

组合测试研究进展

陈 翔 顾 庆 王新平 陈道蓄

(南京大学计算机科学与技术系 计算机软件新技术国家重点实验室 南京 210093)

摘 要 组合测试(Interaction Testing)是一种有效的测试用例生成技术,它假设被测系统中的缺陷往往由少数参数间的交互触发,所以可以使用较小规模的测试用例集(Test Suite)完成高质量的软件测试。对目前组合测试的已有工作予以研究,首先通过介绍常见的组合覆盖标准引入组合测试问题;然后总结目前常见的 4 种组合对象;之后依据生成策略的特征对现有的组合测试技术予以分类并简要介绍其中的典型方法;接着给出一套评价标准并对各个方法予以综合比较;最后总结并提出下一步的研究方向。

关键词 组合测试,组合对象,代数构造,贪婪法,元启发式搜索技术

中图法分类号 TP311.5 **文献标识码** A

Research Advances in Interaction Testing

CHEN Xiang GU Qing WANG Xin-ping CHEN Dao-xu

(National Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract Interaction testing is an effective test suite generation technique and becoming a challenging problem. It is concerned with the interaction among several parameters of system under test (SUT) and can test the SUT effectively with smaller test suite. We described the interaction testing problem based on the interaction coverage criteria, explained four well-known combinatorial objects, classified several well-known interaction testing approaches by their characteristic and gave a brief interpretation of some classical approaches. We also gave a set of evaluation criteria and made a comparison among all the approaches. At last we gave a conclusion and discussed some issues to be further studied.

Keywords Interaction testing, Combinatorial object, Algebraic construction, Greedy algorithm, Metaheuristic search technique

1 引言

软件测试是构建高可信软件的关键环节,统计数据表明,该环节一般占软件开发总成本的 50% 以上^[1]。在测试中选择具有代表性的测试用例驱动被测程序,是测试用例生成的核心问题,但该过程大部分工作往往需要手工完成。组合测试技术是一种有效的测试用例生成技术,可作为其他测试技术(例如等价类划分、边界值分析等)的有效补充。

组合测试生成的测试用例集可映射成矩阵 TS,其中行代表测试用例,列代表输入参数。任意 t 元组在该矩阵中至少出现一次,其中 $t(t \geq 2)$ 代表组合测试的覆盖强度。当 $t=2$ 时,我们称其为成对组合测试。

Mandl 等人最早将组合测试引入到软件测试中,即采用正交拉丁方对 Ada 编译器进行了测试^[2]。其后研究人员对组合测试的缺陷检测能力进行了分析,例如 Kuhn 等人根据 3 份缺陷分析报告分析了组合覆盖强度与缺陷检测率间的关系并发现成对组合测试可以发现 70% 的缺陷,3-way 组合测试可以发现 90% 的缺陷,要发现所有缺陷,覆盖强度仅需求达到

6-way^[3]。目前组合测试已经广泛应用于兼容性测试、GUI 测试^[17]、Web 应用测试^[16] 和高度可配置的系统 (highly-configurable system) 中^[4]。

本文对目前已有的组合测试方法进行了研究。首先对组合测试问题进行描述并介绍常见的组合覆盖标准。接着介绍了常用的组合对象,实际的组合测试问题可以抽象成某个组合对象。然后将目前常见的方法分成 3 类并介绍其中的代表方法,接着提出一套评价标准,最后予以总结并指出下一步的研究方向。

2 问题描述及常见覆盖标准

组合测试是一种有效的测试用例生成技术。考虑表 1 中的一个基于 Internet 的软件系统,系统客户可以选择不同的浏览器、操作系统、网络连接类型和打印机配置。若要对系统做充分的兼容性测试,需要 $3^4=81$ 个测试用例;若采用成对组合测试,如表 2 所列,仅需 9 个测试用例。

