软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(12):2879−2893 [doi: 10.3724/SP.J.1001.2011.03973]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于粒子群优化的成对组合测试算法框架*

陈 翔[1,2], 顾 庆[1,2+], 王子元[1,2], 陈道蓄[1,2]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

[2](南京大学 计算机科学与技术系,江苏 南京 210093)

Framework of Particle Swarm Optimization Based Pairwise Testing

CHEN Xiang[1,2], GU Qing[1,2+], WANG Zi-Yuan[1,2], CHEN Dao-Xu[1,2]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: guq@nju.edu.cn

Chen X, Gu Q, Wang ZY, Chen DX. Framework of particle swarm optimization based pairwise testing. *Journal of Software*, 2011,22(12):2879−2893. http://www.jos.org.cn/1000-9825/3973.htm

Abstract: This paper proposes a framework of particle swarm optimization (PSO) based pairwise testing. To systematically build pairwise test suites, two different PSO based strategies are proposed. One strategy takes on a one-test-at-a-time approach and the other takes on an IPO-like approach. In these two different strategies, PSO is used to complete the construction of a single test and research on how to formulate the search space, define the fitness function, and set some heuristic settings. To verify the effectiveness of this approach, these algorithms are implemented and some typical instances have been chosen. In this empirical study, the paper analyzes the impact factors of this framework and compares this approach to other well-known approaches in test suite size and generation time. Final empirical results show the competitiveness of this approach.

Key words: software testing; pairwise testing; meta-heuristic search; particle swarm optimization

摘 要: 提出一种基于粒子群优化的成对组合测试用例集生成算法框架.在生成测试用例时,该框架采用粒子群优化尝试生成强组合覆盖能力的测试用例,并研究了搜索空间、适应值函数和启发式的合理设定;在构造组合测试用例集时,以上述测试用例生成算法为基础,提出两种策略:一种基于 one-test-at-a-time,另一种基于类 IPO.编程实现该算法框架,并通过实证研究分析了算法框架中不同设定对组合测试用例集规模的影响;最后,与现有的经典方法在组合测试用例集生成规模和算法执行时间上进行了比较.最终结果表明,该算法具有竞争力.

关键词: 软件测试;成对组合测试;元启发式搜索;粒子群优化

中图法分类号: TP311 **文献标识码**: A

构建可信软件离不开高质量的软件测试,这个阶段是软件开发过程中的基本环节,但统计数据表明,该环节往往成本高昂[1].测试用例生成是软件测试中的一个研究热点,它主要考虑在程序输入空间内选择典型测试用

* 基金项目: 国家自然科学基金(60873027); 国家高技术研究发展计划(863)(2006AA01Z177); 国家重点基础研究发展计划(973)(2009CB320705)

收稿时间: 2010-07-15; 定稿时间: 2010-11-24

例来驱动被测程序,组合测试作为一种基于规约的测试用例生成技术,主要检测由不同因素间的组合所触发的软件缺陷.具体来说,通过对被测软件制品的分析,测试人员可以获得由因素和可选取值构成的实例.若需要充分测试,则对应的全组合测试用例集需要覆盖任意因素间所有可能取值组合.但当实例较为复杂,即拥有多个因素或其中部分因素拥有多个可选取值时,全组合测试用例集规模将过于庞大并导致所需测试成本超过项目实际预算.而组合测试通过关注部分因素间的组合,从而有效减小组合测试用例集规模.实证研究表明,被测程序内多数缺陷与小部分因素间的组合相关,故组合测试虽大规模缩减了测试用例集规模,但不会大幅度降低缺陷检测能力[2].

Harman 和 Jones 在 2001 年首次提出基于搜索的软件工程(search based software engineering,简称 SBSE)概念.该领域尝试采用元启发式搜索技术,以合理的计算开销求解一类可转化为组合优化问题的软件工程问题[3].演化测试是 SBSE 领域中的一个活跃分支,其尝试采用元启发式搜索技术辅助生成测试用例[4].组合测试作为一种重要的测试用例生成技术,目前已有研究人员采用模拟退火、蚁群算法和遗传算法来构造组合测试用例集[5,6],本文则采用粒子群优化进行构造.迄今为止,粒子群优化在软件测试领域中的成功应用较少,Windisch 等人将粒子群优化与遗传算法用于结构性软件测试并对两者进行了比较[7].与 Windisch 等人的研究工作不同,本文将粒子群优化与贪心法策略结合,提出一种基于粒子群优化的成对组合测试算法框架.

本文的主要贡献是:

(1) 提出一种基于粒子群优化的单个测试用例生成算法并探讨搜索空间、适应值函数和启发式的合理设定.

(2) 以上述单个测试用例生成算法为基础,提出一种成对组合测试用例集构造算法框架,该框架包含两种贪心策略:一种基于 one-test-at-a-time,另一种基于类 IPO.

(3) 采用 JAVA 编程实现算法框架,通过实证研究分析了算法框架中不同设定对测试用例集规模的影响并予以总结,同时,与现有经典方法在测试用例集生成规模和算法执行时间上进行了比较,验证了本文算法具有竞争力.

1 研究背景

1.1 组合测试

考虑图 1(a)中的系统兼容性测试实例,该实例拥有 4 个因素{Database server,Web server,Browser,Router},每个因素均有 3 个可选取值,例如,因素 Database Server 拥有的可选取值集为{Oracle,SQL server,MySQL}.若生成全组合测试用例集,则测试用例集规模为 3^4=81.若生成最优成对组合测试用例集(如图 1(b)所示),则规模仅为 9.在该测试用例集中,任意两个因素间的所有可选取值组合均至少被一个测试用例覆盖.组合测试的思想最早受制造业中的实验设计法启发[8].首次应用到软件测试中的是 Mandl,他在 1985 年采用正交矩阵测试 Ada 编译器[9].Cohen 等人提出组合测试概念并提出一个贪心算法 AETG[10].下面给出组合测试中涉及的定义.

定义 1(因素和可选取值). 拥有 k 个因素的实例可用 $F=\{f_1,f_2,\ldots,f_k\}$表示.采用常见的黑盒测试技术,如边界值分析和等价类划分,第 i 个因素 f_i 的可选取值集用 $V_i=\{v_{i,1},v_{i,2},\ldots,v_{i,l_i}\}$ 表示.

由定义 1 可知,第 i 个因素 f_i 拥有 l_i 个可选取值.在实际软件测试中,如何确定实例的因素和如何为每个因素确定合理的可选取值集,是组合测试成功应用的关键.此外,传统组合测试往往假设因素间相互独立,即不存在某一个因素的取值选择受其他因素取值选择的影响.

定义 2(t-way 组合覆盖准则). t-way 组合覆盖准则将生成测试需求集 TR,TR 包含任意 t 个因素间所有可选取值组合.

在定义 2 中,t 是组合覆盖准则的参数,在一些文献中又称为覆盖强度.当 t=1 时,称为单因素覆盖准则;t=2 时,称为成对组合覆盖准则;$t=k$ 时,称为全组合覆盖准则.随着覆盖强度的提高,相应生成的组合测试用例集的缺陷检测能力也将提高,但以增加测试用例集规模为代价.

定义 3(测试用例). 在测试用例集中,第 i 个测试用例 $Test_i$ 可用一个 n 元组表示:

$$Test_i=(v_1',v_2',...,v_k')(v_1'\in V_1,v_2'\in V_2,...,v_k'\in V_k).$$

在软件测试教材中,测试用例包括测试输入和预期输出[1].目前,组合测试主要关注测试输入,在实际软件测试中,预期输出可通过手工方式或程序分析技术辅助构建.

