RESEARCH ON CREDIBLE SOFTWARE TEST CASE GENERATION BASED ON BEHAVIOR DECLARATION

Wei ZHUO¹*, Xuejun YU²

In the software testing process, in order to complete the verification of software credibility and improve the efficiency of software testing, this paper proposes a software testing method that combines software credibility with software test case automatic generation technology. First, a credible behavior declaration for the software under test is generated as a standard for verifying software credibility and algorithm initial values. Then, in the selection of test case generation algorithm, this paper proposes an improved particle swarm optimization algorithm (DACPSO). The experimental results show that the automatic generation model of credible test cases based on behavior declaration can verify the credibility of the software, and at the same time improve the efficiency of software testing.

Keywords: software test; software credibility; behavior declaration; particle swarm optimization

1. Introduction

Regarding the research on software credibility, Academician Shen Changxiang of China gave a detailed introduction and analysis of trusted computing in the literature [1]. Guo and others have built a credible framework for testing hardware and software that enables credibility verification of third-party vendor projects [2]. Anurag and others conducted credibility studies on crowdsourcing software development and analyzed the factors and risks that can impact the credibility of crowdsourced software [3]. Wu and others have proposed a framework for assessing the credibility of cloud services, and their decision support can be customized [4]. At the same time, because the scale and complexity of software systems are gradually increasing, we need an efficient and fast testing technology [5]. In the research of automatic generation of test cases, Particle Swarm Optimization (PSO) has the characteristics of fast convergence and strong versatility compared with most evolutionary optimization algorithms [6][7]. However, in practical applications, PSO has problems such as lack of diversity of particles in the late stage of the algorithm, reduced search accuracy,
and poor local search ability. In view of the shortcomings of particle swarm optimization, Micael Couceiro and others introduced the fractional algorithm into the particle swarm optimization algorithm, which improved the local search ability of the algorithm [8]. Li and others made precise adjustments to the fitness value of the particle swarm optimization algorithm, which improved the convergence efficiency of the algorithm [9]. Zheng and others introduced improved genetic algorithms in the particle swarm optimization algorithm, which enabled the population to obtain the characteristics of genetic variation and enhanced the performance of the algorithm [10]. Through the above analysis, this paper proposes an automatic generation method of test cases based on credible behavior declaration.

2. Test case generation model based on behavior declaration

The test case automatic generation model proposed in this study mainly includes three parts, namely test environment construction module based on behavior declaration, algorithm implementation module and test run module, as shown in Fig. 1.

![Fig. 1. Automatic generation model of test cases based on behavior declaration](image-url)
3. Construct a test environment based on behavior declaration

3.1 Credible behavior declaration

A credible behavior declaration describes a collection of all behaviors related to credibility in the software, describing only the expected behavior of the software. The behavior of the software can be described more accurately and more fully through the declaration of credible behavior.

Credible behavior declaration can be defined in a variety of styles for different types and platforms of software. However, the behavior in all styles should include the action name, unique ID, action content, trigger condition, constraint parameters, expected results, and security level. The generic credible behavior declaration structure is shown in Fig. 2 (a).

![Diagram of Credible Behavior Declaration Structure](image)

This paper defines the generic credible behavior declaration by means of XML, as shown in Fig. 2 (b).
3.2 Analysis of behavior declaration

In a behavior declaration, a rule entry represents a constraint on a specific operation, and each rule entry has a security level rule subkey. Validating the software through behavior declaration file ensures that the initial values of the generated algorithms are credible. Through the definition of the behavior declaration, the logical structure analysis of the path of the program under test, you can get the path structure based on the behavior declaration, as shown in Fig. 3.

![Diagram of path structure based on behavior declaration](image)

Fig. 3. Schematic diagram of the path structure based on the behavior declaration
In Fig. 3, 1, 2, ..., S, N represent all path nodes in the program under test, each node has a corresponding rule entry, and the direction of path execution is determined according to the content defined in the rule entry.

