浙江理工大学学报，2019，41(1)：72-78
Journal of Zhejiang Sci-Tech University
DOI:10.3969/j.issn.1673-3851(n).2019.01.011

基于 IRAPSO 的组合测试用例生成方法

李晨晨，丁佐华
（浙江理工大学信息学院，杭州 310018）

摘　要：最小覆盖表的生成是组合测试研究领域的一个关键问题，虽然粒子群优化算法是生成最小覆盖表的方法之一，但该算法存在易陷入局部最优和搜索精度低等问题。针对该问题提出了一种改进的约简自适应粒子群算法。该方法首先对粒子群优化算法的进化方程进行约简，消去其速度项，得到约简的粒子群进化方程；然后提出了惯性权重的自适应调整策略并且在适应值策略中引入汉明距，以提高该算法生成测试用例的覆盖率。与已有算法的比较结果表明，该算法在克服粒子群优化算法易陷入局部最优等问题的同时能够在较短的时间内生成规模更小的覆盖表。

关键词：组合测试；覆盖表；约简粒子群算法；进化方程约简；自适应算法
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1673-3851 (2019) 01-0072-07

Test case generation method based on improved reduced adaptive particle swarm optimization

LI Chenchen, *DING Zuohua*
(School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: In the study of combination testing, the generation of the minimum covering array is a key problem. The particle swarm optimization algorithm is one of methods to generate the maximum covering array. However, the algorithm is easy to fall into the local optimal solution, with low searching precision. For such problems, an improved reduced adaptive particle swarm optimization (IRAPSO) algorithm is proposed to address the above problems. Firstly, the evolutionary equation of the particle swarm optimization algorithm (PSO) was simplified by removing the velocity term to gain the reduced particle swarm optimization equation. Then, in order to improve the coverage of the test cases generated by the algorithm, an adaptive adjustment scheme of inertia weight was proposed and the Hamming distance was introduced into the fitness value strategy. The comparison result with the existing algorithm s showed that, the proposed algorithm overcame the defect of PSO which is easy to fall into the local optimal solution and could generate the smaller-scale covering array in a short time.

Key words: combination testing; covering array; reduced particle swarm optimization; reduction of the evolution equation; adaptive algorithm

0　引　言

软件测试是软件开发过程中必不可少的环节，而且成本高昂。如何以较小的代价检测出软件系统中的故障一直是软件测试研究的热点。组合测试作为一种科学有效的测试方法，能够从庞大的组合空

收稿日期：2018－09－08　　网络出版日期：2018－11－01
基金项目：国家自然科学基金项目(61751210,61572441)
作者简介：李晨晨(1992－)，女，山东泰安人，硕士研究生，主要从事软件测试方面的研究。
通信作者：丁佐华，E-mail:zouhuading@hotmail.com

间中选取少量的测试用例,覆盖待测软件中各因素之间的交互组合[1]。Kuhn 等[2]研究发现,超过70%的软件故障是由任意 2 个因素的相互作用引起的,而全部的软件故障可以由 6 个因素之间的相互作用检测出。因此,组合测试可以以较少的测试用例检测出系统的故障。

目前,国内外学者对于组合测试中最小覆盖表生成的研究已经取得了不少的成果。对于最小覆盖表的生成方法主要分为 3 大类:数学方法、贪心算法、启发式搜索算法。近几年,启发式搜索算法被广泛的应用于组合测试中,如模拟退火(Simulated annealing, SA)[3-4]、禁忌搜索算法、蚁群优化算法(Ant colony optimization, ACO)[5-7]、遗传算法(Genetic algorithm, GA)[8-10]、粒子群优化算法(Particle swarm optimization, PSO)[11-13]等都被应用于覆盖表的生成。其中,粒子群优化算法由于规则简单、可调参数少等特点,被广泛应用于该领域。Chen 等[11]提出了一种基于 PSO 生成成对组合测试用例的框架,并分别结合 one-test-at-a-time 策略和类 IPO 策略生成覆盖表。Ahmed 等[12]介绍了一种基于 PSO 生成可变强度组合测试用例的方法,该方法能够生成任意覆盖强度的覆盖表。包晓安等[13]提出了一种自适应粒子群优化算法,并构造了一个优先级度量函数来提升算法性能。