定义 4(组合测试用例集). 给定某一组合覆盖准则生成的测试需求集 *TR*,测试用例集 *T* 是组合测试用例集当且仅当对任意组合 *tr*(*tr*∈*TR*),至少存在一个测试用例 *t*(*t*∈*T*),使得 *t* 覆盖组合 *tr*.

给定一个实例和采用的组合覆盖准则,一个组合测试用例集若规模最小,则被称为最优组合测试用例集.最优组合测试用例集构造已被证实为是一个 NP 难问题,如 Lei 等人通过将顶点覆盖问题多项式时间内规约到最优成对组合测试用例集构建问题上予以证明[11].

当所有因素包含的取值数均一致时,组合测试用例集可抽象为数学对象覆盖矩阵(covering array),矩阵中的行对应为测试用例,列对应为因素.

定义 5(覆盖矩阵). 覆盖矩阵 *CA*(*N*;*t*,*k*,*v*)是一个 *N*×*k* 的矩阵,每列不同取值个数均为 *v*,任意 *N*×*t* 的子矩阵均覆盖对应的 *t* 个因素间构成的任意 *t* 元组至少一次[5].

但在实际测试场景中,不同因素包含的取值数很难保持一致.在这种情况下,通过对覆盖矩阵的扩展,组合测试用例集可抽象为数学对象混合水平覆盖矩阵(mixed level covering array).

定义 6(混合水平覆盖矩阵). 混合水平覆盖矩阵 *MCA*(*N*;*t*,*k*,(l_1,l_2,…,l_k))是一个 *N*×*k* 的矩阵,并满足如下性质:

(1) 第 *i* 列仅包含属于集合 v_i 的取值;

(2) 任意 *N*×*t* 的子矩阵均覆盖对应的 *t* 个因素间构成的任意 *t* 元组至少一次[5].

采用混合水平覆盖矩阵表示实例时,常将拥有取值数相同的因素合并.例如:若有 5 个因素均拥有 4 个候选取值,则可简写为 4^5.采用上述约定后,*MCA*(*N*;*t*,*k*,(l_1,l_2,…,l_k))可缩写为

$$MCA(N;t,(s_1^{r_1},s_2^{r_2},...,s_j^{r_j})),$$

其中,满足 $k=\sum_{i=1}^{j}r_i$.上述两个数学对象中,参数 *N* 的下界较易获得,例如对于覆盖矩阵,下界为 v^t.

Database server	Web server	Browser	Router
Oracle	Apache	IE	Cisco
SQL server	JBoss	FireFox	Huawei
MySQL	WebSphere	Chrome	Alcatel

(a) An instance
(a) 一个实例

Database server	Web server	Browser	Router
Oracle	Apache	IE	Cisco
Oracle	JBoss	FireFox	Huawei
Oracle	WebSphere	Chrome	Alcatel
SQL server	Apache	FireFox	Alcatel
SQL server	WebSphere	IE	Huawei
SQL server	JBoss	Chrome	Cisco
MySQL	Apache	Chrome	Huawei
MySQL	JBoss	IE	Alcatel
MySQL	WebSphere	FireFox	Cisco

(b) An optimal pairwise test suite
(b) 一个最优成对组合测试用例集

Fig.1 An example of a combinatorial test suite
图 1 一个组合测试用例集实例

1.2 粒子群优化

粒子群优化是一种基于种群的元启发式搜索技术,搜索策略主要受鸟群觅食行为的启发.Eberhart 和 Kennedy 在 1995 年首次将粒子群优化发展为一种经典的元启发式搜索技术[12].相对于其他元启发式搜索技术,如遗传算法和蚁群算法,粒子群优化实现更为方便、需要设定的参数更少且算法执行速度更快.下面依次给出粒子群优化相关定义及粒子群优化演化过程.

定义 7(粒子). 种群中的单个个体称为粒子.每个粒子均记录在当前迭代中的速度、位置、个体迄今为止最优位置(*pbest*$_i$)和群体迄今为止最优位置(*gbest*).

定义 8(粒子位置). 若将粒子搜索空间建模为一个 *N* 维空间,则粒子在第 *t* 次迭代中所处位置对应于 *N* 维

空间中的一个点,可用 N 维坐标表示.

定义 9(粒子适应值函数). 粒子适应值函数的合理设定,是粒子群优化成功应用关键.适应值用于比较不同粒子确定的候选解间的优劣.适应值函数的输入是粒子在当前迭代中所处位置,输出是该粒子确定的候选解的适应值.

粒子群优化的演化流程可大致描述为:首先初始化一个指定规模的种群,种群中每个粒子的速度和位置随机生成;然后,种群不断演化直至满足特定终止条件(终止条件通常为达到指定迭代次数或某个粒子取得指定适应值),并返回迄今为止已找到的最优解.在每一次迭代过程中,第 i 个粒子将依据速度更新公式和位置更新公式算出自己在下一次迭代中的速度和位置并评价对应适应值,若适应值优于 $pbest_i$ 对应的适应值,则用当前位置值更新 $pbest_i$ 值;若适应值优于 $gbest$ 对应的适应值,则用当前位置值更新 $gbest$ 值.

2 基于粒子群优化的成对组合测试算法框架

基于粒子群优化的成对组合测试算法框架支持两种组合测试用例集的生成策略:

- 策略 A 采用 one-test-at-a-time 策略;
- 策略 B 采用类 IPO 策略.

在这两种策略中,在单个测试用例生成时均采用粒子群优化,旨在覆盖更多新的组合.同时,给出一个随机算法用于生成组合测试用例集,并作为本文所提算法框架的比较对象.最后给出一个测试用例集约简算法,可以在保证原有覆盖能力前提下进一步缩减测试用例规模.

2.1 基于粒子群优化的单个测试用例生成算法

在本文提出的算法框架中,组合测试用例集的构造过程中往往会生成一些测试用例,这些测试用例仅有部分因素的取值确定.本节介绍如何采用粒子群优化完成单个测试用例的构建,即给未确定取值的因素选择合适的取值.

在应用粒子群优化前,需要对粒子所处的搜索空间进行建模.在组合测试中,给定一个实例,可用 k 维空间表示搜索空间.第 i 维代表第 i 个参数 f_i,第 i 维的有效取值集可用 $D_i=\{1,2,\ldots,l_i\}$表示.则在第 t 次迭代时,粒子 i 的位置矢量表示为 $Pos_i(t)=(x_{i,1},x_{i,2},\ldots,x_{i,k})(x_{i,1}\in D_1,x_{i,2}\in D_2,\ldots,x_{i,k}\in D_k)$.速度矢量表示为 $Vel_i(t)=(vel_{i,1},vel_{i,2},\ldots,vel_{i,k})$.这个粒子对应的测试用例为 $Test_i(t)=(v_{i,x_{i,1}},v_{i,x_{i,2}},...,v_{i,x_{i,k}})$.粒子的适应值函数返回该粒子对应的测试用例覆盖的新组合数.

基于粒子群优化的单个测试用例生成算法的伪代码见算法 1.算法 1 的输入为一个实例、粒子群的群体规模和一个已固定部分因素取值的测试用例.输出为所有因素取值均固定的测试用例.算法 1 中需注意的是,粒子仅在未固定取值因素对应的维上进行操作.

算法 1. 基于粒子群优化的单个测试用例生成算法.