### 3.3 Equivalence class

After obtaining the path structure based on the behavior declaration, in order to make the designed test case cover all the paths, the equivalence class is introduced as a parameter filling. This paper implements the equivalence class generation algorithm, which is based on the predicate expression of the branch node and generates the equivalence class through the key sentence coverage criterion. Where \( b \) is the branch predicate, \( b_i \) is the key sentence, \( R_b \) is the predicate clause set, \( K_b \) is the true value set, and \( E_b \) is the generated equivalence class set. The specific algorithm steps are shown in Algorithm 1.

Algorithm 1: Equivalence class generation algorithm

Begin

Input: \( R_b, TK_b, FK_b \)

Set: \( TK_b = \{k|k \in K_b(b(k) = True)\} \), \( FK_b = \{k|k \in K_b(b(k) = False)\} \)

For (\( b \) in predicate set \( R_b \))

For (\( k_1 \) in truth set \( TK_b \), \( k_2 \) in truth set \( FK_b \))

\( i f (b_j \in R_b \land (b_i \neq b_j) \land b(k_1) \neq b(k_2)) \)

\( i f (k_1 \notin E_b) \)

\( k_1 \) add to \( E_b \)

\( i f (k_2 \notin E_b) \)

\( k_2 \) add to \( E_b \)

Output: \( E_b \)

End

According to the above algorithm, taking the geolocation operation as an example, an equivalence class of the path node can be obtained, as shown in Table 1.

<table>
<thead>
<tr>
<th>Number</th>
<th>Rule entry</th>
<th>Security Level</th>
<th>Effective equivalence class</th>
<th>Invalid equivalence class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LocationAccuracy</td>
<td>Suspicious behavior</td>
<td>--</td>
<td>100 meters</td>
</tr>
<tr>
<td>2</td>
<td>LocationFrequency</td>
<td>Safe behavior</td>
<td>No limit</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>LocationCoding</td>
<td>Dangerous behavior</td>
<td>--</td>
<td>Unable to locate</td>
</tr>
</tbody>
</table>
3.4 Fitness function and program instrumentation

The fitness function is the only interface that connects the particle swarm algorithm to the actual problem [11]. Since the branches in the program under test have different coverage difficulty and different test priorities, the branch weight $\omega_i$ is introduced. The fitness function is obtained by the formula (1).

$$fitness = 1/[0.1 + \sum_{i=1}^{N} \omega_i f(i)]^2$$ (1)

In the formula (1), $N$ is the total number of branches, $f(i)$ is the branch distance function of the i-th branch, and $\omega_i$ is the weight of the i-th branch, and $\sum_{i=1}^{N} \omega_i = 1$. The value of branch weight $\omega_i$ is obtained by branch nesting and branch predicate. First, the implementation of branch coverage becomes more and more difficult as the branch nesting level increases, so the branch nesting weight $\omega_1$ is introduced. Let $l_i$ be the level of the current branch, and $l_{max}$ and $l_{min}$ be the largest and smallest branch levels in the tested program. Then use the formula to obtain the branch nesting weight.

$$\omega_1 = 0.417 \cdot e^{\frac{l_i-l_{min}}{l_{max}-l_{min}}}$$

Then, at the branch node, several conditions are connected as branch predicates by the relational operators. Since the weights of the operators are different, the branch predicate weight $\omega_2$ is introduced. Let the weight of the basic condition be $\omega_r$. If the current relational operator is “and”, the predicate weight acquisition formula is

$$\omega_2 = \sqrt{\sum_{i=1}^{N} \omega_r}.$$ If the current relational operator is “or”, the predicate weight acquisition formula is $\omega_2 = min\{\omega_r\}$. Finally, the branch weight of the branch $i$ is calculated by the formula $\omega_i = 0.5 \cdot (\omega_1 + \omega_2)$.

Program instrumentation refers to the collection of dynamic information about program execution by inserting branch functions into the program under test [12]. To perform the instrumentation operation, you first need to specify the information to be obtained and select the insertion position of the branch function. Then insert the branch function in front of the selected branch judgment statement. Finally, insert the fitness function of the current target path into the end of the program.