在覆盖表的生成中,为了生成最小覆盖表往往忽略了粒子群算法容易陷入局部最优、搜索精度低等缺点。本文为克服这些缺点采用了史娇娇等[14]提出的约简自适应粒子群算法(Reduced adaptive particle swarm optimization, RAPSO)来生成覆盖表。RAPSO 将 PSO 进化方程中的速度项进行约简,来克服粒子群算法容易陷入局部优化的缺点。本文在 RAPSO 中,引入惯性权重的自适应调整方案以及新的适应值策略,提出了一种改进的约简自适应粒子群算法(Improved RAPSO, IRAPSO),并将其与 one-test-at-a-time 策略相结合,提出了一种能够生成任意覆盖强度覆盖表的方法。本文分析了每个覆盖矩阵的参数,并对 IRAPSO 生成测试用例集的规模以及生成时间进行实验验证。

1　相关理论基础

1.1　组合测试

假设有 n 个独立因素影响待测软件系统(Software under test, SUT),这 n 个因素可用有限集合 $F=\{f_1,f_2,f_3,\cdots,f_n\}$ 表示,其中第 i 个因素 f_i 的候选值有 l_i 个,其对应的候选值集可表示为 $D_i=\{1,2,3,\cdots,l_i\}$。

定义 1(覆盖矩阵)　覆盖矩阵 $\boldsymbol{CA}(N;t,k,v)$ 是一个 $N\times k$ 矩阵,每一列有 v 个不同的取值;$N\times t$ 的子矩阵至少包含任意 t 个因素的所有取值一次[3],其中 t 代表覆盖强度。

定义 2(混合覆盖矩阵)　混合覆盖矩阵 $\boldsymbol{MCA}(N;t,k,(v_1,v_2,\cdots,v_k))$ 是一个 $N\times k$ 的矩阵,每一列的取值数目不同,分别用 $v_1,v_2,\cdots,v_k$ 来表示。

定义 3(可变强度覆盖矩阵)　可变强度覆盖矩阵 $\boldsymbol{VSCA}(N;t,k,v,\boldsymbol{C})$ 是一个 $N\times k$ 的矩阵,其中 $\boldsymbol{C}_1,\boldsymbol{C}_2,\cdots,\boldsymbol{C}_j$ 表示该矩阵中包含的子矩阵,并且其对应的覆盖强度 $t_1,t_2,\cdots,t_j$ 大于覆盖强度 t。

下面以网络软件系统为例(见表 1)来分析其生成的组合测试用例集。该系统包含 5 个不同的因素,每个因素的候选值的个数分别为 2、2、2、3、3。为了详尽地测试所有因素的候选值,需要 $2\times2\times2\times3\times3=72$ 个测试用例。当因素和候选值的数量增加时,要生成的测试用例的数量以及所需代价将呈指数增长。为了减少测试成本,本文采用覆盖强度为 2 的组合测试,即只考虑该软件系统中任意 2 个因素之间的相互作用。实验结果显示只需要 9 条测试用例即可覆盖所有组合,见表 2。

表 1　网络软件系统

浏览器	操作系统	内存/G	连接类型	数据库
Chrome	Windows 7	1	LAN	Oracle
Firefox	Windows 8	2	PPP	MySQL
			ISDN	SQL

表 2　覆盖矩阵 $\boldsymbol{MCA}(9;2,5,2^3 3^2)$

序号	浏览器	操作系统	内存/G	连接类型	数据库
1	Chrome	Windows 8	1	ISDN	Oracle
2	Firefox	Windows 7	2	ISDN	SQL
3	Chrome	Windows 7	1	LAN	MySQL
4	Chrome	Windows 8	2	LAN	SQL
5	Firefox	Windows 7	2	LAN	Oracle
6	Firefox	Windows 8	2	PPP	MySQL
7	Chrome	Windows 7	1	PPP	SQL
8	Firefox	Windows 7	1	ISDN	MySQL
9	Chrome	Windows 7	1	PPP	Oracle

在实际情况中,有些因素之间的交互更容易触发系统故障。为了能够覆盖这些不同的交互,本文采用了可变强度覆盖矩阵。假设在上述系统中浏览器、操作系统、数据库这 3 个因素的组合更容易触发系统的故障,即 $\boldsymbol{C}=$(浏览器,操作系统,数据库)。

为了能够完全覆盖这 3 个因素的组合,则需要在表 2 的基础上再增加表 3 中的测试用例。

表 3 覆盖矩阵 *VSCA*(12;2,5,$2^3 3^2$,*MCA*(3,$2^2 3^1$))

序号	浏览器	操作系统	内存/G	连接类型	数据库
10	Chrome	Windows 8	1	ISDN	MySQL
11	Firefox	Windows 8	2	LAN	Oracle
12	Firefox	Windows 8	2	PPP	SQL