表 1 一个基于 Internet 的软件系统

浏览器	操作系统	连接类型	打印机配置
-----	------	------	-------

到稿日期:2009-04-09 返修日期:2009-06-16 本文受 863 国家重点基金项目(2006AA01Z177),国家自然科学基金(60873027),江苏自然科学基金基础研究项目(BK2006115)资助。

陈 翔 博士生,研究方向为组合测试等,E-mail:chenxiang_nju@gmail.com;顾 庆 教授,研究方向为软件测试、软件可靠性等;王新平 硕士生,研究方向为缺陷定位技术等;陈道蓄 博士生导师,研究方向为分布式计算与并行计算等。

Firefox	Windows	LAN	Local
IE	Macintosh	PPP	Networked
Other	Linux	ISDN	Screen

表2 最优成对组合测试用例集

序列	浏览器	操作系统	连接类型	打印机配置
1	Firefox	Windows	LAN	Local
2	Firefox	Linux	ISDN	Networked
3	Firefox	Macintosh	PPP	Screen
4	IE	Windows	ISDN	Screen
5	IE	Linux	LAN	Networked
6	IE	Macintosh	PPP	Local
7	Other	Windows	PPP	Networked
8	Other	Linux	LAN	Screen
9	Other	Macintosh	ISDN	Local

组合测试问题可描述如下:假设被测系统有 k 个参数,构成参数集 $P = \{p_1, p_2, \dots, p_k\}$, 其中参数 p_i 有 n_i 个取值,其取值集为 $V_i = \{v_{i1}, v_{i2}, \dots, v_{in_i}\}$ 。依据如下某个组合覆盖标准生成的测试用例集 TS 是由多个 tc 构成的集合,每个 tc 可表示为 k 元组 $tc = (v_1, v_2, \dots, v_k) (v_1 \in V_1, v_2 \in V_2, \dots, v_k \in V_k)$ 。在满足覆盖标准的基础上,如果包含的测试用例数量最少,我们将它称为最优测试用例集。

目前常见的组合覆盖标准包括:

(1) 单参数覆盖标准。假设测试用例集 $TS_{m \times k}$, 若集合 V_i 中的任意元素均出现在第 i 列中,则称 TS 符合单参数覆盖标准。易知符合该覆盖标准的最优测试用例集规模为 $\max_{1 \leq i \leq k} \{n_i\}$ 。

(2) 成对组合覆盖标准。假设测试用例集 $TS_{m \times k}$, 若矩阵中第 i 列和第 j 列构成的二元组集合均包含集合 $V_i \times V_j$ 中所有元素,则称 TS 符合成对组合覆盖标准。该标准考虑了参数间的交互对缺陷检测能力的影响。

(3) t -way 组合覆盖标准。假设测试用例集 $TS_{m \times k}$, 若矩阵任意 t 列(假设对应参数为 $p_1', p_2', \dots, p_t'$)构成的 t 元组集合包含 $V_1' \times V_2' \times \dots \times V_t'$ 的所有元素,则称 TS 符合 t -way 组合覆盖标准。该标准考虑了更多参数间的交互,但也会产生更多的测试用例。当 t 赋值为 k (k 为参数总个数)时,构成 k -way 组合覆盖标准(又叫全组合覆盖标准),保证了充分测试,测试用例集规模为 $m = \prod_{i=1}^k n_i$, 但 m 值往往过大,远远超出软件开发实际预算。

(4) 可变强度覆盖标准,即输入参数集 P 的不同参数子集 $P' (P' \subseteq P)$ 满足不同覆盖强度的组合覆盖标准。该标准假设不同的参数子集间的覆盖标准不一致,尤其在某些关键子集中采取更高强度的覆盖标准,故更贴近现实的软件测试。

近年来国内外学者对组合测试的研究比较活跃。大部分组合测试问题也已被证实为 NP-Complete 问题,例如 Y. Lei 等人将 vertex-cover 问题规约到生成最优成对组合测试用例集的问题上,证明该问题属于 NP-Complete 问题^[5]。