输入:实例 F,粒子群群体规模 m,一个部分因素取值固定的测试用例;

输出:所有因素取值均固定的测试用例.

```
1  NC←0;   bestTest←NULL;
2  for (i=0; i<m; i++){   //初始化粒子群
3     随机初始化粒子 p_i 的位置和速度;
4     pBest_i←NULL;      //记录每个粒子个体最优位置
5  }
6  gBest←NULL;           //记录全体最优位置
7  while (NC<NC_max){
8     for (i=0; i<m; i++){
9        计算粒子 p_i 的适应值 f(p_i);
```

```
10      if (f(p_i)>f(pBest_i)) pBest_i←p_i;
11      if (f(p_i)>f(gBest)) gBest←p_i;
12      if (f(p_i)==(k×(k–1)/2)){bestTest←deriveTest(p_i); return bestTest;}
13      for (i=0; i<m; i++){
14          依据公式(1)更新粒子 p_i 的速度矢量并进行后处理;
15          依据公式(2)更新粒子 p_i 的位置矢量并进行后处理;
16      }
17      NC++;
18    }
19    bestTest←deriveTest(p_i); return bestTest;
20 }
```

算法 1 中采用的终止条件是:当找到一个最优解时,立刻返回(第 12 行);或未找到最优解,但迭代次数超过预设次数 $NC_{\max}$,返回迄今为止找到的最优解(第 17 行).在成对组合测试中,其中理论最优解是指覆盖新组合数量达到 $k\times(k-1)/2$ 的测试用例.

粒子群优化中各个粒子的速度矢量更新见公式(1),它综合考虑了粒子个体当前速度矢量、个体迄今为止最优位置矢量(*pBest*)和群体迄今为止最优位置矢量(*gBest*):

$$vel_{i,j}(t+1)=w\times vel_{i,j}(t)+c_1\times r_1\times[pBest_{i,j}(t)-x_{i,j}(t)]+c_2\times r_2\times[gBest_j(t)-x_{i,j}(t)] \tag{1}$$

其中,$vel_{i,j}(t)$表示在第 t 次迭代,第 i 个粒子在第 j 维的速度;$x_{i,j}(t)$表示第 i 个粒子在第 j 维的位置;w 代表惯性权值,用于平衡全局搜索和局部搜索,并且取值随着迭代的进行呈线性递减;c_1 和 c_2 代表比例因子,用于决定 $pBest_i$ 和 *gBest* 的影响力;随机变量 r_1 和 r_2 在取值区间 0 和 1 间符合均匀分布.由于成对组合测试属于离散型组合优化问题,本文将计算出的 $vel_{i,j}(t)$采用四舍五入进行取整并限定在$-l_i$和 l_i之间.即,若 $vel_{i,j}(t+1)<-l_i$,则 $vel_{i,j}(t+1)=-l_i$;若 $vel_{i,j}(t+1)>l_i$,则 $vel_{i,j}(t+1)=l_i$.

一旦粒子速度矢量确定,可计算出粒子在下一次迭代中的位置矢量.采用公式(2):

$$x_{i,j}(t+1)=x_{i,j}(t)+vel_{i,j}(t+1) \tag{2}$$

但若仅采用公式(2),粒子有时会飞出有效的搜索空间.为解决上述问题,本文采用 Robinson 等人提出的吸收墙和反射墙[13],并额外添加循环墙作为 3 种可选的边界处理策略.当粒子在第 i 维越界时,这 3 种墙的简单示意如图 2 所示,具体描述如下:

(1) 策略 1:吸收墙(absorbing walls). 当粒子超越某一维边界时,粒子被吸收到相应的边界上.吸收墙策略用公式(3)对粒子的位置作进一步处理:

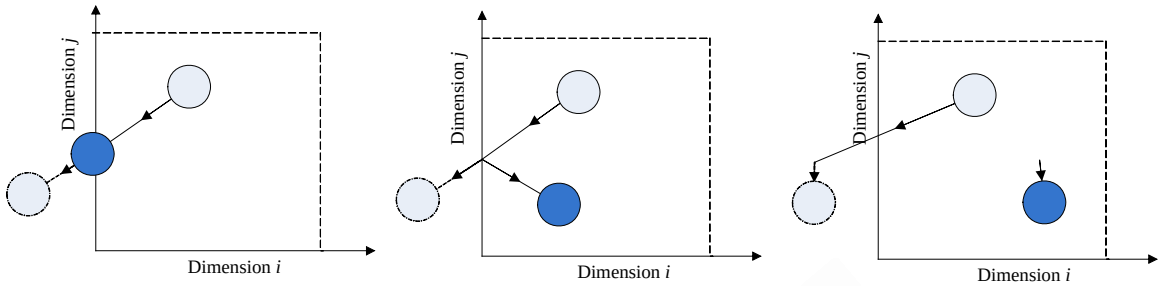
$$f(x_{i,j})=\begin{cases} l_i, & \text{如果} x_{i,j}>l_i \\ 1, & \text{如果} x_{i,j}<1 \\ x_{i,j}, & \text{其他} \end{cases} \tag{3}$$

(2) 策略 2:反射墙(reflecting wall). 当粒子超越某一维边界时,粒子被反弹回来.反射墙策略用公式(4)对粒子的位置作进一步处理:

$$f(x_{i,j})=\begin{cases} 2\times l_i-x_{i,j}+1, & \text{如果} x_{i,j}>l_i \\ (-1)\times x_{i,j}+1, & \text{如果} x_{i,j}<1 \\ x_{i,j}, & \text{其他} \end{cases} \tag{4}$$

(3) 策略 3:循环墙(cyclic wall). 当粒子超越某一维边界时,粒子从该维的另一侧出现并完成剩余的飞行距离.循环墙策略用公式(5)对粒子的位置作进一步处理:

$$f(x_{i,j})=\begin{cases} x_{i,j}-l_j, & \text{如果} x_{i,j}>l_i \\ x_{i,j}+l_j, & \text{如果} x_{i,j}<1 \\ x_{i,j}, & \text{其他} \end{cases} \tag{5}$$



(a) Absorbing walls (a) 吸收墙 (b) Reflecting walls (b) 反射墙 (c) Cyclic walls (c) 循环墙

Fig.2 A schematic diagram of three different walls

图 2 3 种不同墙的示意图

2.2 两种基于粒子群优化的组合测试用例集生成策略

算法 1 可以完成单个测试用例的构建,以该算法为基础,本文提出两种策略来系统的构造组合测试用例集:一种基于 one-test-at-a-time 策略,另一种基于类 IPO 策略.

2.2.1 策略 A:one-test-at-a-time 策略

AETG 算法最早采用 One-test-at-a-time 策略[10],其后,很多组合测试方法也采用该策略生成组合测试用例集[6,14−18].具体来说,该策略首先生成一个空的测试用例集 *TS*,然后依次生成单个测试用例 *t*,移除测试用例 *t* 覆盖的组合并添加该测试用例到 *TS* 中.当所有组合均被覆盖后,算法终止循环并返回 *TS*.本文在单个测试用例生成阶段进行了改造:首先,从集合 *Q* 中随机选择一个成对组合;然后生成一个测试用例,并根据这个组合固定相应因素的取值;最后,使用算法 1 确定其他因素的取值,从而完成该测试用例的构造(第 4 行~第 6 行).伪代码可参考算法 2.以图 1 所示输入为例,假设算法 2 首先从集合 *Q* 中随机选出组合⟨Apache,FireFox⟩并生成测试用例 *t*=(−,Apache,FireFox,−),其中,“−”表示相应因素取值并未确定.接着,算法将使用算法 1 确定这类因素取值,假设最终获得的测试用例是 *t*=(Oracle,Apache,FireFox,Cisco).然后,算法将从集合 *Q* 中移去 *t* 覆盖的组合,并将 *t* 添加到测试用例集中.当集合 *Q* 不为空时将重复上述过程,直到所有组合均被覆盖.