1.2 粒子群优化算法

粒子群优化算法由 Kennedy 和 Eberhart 于 1995 年首次提出,是一种基于种群寻优的元启发式算法,该算法主要受鸟类觅食和鱼群迁徙行为的启发。在粒子群优化算法中,每个粒子的位置代表当前所求问题解的候选值,通过不断地迭代更新使粒子趋向问题的最优解。在每次迭代中,粒子会根据其自身找到的最优解 p 和整个种群找到的最优解 p_g 来更新当前的状态。

粒子群优化算法在解决实际问题时,需要建模分析粒子所在的搜索空间。假设在 M 维的搜索空间中,第 i 维代表的是 SUT 中第 i 个因素 f_i,则第 i 维的候选值集合为 $D_i=\{1,2,\cdots,l_i\}$。第 i 个粒子在第 k 次迭代时,粒子位置可以表示为 $\boldsymbol{X}_i=(x_{i,1},x_{i,2},\cdots,x_{i,n})(x_{i,1}\in D_1,x_{i,2}\in D_2,\cdots,x_{i,n}\in D_n)$,速度可以表示为 $\boldsymbol{V}_i=(v_{i,1},v_{i,2},\cdots,v_{i,n})$。第 i 个粒子在第 $k+1$ 次迭代时,粒子的速度和位置的更新公式如式(1)—(2)所示:

$$v_{i,j}(k+1)=\omega v_{i,j}(k)+c_1 r_1[p_{i,j}(k)-x_{i,j}(k)]+c_2 r_2[p_{g,j}(k)-x_{i,j}(k)] \tag{1}$$

$$x_{i,j}(k+1)=x_{i,j}(k)+v_{i,j}(k+1) \tag{2}$$

其中:$v_{i,j}(k)$表示的是第 i 个粒子在第 j 维时的速度;$x_{i,j}(k)$表示的是第 i 个粒子在第 j 维时的位置。式(1)分为 3 部分,第 1 部分是惯性部分,代表了粒子对当前自身状态的继承,ω 称为惯性权重,其控制着历史速度对当前速度的影响;第 2、3 部分分别是认知部分和社会部分,分别表示粒子对自身找到的最优解的学习以及粒子群中各粒子之间的交互协作,c_1,c_2 称为加速因子,是将粒子推向 p 和 p_g 位置的统计加速项的权,r_1,r_2 是在[0,1]之间均匀分布的随机变量。

为了解决粒子有时会飞出有效搜索空间的问题,需要对粒子速度范围以及位置的边界进行约束。在每一维中,将对应因素的候选值的一半作为最大速度,即将速度限制在$[-l_i/2,l_i/2]$中。通过 Chen 等[11]的实验可知,反射墙策略能够有效的处理粒子飞出搜索空间的问题,本文采用该策略来将粒子约束在搜索范围之内。反射墙策略指的是当粒子在超过某一维的边界时,粒子会被反弹回来。反射墙策略根据式(3)对粒子位置进行处理。

$$h(x_{i,j})=\begin{cases}2l_i-x_{i,j}+1, & x_{i,j}>l_i\\ -x_{i,j}+1, & x_{i,j}<1\\ x_{i,j}, & \text{其他}\end{cases} \tag{3}$$

1.3 one-test-at-a-time 策略

在组合测试用例生成中,粒子群优化算法只能生成单条测试用例。由文献[13,15]可知,为了生成所需的覆盖表,一般将元启发式算法与贪心算法相结合。one-test-at-a-time 策略作为一种便于拓展的贪心算法而被广泛应用于组合测试中[16]。首先,根据实例 E 生成空的测试用例集 TS 以及包含全部组合数的集合 S;然后在 S 中选取一个组合 c,根据粒子群优化算法生成适应值高的单个测试用例 T,将其添加到 TS 中并将其覆盖的组合在 S 中移去;直到 S 中的所有组合均被移除时,返回最终的测试用例集 TS。one-test-at-a-time 策略的伪代码如算法 1 所示。

算法 1 one-test-at-a-time 策略

输入:实例 E。

输出:测试用例集 TS。

1 TS=NULL;

2 将 E 中的所有组合加入到 S 中;

3 while($S\neq$NULL){

4 从 S 中选取一个组合 c;

5 根据粒子群算法生成一条适应值最大的测试用例 T;

6 将测试用例 T 加入到测试用例集 TS 中;

7 将测试用例 T 覆盖的组合在 S 中移除;

8 }

9 返回 TS;

2 基于 IRAPSO 的组合测试用例生成方法

2.1 方法框架

本文提出了一种基于 IRAPSO 的组合测试用例生成方法。该方法的总框架是由测试环境构造模块和测试模块两部分组成,如图 1 所示。

测试环境构造模块:组合测试通过对待测软件系统(SUT)中不同因素之间的交互作用的研究,生成能够检测出系统故障的测试用例。在生成测试用例之前,需要对 SUT 进行静态分析,通过对 SUT 约束条件和覆盖强度的研究,确定需覆盖的全部组合。

