文章编号:1671-4598(2018)03-0050-04 DOI:10.16526/j.cnki.11—4762/tp.2018.03.014 中图分类号:TP311 文献标识码:A

基于改进 PSO 算法的路径软件测试用例生成方法

张倩宜，李 妍
（国网天津市电力公司 信息通信公司，天津 300010）

摘要：针对路径测试中的软件测试用例生成的问题，提出了一种基于改进 PSO 算法的软件测试用例生成方法；首先，采用分值函数叠加方法来构造 PSO 算法中的适应度函数，并对粒子位置更新策略进行改进；接着，构建算法的控制流程图并进行目标路径选择；然后，利用程序插装收集个体的适应度值；最后，测试数据生成程序执行，得到合适的测试数据；通过在三角形分类判别案例程序上的实验结果表明，提出的软件测试用例生成方法能够生成更合适的测试用例，且有效减少了生成所需时间。

关键词：改进 PSO 算法；路径测试；软件测试用例生成

Software Test Case Generation Method Based on Improved PSO Algorithm for Path Testing

Zhang Qianyi, Li Yan
(State Grid Tianjin Electric Power Company Information & Telecommunication Branch, TianJin 300010, China)

Abstract: For the issue that software test case generation in path test, this paper presents a software test case generation method based on improved PSO algorithm. Firstly, the fitness function in PSO algorithm is constructed by using the superposition of the score function, and the particle location updating strategy is improved. Then the control flow chart of the algorithm is built and the target path is chosen. After that we use the program to collect individual fitness values. Finally, the test data generation program is executed and the appropriate test data is obtained. The experimental results on the triangle classification case show that the proposed software test case generation method can generate more suitable test cases and reduce the generation time effectively.

Keywords: improved PSO algorithm; path test; software test case generation

0 引言

计算机技术的不断发展，软件的使用范围以及软件自身质量都显得越来越重要。软件测试作为软件质量保证的主要手段，在实际应用中往往由于要占据大约 50%的运行时间和 50%以上的软件开发总成本，导致软件测试费时费力[1-2]。自动化测试数据生成是软件测试中的一个关键问题，其实施不仅可以显着提高效率，而且可以降低软件测试的高成本[3]。特别值得注意的是，各种结构测试数据生成问题可以转化为面向路径的测试数据生成问题。此外，路径测试策略可以检测到将近 65 个测试程序中的错误。

虽然路径测试下的测试数据生成是一个很难解决的问题，有关学者仍然试图开发各种方法并进行了大量的研究，并取得了一些进展。这些方法可以分为两类：静态的方法和动态的方法。静态方法包括域缩减[4]和符号执行[5]等。当处理无限数组，循环，指针引用和过程调用等问题时这些方法会受到一定的局限性。

与静态方法相比，动态方法包括随机测试，局部搜索方法，目标导向方法，链接方法以及演化方法[6-8]。由于输入变量的值是在程序执行时确定的，动态测试数据的生成可以避免那些静态方法面临的问题。作为复杂空间中的一种强大的搜索方法，粒子群优化（particle swarm optimization，PSO）算法在 1995 年被应用于测试数据生成，并且演化方法从那以后一直是受到广大学者的关注。相关文献也表明基于 PSO 算法的测试数据生成优于其他动态方法，例如随机测试和本地搜索。

基于此，本文采用了基于改进 PSO 算法的软件测试用例生成方法。与传统的遗传算法相比，改进 PSO 算法采用分支函数叠加方法来构造适应度函数。同时，为了提高目标路径选择上的泛化能力，以三角形分类判别程序为例，选择具有最多分支结构的路径为目标路径。最后，利用程序插装收集个体的最优适应值。实验结果表明，本文采用基于改进 PSO 算法的软件测试用例生成方法可以有效减少生成所需时间以及能够生成更合适的测试用例。

1 软件测试用例生成描述

测试用例生成是软件测试中劳动力最密集的任务之一，对软件测试的有效性和效率产生了很大的影响。由于这些原因，测试用例的自动生成是几十年来较为活跃的研究课题之一[9]。一般来说，测试用例作为一个重要的软件工件，必须从一些信息中产生，也就是其他一些类型的软件工件。已经用作生成测试用例的参考输入的工件类型包括：程序结构和/或源代码；软件规格和/或设计模型；关于输入/输出数据空间的信息以及