3 组合对象

根据参数的取值情况、采用的覆盖标准及测试用例集中各个组合出现的次数,可以将组合测试用例集抽象为 4 种组合对象:正交矩阵(Orthogonal Array)、覆盖矩阵(Covering Array)、混合水平覆盖矩阵(Mixed Level Covering Array)和可变强度覆盖矩阵(Variable Strength Covering Array)。

定义 1 正交矩阵 $OA_\lambda(N; k, v, t)$ 是一个 $N \times k$ 的矩阵,

N 为矩阵的行数, k 为输入参数的个数,每个参数均有 v 个取值,规模为 t 的任意组合在相应的 $N \times t$ 子矩阵中出现的次数均为 λ ^[6]。正交矩阵满足等式 $\lambda = N/v^t$, 当 $\lambda = 1$ 时,可简称为 $OA(N; k, v, t)$, 表 2 可用 $OA(9; 4, 3, 2)$ 表示。

正交矩阵假设每个组合出现的次数均相等,这在软件测试中并不合理,且会产生额外的测试用例。同时对于一些 v 和 k 的取值组合,并不存在相应的正交矩阵,所以提出覆盖矩阵。

定义 2 覆盖矩阵 $CA_\lambda(N; k, v, t)$ 是一个 $N \times k$ 的矩阵, N 为矩阵的行数, k 为输入参数的个数,每个参数均有 v 个取值,规模为 t 的任意组合在相应 $N \times t$ 的子矩阵中出现的次数至少为 λ ^[7], 当 $\lambda = 1$ 时,可简称为 $CA(N; k, v, t)$ 。我们称该矩阵最优是指在满足覆盖矩阵性质时行数最少,并把最优值记为 $CAN(k, v, t)$ 。

覆盖矩阵只适用于所有参数取值个数都一致的情况。这并不符合实际情况,为此人们又提出混合水平覆盖矩阵。

定义 3 混合水平覆盖矩阵 $MCA(N; t, k, (n_1, n_2, \dots, n_k))$ 是一个 $N \times k$ 的矩阵, N 为矩阵的行数, k 为输入参数的个数,其中 n_i 代表第 i 个参数可能取值的个数,规模为 t 的任意组合在相应的 $N \times t$ 子矩阵中出现的次数至少为 1。若有多个参数具有相同的取值个数,可以采用缩写,将 $MCA(N; t, k, (n_1, n_2, \dots, n_k))$ 改写为 $MCA(N; t, k, (w_1^{r_1}, w_2^{r_2}, \dots, w_s^{r_s}))$, 其中 $k = \sum_{i=1}^s r_i$ ^[8]。

上述组合对象均假设参数间的覆盖强度一致,但在实际软件测试中,参数间的交互关系较为复杂,不同的参数子集间的覆盖强度也并不一致,对一些关键参数子集人们往往采用更高的覆盖强度,所以又提出可变强度覆盖矩阵。

定义 4 可变强度覆盖矩阵 $VCA(N; k, (n_1, n_2, \dots, n_k), (C_1^1, C_2^2, \dots, C_m^m))$ 是一个 $N \times k$ 的矩阵,其中 N 为矩阵的行数, k 为输入参数的个数, n_i 代表第 i 个参数的可能取值, C_i^j 代表参数集合 P 的某个子集,但这些子集并不构成对集合 P 的划分, t_i 代表该子集需要达到的覆盖强度。

目前大部分组合测试方法都是基于混合水平覆盖矩阵的。本文将现有的组合测试方法分为 3 类: (1) 代数构造法, (2) 贪婪法, (3) 元启发式搜索技术; 并对每一类选取代表方法予以简介。

4 代数构造法

该类方法基于正交矩阵或以正交矩阵为基础的基本构造块,通过组合方式生成测试用例集,生成时间较快,并在某些情况下可生成最优测试用例集。但该类方法并不支持 Test Seeds(即预先指定测试用例)和 Constraints(在测试用例中一定不会出现组合),并难以扩展到更高强度的覆盖标准。