测试运行模块:在该模块中生成最终的测试用

例集。首先 one-test-at-a-time 策略在测试环境模块生成的组合中任意选取一个组合,生成一条缺省的测试用例。然后将缺省的测试用例作为 IRAPSO 的输入,生成一条完整的测试用例。其中,在生成单条测试用例的过程中,本文采用了自适应调整的惯性权重来代替 PSO 中固定的惯性权重。最后,将 IRAPSO 生成的单条测试用例返回到 one-test-at-a-time 策略中。

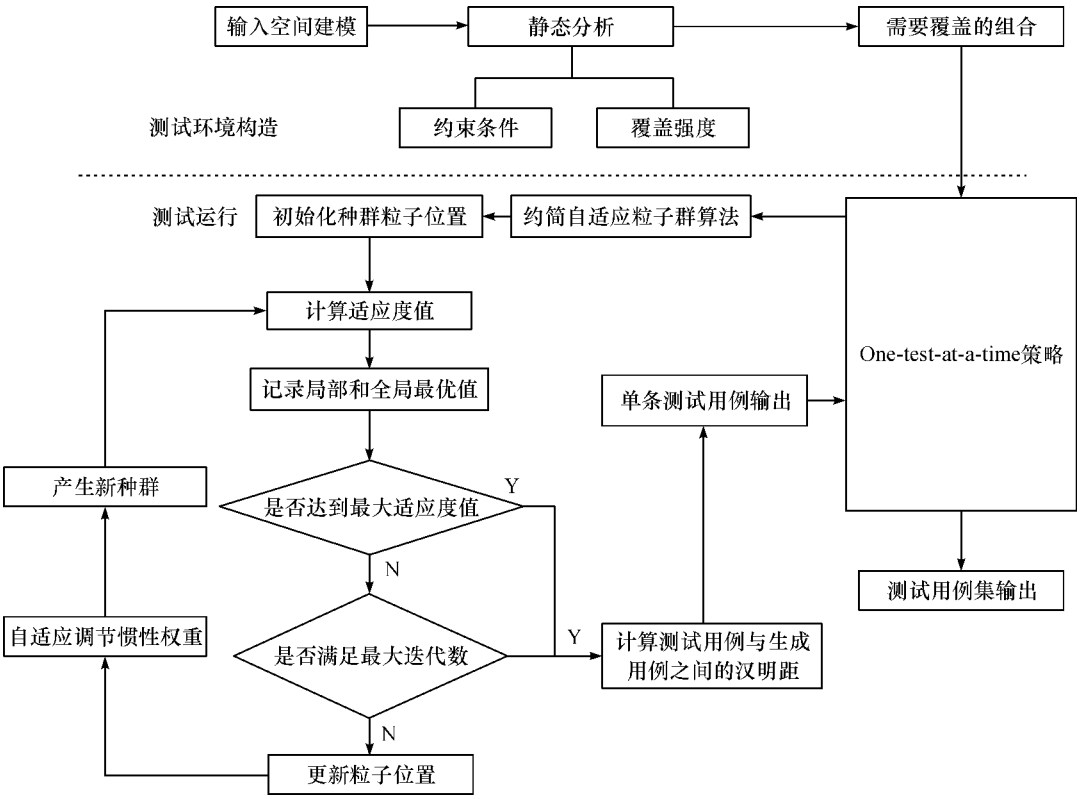

图 1　基于 PAPSO 算法的组合测试框架

2.2　约简自适应粒子群算法

2.2.1　约简粒子群进化方程

PSO 粒子的速度和位置决定了粒子的进化过程。粒子的速度反映了每次迭代更新的快慢,为了防止粒子飞出有效的搜索空间,人为设定粒子速度的最大值。当粒子速度设置不当时,粒子会偏离原来正确方向导致 PSO 后期收敛速度慢、精度降低等问题[14]。为了解决人为设置速度项所带来的问题,本文根据式 $x=v\times t$ 消去式(1)—(2)中粒子的速度项,得到没有速度项的 PSO 进化方程[14],如式(4)所示:

$$x_{i,j}(k+1)=\omega x_{i,j}(k)+c_1r_1[p_{i,j}(k)-x_{i,j}(k)]+c_2r_2[p_{g,j}(k)-x_{i,j}(k)] \tag{4}$$