算法 2. one-test-at-a-time 策略.

输入:实例 *F*;

输出:测试用例集 *TS*.

```
1  TS←∅;
2  将实例 F 的所有成对组合置入集合 Q;
3  while (Q≠∅){
4     从集合 Q 中随机选出一个组合 c;
5     生成测试用例 t,并根据 c 确定相应因素的取值;
6     使用算法 1 确定其他因素的取值;
7     算得测试用例 t 覆盖的组合 Qt;Q←Q−Qt;
8     TS←TS∪{t};
9  }
10 return TS;
```

2.2.2 策略 B:类 IPO 策略

Tai 和 Yu 首次提出 IPO 策略,该策略包括两个阶段:水平扩展阶段和垂直扩展阶段[11].本文借鉴传统的 IPO 策略并做了较大改动.首先,依据实例中因素包含的取值数对因素进行非递增排序,假设排序后第 1 个因素包含 l_1' 个候选取值,第 2 个因素包含 l_2' 个候选取值.然后生成 $n(n=l_1'\times l_2')$ 个测试用例,并保证前两个因素构成的任意

成对组合均被这前 n 个测试用例覆盖.接着依次取出测试用例,利用算法 1 确定其他(k–2)个因素的取值(第 4 行~第 7 行).这个阶段类似于 IPO 的水平扩展阶段.最后,通过添加额外测试用例保证覆盖剩余的成对组合(第 8 行~第 14 行).这个阶段类似于 IPO 的垂直扩展阶段.伪代码可参考算法 3.同样以图 1 所示输入为例,算法 3 将直接生成 9 个测试用例:{(Oracle,Apache,－,－),(Oracle,JBoss,－,－),…,(MySQL,WebSphere,－,－)}.这 9 个测试用例可以保证对前两个因素所有取值组合的完全覆盖.然后依次取出每个测试用例,并使用算法 1 确定后两个因素的取值.接着,从集合 Q 中移去该测试用例覆盖的组合.执行完水平扩展阶段后,若集合 Q 仍非空,则进入垂直扩展阶段,并采用算法 2 依次添加测试用例直至所有组合均被覆盖.

算法 3. 类 IPO 策略.

输入:实例 F;

输出:测试用例集 TS.

1 $TS\leftarrow\varnothing$; 将所有成对组合置入集合 Q;
2 依据因素包含的取值数进行非递增排序,假设排序后第 1 个因素包含 l_1' 个候选取值,第 2 个因素包含 l_2' 个候选取值;
3 生成 n 个测试用例,并保证前两个因素构成的任意组合均被覆盖,将 n 个测试用例添加到 TS;
4 **for** (i=0; i<n; i++){ //水平增长阶段
5 取出第 i 个测试用例 t,并使用算法 1 确定其他(k–2)个因素的取值;
6 算得测试用例 t 覆盖的组合 Q_t;$Q\leftarrow Q-Q_t$;
7 }
8 **while** ($Q\neq\varnothing$){ //垂直增长阶段
9 从集合 Q 中随机选出一个组合 c;
10 生成测试用例 t,并根据 c 确定相应因素的取值;
11 使用算法 1 确定其他因素的取值;
12 算得测试用例 t 覆盖的组合 Q_t;$Q\leftarrow Q-Q_t$;
13 $TS\leftarrow TS\cup\{t\}$;
14 }
15 **return** TS;

2.3 随机算法

本文提出一个随机算法生成组合测试用例集,并作为初始比较对象.该算法采用 one-test-at-a-time 策略.在生成单个测试用例时,先从集合 Q 中随机取出一个成对组合 c,确定相应因素取值;然后,在该测试用例基础上生成 $NC_{\max}$ 个候选解,对这些候选解中剩余的未固定取值因素随机确定相应取值,并从中选择一个覆盖能力最强的测试用例添加到 TS 中去.伪代码可参考算法 4.

算法 4. 随机算法.

输入:实例 F;

输出:测试用例集 TS.

1 $TS\leftarrow\varnothing$; 将所有成对组合置入集合 Q;
2 **while** ($Q\neq\varnothing$){
3 从集合 Q 中随机选出一个组合 c; 生成测试用例 t,并根据 c 确定相应因素的取值;
4 $NC\leftarrow$0;$bestTest\leftarrow null$;$Q_{best}\leftarrow\varnothing$;
5 **while** (NC<$NC_{\max}$){
6 随机确定测试用例 t 其他因素的取值; 算得测试用例 t 覆盖的新组合 Q_t;
7 **if** ($|Q_t|$>$|Q_{best}|$){$bestTest\leftarrow t$;$Q_{best}\leftarrow Q_t$;}
8 **if** ($|Q_t|$==(k×(k–1)/2)){$bestTest\leftarrow t$;$Q_{best}\leftarrow Q_t$; **break**;}

```
9        NC++;
10    }
11    Q←Q−Q_best;TS←TS∪{bestTest};
12 }
13 return TS;
```

2.4 组合测试用例集约简算法

上述两种测试用例集生成策略和随机算法均可采用约简算法进一步缩减测试用例集规模.本节提出约简算法受文献[6]中压缩算法启发并作了适当改造.与文献[6]中的算法不同,本节提出的算法具有确定性(即针对同一测试用例集,算法独立执行多次,返回结果均一致)且操作性更强.伪代码见算法 5.

算法 5. 组合测试用例集约简算法.

输入:约简前的测试用例集 *TS*,*TS* 的规模 *m*;

输出:约简后的测试用例集 *TS*.

```
1  while (TRUE){
2     for (i=1;i≤m;i++){   //阶段 1:识别出所有 don't care 取值
3        识别出第 i 个测试用例 ts_i 中的 don't care 取值;
4     }
5     flag←FALSE;
6     for (j=m;j≥1;j−−){   //阶段 2:合并测试用例
7        for (k=j−1;k≥1;k−−){
8           if (测试用例 ts_k 和测试用例 ts_j 满足合并规则){
9              依据 ts_j 更新 ts_k;从 TS 中移除 ts_j;flag←TRUE; goto LABEL;
10          }
11       }
12    }
13    LABEL:
14       if (flag==TRUE) continue; else break;
15 }
16 return TS;
```

在阶段 1 中先识别出测试用例集中所有 don't care 取值,don't care 取值是指在该取值所处的测试用例中,用其他有效取值代替该取值,并不会影响测试用例原有覆盖能力.阶段 2 尝试采用合并规则压缩测试用例集.合并规则为:若两个测试用例除了拥有 don't care 取值的因素,其他因素取值均一致,则这两个测试用例可合并为单个测试用例.当不存在符合合并规则的成对测试用例时,算法终止并返回约简后的测试用例集 *TS*.

3 实证研究

3.1 实验设计和研究问题

为了评价本文提出的算法框架,我们采用 JAVA 语言(JDK 1.6)编程实现了文中的算法,算法运行的操作系统是 Windows 7,硬件平台是 Thinkpad T400(Intel CPU P8400 处理器,3G 内存).由于不能穷尽所有实例,文中采用了若干典型实例,这些实例被研究人员广泛用于评价各自提出方法的有效性.实例中,有的属于覆盖矩阵,有的属于混合水平覆盖矩阵.在本文提出的算法框架中,对粒子群优化中各个设计参数的取值选择不在讨论范围之内,表 1 列出了本文实证研究中采用的参数取值,这些取值同时参考了文献[12]和实际实验效果.

