Research of Automatic Test Case Generation Algorithm Based on Improved Particle Swarm Optimization

Weiwei WU
China Software Testing Center, Beijing, 100048, China
email: wuweiwei_www@sina.com

Keywords: Test case; Particle swarm optimization; Neighbor; Software test; Morlet.

Abstract. The software testing is an important way to find bugs, and guarantee the quality and reliability of software. Automatic software case generation can effectively improve test efficiency, reduce test time and cost of development, so it has been widely concerned. Aiming at premature convergence and local optimum problems of automatic software case generation based on particle swarm optimization algorithm, an automatic test case generation algorithm based on improved position and particle swarm optimization is proposed. The proposed algorithm can effectively solve the premature convergence problem by dynamically adjusting the inertia factor and Morlet variation to change the position of particles. Meanwhile, neighbor position information is used to solve the locally optimum problem. Simulation demonstrates that compared with genetic algorithm, artificial immune algorithm and standard particle swarm optimization algorithm, the proposed algorithm is the best in the term of iterations and overhead time.

Introduction

Design defect and code error seriously affect the reliability and availability of software. Software testing is widely focused as a method for discovering software problems [1]. Wherein, automatic generation of test cases not only can greatly reduce the workload of software testing, but also can improve the test efficiency. Therefore, it has become one of the research hotspots in software testing [2-3].

Static test case generation method wastes time and labor on one hand, it also has the hidden danger of test case deficiency on the other hand. Meanwhile, its robustness is poor aiming at diversified test target [4-5]. Currently, test case generation is firstly regarded as function minimization issue under general condition according to the above problems. Then, immune algorithm, genetic algorithm and other heuristic algorithms are utilized for optimization and solution. However, such heuristic algorithms still has problems of high complexity, low convergence, low efficiency, etc. [6]. Simple and efficient PSO (Particle Swarm Optimization) algorithm can automatically produce test cases according to interaction between groups and individual in particle swarm with better guidance and convergence, thereby it has become a new research hotspot. However, standard PSO algorithm has the problems of premature convergence and local optimum [7-8]. In the paper, Automatic Test Case Generation Algorithm Based on an Improved Particle Swarm Optimization (ATCGIPSO) is proposed in order to solve these problems. The problem of premature convergence is effectively inhibited by the algorithm through dynamically adjusting inertia factor in standard PSO algorithm and use of Morlet variation. Meanwhile, neighbor position information is introduced to solve local optimum problem.

Particle swarm optimization algorithm

PSO algorithm is an optimization method which was proposed by Kennedy and others in 1995. A group of particles were firstly selected in the algorithm as initial population. In addition, each particle in the population is regarded as a feasible solution of optimum problem. Then, each particle can move to the direction of optimal particle in feasible solution space. Finally, the optimal solution can be discovered through many iterations [9].
Its mathematical model is shown as follows: \((P): \min \left\{ f(x) : x \in \Omega \subseteq R^n \right\}, \quad f : \Omega \subseteq R^n \rightarrow R^i \)
for the global optimization problem, the population is a collection of multiple feasible solutions of \((P)\). Members of the population are called particles. The number of particles is called scale. It is assumed that the population scale is \(m\) in \(n\)-dimensional space. Then the speed and position of the \(i\)th particle are respectively \(V_i = (v_{i1}, v_{i2}, \cdots, v_{inn})^T\), \(X_i = (x_{i1}, x_{i2}, \cdots, x_{inn})^T \in \Omega\). In the particle movement process, \(P_i = (p_{i1}, p_{i2}, \cdots, p_{inn})^T\) is used for recording the optimal position passed by \(i\) particles. Then, the optimal positions passed by all particles are \(P = (p_{g1}, p_{g2}, \cdots, p_{gmn})^T\). Each particle can update own position and speed according to formula (1) and formula (2).

\[
\begin{align*}
\dot{x}_{id}^{(t+1)} &= w^{(t)}\dot{x}_{id}^{(t)} + c_1r_{1id}(p_{id}^{(t)} - x_{id}^{(t)}) + c_2r_{2id}(p_{g_{id}}^{(t)} - x_{id}^{(t)}) \\
\dot{v}_{id}^{(t+1)} &= \dot{x}_{id}^{(t)} + v_{id}^{(t)}
\end{align*}
\]

Wherein, \(i = 1, 2, \cdots, m\), \(d = 1, 2, \cdots, n\), \(r_{1id}\) and \(r_{2id}\) are random numbers subject to \(U(0,1)\) distribution. \(c_1\) and \(c_2\) are non-negative constant learning factors. \(w\) is inertia factor.