在式(4)中,各个参数与 PSO 中的参数是一一对应的,但是进化方程由原来的二阶转为一阶,使算法的复杂度降低。

2.2.2　基于惯性权重的自适应调整方案

RAPSO 算法相较于 PSO 算法的不同之处在于消去了速度项,所以 RAPSO 算法中惯性权重对粒子的收敛速度以及搜索精度方面起着至关重要的作用。通过对 PSO 研究可知:当 ω 较大时,粒子更容易跳出局部最优,有利于全局搜索;当 ω 较小时,有利于粒子群优化算法的局部精确搜索。本文将文献[13]提出的惯性权重调整策略进行改进,并采用此策略来代替 *PSO* 中固定的惯性权重。惯性权重的调整策略是根据当前粒子 $\boldsymbol{X}_i$ 与最优粒子 $\boldsymbol{X}_g$ 之间的曼哈顿距离来评价粒子优劣。已知有两个向量 $\boldsymbol{X}_1=(x_1,y_1)$ 和 $\boldsymbol{X}_2=(x_2,y_2)$,两者的曼哈顿距离 d 如式(5)所示:

$$d(\boldsymbol{X}_1,\boldsymbol{X}_2)=|x_1-x_2|+|y_1-y_2| \tag{5}$$

为了更好地评价当前粒子 $\boldsymbol{X}_i$ 与当前最优粒子 $\boldsymbol{X}_g$ 之间的差异,定义函数 f 来进行衡量。式(6)给出了衡量粒子间差异的公式:

$$f(X_i)=\frac{d(\boldsymbol{X}_i,\boldsymbol{X}_g)}{Dist_{\max}} \tag{6}$$

其中:$d(\boldsymbol{X}_i,\boldsymbol{X}_g)$表示 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_g$ 之间的曼哈顿距离;$Dist_{\max}$表示粒子群中粒子与 $\boldsymbol{X}_g$ 之间的最大距离。$f(\boldsymbol{X}_i)$值越大,代表 $\boldsymbol{X}_i$ 与 $\boldsymbol{X}_g$ 之间的差异越大,此时应该增大 ω 的值,提高粒子的全局搜索能力;反之,应该减小 ω 的值,提高粒子的局部搜索精度。根据

上述关系,定义了惯性权重的自适应调整策略。ω 与 f 之间的关系如式(7)所示:

$$\omega=\begin{cases}\omega_{\max}, & f(\boldsymbol{X}_i)=1\\ \omega_{\max}-(\omega_{\max}-\omega_{\min})\mathrm{e}^{(f(X_i)/(f(X_i)-1))}, & \text{其他}\end{cases} \tag{7}$$

由式(7)可知,当 $f(\boldsymbol{X}_i)$取值为 1 时,指数部分分母为 0,会导致程序在运行到该值时出现异常。由图 2 可知,当 $f(\boldsymbol{X}_i)$的值为 1 时,ω 是无限接近 $\omega_{\max}$,本文为了避免程序异常退出,将 $f(\boldsymbol{X}_i)=1$ 时的 ω 值设置为$\omega_{\max}$。其中,$\omega_{\max}$和$\omega_{\min}$分别表示惯性权重的最大值和最小值,本文根据文献[11]将其设置为:$\omega_{\max}=0.9$,$\omega_{\min}=0.4$。其中,图 2 给出了 ω 随 f 值的变化曲线图,由图可知 ω 随 f 值单调递增,与上述惯性权重的自适应调整策略一致。

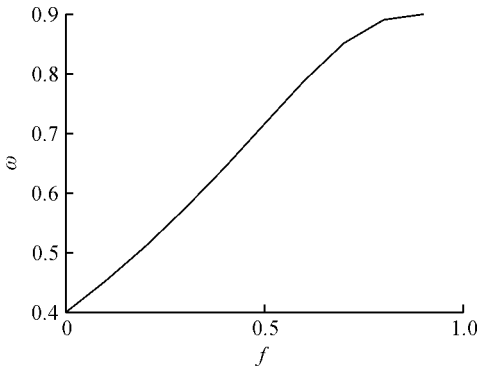

图 2 随 f 的变化曲线

2.2.3 新的适应值策略

在解决不同的问题时,PSO 中的适应值函数是不同的。在组合测试中,一般将单个测试用例覆盖新组合的数量设置为 PSO 的适应值函数。由于一条测试用例所能覆盖的最多的组合数为 $\boldsymbol{C}(n,t)$个,所以当生成的测试用例覆盖的组合数达到 $\boldsymbol{C}(n,t)$个时,将该测试用例作为最优测试用例添加到测试用例集中。但是该适应值函数只是选择当前覆盖组合数最多的测试用例,没有分析已经生成的测试用例集与当前生成的测试用例之间的关系,会导致粒子无法跳出局部最优。所以本文在原来的适应度函数的基础上引入 Wu 等[15]提出的新的适应值策略,即在适应值函数中引入汉明距,使得测试用例的构造更加的紧密。