Table 1 Design parameters in particle swarm optimization

表 1 与粒子群优化相关的设计参数取值

Design parameter	Value
Number of particles m	10
Inertia weight w	Linear decreased from 0.9 to 0.4
Coefficient c_1 and c_2	2

本文的实证研究尝试回答以下 3 个研究问题:

问题 1:分析不同的组合测试用例集构造策略和不同的边界处理策略的组合对生成的测试用例集规模的影响,同时分析测试用例集约简算法的效果;

问题 2:分析迭代次数的设置对测试用例集规模的影响;

问题 3:在测试用例集生成规模和执行时间上对本文提出的算法和其他经典组合测试算法进行比较.

3.2 数据分析

由于本文提出的算法框架中存在随机影响因素,故在实证研究中,针对每组参数设定均独立运行 10 次.

本文通过表 2 和图 3 回答问题 1.这里,迭代次数均设置为 1 000.其中,表 2 比较了不同组合测试用例集生成策略、不同边界处理策略和测试用例集约简算法对最终生成的测试用例集规模的影响.PSO_A_1 表示在生成测试用例集上采用策略 A,在边界处理策略上采用策略 1.本文主要有两种组合测试用例集生成策略:策略 A(即 one-test-at-a-time 策略)和策略 B(即类 IPO 策略),同时有 3 种边界处理策略:策略 1(即吸收墙策略)、策略 2(即反射墙策略)和策略 3(即循环墙策略).表中每一个单元格内表示的是不同实例在不同策略组合下最终生成的测试用例集规模的均值.例如:如果单元格内数据为 19.7+0.3,则表示采用约简算法后规模均值为 19.7,未采用约简算法时规模均值为 20.图 3 采用盒图(boxplot)描述了数据的分布情况,其中,盒内的小矩形表示该组数据的均值.图 3 中盒图的水平轴表示不同测试用例集生成策略和不同边界处理策略的组合,垂直轴代表采用约简算法后得到的测试用例集规模.

Table 2 Test suite size comparison in different strategies, different walls and test suite reduction algorithm

表 2 比较不同测试用例集生成策略、不同边界处理策略、约简算法对测试用例集规模的影响

ID	Instance	Random	PSO_A_1	PSO_A_2	PSO_A_3	PSO_B_1	PSO_B_2	PSO_B_3
1	$CA(N;2,4,3)$	10.5+0.0	11.4+0.0	10.9+0.0	11.2+0.0	9.0+0.0	9.0+0.0	9.0+0.0
2	$CA(N;2,10,10)$	193.3+1.6	165.1+0.3	160.9+0.0	179.3+1.3	161.9+0.3	158.1+0.1	177.4+0.8
3	$CA(N;2,20,10)$	282.3+5.0	221.4+1.1	214.3+0.6	266.1+2.5	220.2+0.9	214.1+0.1	266.6+2.0
4	$CA(N;2,13,3)$	19.8+0.4	19.4+0.0	19.1+0.0	19.7+0.3	19.5+0.0	19.4+0.0	19.9+0.0
5	$CA(N;2,100,2)$	15.1+1.0	14.7+1.0	14.0+0.6	14.6+0.0	14.7+0.7	14.1+0.7	14.3+0.1
6	$MCA(N;2,5^1 3^8 2^2)$	22.0+0.1	22.6+0.2	21.7+0.0	21.6+0.1	20.0+0.0	18.6+0.0	19.1+0.1
7	$MCA(N;2,7^1 6^1 5^1 4^5 3^8 2^3)$	52.0+0.0	52.2+0.0	52.2+0.0	51.1+0.0	46.1+0.0	45.1+0.0	45.9+0.2
8	$MCA(N;2,5^1 4^4 3^{11} 2^5)$	32.7+0.0	31.5+0.0	31+0.0	32.7+0.1	29.4+0.0	28.6+0.0	28.8+0.1
9	$MCA(N;2,6^1 5^1 4^6 3^8 2^3)$	41.4+0.1	40.6+0.0	39.4+0.0	40+0.0	36.2+0.1	35.8+0.0	36.1+0.3
10	$MCA(N;2,4^{15} 3^{17} 2^{29})$	44.1+0.7	39.8+0.2	38.4+0.0	41.3+0.2	39.5+0.2	38.9+0.0	41.0+0.6
11	$MCA(N;2,4^1 3^{39} 2^{35})$	32.8+1.1	30.1+0.6	28.9+0.0	31.1+0.5	30.0+0.3	27.4+0.1	30.3+0.5

首先,分析约简算法对测试用例集规模的影响.从表 2 中可以看出,约简算法往往都能取得一定效果.当实例较为复杂或生成的测试用例集规模较大时,例如实例 2、实例 3、实例 10 和实例 11,约简算法在缩减测试用例集规模时效果比较显著,往往能约简 2~3 个;但当实例较为简单时,例如实例 1,约简算法很难进一步缩减测试用例集规模.接着,分析不同测试用例集生成策略对结果的影响.从表 2 和图 3 中发现,在大部分实例中,策略 B 要优于策略 A,尤其是面对因素取值数并不一致的实例时(例如实例 6~实例 9);在一些因素取值数均一致的实例中(例如实例 4、实例 5),策略 B 的优势并不非常明显.其次,分析 3 种边界处理策略对结果的影响.从表 2 和图 3 中不难发现:在大部分实例中,反射墙要优于吸收墙和循环墙,而循环墙在 3 种边界处理策略中效果最差.最后,对本文所提的随机算法进行分析比较.在大部分实例中,随机算法生成的测试用例集规模都要高于本文提出的基于粒子群优化的算法,尤其在较为复杂的实例中(例如实例 2、实例 3),这种差距更为明显.但在一些较为简单的实例中或最终生成的测试用例集规模较小的情况下,本文给出的随机算法也能取得不错的效果,即在最终生成

的测试用例集规模上与基于粒子群优化算法差距并不大;而且通过表 4 可以看出,随机算法在测试用例集生成时间上相对其他方法具有明显的优势.通过上述分析可以看出,采用类 IPO 策略生成组合测试用例集,在处理边界越界情况下,采用反射墙策略往往生成的组合测试用例集规模较小.当实例较为复杂或最终生成的测试用例集规模较大时,可以采用本文提出的约简算法对测试用例集做进一步压缩.