**Test case automatic generation algorithm based on improved particle swarm optimization**

**Test case generation model**

Figure 1 shows ATCGIPSO test case generation model. It is obvious in Figure 1 that the model consists of three parts:

1. Test environment generation module: the module is used for static analysis of source program, thereby obtaining initial parameters of the algorithm module. Meanwhile, function instrumentation is also implemented on the source program for testing operation module.
2. Testing operation module: in the module, the tested program after instrumentation operates test cases provided by algorithm module, and the test results are sent to algorithm module for the algorithm module to calculate the fitness of corresponding particles.
3. Algorithm module: first of all, the module can generate parameter generation initial population provided by the module according to test environment. Next, the generated test cases are transmitted to test operation module. The fitness of each particle is calculated according to feedback information of the test operation module. Individuals are evaluated for judging whether the search is finished or not. If it is finished, results can be directly output, and otherwise the particle position and speed are adjusted for generating population in next generation.

![Figure 1 ATCGIPSO test case generation model diagram](image-url)
**Improved particle swarm optimization algorithm**

The Improved Particle Swarm Optimization (IPSO) algorithm proposed in the paper improves the standard PSO algorithm from two aspects aiming at premature convergence and local optimum problems in standard PSO algorithm.

First of all, IPSO algorithm can dynamically adjusts inertia factor $w$, and the adjustment formula is shown as follows:

$$
\omega_{i}^{(t+1)} = \omega_{i}^{(t)} \times (1 - (dist_{i}^{(t)} / dist_{\max}^{(t)}))
$$

(3)

Wherein, $dist_{i}^{(t)}$ represents Euclidean distance of particle $i$ and the global optimal value in generation $t$, namely $dist_{i}^{(t)} = \left( \sum_{j=1}^{n} \left( p_{g}^{(t)} - x_{j}^{(t)} \right) \right)^{1/2}$, $dist_{\max}^{(t)} = \max(dist_{1}^{(t)}, dist_{2}^{(t)}, \cdots, dist_{N}^{(t)})$. It can ensure that the particle is always located around the global optimal solution, thereby effectively inhibiting the occurrence of premature convergence. Meanwhile, the particles can undergo Morlet variation according to probability $p_{m} \in [0, 1]$ in order to further overcome premature convergence. The particle after variation is shown as follows:

$$
mut(x_{i}^{(t)}) = \begin{cases} 
  x_{i}^{(t)} + \sigma \times (x_{\max}^{(t)} - x_{i}^{(t)}), \sigma > 0 \\
  x_{i}^{(t)} + \sigma \times (x_{i}^{(t)} - x_{\min}^{(t)}), \sigma \leq 0
\end{cases}
$$

(4)

In the formula, $mut(x_{i}(t))$ is compiled $x_{i,j}(t)$, $x_{\min}$ and $x_{\max}$ are respectively minimum value of $x$ and the minimum value, wherein $\sigma$ is calculated through formula (5):

$$
\sigma = \frac{1}{\sqrt{a}} e^{-\frac{\phi^2}{2}} \cos(5(\phi^2))
$$

(5)

Wherein: $\phi \in [-2.5a, 2.5a]$.

$$
a = e^{-\ln(g) \times (1 - \frac{t}{t_{\max}})^{\zeta_{wm}} + \ln(g)}
$$

(6)

Wherein, $g$ is upper boundary of $a$, $\zeta_{wm}$ is shape parameter of monotone increasing equation. $t$ is current iteration frequency, and $t_{\max}$ is upper limit of iteration frequency.

Secondly, when IPSO algorithm is used for speed updating, optimal neighbor location information is introduced, then formula (1) is changed into follows:

$$
v_{i}^{(t+1)} = v_{i}^{(t)} + c_{1}rand_{1}(p_{g}^{(t)} - x_{i}^{(t)}) + c_{2}rand_{2}(p_{g}^{(t)} - x_{i}^{(t)}) + c_{3}rand_{3}(p_{g}^{(t)} - x_{i}^{(t)})
$$

(7)

Wherein, $rand_{1}$ is a random number subject to $U(0,1)$ distribution, $c_{3}$ is non-negative constant learning factor. Particle $k$ is a particle with the optimal fitness value in neighbors of particle $i$. Other parameter meanings are the same as formula (1).

**Workflow**

Figure 1 shows that the algorithm module is related to calculation of fitness value. Namely, test cases corresponding to particles with higher fitness value can cover more paths, thereby more software defects can be discovered, thereby improving testing efficiency. The fitness value is determined by fitness value function. Fitness value function determines the speed of ATCGIPSO algorithm in searching test data. Excellent fitness function can make each particle in the population to approach to the optimal solution space more rapidly. In the paper, branch function superposition method is used for generating the fitness value function.