为了衡量已经生成的测试用例集 TS 中测试用例与当前生成的测试用例 T_i 之间的关系,在此定义了平均汉明距。假如有两个测试用例 T_1 和 T_2,本文将两个测试用例之间不同取值的个数定义为汉明距 $z_{1,2}$,故将平均汉明距 $H(T_i,TS)$定义为当前测试用例 T_i 与测试用例集 TS 中测试用例汉明距之和的平均值,如式(8)所示:

$$H(T_i,TS)=\frac{1}{|TS|}\sum_{k\in TS}z_{i,k} \tag{8}$$

通过将汉明距作为新的适应值函数,可以在多个具有相同覆盖组合个数的测试用例中选取更合适的测试用例,即选取与测试用例集中的测试用例之间的平均汉明距最小的测试用例。

2.3 IRAPSO 生成单条测试用例的伪代码

本文采用上述策略,提出了适用于组合测试的改进的约简自适应粒子群算法。在算法 2 中给出了该算法生成单个测试用例的伪代码。

算法 2 基于改进的约简自适应粒子群算法生成单个测试用例

输入:粒子数 m,覆盖强度 t,因素个数 n,各个因素的值 D_i,需覆盖的组合数 S,迭代次数 $iter$,迭代的最大次数 $iter_{\max}$,适应值函数 $fitness(\boldsymbol{X}_i)$。

输出:最好的单个测试用例 p_g。

```
1 iter=0;
2 for(i=0;i<m;i++){//初始化粒子群;
3 初始化粒子群的位置和速度;
4 p_i=NULL; //记录每个粒子个体的最优位置;
5 }
6 p_g=NULL;//记录全局最优位置;
7 while(iter<iter_max){
8 for(i=0;i<m;i++){
9 计算适应值函数 fitness(X_i);//单个测试用例所覆盖的组合数量;
10 if(fitness(X_i)==C(n,t)&&min(H(T_i,TS))) return X_i;
11 if(fitness(X_i)>fitness(p_i) p_i=X_i;
12 if(fitness(X_i)>fitness(p_g) p_g=X_i;
13 }
14 for(i=0;i<m;i++){
15 根据式(7)对惯性权重进行调整;
16 根据式(3)和式(4)对粒子位置进行处理,并对位置进行取整运算;
17 }
18 iter++;
19 }
20 return p_g;
```

3 实 验

为了验证本文所提方法的有效性,选取了文献

[13,15]中 16 个比较经典的覆盖矩阵来进行分析,其包含了任意强度的覆盖矩阵。本文根据文献[11]对参数进行设置:最大迭代次数 $iter_{max}=1000$,学习因子 $c_1=2$,$c_2=2$,惯性权重的范围为 $0.4\leqslant\omega\leqslant0.9$。由于 PSO 具有随机性,每个覆盖矩阵中种群数量和迭代次数是不同的。由文献[14]的实验分析可知,种群数量与迭代次数共同决定 PSO 搜索复杂程度且两者的乘积为常数 20000,因此本文将初始迭代次数设为 100,并以 100 为间隔逐渐增加到最大迭代次数。为了减少 PSO 的随机性对实验结果的影响,本文对每一组实验参数独立运行 20 次,取其生成测试用例集的平均值来进行研究。本文所选覆盖矩阵如下所示,并用下标对每个覆盖矩阵进行区分:

$\boldsymbol{CA}_1(2,4,3)$,$\boldsymbol{CA}_2(4,8,5)$,$\boldsymbol{MCA}_3(2,9,8^2 7^3 5^2 3^2)$,$\boldsymbol{MCA}_4(3,10,5^1 3^4 2^5)$,$\boldsymbol{MCA}_5(4,30,4^6 3^{14} 2^{10})$,$\boldsymbol{VSCA}_6(2,15,3,\boldsymbol{CA}(3,3^4))$,$\boldsymbol{VSCA}_7(2,15,5^3 4^5 3^7,\boldsymbol{MCA}(3,5))$,$\boldsymbol{VSCA}_8(3,15,4^5 2^{10},\boldsymbol{MCA}(3,2^5))$,$\boldsymbol{CA}_9(2,3^{13})$,$\boldsymbol{CA}_{10}(3,3^6)$,$\boldsymbol{CA}_{11}(3,5^7)$,$\boldsymbol{MCA}_{12}(3,5^2 4^2 3^2)$,$\boldsymbol{VSCA}_{13}(2,3^{15},\boldsymbol{CA}(3,3^5))$,$\boldsymbol{VSCA}_{14}(2,3^{15},\boldsymbol{CA}(3,3^6))$,$\boldsymbol{VSCA}_{15}(2,3^{15},\boldsymbol{CA}(3,3^7))$,$\boldsymbol{VSCA}_{16}(2,3^{15},\boldsymbol{CA}(3,3^9))$。