4.1 正交实验设计

正交实验设计(orthogonal test design)基于正交矩阵。Mandle 早在 1985 年就将该方法用于测试 Ada 编译器^[2], Brownlien 等人也将正交实验设计应用到软件测试中,并开发了 OATs 工具^[9]。正交实验设计根据参数和参数取值,选择合适的正交矩阵快速生成测试用例集。

该方法产生测试用例的时间极短,在某些情况下,可生成最优测试用例集。但也存在不足: (1) 每个参数的取值需一样,虽可通过添加不相关取值解决该问题,但会产生额外的测

试用例; (2) 每个组合在正交矩阵中出现的次数一样, 这在统计实验中是必需的, 但在软件测试中并不需要。

4.2 TConfig 法

TConfig 法^[10]由 A. W. Williams 等人提出。它是对正交实验设计的改进, 并有效解决了上述问题。

TConfig 法以正交矩阵为基础, 通过如下 5 个基本构造块递归地构造测试用例集: (1) $O(k^2, k+1, k)$ 是一个正交矩阵, 该矩阵有 k^2 行、 $k+1$ 个参数, 其中每个参数最多有 k 个取值。 (2) $B(k^2-1, k+1, k, d)$ 基于正交矩阵 $O(k^2, k+1, k)$, 其中将正交矩阵 O 删除第一行, 将每一列重复 d 次。 (3) $R(k^2-k, k, k, k, d)$ 基于正交矩阵 $O(k^2, k+1, k)$, 其中将正交矩阵 O 删除前 k 行, 删除第一列, 将剩下的每一列重复 d 次。 (4) $I(c, d)$ 是一个 $c \times d$ 矩阵, 其中每个元素的取值均为 1。 (5) $N(k^2-k, k, d)$ 是一个 (k^2-k) 行、 $(k \times d)$ 列的矩阵, 其中第一个 $k \times d$ 子矩阵的元素值均为 2, 第二个 $k \times d$ 子矩阵的元素值均为 3, ..., 第 $k-1$ 个子矩阵的元素值均为 k 。

TConfig 法生成测试用例集的速度较快, 且适用于各种输入, 但最终产生的测试用例集规模相比其他方法往往偏大, 在混合水平覆盖矩阵中尤为明显。

5 贪婪法

目前大部分组合测试问题都是 NP-Complete 问题, 研究人员提出大量基于贪婪法的组合测试方法, 力争在多项式时间内获得近似解。

5.1 CATS 法

CATS 法^[11]由 Sherwood 提出, 它采用 one-test-at-a-time 策略生成最终测试用例集。以成对组合测试为例, One-test-at-a-time 策略如图 1 所示。

```

输入: 参数集 P
TestSuite ← ∅
将参数集 P 所有的成对组合置入 Uncover 集
while Uncover ≠ ∅ do
    生成单个测试用例 test
    从 Uncover 中移去 test 覆盖的组合
    添加 test 到 TestSuite
end while
输出: 测试用例集 TestSuite

```

图 1 One-test-at-a-time 策略

在生成单个测试用例时, 该方法遍历尚未选择的所有可能测试用例, 并从中选择一个覆盖 Uncover 中最多组合的测试用例。如果存在多个, 则选择第一个作为测试用例。

在参数规模较小的情况下, 该方法可获得最优测试用例集。但当参数个数或者参数取值增多时, 候选的测试用例个数将呈指数级增长。

5.2 AETG 法和 TCG 法

为了克服 CATS 法存在的不足, D. M. Cohen 等人提出了 AETG 法^[12]。

该方法也采用 one-test-at-a-time 策略完成测试用例集的构建。但在生成单个测试用例时, 引入了随机技术。具体步骤如下: 首先找出 Uncover 中出现次数最多的参数取值, 并将该参数设为第一个参数, 接着对剩下的参数随机排列 M 次, 假设某次排列后参数的次序为 $p_1', p_2', \dots, p_k'$ 。从 p_2' 开始

