. In Figure 3, the member includes member variables and member methods and it is same to calculate the weights. It is found that there are three members in class A that depend on class B, funcA1 () and funcA2 are affected only by b1 and funcA3 () is only affected by funcB2 (). Therefore, if funcB1 () has an error and it is not passed to A. Therefore, this paper defines the weight function between two nodes (see function 1).

$$\omega_{ij} = \frac{f_i \times f_j}{f_j}$$

In function 1, f_i is the member number of node i and f_j is the the member number of node j. f_ij represents the member number of node i depending on node j and f_ji represents the member number of node j depending on node i. \(\omega_{ij}\) indicates the weight from node i to node j. In general, \(\omega_{ij}\) is different from \(\omega_{ji}\). In Figure 3, f_ab = 3, f_ba =2, f_a =5, f_b = 3, therefore, \(\omega_{AB} = 3/5 \times 2/5 = 0.4\)

**Evaluation of Software Key Nodes Based on KNR Algorithm**

**KNR Algorithm**

The PageRank algorithm is a tool for evaluating the importance of Web pages and is very important for Google search technology [13,14]. PageRank ranks the pages based on the value of the link. The link from other pages to this page is a vote to this page. The page is more important if has more votes from other pages and the rank value of this page is higher. In fact, in Google search system, the PageRank value of different pages is based on the number of votes, which measures the importance of the page. The PageRank is as follows:

$$PR(A) = (1-\alpha) + \alpha \left( \frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \ldots + \frac{PR(T_n)}{C(T_n)} \right) = (1 - \alpha) + \alpha \sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)}$$

PR (A) represents the PageRank value of page A and T1, T2, ..., Tn is the other page that links to page A. PR (Ti) represents the PageRank value of page Ti, C (Ti) represents the number of pages from page Ti, so the PageRank value of page Ti is passed to that pages that it points to. \(\alpha\) represents a random jump factor, which imitates the user behavior and represents the probability of browsing a page. The probability of staying on the current page is 1-\(\alpha\). In general, the value of \(\alpha\) is 0.85.

KeyNodeRank algorithm improves the PageRank algorithm, which is applied to a weighted network. This algorithm not only considers the direction between nodes, but also takes into account the weight between nodes. Function 2 shows the KeyNodeRank algorithm.

In the directed weighted network, the number of nodes are n. Node v has the links form u1,u2,u3 , Function 3 shows the KNR value of v.
\[
\text{KNR}(v) = \frac{1 - \alpha}{N} + \alpha \left( \sum_{j} \omega_{\mu jv} \text{KNR}(\mu) + \sum_{j} \omega_{\mu jv} \text{KNR}(\mu_j) + \cdots + \sum_{j} \omega_{\mu jv} \text{KNR}(\mu_n) \right) = \frac{1 - \alpha}{N} + \alpha \sum_{j} \omega_{\mu jv} \text{KNR}(\mu_j). \quad (3)
\]

\(\alpha\) is the jump factor, generally, \(\alpha = 0.85\). \(\text{KNR}(\mu_j)\) is the KeyNodeRank value of node \(\mu_j\) that is linked to \(v\). \(\omega_{\mu jv}\) is the weight from node \(\mu_j\) to node \(v\). \(\sum_{j} \omega_{\mu jv}\) is the sum of weight from node \(\mu_j\) to other nodes. \(z_1, z_2, \ldots, z_n\) (including node \(v\)) is pointed by node \(\mu_i\). Therefore, the KeyNodeRank proportion of node \(v\) from \(\mu_i\) is \(\frac{\sum_{j=1}^{n} \omega_{\mu jv}}{\sum_{j=1}^{n} \omega_{\mu jv}}\).

**Steps of KNR Algorithm**

KeyNodeRank algorithm takes the depth analysis of the network topology, which is based on the adjacency matrix of the network. KNR algorithm utilizes the power iteration to evaluate the software system [15]. In specifically, the KeyNodeRank value of each node in the network is calculated and the key nodes are selected according to the rank of KeyNodeRank value. However, the times of iteration is so huge when calculating the KeyNodeRank value. In order to reduce the time complexity, the initial vector of the iteration uses in-degree, which can combine the local attributes and the global network. The process of KNR algorithm is as follows (see Figure 4).