将 IRAPSO 与 CPSO[15]、AP-PSO[13]、GA[5]、ACO[5-6]等算法进行比较,结果见表 4—表 5。表 4—表 5 列出了这几种算法生成的最优测试用例集的规模,这些数据均来源于文献[5-6,14-15]。

表 4 给出了 AP-PSO、RAPSO 和 IRAPSO 生成的组合测试用例集的规模和运行时间。其中,RAPSO 代表了文献[14]提出的用于生成测试数据的约简自适应粒子群算法,为了验证本文所提算法的有效性,在 RAPSO 中引入惯性权重的自适应调整策略并将其用于组合测试用例生成中。从测试用例集的规模上看,IRAPSO 在大多数覆盖矩阵中优于 AP-PSO 和 RAPSO,尤其是在复杂的覆盖矩阵 $\boldsymbol{MCA}_3$、$\boldsymbol{MCA}_5$、$\boldsymbol{VSCA}_7$ 中。在覆盖矩阵 $\boldsymbol{CA}_1$ 中,由于 3 种方法生成的每条测试用例的适应值都达到了最大适应值 $\boldsymbol{C}(n,t)$,所以生成的测试用例集规模相同,但是通过分析 3 种方法的时间可知,IRAPSO 所需时间少于其他方法。表 4 中 p-value 表示 IRAPSO 相较于 AP-PSO 所约简的测试用例的百分比。通过 p-value 可知,IRAPSO 能够平均约简 AP-PSO 测试用例集的 18.01%。从 3 种算法的运行时间上来看,虽然 AP-PSO 与 IRAPSO 运行的平台存在差异,但是 IRAPSO 运行时间要少于 AP-PSO,其约简的时间达到 6%～49%,表明本文所提方法可以有效的减少运行时间。

表 4　IRAPSO、AP-PSO、RAPSO 算法的实验结果

覆盖矩阵	AP-PSO[13]		RAPSO		IRAPSO		p-value
	规模	时间/s	规模	时间/s	规模	时间/s	/%
$\boldsymbol{CA}_1$	9.0	12	10.0	10	9.0	6	0
$\boldsymbol{CA}_2$	1241.6	3432	1275.0	4076	1040.0	3223	16.23
$\boldsymbol{MCA}_3$	111.5	58	81.0	48	78.7	41	29.36
$\boldsymbol{MCA}_4$	44.7	79	46.8	47	43.2	40	3.35
$\boldsymbol{MCA}_5$	119.2	126	90.4	129	83.2	67	30.10
$\boldsymbol{VSCA}_6$	32.7	32	32.5	25	31.6	23	3.21
$\boldsymbol{VSCA}_7$	241.3	903	14.8	390	125.0	712	48.19
$\boldsymbol{VSCA}_8$	125.9	612	116.6	586	108.8	405	13.58

表 5　IRAPSO、CPSO、GA、ACO 算法的最小及平均测试用例集

覆盖矩阵	CPSO[15]		IRAPSO		GA[5]	ACO[5-6]
	最优	均值	最优	均值	最优	均值
$\boldsymbol{CA}_9$	18	19.1	17	19.0	17	17
$\boldsymbol{CA}_{10}$	42	45.3	33	36.8	33	33
$\boldsymbol{CA}_{11}$	223	225.2	216	220.1	218	218
$\boldsymbol{MCA}_{12}$	117	123.6	101	106.2	108	106
$\boldsymbol{VSCA}_{13}$	38	41.4	39	41.5	—	38
$\boldsymbol{VSCA}_{14}$	45	47.3	44	46.1	—	45
$\boldsymbol{VSCA}_{15}$	49	52.1	45	48.7	—	48
$\boldsymbol{VSCA}_{16}$	58	60.3	56	57.8	—	57

注:“—”表示在文献中没有针对该覆盖矩阵的实验结果。

表 5 给出了 IRAPSO、CPSO、GA、ACO 生成的最小测试用例集以及平均测试用例集。从表 5 整体的实验结果来看,IRAPSO 在生成测试用例的规模上要优于其他 3 种算法。从最小测试用例集规模上来看,除了在覆盖矩阵 $\boldsymbol{VSCA}_{13}$ 中 IRAPSO 的实验结果要略差于其他 3 种算法之外,IRAPSO 能够得到与其他 3 种算法相同或者更好的结果。在覆盖矩阵 $\boldsymbol{CA}_9$、$\boldsymbol{CA}_{10}$ 中,4 种算法得到相同的最小测试用例规模,但是 IRAPSO 的测试用例集的均值要明显优于其他 3 种算法。

