Important Usage Paths Selection for GUI Software Testing

Ye Mao, Feng Boqin, Huang Zhenfang and Zhu Li
Department of Computer Science Technology,
Xi’an Jiaotong University, Xianning Road, Xi’an 710049, China
Institute of Software, Xi’an Jiaotong University, Xianning Road,
Xi’an 710049, China

Abstract: From the user’s point of view, the reliability of software depends greatly on the manner in which the software is used. As a result, it is necessary to test the software according to some model that highlights critical usage. Windows Navigation Networks (WNN) were proposed to model the usage of GUI software. Vertexes in the model represent windows and arcs represent transitions between windows. Each transition has probability of occurrence. Algorithm was proposed to obtain the transition probability from software usage log automatically. Important paths can be selected based on the WNN. Existing testing technologies then be used to test the important paths. WNN can describe the usage of GUI software from users’ view and reduce the complexity of modeling GUI software. Important usage paths of GUI software can be got from WNN. It can focus the testing to reveal more important faults.

Key words: Software testing, GUI, WNN model, statistical testing, usage model

INTRODUCTION

GUI (Graphical User Interface) testing is difficult because it has characteristics different from those of traditional software (Memon, 2001). It is graph oriented and event driven. The input space size is extremely large due to the number of different permutations of inputs and events that affect the GUI (White and Almezen, 2000). Testing all these permutations is time and labor consuming. To reduce the number of test cases, different methods were proposed. When FSM was used to model GUI software (Chen and Subramaniam, 2002; Belli, 2001; Shehady and Siewiorek, 1997; Petrenko and Yevtushenko, 2005; Yip and Robson, 1991). Offutt et al. (2003) presented formal testing criteria for generating test inputs, which are transition coverage, full predicate coverage, transition-pair coverage, and complete sequence coverage. When the test cases generated satisfied the criterion, it is no need to generate more test cases. White et al. (2003) can reduced the number of test cases from 50 to 8 by identifying CIS of GUI software. Bogdanov and Holcombe (2004) discovered that the existing state chart testing method requires a rather large test set. By introduction of additional constraints, test set size could be reduced without weakening of the conclusions obtained by testing. Memon et al. (2005) used event-flow graph to represent GUI. Intra-component coverage and inter-component coverage criteria were used to control the test cases generated for testing GUI. Du et al. (2002) used interface component relating chart to model GUI. Vertex coverage, arc coverage, path coverage, and input/output space coverage criteria were presented to control the size of test cases. Nie and Xu (2003) proposed a minimal test suite generation method based on the interrelations among the testing requirements. All the effort above was to reduce the number of test cases and at the same time insure the reliability of the software.

In last years, the software engineering community has turned its attention to statistical software testing (Sayre, 1999; Cristiano et al., 2004; Bjorn et al., 2000). The main idea is that the reliability of software depends greatly on the manner in which the software is used. Not all software failures have an equal impact on user. The likelihood (probability) of encountering the failure has a significant affect on the importance of the failure to the user. As a result, it is necessary to focus testing so as to reveal the more important faults, i.e., test the software according to some model that highlights critical usage. Usage model is important in statistical testing. It characterizes the operational use of a software system. For almost a decade the software engineering community has been using Markov Chain (MC) to describe usage models (Jame and Michael, 1994; Yan et al., 2005; Prowell, 2005; Beyer et al., 2003). Farina et al. (2002) admit the use of Stochastic Automata Networks (SAN) to model applications. SAN formalism has exactly the same application scope as the

Corresponding Author: Ye Mao, Department of Computer Science Technology, Xi’an Jiaotong University, Xi’an 710049, China

6-48
MC. SAN formalism may cope the state space explosion by a modular way to model and an efficient numerical treatment of the infinitesimal generator matrix.

A usage model of Windows Navigation Networks (WNN) was proposed in this study. It models the usage of GUI software from its characteristics. The purpose of this paper is to show WNN can model GUI effectively and can be used to select important usage paths of GUI software for testing. The specific contributions of this work include:

- Proposing WNN to model the use of GUI software.
- Presenting the algorithm to obtain transition distribution of WNN from usage log automatically.
- Selecting important usage paths from WNN for testing.