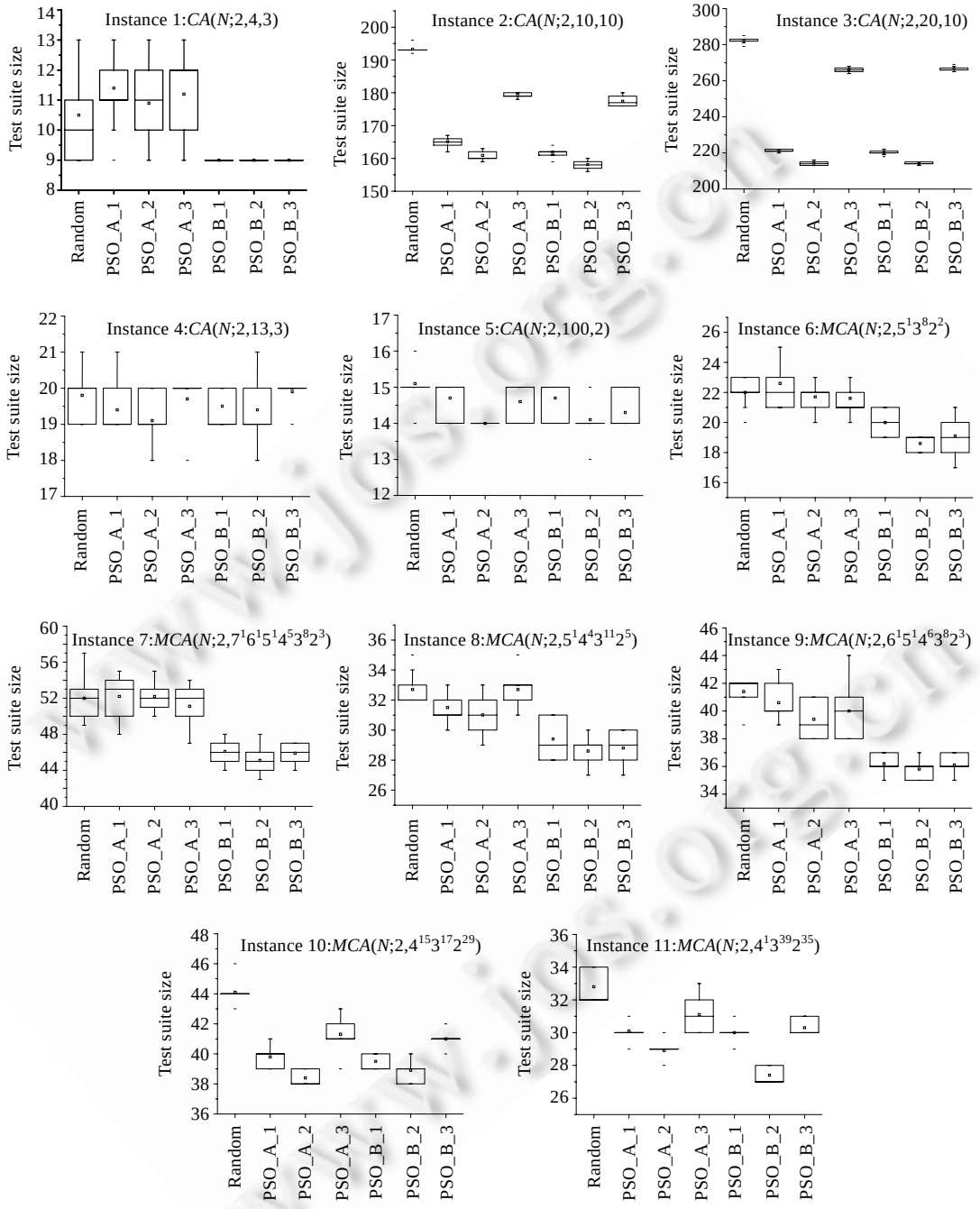


Fig.3 Distribution of test suite size in different strategies

图 3 不同策略下组合测试用例集规模分布

本文通过图 4 回答问题 2.这里采用的是 PSO_B_2 的组合,并结合使用约简算法.图 4 主要考虑了实例 2、实例 3、实例 7、实例 9~实例 11,并采用盒图描述了最终数据集的分布情况.采用的这几个实例在复杂度上要高于其他实例,所以更容易观察出迭代次数对最终结果的影响.图 4 考虑的迭代次数主要包括 50,100,200,500,1 000 和 1 500.当实例较为复杂时(例如实例 2、实例 3),增加迭代次数可以显著降低测试用例集规模.但同时,在一些较为简单的实例中,过度的增加迭代次数反而会增加测试用例集规模.出现这种情况的原因是,如果早期生成的测试用例的组合覆盖能力过强,可能会影响到后期测试用例的构造.

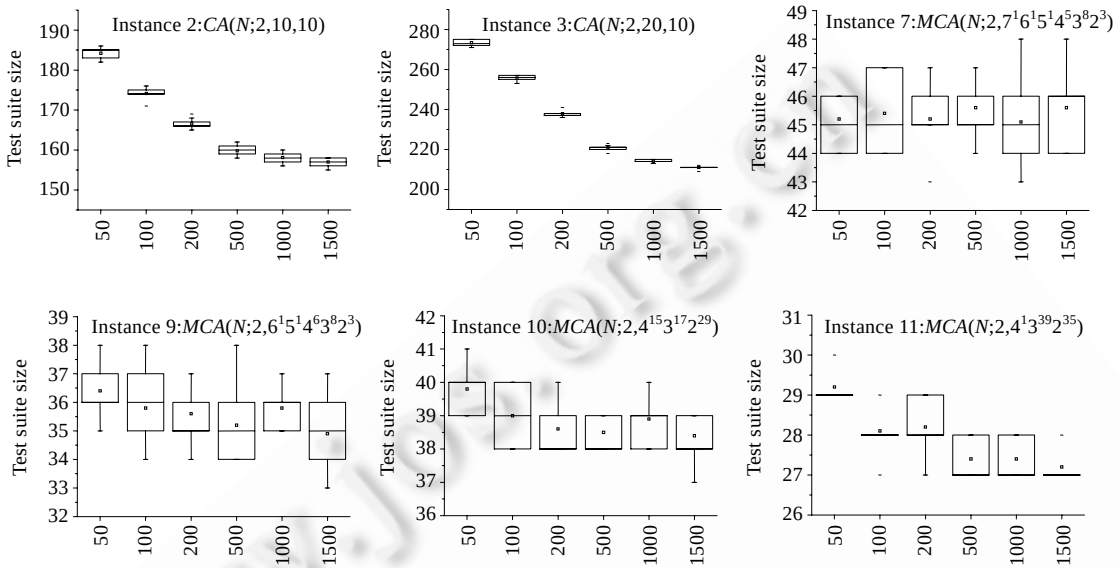


Fig.4 Distribution of test suite size in different iteration number

图 4 不同迭代次数下组合测试用例集规模分布

本文通过表 3 和表 4 回答问题 3.这里采用的也是 PSO_B_2 的组合并结合使用约简算法,设置的迭代次数为 1 000.主要比较对象是 AETG[5]、DDA[15]、IPO[11]、PSST[17]、TConfig[18]、模拟退火(SA)[5]、蚁群优化算法(ACA)[6]和遗传算法(GA)[6].表 3 列出了这些算法算得的最优规模,这些数据来自于上述文献,单元格若标记为 N/A,则代表相关文献中没有针对该实例的结果.从表 3 中可以发现,模拟退火总能生成最小规模测试用例集并可作为其他方法的基线,这主要与模拟退火策略相关.该策略首先指定一个稍大的测试用例集规模,采用模拟退火策略检测能否生成相应的组合测试用例集,接着采用二分法确定下一个规模.模拟退火策略的缺点是计算量大且耗时长,不仅需要检测多个候选规模,而且即使在指定规模下,要获得相应的组合测试用例集也需要更长的搜索时间.表 3 同时表明,除了模拟退火,没有一种方法总是优于其他方法.一般情况下,元启发式搜索策略生成的测试用例集规模要小于贪心法,但同时也需要更长的执行时间.在元启发式搜索策略中,本文提出的基于粒子群优化的算法与其他同类方法(如遗传算法和蚁群算法)相比,测试用例集规模相近.在一些较为复杂的实例中(例如实例 2、实例 3),粒子群优化算法要优于遗传算法和蚁群算法;若增加迭代次数,则可以获得更小的测试用例集规模.例如,当迭代次数设置为 1 500 次时,实例 2 的最优规模可为 155,实例 3 的最优规模可为 209.表 4 对部分算法在测试用例集生成时间上进行比较.由于无法获得相应方法的源代码,我们只能根据文献中给出的数据进行比较,同时,表 5 列出了这些方法采用的编程语言和运行平台.虽然算法运行的平台存在差异,但从表 4 中可以看出,本文提出的算法在执行时间上要优于其他方法.