收稿日期:2018-01-02； **修回日期**:2018-01-18。
基金项目:国网天津市电力公司 2017 年科技项目(KJ17—1—16)。
作者简介:张倩宜(1984-),女,天津市人,副高级工程师,硕士学位,主要从事电力企业信息通信工作方向的研究。

从程序执行中动态获得的信息。因此，可以将自动生成软件测试用例方法分为以下几类[10]，即（a）基于符号执行的结构测试，（b）基于模型的测试，（c）组合测试，（d）随机测试，以及（e）基于搜索的测试。

基于符号执行的结构测试是一种程序分析技术，它分析程序的代码，为程序自动生成测试数据。然而，该技术存在一些基本问题，限制了其对现实软件的有效性。基于模型的测试是一种使用软件系统模型推导测试套件的轻量级形式化方法。与传统的形式化方法相比，其旨在通过不完整的测试方法来收集程序正确性的见解。组合测试已经成为软件测试工具箱中常见的技术。在组合测试中，重点是选择输入参数（或配置设置）的样本，覆盖要测试的元素组合子集。随机测试中，测试用例均匀分布在输入域中。一组测试用例的选择旨在通过强制随机测试的均匀扩展来提高随机测试的失败检测效率。即如果之前执行的测试用例没有发现失败，那么新的测试用例应该远离已执行过的非失败测试用例。基于搜索的测试是基于一些智能搜索算法来自动化查找测试数据，从而最大限度地实现测试目标，同时最小化测试成本。

本文提出的方法属于基于搜索的测试方法，使用了 PSO 智能搜索算法来生成测试数据。

2　改进 PSO 算法

2.1　传统 PSO 算法

粒子群优化（PSO）算法最早由 Berhart 和 Kennedy 两位学者提出[11]。与遗传算法相比，PSO 算法从随机解开始，通过种群中个体之间的相互协作和信息共享来实现最优解的搜索。其中，PSO 算法中的每一个粒子代表问题的一个可能解，调节粒子位置和速度的适应度特征参数，以达到所选粒子的最优情形。

PSO 算法实现步骤如下，首先初始化一群随机粒子，让每个粒子都是所对应问题的一个历史最优解，并确定各自的适应值。确定适应度值后的每个粒子在整个算法过程中，其运动的距离和方向通过速度来确定。经过迭代搜索后的最优粒子将从局部最优解中求出全局最优解。PSO 算法的基本思想是加速每个粒子朝各自的历史和种群的最好位置逼近[12]。

设第 i 个粒子的当前位置为 $x_i=(x_{i1},x_{i2},\cdots,x_{im})$，当前速度为 $v_i=(v_{i1},v_{i2},\cdots,v_{im})$，其中 m 表示搜索空间维数。通过比较更新粒子群的历史最优解和找到的全局极值解，根据公式（1）（2）来更新粒子速度和位置。

$$v_{ij}(t+1)=\omega v_{ij}(t)+c_1r_1(p_{ij}(t)-x_{ij}(t))+c_2r_2(p_{gj}(t)-x_{ij}(t)) \quad (1)$$

$$x_{ij}(t+1)=x_{ij}(t)+rv_{ij}(t+1) \quad (2)$$

式中，$i\in(1,2,\cdots,n)$，$j\in(1,2,\cdots,m)$，t 表示迭代次数，ω 表示权重系数，其取值大小决定了当前粒子速度占下一刻速度的比例。c_1，c_2 表示学习因子系数，用来调节粒子的位置方向的步长。r_1，r_2 表示约束因子，为［0，1］范围内的随机数。

2.2　改进 PSO 算法

为了避免 PSO 算法在执行时所带来的负面影响，尤其是处理高维复杂问题时，易陷入局部最优解的困境，在求解全局最优解时，加入考虑相邻粒子的位置信息。即，式（1）进行如下的改进：

$$v_{ij}(t+1)=\omega v_{ij}(t)+c_1r_1(p_{ij}(t)-x_{ij}(t))+c_2r_2(p_{gj}(t)-x_{ij}(t))+c_2r_2(p_{kj}(t)-x_{ij}(t)) \quad (3)$$

式中，$p_{kj}(t)$ 表示相邻粒子的位置信息，而且为了减少计算量，实际运算过程中，并不是纳入全部粒子的位置信息，只选择具有最佳适应度值的相邻粒子的位置信息。同时，式（3）中由于考虑了相邻粒子位置对第 i 个粒子的影响，一定程度地提高了算法的收敛性。