**PROBLEM DOMAIN**

The field reliability, or reliability of software from the user’s point of view, depends greatly on the manner in which the software is used. For example, consider a system that performs a number of functions correctly and one function incorrectly. If the incorrect function is rarely used few field failures will be observed and the software will have high field reliability. However, if the incorrect function is frequently used, the software will exhibit a large number of field failures and have low field reliability. As a result, it is necessary to focus testing so as to reveal faults on functions that are used frequently. We name these functions as critical usage. The important characteristics of GUI are their graphical orientation that is implemented by windows. Windows supply usage paths for users to accomplish functions.

**Definition:** GUI is a graphical front-end to a software system that accepts as input user-generated and system-generated events and produces deterministic graphical output. GUI contains windows. At any time during the execution of GUI, only one window is active.

We represent windows in GUI as \( W = \{w_i | 1 = i = k\} \) \( k \) is the number of windows in GUI software. It needs to note that windows here include dialog and menu. When user operates GUI software, windows are invoked in sequence. Let it be \( w_1, w_2, \ldots, w_n \). After many users have operated the software, there must be windows sequences that have high occurrence probability. These windows sequences represent the critical usage of GUI software from the views of users. For example, if one sequence is \( w_1, w_2, \ldots, w_n \), another sequence is \( w_1, w_2, w_1, w_2, w_1, w_2 \), the sequence \( w_1, w_2, w_1, w_2 \) has high probability to occur. It is necessary to focus testing so as to reveal the faults in \( w_1, w_2, w_1, w_2 \). But to find the critical usage of GUI software is difficult because the number of possible usage sequences is extremely large due to the different permutations of inputs and events. The purpose of the paper is to find these sequences and focus testing to them.

**WINDOWS NAVIGATION NETWORKS**

In software testing there are several model to describe the usage of software, for example FSM, MC. In the paper, we proposed a model of WNN. It can model the GUI software from its characteristics.

**Definition:** WNN is a structure \((S, T, R)\), where:

\[
S = \{s_i | 0 \leq i \leq n\} \] is a set of states. \( s_i \) is initial state. Application is in state \( s_0 \) before being invoked. \( s_n \) is terminal state. Application is in state \( s_n \) after being terminated.

\[
s_1 (1 \leq i \leq n - 1) \] is the state when window in GUI software is active.

\[
T = \{t_i | 0 \leq i \leq m\} \] is a set of transitions. Each transition is a function \( t_i : S \rightarrow S \). Let \( t_i = \{s_i, s_j\} \), \( s_i \) and \( s_j \) are source state and destination state of the transition \( t_i \), respectively, \( 0 \leq i \leq n \).

\[
R = \{r_i | 0 \leq i \leq m\} \] is a set of transition probability functions, one for each transition. Each function \( r_i : T \rightarrow [0, 1] \) describes the probability of occurrence of the transition.

Usage matrix \( M \) is used to maintained transition probability. \( M[u][v] = R(t)\) if \( t \in T \) and \( M[u][v] = 0 \) if \( t \not\in T \).

As a result, \( M \) is the probability distribution of WNN. \( M \) represents usage profile of GUI software.

Usage path starts from state \( s_0 \) and ends at state \( s_n \). Let \( p[k] \) be the \( k \)th state in path \( P \). For example, in path \( p = s_0 s_1 \ldots s_n \), \( P[1] = s_0 \) and \( P[2] = s_1 \).

**Definition:** Usage path \( P \) is a sequence of states \( s_0, s_1, \ldots, s_n \). Usage path \( P \) is same as \( P \) iff \( \forall k, p[k] = p[k] \). Usage path \( P \) is different from \( P \) iff \( \exists k, p[k] \neq p[k] \).

Usage path can also be regarded as a sequence of transitions because each transition is a function \( t_i : S \rightarrow S \). Assume that the probability of occurrence of every transition in WNN is independent statistically. Let \( t_i \) be transitions in path \( P \).

**Definition:** The occurrence probability of the path \( P \) is

\[
\eta_1 = \prod_{t_i} \eta_i
\]

It means that occurrence probability of a path depends on each transition in the path.
Fig. 1: The example GUI

Fig. 2: WNN model

WNN has been shown to have a number of advantages for modeling GUI software in comparison with FSM and MC. Firstly, the construct of WNN is intuitive because each physical window is represented by one state. Secondly, the number of states in WNN is bounded by \( k + 2 \), where \( k \) is the number of windows in GUI. As a result, the number of states is reduced largely. Thirdly, each transition is associated with a probability, which can reflect the usage frequency as well as the importance of the transition. Finally, the path from the initial state to terminal state can describe a usage of the GUI software.