The fitness value function obtained by utilizing branch function superposition method is shown as follows:

$$
F = C - \left( F(f_{1}) + F(f_{2}) + \cdots + F(f_{m}) \right)
$$

(8)

$$
F(x) = \begin{cases} 
  0, x \leq 0 \\
  x, x > 0
\end{cases}
$$

(9)

In the formula, $f_{i} = f_{i}(x_{1}, x_{2}, \cdots, x_{n})$ is branch function, $i = 1, 2, \cdots, m$. $M$ represents the score
point on the to-be-tested path. C is a positive integer, which meets
\[ C - (F(f_1) + F(f_2) + \cdots + F(f_m)) \geq 0. \]

ATCGIPSO workflow is shown as follows according to Figure 1:

1. Generate the fitness function, and implement static analysis and instrumentation on source program;
2. Determine the threshold of fitness value, \( c_1, c_2, c_3 \), rand1, rand2, rand2, w and maximum circulation frequency \( T \) according to the parameters obtained by static analysis;
3. Generate to-be-tested program according to source program after instrumentation;
4. Generate population;
5. Implement test cases corresponding to each particle by to-be-tested program, and utilize formula (8) and formula (9) for calculating corresponding fitness value according to implementation results;
6. Finish circulation and output results if the fitness value is greater than the threshold, otherwise, judge whether the circulation frequency is higher than \( T \) or not; finish circulation and output results if it is higher than \( T \);
7. Update speed and position of all particles according to the formulas (2), (3), (4) and (7), and jump to step (4);

Simulation experiments

Because the triangle type judgment program has clear logic and moderate complexity, it has become one of software testing benchmark programs [10]. In the paper, triangle type judgment program is adopted as an example, performance of ATCGIPSO, PSO, artificial immune algorithm and genetic algorithm is compared from two aspects of time expense and iteration frequency.

<table>
<thead>
<tr>
<th>( m )</th>
<th>Artificial immune algorithm</th>
<th>Genetic algorithm</th>
<th>PSO</th>
<th>ATCGIPSO</th>
<th>Artificial immune algorithm</th>
<th>Genetic algorithm</th>
<th>PSO</th>
<th>ATCGIPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>186.2</td>
<td>324.8</td>
<td>110.4</td>
<td>20.5</td>
<td>123.1</td>
<td>262.2</td>
<td>96.9</td>
<td>24.6</td>
</tr>
<tr>
<td>20</td>
<td>163.7</td>
<td>287.6</td>
<td>101.7</td>
<td>19.7</td>
<td>111.5</td>
<td>238.7</td>
<td>87.5</td>
<td>20.9</td>
</tr>
<tr>
<td>30</td>
<td>147.3</td>
<td>249.3</td>
<td>89.2</td>
<td>18.5</td>
<td>87.5</td>
<td>161.4</td>
<td>76.2</td>
<td>16.7</td>
</tr>
<tr>
<td>40</td>
<td>154.6</td>
<td>258.6</td>
<td>94.3</td>
<td>19.2</td>
<td>82.7</td>
<td>153.8</td>
<td>69.5</td>
<td>12.5</td>
</tr>
<tr>
<td>50</td>
<td>132.9</td>
<td>228.5</td>
<td>81.5</td>
<td>17.4</td>
<td>72.8</td>
<td>117.9</td>
<td>60.2</td>
<td>9.9</td>
</tr>
</tbody>
</table>

After the experiment is repeated for 100 times, the average value of time expense and iteration frequency of four methods is shown in Table 1. Table 1 shows that the iteration frequency required by four methods is smaller with increase of population scale \( m \) because population diversity can be improved by \( m \) increase, thereby discovering the required test cases more easily. However, time expense of each iteration time can be increase with \( m \) increase. For example, the expense of four methods at the time of \( m=30 \) is larger than that at the time \( m = 40 \) in table 1, therefore when \( m \) is selected, time expense and iteration frequency should be comprehensively considered. ATCGIPSO time expense and iteration frequency are the optimal according to table 1 because ATCGIPSO not only uses global optimal position to adjust the current particle position, but also uses neighbor optimal position to guide the particles to rapidly approach to the particle with the highest fitness value, thereby greatly improving the convergence speed of IPSO algorithm. Meanwhile, ATCGIPSO dynamically adjusts inertia factor \( w \) and adjusts the position of the particles through Morlet mutation, thereby avoiding the PSO algorithm from being caught in the local optimum, and realizing improvement of test case generation efficiency.
Conclusion

Since test case automatic generation algorithm based on standard PSO algorithm has the problems of premature convergence and local optimum, ATCGIPSO is proposed in the paper aiming at the above problems. The algorithm not only has the characteristics of low complexity and simple implementation, but also can effectively improve the generation efficiency of test cases, thereby effectively reducing software development costs and reducing software development cycle.

References


[4] Liu Jiaxin. Research and design of embedded software static testing and automatic path test tools [D]. South China University of Technology, 2012;