采用贪婪策略依次确定参数取值 (即选取与前面参数取值构成组合在 Uncover 中出现次数最多的取值)。其算法如图 2 所示。

```

输入: 已生成的测试用例  $tc_i - tc_i - 1$ 
for  $i=0$  to  $M$  do //生成  $M$  个候选测试用例
    找出在 Uncover 中出现次数最多的参数取值  $v_{ij}$ , 并将对应
    参数  $p_i$  设为第一个参数
    对剩下的参数进行随机排列, 然后采用贪婪策略依次确定参
    数取值, 产生候选测试用例  $Test_i$ 
end for
从  $M$  个候选测试用例中选择覆盖 Uncover 中组合最多的测试用
例作为  $tc_i$ 
输出: 测试用例  $tc_i$ 

```

图 2 AETG 算法

最终测试用例集规模与候选的测试用例数 M 有关。文献^[12]通过实验发现 M 的取值为 50, 即可获得理想规模的测试用例集。

TCG (Test Case Generation) 法^[13]与 AETG 法类似, 由 Y. W. Tung 等人提出。AETG 法是一种不确定的方法, 即相同的输入每次并不一定生成相同的测试用例集。但 TCG 法是一种确定的方法, 它在确定测试用例 tc_i 时首先将参数按照取值个数非递增排序, 然后根据第一个参数的取值个数生成 n_1 个候选测试用例, 每个候选测试用例依据贪婪策略 (同 AETG 法) 按序选定余下参数的取值。最后从 n_1 个候选测试用例中选择一个覆盖能力最强的测试用例。

5.3 DDA 法

DDA (Deterministic Density Algorithm) 法^[14]由 C. J. Colbourn 等人提出。DDA 法通过 one-test-at-a-time 策略构建测试用例集。在构建单个测试用例时引入局部密度和全局密度, 其局部密度定义如下:

$$\delta_{i,j} = \begin{cases} u_{i,j} / (\max_i \{n_i\})^2, & p_i, p_j \text{ 均未赋值} \\ u_{i,j} / \max_i \{n_i\}, & p_i, p_j \text{ 中之一已赋值} \\ u_{i,j}, & p_i, p_j \text{ 均已赋值} \end{cases}$$

其中, $u_{i,j}$ 指参数 p_i 与参数 p_j 构成的可覆盖 Uncover 中的组合数量。

$$\text{全局密度} = \delta = \sum_{1 \leq i < j \leq k} \delta_{i,j}$$

DDA 法在参数赋值次序和参数赋值上均采用贪婪策略, 先依据参数密度确定参数赋值次序 (参数 p_i 的密度计算公式为 $\delta_i = \sum_{1 \leq j \leq k, j \neq i} \delta_{i,j}$)。次序确定后, 依据全局密度最大原则依次确定参数取值。

DDA 法是确定性方法, 同时基于密度的策略使得它: (1) 不需要生成大批候选测试用例; (2) 当确定参数取值时, 不仅考虑了与已确定取值的参数间的组合, 而且考虑了与未确定取值参数间的组合。因此最终生成的测试用例集规模以及生成时间往往优于其他同类方法。

5.4 IPO 法

IPO (In-Parameter-Order) 法^[5]由 Y. Lei 等人提出。该方法首先依据前两个参数生成满足成对组合覆盖标准的测试用例集, 然后通过扩展测试用例集使之满足前 3 个参数的成对组合覆盖。如此往复, 直至所有的参数都包括到测试用例集中。