The benefit of using a WNN can be seen clearly with an example. Consider a simple user interface with five windows. The first window, \( s_0 \), is the main window, where user can choose to go to other windows. \( s_i \) is pull-down menu of file where user can choose to go to either save window (\( s_3 \)) or open window (\( s_7 \)). If user clicks edit menu in the main window, pull-down menu (\( s_j \)) of edit appears. An intuitive diagram of this interface is presented in Fig. 1. Note pull-down menu is treated as a window.

A WNN model of this interface is presented in Fig. 2. Initial and terminal states are \( s_0 \) and \( s_9 \), respectively. State from \( s_i \) to \( s_j \) corresponding to one window in GUI respectively. The transition from \( s_i \) to \( s_j \) means user can either save the data or cancel back to the main window. Transitions of \( s_{a,b} \), \( s_{a-c} \) and \( s_{d,e} \) have similar means.

**OBTAIN SOFTWARE USAGE PROFILE**

Transition probability is needed for generating usage paths statistically. It is obtained from the usage log. Usage log records sequence of the windows operated by user. The usage log has the form as Fig. 3. Technique such as window hook can be used to generate usage log. Alternatively, the codes record the windows usage can be inserted into the debug version of the software to generate usage log.

In the testing process, users are asked to use software. Usage log can be got at the same time. After obtaining the usage log files, we can compute the transition probability automatically. Let FILE be the set of all log files. Procedure CompTranProb was used to compute transition probability of WNN model. The input to procedure is FILE and the output is usage matrix M. The algorithm is as follows. Firstly, compute the frequency of every transition from usage log files (lines 2...13). Secondly, compute transition probability according to frequency (lines 14...28). Variable of freqSum (line 17, 21, 26) is used to maintain the sum of frequency of each transition leading from a state.

PROCEDURE CompTranProb(FILE, M)
Input: FILE;
Output: M;
Begin
1. Initiate M to zero;
2. For All file∈FILE do

Fig. 3: The example of usage log
3. {
4.     w = Get the first window from file;
5.     u = Get the state in WNN corresponding to w;
6.     while(!eof)
7.         {
8.             w = Get the next window from file;
9.             v = Get the state in WNN corresponding to w;
10.            M[u][v] += 1;
11.            u = v;
12.         }
13.     }
14. For All u∈S do
15.     {
16.         E = Get all transitions leading from state u;
17.         freqSum = 0;
18.         For All transition ∈ E do
19.             {
20.                 v = Get destination state of the transition;
21.                 freqSum += M[u][v];
22.             }
23.         For All transition ∈ E do
24.             {
25.                 v = Get destination state of the transition;
27.             }
28.     }
29. End.

SELECTING IMPORTANT USAGE PATHS

Matrix M maintains the probability of each transition. It represents usage profile of GUI software. The next step is to generate usage paths according usage profile and select important ones. Paths are generated by randomly walking in the WNN according to probability distribution. Each path begins from state \( s_i \) and end at the state \( s_o \). The problem is when to stop the paths generation process. We adopted Euclidean distance as stop criterion. Euclidean distance is commonly used to provide an indication of the degree to which the paths generated from the Markov chain matches the probability distribution. It is computed as indication \( \sqrt{\sum_{i,j}(u_{ij} - t_{ij})^2} \)

where, \( u_{ij} \) and \( t_{ij} \) are probabilities of going from state \( i \) to state \( j \) in the usage chain and testing chain, respectively (Sayre, 1999). In our paths generating process, \( u_{ij} \) is equal to \( M[i][j] \). \( t_{ij} \) is the distribution of paths-generating process. It is computed as follows. During paths generating process, let \( t_{ij} \) maintain the frequency that the transition \( t = s_i \rightarrow s_j \) is selected. After process finished, compute the probability that each transition is selected. The algorithm is same as line 14 to 28 of the procedure CompTranProb. Once \( \sqrt{\sum_{i,j}(u_{ij} - t_{ij})^2} \leq \varepsilon \), where \( \varepsilon \) is a threshold, stop path-generating process.