4　结　论

目前,组合测试较多地关注最小覆盖表的生成而忽略了粒子群优化算法易陷入局部最优的问题。为了解决 PSO 容易陷入局部优化以及搜索精度低等问题,本文在 RAPSO 中引入自适应调整惯性权重策略以及新的适应值策略,提出了一种基于 IRAPSO 的组合测试用例生成方法。该方法具有广泛的适用性,可以适用于任意覆盖强度的覆盖表的生成。通过与 CPSO、AP-PSO、GA、ACO 等经典

算法的比较可知,本文所提方法可以在较短的时间内生成规模更小的测试用例集。

由于粒子群算法具有随机性,导致每个覆盖矩阵中各个参数的取值不是固定不变,为了得到每个覆盖矩阵最优的参数设置需要进行多次实验,这无疑会加大组合测试的工作量以及测试的代价。在未来的工作中,可以对各种变种的粒子群算法进行研究,寻找合适的方法来解决该问题,减少组合测试的工作量以及代价。

参考文献:

[1] Nie C, Leung H. A survey of combinatorial testing[J]. Acm Computing Surveys, 2011, 43(2):1-29.

[2] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing [C]// Proceedings of the 27th Software Engineering. Greenbelt:IEEE Software Engineering Workshop,2002: 91-95.

[3] Cohen M B, Gibbons P B, Mugridge W B, et al. Constructing test suites for interaction testing [C]// Proceedings of the 25th International Conference on Software Engineering, Portland. IEEE Computer Society,2003:38-48.

[4] Torres-Jimenez J, Rodriguez-Tello E. New bounds for binary covering arrays using simulated annealing [J]. Information Sciences, 2012, 185(1):137-152.

[5] Shiba T, Tsuchiya T, Kikuno T. Using artificial life techniques to generate test cases for combinatorial testing [C]// Computer Software and Applications Conference. Proceedings of 25th International, Hong Kong. IEEE Computer Society,2004:72-77.

[6] Chen X, Gu Q, Li A, et al. Variable strength interaction testing with an ant colony system approach [C]// Proceedings of Software Engineering Conference, Penang. IEEE Computer Society, 2009:160-167.

[7] Chen X, Gu Q, Zhang X, et al. Building prioritized pairwise interaction test suites with ant colony optimization [C]// Proceedings of 9th International Conference on Quality Software, Jeju-do. IEEE Computer Society,2009:347-352.

[8] Mccaffrey J D. An empirical study of pairwise test set generation using a genetic algorithm[C]// Proceedings of Seventh International Conference on Information Technology, Las Vegas. IEEE Computer Society,2010: 992-997.

[9] Flores P, Cheon Y. PWiseGen: Generating test cases for pairwise testing using genetic algorithms [C]// Proceedings of IEEE International Conference on Computer Science and Automation Engineering, Shanghai. Institute of Electrical and Electronics Engineers, 2011:747-752.

[10] Bansal P, Mittal N, Sabharwal A, et al. Integrating greedy based approach with genetic algorithm to generate mixed covering arrays for pair-wise testing [C]// Proceedings of 7th International Conference on Contemporary Computing, Noida. IEEE Computer Society,2014:629-634.

[11] Chen X, Gu Q, Qi J, et al. Applying particle swarm optimization to pairwise testing[C]// Proceedings of 34th of Computer Software and Applications Conference, Seoul. IEEE Computer Society, 2010: 107-116.

[12] Ahmed B S, Zamli K Z. A variable strength interaction test suites generation strategy using Particle Swarm Optimization[J]. Journal of Systems & Software, 2011, 84(12):2171-2185.

[13] 包晓安, 杨亚娟, 张娜, 等. 基于自适应粒子群优化的组合测试用例生成方法[J]. 计算机科学, 2017, 44(6):177-181.

[14] 史娇娇, 姜淑娟, 韩寒, 等. 自适应粒子群优化算法及其在测试数据生成中的应用研究[J]. 电子学报, 2013, 41(8):1555-1559.

[15] Wu H, Nie C, Kuo F C, et al. A discrete particle swarm optimization for covering array generation[J]. IEEE Transactions on Evolutionary Computation, 2015, 19(4):575-591.

[16] Colbourn C J. One-test-at-a-time heuristic search for interaction test suites [C]// Proceedings of 9th Conference on Genetic and Evolutionary Computation, London. ACM,2007: 1082-1089.

(责任编辑: 康 锋)