IPO 包括两个阶段: (1) 水平扩展阶段 (horizontal

growth), 算法如图 3 所示; (2) 垂直扩展阶段 (vertical growth), 算法如图 4 所示。

```

输入: 已有测试用例集  $T$ , 水平扩展参数  $p_i$ 
 $\pi \leftarrow$  参数  $p_i$  与前  $(i-1)$  个参数需要覆盖的组合
if  $|T| \leq n_i$  then
    for  $j=1$  to  $|T|$  do
        通过给参数  $p_i$  赋值  $v_{i,j}$  扩展第  $j$  个测试用例, 并从  $\pi$  中移去已覆盖的组合
    end for
else
    for  $j=1$  to  $n_i$  do
        通过给参数  $p_i$  赋值  $v_{i,j}$  扩展第  $j$  个测试用例, 并从  $\pi$  中移去已覆盖的组合
    end for
    for  $j=n_i+1$  to  $|T|$  do
        选择参数  $p_i$  中覆盖  $\pi$  中组合最多的取值来扩展第  $j$  个测试用例, 并从  $\pi$  中移去已覆盖的组合
    end for
end if
输出: 完成水平扩展后的测试用例集  $T$ 

```

图 3 IPO 水平扩展算法

```

输入: 水平扩展后的测试用例集  $T$ , 尚未覆盖的组合集  $\pi$ 
 $T' \leftarrow \emptyset$ 
for  $\forall i \in \pi$  do
    假设组合  $i$  由参数  $p_k$  的取值  $w$  和参数  $p_i$  的取值  $u$  构成
    if  $T'$  中包括测试用例, 其参数  $p_k$  的取值为“—”, 参数  $p_i$  的取值为  $u$  then
        修改该测试用例, 将“—”替换为  $w$ 
    else
        添加新的测试用例  $t$ , 参数  $p_k$  的取值为  $w$ , 参数  $p_i$  的取值为  $u$ , 其他参数的取值为“—”
        将  $t$  添加到  $T'$ 
    end if
end for
 $T \leftarrow T \cup T'$ 
输出: 完成垂直扩展后的测试用例集  $T$ 

```

图 4 IPO 垂直扩展算法

CAT, AETG, TCG 和 DDA 均采用了 one-test-at-a-time 策略, 而 IPO 法则更多考虑复用已有的测试用例集, 所以在参数或者参数取值发生变化时可以快速生成新的测试用例集。

6 元启发式搜索技术

采用元启发式 (metaheuristic) 搜索技术来生成测试用例, 是近几年的一个研究热点。特定的元启发式搜索技术都具有固定的方法框架, 采用特定的启发技术以合理的计算成本求解组合优化问题。组合测试是一种典型的组合优化问题, 目前大量元启发式搜索技术应用于组合测试中。采用元启发式搜索技术, 测试用例集的构造一般采用 one-test-at-a-time 策略。在生成单个测试用例时, 可以采用各种元启发式搜索技术, 包括爬山法、模拟退火、遗传算法、蚁群算法等, 下面将选取典型技术予以简介。

6.1 爬山法

爬山法 (Hill Climbing, HC) 是一种局部搜索算法, 在生

成单个测试用例时, 其搜索过程如图 5 所示。其中测试用例 tc 的邻域集 $Neighbour(tc)$ 通过随机选择参数并修改其取值后的测试用例构成。测试用例 tc 的目标函数 $obj(tc)$ 定义为 tc 覆盖 $Uncover$ 中的组合数量。

```

输入: 已生成的测试用例  $tc_1 \sim tc_{i-1}$ 
随机生成初始测试用例  $tc$ 
repeat
    选择  $tc' \in Neighbour(tc)$ , 使得  $obj(tc') > obj(tc)$ 
     $tc \leftarrow tc'$ 
until  $obj(tc) \geq obj(tc')$ ,  $tc' \in Neighbour(tc)$ 
输出:  $tc$ 