4. Improvement of test case generation algorithm

4.1 Basic particle swarm optimization

The running process of particle swarm optimization algorithm is as follows, the corresponding flowchart is shown in Fig. 4.

1. Initialize the particle swarm optimization algorithm to randomly initialize the speed and position of each particle in the population;
(2) After the initialization is successful, the fitness of the particles, the individual optimal value $L_i$ and the global optimal value $L_g$ in the population are calculated;

(3) Calculate the latest speed and position of the particles through the speed and position update formula;

(4) Adjust according to the fitness value of the particles. If the fitness value of the selected particle is better than the individual optimal value $L_i$ in the population, the fitness value of the particle is assigned to $L_i$. If the fitness value of the selected particle is better than the global optimal value $L_g$ of the population, then the fitness value of the particle is assigned to $L_g$;

(5) If the number of iterations of the population has reached the maximum value or the optimal value found by the population meets the requirements, step (6) is performed, otherwise step (3) is performed;

(6) End the algorithm and output the global optimal value of the population at this time.

Fig. 4. Basic particle swarm optimization flowchart

### 4.2 Improved particle swarm optimization

For the automatic generation of test cases, although particle swarm optimization has advantages over other optimization algorithms, such as fast convergence and strong versatility. However, particle swarm optimization also has some shortcomings, such as lack of diversity in the late stage of the algorithm, reduced search accuracy, and poor local search capabilities. In view of the shortcomings of particle swarm optimization, this article uses an improved
version of particle swarm optimization algorithm I proposed before, which is based on dynamic adaptive and chaotic search, see the literature [13] for details. The specific flow of the algorithm is shown in the figure below:

![Improved particle swarm optimization flowchart](image)

Fig. 5. Improved particle swarm optimization flowchart

5. Experiment and result analysis

5.1 Experimental purpose and experimental steps

Experimental purpose: In order to verify the credible test case generation model proposed in this paper, the efficiency of software test can be improved while verifying the credibility of the software. In this paper, the iOS application software is used as the program to be tested, and the general function of the tested software is selected as an example to verify the model.
Experimental steps: First step, upload the test program source file and behavior declaration file to the model proposed in this paper. The model analyzes the sequence of program tuning through the behavior declaration file to obtain a collection of sensitive behaviors of each functional module. Then check the behavior name, trigger condition and constraint indicator of the current behavior of each function. Then, in order for the designed test case to cover all the paths, the corresponding equivalence class is populated as a parameter into the current behavior sub-item. Finally, the improved particle swarm algorithm is used to generate the credible test cases automatically, and the test cases are analyzed to obtain experimental conclusions.

5.2 Experimental results and experimental analysis

By analyzing the program under test and selecting its various functional modules as objects to generate credible test cases, the relevant test cases are generated by the model proposed in this paper, as shown below.

<table>
<thead>
<tr>
<th>Serial number</th>
<th>File Type</th>
<th>File Number</th>
<th>File Size</th>
<th>expected results</th>
<th>actual results</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>The most accurate positioning</td>
<td>3</td>
<td>6.7 MB</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>002</td>
<td>Range of kilometer error</td>
<td>1</td>
<td>18.2 MB</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>003</td>
<td>The most accurate positioning</td>
<td>1</td>
<td>52 KB</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>004</td>
<td>Range of hundred meters error</td>
<td>1</td>
<td>2.1 MB</td>
<td>Credible behavior</td>
<td>Consistent with expectations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Location Accuracy</th>
<th>Location Frequency</th>
<th>Location Coding</th>
<th>expected results</th>
<th>actual results</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>The most accurate positioning</td>
<td>400 meters</td>
<td>Unable to locate</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>002</td>
<td>Range of kilometer error</td>
<td>No limit</td>
<td>Positioning successful</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>003</td>
<td>The most accurate positioning</td>
<td>No limit</td>
<td>Positioning successful</td>
<td>Credible behavior</td>
<td>Consistent with expectations</td>
</tr>
<tr>
<td>004</td>
<td>Range of hundred meters error</td>
<td>50 meters</td>
<td>Positioning successful</td>
<td>Not credible behavior</td>
<td>Consistent with expectations</td>
</tr>
</tbody>
</table>