Let \( P \) be the set of all paths generated from initial state to terminal state. Paths in \( P \) are different from each other. Each \( p_i \in P \) has an occurrence probability \( \eta_i \) which has been defined. We define indicator \( \rho_i = \eta_i / \sum_{i=1}^{n} \eta_i \). As a result, \( \sum_{i=1}^{n} \rho_i \) must be equal to 1. We will use the \( \rho \) instead of \( \eta \) as occurrence probability because \( \rho \) has same distribution as \( \eta \) and \( \sum_{i=1}^{n} \rho_i = 1 \).

Occurrence probability \( \rho \) is computed for every path. The paths will be select according to the value of \( \rho \). The paths with large value are importance paths, and then these paths can be tested by existing testing technology (White and Almezeni, 2000; Chen and Subramaniam, 2002; Belli, 2001).

A CASE STUDY

We will show that the method is effective for GUI software. We performed a preliminary case study, which consisted of the following steps:

- Choice an example GUI application;
- Construct WNN for objective application;
- Generate usage log files;
- Compute the transition probability of WNN from the usage log files;
- Generate usage paths from WNN;
- Select important paths according to occurrence probability.

The application we selected was Microsoft WordPad5.0. Figure 4 gives the WNN of the application. It only contains twenty-one windows, which is a part of windows in WordPad5.0. \( s_i \) is initial state and \( s_{21} \) is terminal state. \( s_1 \) is the main window. Every state \( s_i (1 \leq i \leq 21) \) in the WNN corresponds to a window in WordPad. There are totally twenty-three states and thirty-six transitions in WNN.

Usage log files can be got after users use the software in testing process. According to Fig. 4, we generated files manually by walking in the graph from state \( s_1 \) to \( s_{21} \). The number of log files we generated is 100. Figure 5 is a fraction of records from the 99th log file. We can see from the log file that the first state is \( s_1 \).

From the log files, we can compute the probability of every transition in WNN by running the algorithm CompTranProb. The probability of every transition that
Fig. 4: WNN for WordPad5.0

```
//begin
S0
S1
S16
S19
S1
S11
S12
S1
S13
S14
```

Fig. 5: A fraction of log records

results from all the 100 log files is shown in Table 1. The column of ID shows transition number and Probability shows the probability of occurrence. The table describes the usage profile (probability distribution) of the WordPad. We can see that the probability is different from transition to transition.

The next step was to generate all usage paths by probability distribution. Euclidean distance is used in this step as stop criteria. Threshold was set to 0.0008. When Euclidean distance converges to threshold, the process of generating paths was stopped. From Fig. 6 that when path is generated one by one, paths-generating process can converge to usage profile. The general trend of curve is downward because the generating process approach to usage profile statistically. It shows that Euclidean distance is an effective indicator for stop criterion. The number of paths generated was 538.

In this step we selected the important paths according the value of \( \rho \). Before selecting, some path same to other in these 538 paths must be deleted to form the paths set of \( P \). After processing, there were total 520 paths left in \( P \). Table 2 gave some paths from \( P \). The first column is path number. The second column is \( \rho \) and the third column is the path described by state sequence. Paths have been sorted by the value of \( \rho \) from large to small. It shows that some paths have larger value of \( \rho \) than others. The first 15 paths have the value of \( \rho = \sum \rho_i = 0.992811 \). It means that these paths out of all paths have very high probability to occur statistically while other 507 paths just have occurrence probability of \( 1 - \rho = 0.007189 \). It is needed to emphasize again here that