另外，w 设置为固定值不能适应动态的收敛过程。为此，引入了动态惯性权重，使其在一定范围内，随着迭代次数的增加而线性递减，表达式如下：

$$w=w_{\min}+\frac{w_{\max}-w_{\min}}{T_{\max}}\times(T_{\max}-ite) \quad (4)$$

式中，$w_{\max}$ 和 $w_{\min}$ 为 w 的最大值和最小值，取值为 0.9 和 0.4；$T_{\max}$ 为最大迭代次数，ite 为当前迭代次数。

适应度函数对 PSO 算法的搜索寻找测试数据速度有直接的影响[13]。本文通过分值函数叠加方法来构造适应度函数。其中，当分支谓词为假时，$F(x)$ 为正或零，当分支谓词为真时，$F(x)$ 为负或零。

分支函数表达式为：

$$F=C-(F(f_1)+F(f_2)+\cdots+F(f_m)) \quad (5)$$

其中：$F(x)=\begin{cases}0,x\leqslant 0\\ x,x>0\end{cases}$；通过对 $F(x)$ 的求解比较，可以清楚知道测试数据对目标路径的覆盖程度情况。

3　基于改进 PSO 算法的测试数据生成

采用 PSO 算法自动生成路径测试的测试用例的原因之一是实现简单。PSO 算法的每次迭代都会生成一代个体。在实践中，计算时间不能是无限的，所以迭代在算法中应该是有限的。在有限的世代内，由 PSO 算法得出的求解结果可能会是局部最优解，结果就是不能定位需求可能被困在不需要的路径周围，无法找到所需的全局最优值[14]。PSO 在第一代的测试用例正常地分布在被测试的领域程序中，受困概率非常低。另外，其产生的测试用例的质量高于随机产生的测试用例的质量，因为该算法可以将测试用例的产生快速地引导到期望的范围[15]。

PSO 算法实现首要是选择合理的目标路径，将输入向量（测试数据）作为一个个体。为了使用 PSO 为被测程序生成面向路径的测试数据，本文设计了 5 个步骤，图 1 描述了整体的软件测试框架。

1）控制流程图构建。被测试程序的控制流程图可以通过相关工具手动或自动构建。控制流程图有助于测试人员选择具有代表性的目标路径。

2）目标路径选择。一般来说，被测试的程序有太多的路径可以完全测试。因此，测试人员必须选择有意义的路径作为目标路径。

3）适应度函数构建。为了评估执行路径和目标路径之间的距离，必须构建适应度函数。

4）程序插装。这意味着在每个源代码块的开始插入探测器来监视程序执行并收集相关信息（例如个体的适应度值）。

5）测试数据生成和插装程序执行。为实现目标路径，原始测试数据主要来自于从测试数据的领域随机选择和 PSO 算

法产生的新测试数据。最后，可能会产生沿着目标路径执行的合适的测试数据，或者由于超过最大生成而不能找到合适的测试数据。

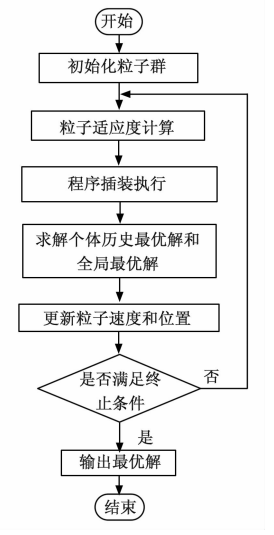

图 1 基本流程图

4 实验与分析

4.1 案例程序

为了验证本文算法在收敛性及运行时间上的性能表现，选择三角形分类判别程序为示例程序。三角形分类程序广泛应用于软件测试研究领域。它旨在确定 3 个输入边是否可以形成一个三角形，以及它们可以形成什么类型的三角形。三角形分类程序如下所示。

```
算法:三角形分类程序
TraversedPath= [];
TriangleType='Not a Triangle';
If((SideA+SideB>SideC)&&(SideB+SideC>SideA)&&(SideC+SideA>SideB))
    TraversedPath =[TraversedPath'a'];
    if
((SideA~=SideB)&&(SideB~=SideC)&&(SideC~=SideA))
        TraversedPath=[TraversedPath'e'];
        TriangleType='Scalene';
    else
        TraversedPath =[TraversedPath 'b'];
        if(((SideA==SideB)&&(SideB~=SideC))||((SideB==SideC)&&(SideC~=SideA))||((SideC==SideA)&&(SideA~=SideB)))
        TraversedPath = [TraversedPath 'f'];
        TriangleType ='Isosceles';
    else
        TraversedPath=[TraversedPath 'c'];
        TriangleType='Equilateral';
      end
    end
  else
    TraversedPath =[TraversedPath'd'];
  end
```