Table 3 Comparison in test suite size for different approaches

表 3 不同方法生成的测试用例集规模比较

ID	Instance	AETG	DDA	IPO	PSST	TConfig	SA	GA	ACA	Random	PSO
1	$CA(N;2,4,3)$	9	N/A	9	9	9	9	9	9	9	9
2	$CA(N;2,10,10)$	N/A	N/A	169	N/A	N/A	N/A	157	159	192	156
3	$CA(N;2,20,10)$	198	201	212	N/A	231	183	227	225	279	213
4	$CA(N;2,13,3)$	17	18	17	20	15	16	17	17	19	18
5	$CA(N;2,100,2)$	N/A	15	15	16	14	N/A	12	13	14	13
6	$MCA(N;2,5^1 3^8 2^2)$	20	21	N/A	N/A	21	15	15	16	20	17
7	$MCA(N;2,7^1 6^1 5^1 4^5 3^8 2^3)$	44	43	N/A	N/A	91	42	42	42	49	43
8	$MCA(N;2,5^1 4^4 3^{11} 2^5)$	28	27	N/A	N/A	32	21	26	25	32	27
9	$MCA(N;2,6^1 5^1 4^6 3^8 2^3)$	35	34	N/A	N/A	50	30	33	32	39	35
10	$MCA(N;2,4^{15} 3^{17} 2^{29})$	37	35	34	N/A	40	30	37	37	43	38
11	$MCA(N;2,4^1 3^{39} 2^{35})$	27	27	26	30	30	21	27	27	32	27

Table 4 Comparison in generation time for different approaches (s)

表 4 不同方法生成测试用例集时间比较(秒)

Instance	AETG	IPO	SA	GA	ACA	Random	PSO
$CA(N;2,10,10)$	N/A	0.3	N/A	886	1 180	7	65
$CA(N;2,20,10)$	6 001	N/A	10 833	6 365	7 083	104	715
$MCA(N;2,5^1 3^8 2^2)$	58	N/A	214	22	31	0.4	4
$MCA(N;2,7^1 6^1 5^1 4^5 3^8 2^3)$	489	N/A	874	207	258	6	58
$MCA(N;2,5^1 4^4 3^{11} 2^5)$	368	N/A	579	124	154	4	40
$MCA(N;2,6^1 5^1 4^6 3^8 2^3)$	359	N/A	13 495	1 787	1 881	4	42

Table 5 Implementation detail of different approaches

表 5 不同方法的实现细节

Approach	Programming language	Running platform
AETG and SA	C++	Linux, Intel P4 1.8GHZ
IPO	JAVA	Win 98, Intel P2 450MHZ
GA and ACA	C	Win XP, Intel P4 2.26GHZ
Random and PSO	JAVA	Win 7, Intel P8400, 2.27GHZ

3.3 有效性影响因素分析

本节对威胁实证研究有效性的影响因素进行分析.首先分析内部有效性,主要包括 3 个因素:(1) 更好的参数取值组合有助于获得更好的结果,但如何确定更好的参数取值组合仍是粒子群优化研究中的一个热点;(2) 本文仅考虑了传统粒子群优化,可考虑采用粒子群优化变体以期获得更好的效果;(3) 本文的算法实现可能存在缺陷,我们通过编写验证程序和相互检查来避免缺陷的出现.其次分析外部有效性.本文仅选择了部分实例进行研究,但这些实例已普遍被其他方法用来作评价,从而保证了它们的代表性.最后分析结论有效性.为了更好地分析算法中存在的随机因素,可考虑采用更为复杂的统计假设实验方法.

4 相关研究工作

4.1 组合测试

目前,国内外学者提出许多传统组合测试方法和工具,这些方法和工具可大致分为 3 类[19]:代数构造法、贪心法和元启发式搜索.目前,组合测试已经被成功应用于不同应用领域的测试研究中,如 Web 应用测试[20]、回归测试[21]、可配置软件测试[22,23]和缺陷定位[23].

代数构造法:当参数 t,k 和 v 满足一定条件时,可以直接采用正交矩阵进行构造并有时可生成最优组合测试用例集[8,9].Williamas 开发了 TConfig 工具,它以正交矩阵为基础,通过 5 个基本构造块递归地构造组合测试用例集[18].该工具适用于任意实例且执行速度较快,但生成的测试用例集往往规模偏大.

贪心法:这类方法可分为两类:(1) One-test-at-a-time 策略:Cohen 等人开发了一种基于 Web 服务的商业工具 AETG[10],Bryce 和 Colbourn 提出一种确定的基于密度的算法 DDA[15],史亮等人提出 PSST 法,采用解空间树表

示实例的搜索空间,然后通过搜索解空间树来生成单个测试用例[17];(2) IPO 策略:Tai 和 Yu 提出的 IPO 法属于另一种贪心策略,并开发了工具 FireEye.具体来说,首先为前两个因素生成满足成对组合覆盖准则的测试用例集,然后通过扩展测试用例集来覆盖前 3 个因素间的成对组合.如此反复,直到覆盖所有因素.在每一次扩展时,均包括两个阶段:水平扩展阶段和垂直扩展阶段[11].

元启发式搜索技术:此技术已成功应用到组合测试用例集的构建中.依据构建特点可分为两类: (1) One-test-at-a-time 策略:在确定单个测试用例时,Shiba 等人采用遗传算法和蚁群算法[6],Bryce 等人则提出一种混合策略,即先用近似法生成候选解,然后分别采用爬山法、模拟退火、禁忌搜索和大洪水算法策略转换该候选解,以提高新组合覆盖能力[14];(2) 直接生成策略:Cohen 等人采用模拟退火策略直接生成组合测试用例集[5].虽然现有元启发式搜索技术在大部分情况下可获得更小规模的组合测试用例集,但往往以牺牲时间为代价.除上述元启发式搜索外,严俊等人将组合测试问题转化为可满足问题(SAT),使用 Shatter 工具减小问题的搜索空间,然后调用 SAT 求解工具 Zchaff 进行处理[24].

除了上述传统组合测试问题,还有其他变种问题同样值得关注,主要包括:(1) 测试人员在实际的软件测试中依据自身的经验,希望特定的测试用例或组合必须出现在测试用例集中,这种测试用例或组合被称为种子(seed)[10].目前,AETG[10]和 DDA[25]均对种子提供支持;(2) 测试用例优先级技术通常针对一个特定目标将测试用例按照重要程度依次排序,使得高优先级的测试用例优先执行.目前,测试用例优先级技术已引入到组合测试中[25];(3) 在实际的测试场景中,不同因素子集间的覆盖强度往往并不一致,对一些关键的因素子集,测试人员往往需要更高的覆盖强度.这类问题被称为可变强度组合测试[26];(4) 在实际测试中,因素间存在的依赖关系将生成一系列约束,并提高了组合测试用例集生成难度.Cohen 等人将约束处理机制引入到传统的组合测试用例集生成方法[22];(5) Cohen 和 Porter 等人采用决策树方法分析缺陷特征,他们的实证研究表明,基于较小覆盖强度的组合测试用例集的训练结果和基于全组合测试用例集的训练效果基本相当[23].