```

图 5 基于爬山法的算法框架

爬山法虽然简单, 但容易陷入局部极值。所以研究人员提出模拟退火、遗传算法等全局搜索算法。

6.2 模拟退火

模拟退火 (Simulated Annealing, SA) 是一种全局搜索算法, 由 Metropolis 等人提出。思想来源于固体退火原理。在生成单个测试用例时, 模拟退火搜索过程如图 6 所示。模拟退火方法依据 Metropolis 准则由一定的概率接收劣解, 温度越高, 接受的概率越高, 从而一定程度内避免了局部最优问题。

```

输入: 已生成的测试用例  $tc_1 \sim tc_{i-1}$ 
随机生成初始测试用例  $tc$ 
设置初始温度  $t$ 
repeat
    随机选择  $tc' \in Neighbour(tc)$ 
     $\Delta e \leftarrow obj(tc') - obj(tc)$ 
    if  $\Delta e > 0$  then
         $tc \leftarrow tc'$ 
    else
        生成随机数  $r$ ,  $0 \leq r < 1$ 
        if  $r < e^{-\Delta e/t}$  then  $tc \leftarrow tc'$ 
    end if
    依据冷却进度表降低温度  $t$ 
until  $t$  到达特定温度
输出:  $tc$ 

```

图 6 基于模拟退火的算法框架

6.3 遗传算法

遗传算法 (Genetic Algorithm, GA) 是一种全局搜索算法, 它受生物界的进化规律 (适者生存、优胜劣汰机制) 所启发。针对具体问题, 遗传算法需要对候选解进行染色体编码和适应值函数设定。在组合测试中, 测试用例可以直接表示为染色体, 适应值函数可以设置为测试用例覆盖 $Uncover$ 中的组合数量。在生成单个测试用例时, 遗传算法搜索过程如图 7 所示。其中选择策略、交叉操作和变异操作可以参考文献[15]。

```

输入: 已生成的测试用例  $tc_1 \sim tc_{i-1}$ 
创建初始种群  $P$ , 规模为  $m$ 
repeat
    从  $P$  中选择  $\sigma$  个适应值函数最高的个体保留到下一代  $P'$ 
    应用选择策略选择  $(m-\sigma)$  个个体存入  $P_{mating}$ 
    对  $P_{mating}$  采取交叉操作
    对  $P_{mating}$  采取变异操作
     $P \leftarrow P' \cup P_{mating}$ 

```

until 达到指定迭代次数

输出:种群 P 中适应值最高的测试用例

图 7 基于遗传算法的算法框架

7 组合测试方法评价标准

目前存在大量工具支持组合测试。例如商业工具 AETG 系统以 Web 服务方式提供服务;基于 Java 的 TConfig 由 A. W. William 开发并可免费下载。更为详细的工具列表可参考 <http://www.pairwise.org>。南京大学计算机软件新技术国家重点实验室对组合测试进行了初步的研究,开发了 ISGT (Interaction Test Suite Generation Tool) 工具,集成了上述多种组合测试方法。

为了客观地比较各种组合测试方法,本文给出如下 5 条评价标准:(1)生成测试用例集规模。规模越小,软件测试执行时间越短。表 3 列出了各个方法在 4 个不同输入中生成的测试用例集规模。(2)生成时间。精确的度量需要各个方法用同一种语言编码并运行在相同的硬件平台上,本文仅根据算法描述的时间复杂度将运行时间大致分为快、中、慢 3 个级别。(3)是否支持 Test Seeds。测试人员往往可以根据经验预先生成一些测试用例,并希望最终生成的测试用例集中包含这部分测试用例并优先执行。(4)是否支持 Constraints。参数之间往往并不相互独立,一个参数的某个取值可能导致其他参数的某些取值无效。在构造测试用例时,必须禁止这些组合的出现。(5)确定性。在同样输入的情况下,算法的每次运行是否产生相同的测试用例集以便更好地复用。表 4 针对评价标准(2)~(5)对常见组合测试方法予以比较。

表 3 常见组合测试方法生成的测试用例集的最小规模

被测系统	TConfig	AETG	TCG	IPO	DDA	HC	SA
MCA($N_1; 2, 5^1 3^8 2^2$)	21	19	20	19	21	16	15
MCA($N_2; 2, 7^1 6^1 5^1 4^5 3^8 2^3$)	91	45	45	45	43	42	42
MCA($N_3; 2, 5^1 4^1 3^1 1^2 5$)	32	30	30	25	27	23	21
MCA($N_4; 2, 6^1 5^1 4^6 3^8 2^3$)	50	34	33	33	34	30	30