三角形分类测试程序确定任何 3 个输入长度的边可以形成哪种三角形。程序控制流程图如图 2 所示，其中包含 4 个路径。

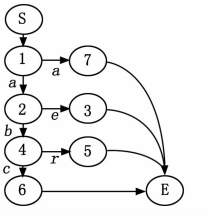

图 2 示例程序的流程图

这里设置输入的三角形边的取值为 1 到 2^N 的整数。另外，对于本文改进 PSO 算法，设置粒子编码长度为 $3N$，种群大小为 1 到 1000，学习因子 c_1，c_2 都为 2，w 从 0.9 线性降低到 0.4，最大迭代次数为 20。

4.2 目标路径选择

上图三角形分类程序的控制流程图，由四条路径组成：

路径 1：＜d＞ //非三角形

路径 2：＜ae＞ //不等边三角形

路径 3：＜abf＞ //等腰三角形

路径 4：＜abc＞ //等边三角形

根据概率论知识，路径＜abc＞是路径测试中最难涵盖的路径。因此，选择路径＜abc＞作为目标路径。

4.3 测试数据生成和执行

测试用例的自动生成可以按照如下步骤进行：

1）分析测试程序的整体框架，绘制出控制流程图，如图 1 所示；

2）根据测试路径要求，绘制合适的测试路径，如图 2 所示，并制定适应度函数；

3）通过程序插装对源程序进行插装；

4）根据程序的需求，随机产生定义域内的初始化测试用例集合；

5）在获得对应的适应度值后，检测是否执行了预定目标路径；

6）根据每个粒子的适应度值，执行改进 PSO 算法，生成测试用例，转到步骤 5）

7）运行结束，输出合适的测试用例。

表 1 为当设定测试总数为 500 时，10 次迭代中各种路径上的测试用例数量。图 3 显示了每一次迭代路径上测试用例数量曲线。

表 1 各次迭代中的测试用例数量(测试总数为 500)

迭代次数	＜abc＞	＜d＞	＜ae＞	＜abf＞	时间	测试总数
1	498	366	133	498	0.0 875	1 000
2	372	82	45	372	0.0 523	500
3	354	103	43	354	0.0 453	500
4	339	103	57	339	0.0 432	500
5	341	112	46	341	0.0 448	500
6	319	125	56	319	0.0 448	500
7	334	119	47	334	0.0 446	500
8	351	81	67	351	0.0 453	500
9	339	95	65	339	0.0 443	500
10	361	82	56	361	0.0 443	500

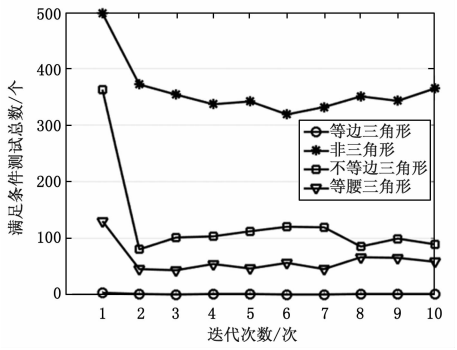

图 3　各次迭代中的测试用例数量
（测试总数为 500）

表 2 显示了当测试总数为 1000 时，10 次迭代中测试用例的平均数量。图 4 显示了每一次迭代路径上测试用例数量曲线。

表 2　各次迭代中的测试用例数量(测试总数为 1 000)

迭代次数	＜abc＞	＜d＞	＜ae＞	＜abf＞	时间	测试总数
1	4	487	354	155	0.0 752	1 000
2	2	647	266	85	0.0 956	1 000
3	4	703	212	81	0.0 921	1 000
4	5	759	153	84	0.0 899	1 000
5	1	792	148	59	0.0 871	1 000
6	2	801	143	59	0.0 879	1 000
7	2	845	104	49	0.0 869	1 000
8	4	845	107	44	0.0 912	1 000
9	2	835	106	57	0.0 865	1 000
10	1	843	113	43	0.0 865	1 000