4.2 基于粒子群优化的软件测试

相对于遗传算法和模拟退火,粒子群优化在软件测试中的应用刚起步.Windisch 等人将粒子群优化应用于基于结构的演化测试中,他们在一些小规模的程序对象和一些稍复杂的来自工业界的程序对象上搜集了执行结果,说明粒子群优化在代码覆盖和执行效率上要优于遗传算法[7].

5 总结与展望

本文提出一种基于粒子群优化的成对组合测试算法框架.该框架提供两种测试用例集生成策略:一种基于 one-test-at-a-time 策略,另一种基于类 IPO 策略.在生成单个测试用例时,采用粒子群优化策略,旨在提高单个测试用例的新组合覆盖能力.实证研究分析了算法框架中的存在因素对测试用例集规模的影响,同时,通过在测试用例集生成规模和执行时间上与其他经典方法相比,验证本文所提方法具有竞争力.

下一步的研究工作包括:(1) 采用并行编程方式实现本文提出的算法,进一步缩减测试用例集生成时间;(2) 在实际软件开发过程中验证本文所提方法的缺陷检测能力和效果;(3) 扩展本文方法支持其他组合测试变种问题,如可变强度组合测试、基于优先级的组合测试和更高覆盖强度的组合测试等.

References:

[1] Ammann P, Offutt J. Introduction to Software Testing. Cambridge: Cambridge University Press, 2008.

[2] Kuhn DR, Wallace DR, Gallo AM. Software fault interactions and implications for software testing. IEEE Trans. on Software Engineering, 2004,30(6):418−421. [doi: 10.1109/TSE.2004.24]

[3] Harman M, Jones BF. Search-Based software engineering. Information and Software Technology, 2001,43(14):833−839. [doi: 10.1016/S0950-5849(01)00189-6]

[4] McMinn P. Search-Based software test data generation: A survey. Software Testing, Verification and Reliability, 2004,14(2): 105−156. [doi: 10.1002/stvr.294]

[5] Cohen MB, Colbourn CJ, Gibbons PB, Mugridge WB. Constructing test suites for interaction testing. In: Proc. of the 25th Int'l Conf. on Software Engineering. Portland: ACM Press, 2003. 38−48. [doi: 10.1109/ICSE.2003.1201186]

[6] Shiba T, Tsuchiya T, Kikuno T. Using artificial life techniques to generate test cases for combinatorial testing. In: Proc. of the Int'l Conf. on Computer Software and Applications Conf. Hong Kong: IEEE Press, 2004. 72−77. [doi: 10.1109/CMPSAC.2004.1342808]

[7] Windisch A, Wappler S, Wegener J. Applying particle swarm optimization to software testing. In: Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation. London: ACM Press, 2007. 1121−1128. [doi: 10.1145/1276958.1277178]

[8] Hedayat AS, Sloane NJA, Stufken J. Orthogonal Arrays: Theory and Applications. Springer-Verlag, 1999.

[9] Mandl R. Orthogonal latin squares: An application of experiment design to compiler testing. Communications of the ACM, 1985, 28(10):1054−1058. [doi: 10.1145/4372.4375]

[10] Cohen DM, Dalal SR, Fredman ML, Patton GC. The AETG system: An approach to testing based on combinatorial design. IEEE Trans. on Software Engineering, 1997,23(7):437−444. [doi: 10.1109/32.605761]

[11] Tai KC, Lei Y. A test generation strategy for pairwise testing. IEEE Trans. on Software Engineering, 2002,28(1):109−111. [doi: 10.1109/32.979992]

[12] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Neural Networks. Perth: IEEE Press, 1995. 1942−1948. [doi: 10.1109/ICNN.1995.488968]

[13] Robinson J, Rahmat-Samii Y. Particle swarm optimization in electromagnetics. IEEE Trans. on Antennas and Propagation, 2004, 52(2):397−407. [doi: 10.1109/TAP.2004.823969]

[14] Bryce RC, Colbourn CJ. One-Test-at-a-Time heuristic search for interaction test suites. In: Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation. London: ACM Press, 2007. 1082−1089. [doi: 10.1145/1276958.1277173]

[15] Bryce RC, Colbourn CJ. The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability, 2007, 17(3):159−182. [doi: 10.1002/stvr.365]

[16] Bryce RC, Colbourn CJ, Cohen MB. A framework of greedy methods for constructing interaction tests. In: Proc. of the 27th Int'l Conf. on Software Engineering. St. Louis: ACM Press, 2005. 146−155. [doi: 10.1145/1062455.1062495]

[17] Shi L, Nie CH, Xu BW. Pairwise test data generation based on solution space tree. Chinese Journal of Computers, 2006,29(6): 849−857 (in Chinese with English abstract).

[18] Williams AW. Determination of test configurations for pair-wise interaction coverage. In: Proc. of the Testing of Communicating System: Tools and Techniques. Netherland: IEEE Press, 2000. 59−74.

[19] Yan J, Zhang J. Combinatorial testing: principles and methods. Journal of Software, 2009,20(6):1393−1405 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3497.htm

[20] Sampath S, Bryce RC, Viswanath G, Kandimalla V, Koru AG. Prioritizing user-session-based test cases for Web application testing. In: Proc. of the 1st Int'l Conf. on Software Testing, Verification, and Validation. Lillehammer: IEEE Press, 2008. 141−150. [doi: 10.1109/ICST.2008.42]

[21] Qu X, Cohen MB, Woolf KM. Combinatorial interaction regression testing: A study of test case generation and prioritization. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. Paris: IEEE Press, 2007. 255−264. [doi: 10.1109/ICSM.2007.4362638]

[22] Cohen MB, Dwyer MB, Shi JF. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Trans. on Software Engineering, 2008,34(5):633−650. [doi: 10.1109/TSE.2008.50]

[23] Yilmaz C, Cohen MB, Porter AA. Covering arrays for efficient fault localization in complex configuration space. IEEE Trans. on Software Engineering, 2006,32(1):20−34. [doi: 10.1109/TSE.2006.8]

[24] Yan J, Zhang J. A backtracking search tool for constructing combinatorial test suites. Journal of System and Software, 2008,81(10): 1681−1693. [doi: 10.1016/j.jss.2008.02.034]

[25] Bryce RC, Colbourn CJ. Prioritized interaction testing for pair-wise coverage with seeding and constraints. Information and Software Technology, 2006,48(10):960−970. [doi: 10.1016/j.infsof.2006.03.004]

[26] Cohen MB, Gibbons PB, Mugridge WB, Colbourn CJ, Collofello JS. Variable strength interaction testing of components. In: Proc. of the 27th Int'l Computer Software and Applications Conf. Dallas: IEEE Press, 2003. 413−418. [doi: 10.1109/CMPSAC.2003.1245373]

附中文参考文献:

[17] 史亮,聂长海,徐宝文.基于解空间树的组合测试数据生成.计算机学报,2006,29(6):849−857.

[19] 严俊,张健.组合测试:原理与方法.软件学报,2009,20(6):1393−1405. http://www.jos.org.cn/1000-9825/3497.htm

陈翔(1980－),男,江苏南通人,博士生,CCF 会员,主要研究领域为软件测试,程序分析.

王子元(1982－),男,博士,主要研究领域为软件测试.

顾庆(1972－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为软件质量保障.

陈道蓄(1947－),男,教授,博士生导师,CCF 会员,主要研究领域为分布式计算,并行处理.