1. Construct the adjacency matrix \(H\). The associated information of each node is stored in \(H\) and \(h_{ij}\) means the weight of the edge that is from node \(i\) to node \(j\).
2. Normalize the adjacency matrix \(H\). For each element \(h_{ij}\) in \(H\), the division operation should be operated that \(h_{ij}\) is divided by the sum of all the elements in the same row of \(H\). After the operation, the matrix \(H\) is transferred to \(\frac{1}{\sum_{j} h_{ij}} h_{ij} = \frac{1}{\sum_{j} h_{ij}} \). In \(M\), the values in a row are all 0, the node is called a dangling node and use the vector \(\left( \frac{1}{n} \right)^T\) to take the place of this node. After that, \(M\) is transferred to the random matrix \(M' = M + \eta \left( \frac{1}{n} \right)^T\). In \(M'\), \(\eta\) is a Boolean vector, 1 is the dangling node and 0 is a normal node.
3. Transpose matrix \(M'\). Matrix \(M'\) transitions to matrix \(P = M'^T\). The KeyNodeRank algorithm concerns the in-degree of the nodes, so this operation is very important.
(4) Calculate matrix A. Matrix A is an abstract of the KeyNodeRank algorithm.

\[ A = \frac{1-\alpha}{N} e \times e^T + \alpha \times P. \]

P is the probability transfer-matrix, \( \alpha \) is a random jump factor and \( n \) is the total number of the nodes in the network. Each element of \( e \times e^T \) is 1. The KNR value of all the nodes is the N-dimensional KNR vector, which is the smooth distribution of the A.

(5) Solve the smooth distribution of matrix A. \( \chi \) is the initial KNR value of a node, so the vector is \( \chi^T = (\chi_0, \chi_1, \chi_2, ..., \chi_n) \). For X, the iterative operation is operated and the result is \( \gamma = AX \). At the same time, the limit value \( e = \tau e^k \) is used to judge the end of iteration. When the result between two iterations is less than \( \varepsilon \), the process is end up. Finally, the smooth matrix \( \gamma \) of the matrix A is obtained, which is the N-dimensional KNR vector in the network.

(6) Rank the nodes with the KNR value. The top 10% among the all nodes in the network is the key nodes.

**KNR Initial Value**

It is need to set the initial vector before the iteration using the algorithm KNR. The initial vector is related to the times of iteration, so it is important to set the proper value. In general, \( \chi = (1/n, 1/n, ..., 1/n) \) is the initial vector and \( n \) is the number of the nodes in the network, which shows that the importance of each node is the same. However, in this paper, the KNR value is more concerned the in-degree of the nodes. Therefore, the in-degree of nodes is used to initialize the vector \( \chi \). The specific steps are as follows.

1. Calculate in-degree of each node in the adjacency matrix. The in-degree of each node is \( S_{i(n)} \). The in-degree of nodes corresponds to the sum of each column in the adjacency matrix.
2. Calculate the sum of the nodes in-degree in the global network.
   \[ \text{sum} = \sum_{i=1}^{n} S_{i(n)} \]
3. Calculate each component of the initial vector. Each component is \( \chi_i = S_{i(n)} / \text{sum} \)

**Design of the Experiment**

In order to examine the validity and accuracy of the software key node measurement based on complex network, an object-oriented open source software JEdit is used. The network architecture diagram of software system is generated at the class level, and then the key nodes value is measured on the basis of the KNR. In addition, the defects of JEdit are collected. Therefore, the defect prediction is verified by calculating the hit ratio of the key nodes to the software defects. In this paper, JEdit5.2.0 and JEdit4.5.1 are used and the defects are collected in the website http://sourceforge.net/p/jedit/bugs/.

Only the software network is a complex network, can the method related to the complex network be used. Therefore, there are two steps in this experiment. On the one hand, whether the built software network is a complex network, which means to the software network has “small-world” and “free-scale” features. On the other hand, KNR algorithm in software defect prediction is examined. The figure 5 shows the specific process of this experiment.

1. Software source code is preprocessed and xml file with dependency relationship is extracted. In this paper, Dependency Finder tool takes some file type as input, such as .jar, .class, .zip, so it is need to preprocess the source code.
2. Extract useful dependency information from the xml file by programming. The xml file contains the dependency information of the package level, class level, and feature level. This paper only extracts the dependencies of the classes and features level.
3. According to the dependency, the weighted network model of software source code is constructed by class and the interaction between classes.
(4) Calculate the KNR values of the nodes in the software network and rank them, choosing the top 10% as the key nodes.
(5) Software defects are predicted by the key nodes and the KNR algorithm in software defect prediction is verified.

![Diagram](image)

Figure 5. Specific process of experiment.