It has been verified by Table 2 and Table 3 that the credible test case automatic generation model proposed in this paper can detect the credibility of the tested software. At the same time, in order to verify that the model improves the efficiency of test case generation, this paper introduces the basic particle swarm
optimization algorithm and genetic algorithm as the comparison algorithm of DACSPSO, and carries out related experiments. The three algorithms are compared and analyzed mainly from the average iteration number of the algorithm and the average iteration time. Taking the test geolocation function as an example, the experimental results obtained are shown in Table 4.

Table 4

<table>
<thead>
<tr>
<th>Number of population</th>
<th>Number of executions</th>
<th>Average number of iterations</th>
<th>Average iteration time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DACSPSO</td>
<td>PSO</td>
<td>GA</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>70.2</td>
<td>113.7</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>73.6</td>
<td>105.8</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>65.5</td>
<td>102.6</td>
</tr>
<tr>
<td>150</td>
<td>10</td>
<td>64.1</td>
<td>123.5</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>75.6</td>
<td>107.3</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>72.9</td>
<td>103.1</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>74.5</td>
<td>116.3</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>66.2</td>
<td>112.9</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>71.3</td>
<td>124.2</td>
</tr>
</tbody>
</table>

From Table 4, it can be found that the average number of iterations and the average iteration time of the DACSPSO-based test case generation model are less than PSO and GA when the number of populations and the number of algorithm executions are the same, which indicates the search speed of DACSPSO is more excellent. At the same time, compared with the comparison algorithm, DACSPSO has stronger stability because of its effective regulation ability.

Finally, in order to fully prove the efficiency of DASPSO here, some common functions are selected for performance testing. The advantages and disadvantages of the algorithm can be judged by the process of searching the extremum of each function. The control algorithms selected in the experiment are DNSPSO [14], OLPSO [15], and PSO. The population size M is 20, the particle dimension D is 30, and the maximum iteration number of the algorithm is 1500. Perform 100 searches and calculate the average and standard deviation of the optimal fitness value. The specific experimental results are shown in Table 5.

Table 5

<table>
<thead>
<tr>
<th>Test function</th>
<th>Sphere</th>
<th>Ackley</th>
<th>Rosenbrock</th>
<th>Rastrigrin</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>Average fitness</td>
<td>2.41E-04</td>
<td>5.91E-02</td>
<td>2.57E+04</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>1.72E-09</td>
<td>3.82E-01</td>
<td>1.74E+05</td>
</tr>
<tr>
<td>OLPSO</td>
<td>Average fitness</td>
<td>6.25E-09</td>
<td>7.13E-09</td>
<td>5.23E+02</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>9.74E-02</td>
<td>5.36E-07</td>
<td>8.41E+03</td>
</tr>
</tbody>
</table>
Research on credible software test case generation based on behavior declaration

It can be concluded from Table 5 that the PSO algorithm has the largest average fitness value, OLPSO and DNSPSO are smaller than PSO, and the average value of DASPSO is the smallest, indicating that the convergence accuracy of DASPSO is the highest. At the same time, the standard deviation of DASPSO is the smallest among these algorithms, indicating that the stability and robustness of the algorithm are optimal. In summary, the proposed automatic generation model of credible test cases based on behavior declaration can verify the credibility of the software and improve the efficiency of software testing.

6. Conclusions

For software testing, this paper proposes an automatic generation model of test case based on credible behavior declaration. First, generate a credible behavior declaration for the software under test and build a test environment based on the behavior declaration. Then the test case generation algorithm is improved, and finally the test case generation model is realized. The experimental results show that the model can improve the testing efficiency of the software while verifying the credibility of the software.

Acknowledgement

The paper was supported by National Key Research and Development Plan of China (2017YFF0211801).

REFERENCES