表 4 常见组合测试方法比较

方法	生成时间	Seeds	Constraints	确定
OTD	fast	no	no	yes
TConfig	fast	no	no	yes
Hill Climbing	slow	yes	yes	no
Simulate Annealing	slow	yes	yes	no
Genetic Algorithm	slow	yes	yes	no
CATS	medium	yes	yes	yes
AETG	medium	yes	yes	no
TCG	medium	yes	yes	yes
IPO	medium	yes	yes	yes
DDA	medium	yes	yes	yes

结束语 本文研究了目前组合测试的进展。组合测试在很多方面还亟待拓展和深化,其中包括可变强度组合测试、覆盖强度更高的组合测试、基于优先级的组合测试、基于约束的组合测试以及组合测试在实际软件测试的有效性验证。上述问题的有效解决将使组合测试更好地应用于软件测试中,在保障对软件高质量测试的基础上大幅度降低测试成本。

参考文献

[1] Ammann P, Offutt J. Introduction to Software Testing [M].

Cambridge University Press, 2008

- [2] Mandl R. Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing [J]. Communications of the ACM, 1985, 28(10): 1054-1058
- [3] Kuhn D R, Wallace D R, Gallo A M. Software Fault Interactions and Implications for Software Testing [J]. IEEE Transactions on Software Engineering, 2004, 30(6): 418-421
- [4] Cohen M B, Dwyer M B, Shi J F. Constructing Interaction Test Suites for Highly-configurable Systems in the Presence of Constraints: A Greedy Approach [J]. IEEE Transaction on Software Engineering, 2008, 34(5): 633-650
- [5] Tai K C, Lei Y. A Test Generation Strategy for Pairwise Testing [J]. IEEE Transaction on Software Engineering, 2002, 28(1): 109-111
- [6] Hedayat A, Sloane N, Stufken J. Orthogonal Arrays: Theory and Applications [M]. Springer-Verlag, 1999
- [7] Sloane N. Covering arrays and intersecting codes [J]. Journal of combinatorial Designs, 1993, 1(1): 51-63
- [8] Cohen M B, Gibbons P B, Mugridge W B, et al. Constructing Test Suites for Interaction Testing [C]// The 25th International Conference on Software Engineering. 2003: 38-48
- [9] Brownlie R, Prowse J, Phadke M. Robust Testing of AT&T PMX/StarMail Using OATS [J]. AT&T Technical Journal, 1992, 3(71): 41-47
- [10] Williams A W. Determination of test configurations for pair-wise interaction coverage [C]// The 13th International Conference on the Testing of Communicating Systems. 2000: 59-74
- [11] Sherwood G. Effective Testing of Factor Combinations [C]// The Third International Conference on Software Testing, Analysis and Review. 1994
- [12] Cohen D M, Dalal S R, Fredman M L, et al. The AETG system: an approach to testing based on combinatorial design [J]. IEEE Transactions on Software Engineering, 1997, 23(7)
- [13] Tung Y W, Aldiwan W S. Automating Test Case Generation for the New Generation Mission Software System [C]// IEEE Aerospace Conference. 2000: 431-437
- [14] Colbourn C J, Cohen M B, Turban R C. A Deterministic Density Algorithm for Pairwise Interaction Coverage [C]// IASTED International Conference on Software Engineering. 2004: 345-352
- [15] Shiba T, Tsuchiya T, Kikuno T. Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing [C]// International Conference on Computer Software and Applications Conference. 2004: 72-77
- [16] Sampath S, Bryce R C, Viswanath G, et al. Prioritizing User-Session-based Test Cases for Web Application Testing [C]// The 1st International Conference on Software Testing, Verification, and Validation. 2008: 141-150
- [17] Yuan X, Cohen M B, Memon A M. Covering Array Sampling of Input Event Sequences for Automated GUI Testing [C]// The 22nd International Conference on Automated Software Engineering. 2007: 405-408