Results and Analysis of the Experiment

Experiment of Software Network Complexity

The source codes of JEdit5.2.0 and JEdit4.5.1 are extracted into complex networks. In the complex networks, classes correspond to the nodes and the interaction between classes corresponds to the edges. The classes that are developed by the developers are considered and the classes in the system are ignored. The results of software network statistics are shown in table 1.

The random network with the same numbers of nodes and edges as the experiment is built. The result shows that the average path length is similar and the average aggregation coefficient of the random network is very different from the experiment network, which indicates the experiment network has the feature of “small-world”. Besides, the power distribution index of the two versions is larger than 3, which shows the experiment network has the feature of “free-scale”. Therefore, the experiment software is a complex network and the method based on complex network is useful.

<table>
<thead>
<tr>
<th>Version</th>
<th>Total nodes numbers N</th>
<th>Total edges numbers M</th>
<th>Average degree k</th>
<th>Average aggregation coefficient C</th>
<th>Random network aggregation coefficient CRand</th>
<th>Average path length L</th>
<th>Random network Lrand</th>
<th>Power distribution index γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.0</td>
<td>1264</td>
<td>4045</td>
<td>3.20</td>
<td>0.538</td>
<td>0.0056</td>
<td>4.203</td>
<td>3.934</td>
<td>3.0761</td>
</tr>
<tr>
<td>4.5.1</td>
<td>1169</td>
<td>3711</td>
<td>3.175</td>
<td>0.545</td>
<td>0.0071</td>
<td>4.368</td>
<td>3.874</td>
<td>3.0542</td>
</tr>
</tbody>
</table>

Table 1. Statistics of JEdit resource code.
Application of KeyNodeRank in Software Defects Prediction

For the prediction of defects using the KNR algorithm, this paper collects the defects of JEdit5.2.0. The defects are distinguished by the key nodes in this experiments, so if the recognition accuracy is relatively high, the KNR algorithm is valid.

Table 2 shows the found defects in this experiment. In the table 2, each defect has a unique id and a short description of the defect. Besides, the class where the defect is located is depicted to examine the accuracy of the KNR algorithm. These defects of JEdit5.2.0 are from the JEdit development site and have been confirmed and accepted. In this paper, the KNR algorithm is use to calculate the key nodes to predict the defects. If these key nodes can correspond the software defects, the algorithm is feasible in software defect prediction.

Table 2. Defects of JEdit5.2.0.

<table>
<thead>
<tr>
<th>Defect ID</th>
<th>Defect description</th>
<th>Defect classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3843</td>
<td>Russian localization breaks Search dialog</td>
<td>SearchMatcher</td>
</tr>
<tr>
<td>#3903</td>
<td>jEdit pops up hidden modal dialogs</td>
<td>OptionsDialog</td>
</tr>
<tr>
<td>#3917</td>
<td>Undetected fold end at end of buffer</td>
<td>Buffer</td>
</tr>
<tr>
<td>#3911</td>
<td>When saving a buffer, the text area scrolls to left</td>
<td>Gutter</td>
</tr>
<tr>
<td>#3872</td>
<td>NPE on startup due to EditPane.loadCaretInfo failing</td>
<td>EditPane</td>
</tr>
<tr>
<td>#3857</td>
<td>Duplicate buffer created after click on hypersearch</td>
<td>HyperSearchResults</td>
</tr>
<tr>
<td>#3882</td>
<td>Regex search and replace with empty replace string eats the line</td>
<td>SearchMatcher$Match</td>
</tr>
<tr>
<td>#3907</td>
<td>jEdit doesn't see if system time zone was changed</td>
<td>StatusWidgetFactory</td>
</tr>
<tr>
<td>#3898</td>
<td>Bracket matching not working on very large files with brackets very far apart</td>
<td>SyntaxStyle</td>
</tr>
<tr>
<td>#3883</td>
<td>Selecting text by mouse sets X primary selection but the keyboard does not</td>
<td>TextAreaPainter</td>
</tr>
<tr>
<td>#3894</td>
<td>EnhancedDialog does not throw NullPointerExceptions anymore when checking whether a pressed key is the buffer close shortcut</td>
<td>EnhancedDialog</td>
</tr>
</tbody>
</table>

Table 3 shows the defect evaluation of JEdit5.2.0 including the name, the KNR value, the ranking of the KNR value of the defect class and the evaluation. It is found that the KNR algorithm can evaluate the most of the defects. Besides, the accuracy of 72.7% illustrates that KNR algorithm is accurate to evaluate the software system. For example, org.gjt.sp.jedit.Buffer class operates when the files are opened in the memory. Form the complex network, this node can be referenced by many other nodes. Therefore, this class is important. In fact, the common functions such as opening files, editing files are related to this class. In addition, KNR algorithm also shows the importance of this class.