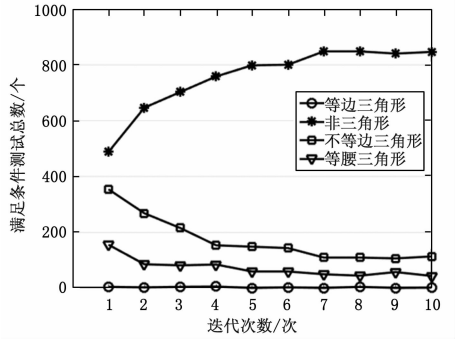

图 4　各次迭代中的测试用例数量
（测试总数为 1 000）

通过以上两组实验可以看出，所选择的种群规模越小，则找到全部用例的迭代次数越多，反之亦然。但是，需注意的是，运行时间会随着种群规模的增大而增加。在实际操作中，需要折衷权衡迭代次数和运行时间两者之间的比重。

5　结束语

针对路径测试中的软件测试用例生成的问题，本文提出了一种基于改进 PSO 算法的软件测试用例生成方法。首先，考虑相邻粒子的位置信息和动态权重来改进传统 PSO，并利用分值函数叠加方法来构造适应值函数。接着构建算法的控制流程图并进行目标路径选择。然后，利用程序插装收集个体的适应度值。最后，测试数据生成程序执行，得到合适的测试数据。

实验表明，PSO 算法能够有效生成合适的路径测试的测试用例，并大大减少生成测试所需的时间。

参考文献：

[1] 邓璐娟，林　楠，卢华琦，等. 基于改进粒子群算法的测试数据自动生成研究 [J]. 计算机测量与控制，2011，19 (2)：250－252.

[2] 贺　滢，徐蔚鸿，李杨林. 基于 RACPSO 的测试用例自动生成方法 [J]. 计算机工程，2016，42 (5)：66－70.

[3] 杨　波，吴　际，徐　珞，等. 一种软件测试需求建模及测试用例生成方法 [J]. 计算机学报，2014，37 (3)：522－538.

[4] 周虹伯，金大海，宫云战. 基于域敏感指向分析的区间运算在软件测试中的应用 [J]. 计算机研究与发展，2012，49 (9)：1852－1862.

[5] Bertolino A. Software Testing Research：Achievements，Challenges，Dreams [C]. international conference on software engineering，2007：85－103.

[6] Haga H，Suehiro A. Automatic test case generation based on genetic algorithm and mutation analysis [C]. IEEE International Conference on Control System，Computing and Engineering. IEEE，2013：119－123.

[7] Zhang N，Wu B，Bao X. Automatic Generation of Test Cases Based On Multi－population Genetic Algorithm [J]. International Journal of Multimedia & Ubiquitous Engineering，2015，10 (6)：113－122.

[8] Huang M，Zhang C，Liang X. Software test cases generation based on improved particle swarm optimization [C]. International Conference on Information Technology and Electronic Commerce. IEEE，2015：52－55.

[9] 耿　技，聂　鹏，秦志光. 软件确保智能测试用例生成 PSO 算法进展研究 [J]. 电子科技大学学报，2012，41 (6)：905－910.

[10] Anand S，Burke E K，Chen T Y，et al. An orchestrated survey of methodologies for automated software test case generation [J]. Journal of Systems & Software，2013，86 (8)：1978－2001.

[11] 陈云飞，李　征，赵瑞莲. 基于 PSO 的多目标测试用例预优化 [J]. 计算机科学，2014，41 (5)：72－77.

[12] Tiwari S，Mishra K K，Misra A K. Test Case Generation for Modified Code Using a Variant of Particle Swarm Optimization (PSO) Algorithm [C]. International Conference on Information Technology：New Generations. IEEE Computer Society，2013：363－368.

[13] Liu D，Feng Q Y. PSO fitness function used in antenna arrays pattern synthesis [J]. Chinese Journal of Radio Science，2011，26 (3)：581－586.

[14] 郭书杰，黄　明，梁　旭，等. 基于差分进化算法的组合测试用例集生成 [J]. 计算机应用研究，2014，31 (5)：1449－1451.

[15] Devasena M S G，Gopu G，Valarmathi M L. Automated and Optimized Software Test Suite Generation Technique for Structural Testing [J]. International Journal of Software Engineering & Knowledge Engineering，2016，26 (1)：1－13.