<table>
<thead>
<tr>
<th>ID</th>
<th>Probability</th>
<th>ID</th>
<th>Probability</th>
<th>ID</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0000</td>
<td>12</td>
<td>0.3924</td>
<td>24</td>
<td>1.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.1115</td>
<td>13</td>
<td>0.4522</td>
<td>25</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.1880</td>
<td>14</td>
<td>1.0000</td>
<td>26</td>
<td>0.2051</td>
</tr>
<tr>
<td>3</td>
<td>0.2382</td>
<td>15</td>
<td>1.0000</td>
<td>27</td>
<td>0.0590</td>
</tr>
<tr>
<td>4</td>
<td>0.2209</td>
<td>16</td>
<td>0.2807</td>
<td>28</td>
<td>0.4316</td>
</tr>
<tr>
<td>5</td>
<td>0.1289</td>
<td>17</td>
<td>1.0000</td>
<td>29</td>
<td>0.5604</td>
</tr>
<tr>
<td>6</td>
<td>1.0000</td>
<td>18</td>
<td>1.0000</td>
<td>30</td>
<td>1.0000</td>
</tr>
<tr>
<td>7</td>
<td>1.0000</td>
<td>19</td>
<td>0.1554</td>
<td>31</td>
<td>1.0000</td>
</tr>
<tr>
<td>8</td>
<td>1.0000</td>
<td>20</td>
<td>1.0000</td>
<td>32</td>
<td>1.0000</td>
</tr>
<tr>
<td>9</td>
<td>0.2075</td>
<td>21</td>
<td>0.1234</td>
<td>33</td>
<td>0.1053</td>
</tr>
<tr>
<td>10</td>
<td>0.2146</td>
<td>22</td>
<td>0.9225</td>
<td>34</td>
<td>1.0000</td>
</tr>
<tr>
<td>11</td>
<td>1.0000</td>
<td>23</td>
<td>0.6875</td>
<td>35</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>( \rho )</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>0.558595</td>
<td>S651S1S2S22</td>
</tr>
<tr>
<td>161</td>
<td>0.160248</td>
<td>S651S1S1S1S2S1S2S22</td>
</tr>
<tr>
<td>76</td>
<td>0.062924</td>
<td>S651S1S1S1S1S2S2S2</td>
</tr>
<tr>
<td>314</td>
<td>0.058806</td>
<td>S651S2S2S2S2S2S2S2S2</td>
</tr>
<tr>
<td>165</td>
<td>0.042449</td>
<td>S651S1S1S1S2S2S2S2S2</td>
</tr>
<tr>
<td>137</td>
<td>0.016893</td>
<td>S651S1S1S1S1S2S2S2S2</td>
</tr>
<tr>
<td>151</td>
<td>0.016876</td>
<td>S651S1S1S1S1S2S2S2S2</td>
</tr>
<tr>
<td>491</td>
<td>0.01459</td>
<td>S651S1S1S1S1S1S2S2S2S2</td>
</tr>
<tr>
<td>471</td>
<td>0.014226</td>
<td>S651S1S1S1S1S1S2S2S2S2</td>
</tr>
<tr>
<td>503</td>
<td>0.010563</td>
<td>S651S1S1S1S1S1S1S2S2S2</td>
</tr>
<tr>
<td>384</td>
<td>0.006627</td>
<td>S651S1S1S1S1S1S1S2S2S2</td>
</tr>
<tr>
<td>211</td>
<td>0.006151</td>
<td>S651S1S1S1S1S1S1S2S2S2</td>
</tr>
<tr>
<td>293</td>
<td>0.000489</td>
<td>S651S1S1S1S1S1S1S2S2S2</td>
</tr>
</tbody>
</table>
paths we get may not be real paths but reflect the statistical distribution of real paths. We determined these 13 paths as important paths. Existing testing technique then will be used to test these paths thoroughly.

It is shown from the case study that the important paths in GUI software are not too much but they have more chance to occur than other paths. By testing these important paths, we can insure the reliability of important usage. The cost of testing is not too expensive because important paths are just a little part from all possible paths of the GUI application.

CONCLUSIONS

Testing GUI software has become extremely important as GUIs become increasingly complex and popular. Testing all function is expensive because the input space size is extremely large due to the number of different permutations of inputs and events that affect the GUI. It is necessary to focus testing so as to reveal the more important faults. We construct model for GUI software from users' view. Important usage paths can be selected from the model. Existing technology then can be applied to test these paths. At this point, some preliminary conclusions are available:

- WNN can reduce the complexity of modelling GUI software.
- WNN can describe the usage of GUI software from users' view.
- Important usage paths of GUI software can be easily got from WNN. It can focus the testing to reveal more important faults.
- It is possible to obtain transition probability of WNN from usage log automatically.

The experiment described here is first case study. The target application is relatively simple, and the usage log we used here is generated from simulation. The next steps include: analysing the number of importance paths from the model, obtaining the usage log from real testing environment. Also, we will develop experiments with more complex target applications.

ACKNOWLEDGMENT

This research was supported by the National High-Tech Research and Development Plan of China under Grant No. 2003AA11Z2610.

REFERENCES