Table 3. Defects evaluation of JEdit5.2.0.

<table>
<thead>
<tr>
<th>Defect classes</th>
<th>KNR value</th>
<th>Rank</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchMatcher</td>
<td>4.19E-04</td>
<td>94</td>
<td>Yes</td>
</tr>
<tr>
<td>OptionsDialog</td>
<td>3.34E-04</td>
<td>235</td>
<td>No</td>
</tr>
<tr>
<td>Buffer</td>
<td>5.92E-04</td>
<td>26</td>
<td>Yes</td>
</tr>
<tr>
<td>Gutter</td>
<td>3.62E-04</td>
<td>147</td>
<td>Yes</td>
</tr>
<tr>
<td>EditPane</td>
<td>4.87E-04</td>
<td>72</td>
<td>Yes</td>
</tr>
<tr>
<td>HyperSearchResults</td>
<td>3.87E-04</td>
<td>128</td>
<td>Yes</td>
</tr>
<tr>
<td>SearchMatcher$Match</td>
<td>3.43E-04</td>
<td>159</td>
<td>Yes</td>
</tr>
<tr>
<td>StatusWidgetFactory</td>
<td>3.05E-04</td>
<td>359</td>
<td>No</td>
</tr>
<tr>
<td>SyntaxStyle</td>
<td>2.97E-04</td>
<td>383</td>
<td>No</td>
</tr>
<tr>
<td>TextAreaPainter</td>
<td>3.91E-04</td>
<td>103</td>
<td>Yes</td>
</tr>
<tr>
<td>EnhancedDialog</td>
<td>4.53E-04</td>
<td>84</td>
<td>Yes</td>
</tr>
</tbody>
</table>

However, there are some defects class that can not be found by the KNR algorithm. For example, SyntaxStyle class is about the grammar of the text and is a relatively independent class. KNR algorithm is based on the global network, so the KNR value of SyntaxStyle is relatively low and can not be detected.
Comparison of KNR with Traditional Method

In order to explain the advantages of the method based on the KNR, the traditional method is used to measure JEdit5.2.0 and calculate the hit rate of defect class. In traditional method, LOC is selected. Generally, for CK model, six metrics are selected. However, MOOD model measure the complexity of the software from the system level, which is not comparable with software security defect based on the complex network. For the selected metric, the value of node is calculated and ranked and the top 10% of nodes are selected as the key nodes. Table 4 shows the results of the defects of the software by traditional method.

<table>
<thead>
<tr>
<th>metrics</th>
<th>the number of hitting defective classes</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>3</td>
<td>27.27%</td>
</tr>
<tr>
<td>WMC</td>
<td>4</td>
<td>36.36%</td>
</tr>
<tr>
<td>DIT</td>
<td>5</td>
<td>45.45%</td>
</tr>
<tr>
<td>NOC</td>
<td>5</td>
<td>45.45%</td>
</tr>
<tr>
<td>CBO</td>
<td>6</td>
<td>54.54%</td>
</tr>
<tr>
<td>RFC</td>
<td>4</td>
<td>36.36%</td>
</tr>
<tr>
<td>LCOM</td>
<td>3</td>
<td>27.27%</td>
</tr>
</tbody>
</table>

It is found that the result of traditional method is lower than the software security defect method based on complex network. For LOC, it is so simple to evaluate the software only by the number of codes. For CK, the index only concerns the local attributes. For example, NOC only considers the number of the sub-classes and DIT only concerns the depth of inheritance. Therefore, the traditional methods are not good and accurate to evaluate the software. By contrary, KNR measures the network in the global network, which is valid and accurate to evaluate the software.

Conclusion

Good and pleasing method has been still of great importance for measuring the software defects. Due to the limits of accuracy, there is a need to develop a new model to measure the software. What could be found is the software security defect method based on complex network, which is valid, accurate and reasonable.

In software security defect method based on complex network is introduced and KNR algorithm based on PageRank is developed. KeyNodeRank algorithm calculate the KeyNodeRank value of nodes and find the key nodes in the network. Compared to the traditional methods, this software security defect method not only utilizes the complex network, but also measures the software defects for the global not the local network. In order to examine the validity of the software security defect method based on the complex network, the experiment of JEdit5.2.0 and JEdit4.5.1 is carried. The results show the software security defect method based on the complex network and KNR algorithm can evaluate the software effectively and accurately.

Acknowledgement

This work was supported the National Key R&D Program of China (Grant No. 2016YFB0800700).

References


