

基于 One-test-at-a-time 策略的可变力度组合 测试用例生成方法

王子元^{1),2),3)} 钱 巨^{4),5)} 陈 林^{1),2)} 徐宝文^{1),2)}

¹⁾(南京大学软件新技术国家重点实验室 南京 210093)

²⁾(南京大学计算机科学与技术系 南京 210093)

³⁾(南京邮电大学计算机学院 南京 210006)

⁴⁾(上海市计算机软件评测重点实验室 上海 201112)

⁵⁾(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 组合测试可以有效地检测软件系统中由各个因素间交互作用所引发的软件故障,但传统的组合测试方法对系统中各因素之间的实际交互关系考虑不足,难以有效处理交互力度不统一的情况,进而可能导致测试用例的冗余和检错能力的降低.针对该问题,应在充分考虑因素间实际交互关系的基础上,使用可变力度组合测试方法,从而实现对于因素间实际交互关系的覆盖.为此,文中针对一种新的可变力度组合测试模型,提出了两种基于 one-test-at-a-time 策略的可变力度组合测试用例集生成算法.实验表明,相对于已有的具备类似功能的测试用例生成算法和工具,文中提出的算法在测试用例集规模和算法运行时间上均具备一定优势,并可适用于固定力度组合测试、可变力度组合测试等不同测试模型.

关键词 软件测试;可变力度组合测试;测试用例生成;交互关系

中图法分类号 TP311

DOI号: 10.3724/SP.J.1016.2012.02541

Generating Variable Strength Combinatorial Test Suite with One-test-at-a-time Strategy

WANG Zi-Yuan^{1),2),3)} QIAN Ju^{4),5)} CHEN Lin^{1),2)} XU Bao-Wen^{1),2)}

¹⁾(State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093)

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

³⁾(School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing 210006)

⁴⁾(Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112)

⁵⁾(College of Computer Science and Technology Nanjing University of Aeronautic and Astronautics, Nanjing 210016)

Abstract Combinatorial testing aims to detect faults triggered by interactions among factors in software. However, traditional combinatorial testing may lead to the redundancy of test suite and the decrease of fault detecting ability, since it can not handle scenarios where the strengths of different interactions are not uniform. To avoid this limitation, the actual interaction relationship should be made sufficient consideration, and the strength of each interaction should be varied depending on the actual interaction relationship. Therefore, for a new model of variable strength combinatorial testing, two greedy heuristic algorithms, which are based on one-test-at-a-time strategy, are proposed to generate variable strength combinatorial test suite to cover the interaction relationship. Experimental results show that, compared to some existed similar algorithms and tools, the two proposed algorithms could leverage the execution effectiveness and optimality on the size of generated test suite.

收稿日期:2010-10-31;最终修改稿收到日期:2012-09-24. 本课题得到国家自然科学基金重大研究计划重点项目(90818027,91018005)、国家自然科学基金面上项目(60973046,61003020)、江苏省自然科学基金项目(BK2009426,BK2011190)及上海市计算机软件评测重点实验室开放基金项目(09DZ2272600)资助. 王子元,男,1982年生,博士,主要研究方向为软件测试. E-mail: wangziyuan@njupt.edu.cn. 钱 巨,男,1980年生,博士,副教授,主要研究方向为程序分析. 陈 林,男,1979年生,博士,主要研究方向为程序分析. 徐宝文(通信作者),男,1961年生,博士,教授,博士生导师,主要研究领域包括程序设计语言、软件工程等. E-mail: bwxu@nju.edu.cn.

Keywords software testing; variable strength combinatorial testing; test generation; interaction relationship

1 引 言

软件系统是一个复杂的逻辑系统,有很多因素都可能会影响软件系统的正常运行,这些因素可能包括系统的配置、内部事件、外部输入等.除了单个因素之外,这些因素之间的相互作用也可能会对系统的运行造成影响.组合测试作为一种科学、有效的软件测试方法,可以使用较少的测试用例有效地检测软件系统中各个因素以及它们之间的相互作用对系统产生的影响.

组合测试方法主要可分为固定力度组合测试和可变量度组合测试这两种类型.其中,固定力度组合测试方法得到了最为广泛的研究和应用,并在很多实践中都取得了良好的效果.在固定力度的 τ 维组合测试中,人们一般简单地假设系统中任意 τ 个因素间均存在交互作用,并在上述假设的基础上设计 τ 维组合测试用例集以检测这些交互作用.但 Cohen 等人^[1]提出,在很多软件系统中,不同因素之间的交互力度往往是不同的, Kuhn 等人^[2-3]对一些具体软件系统的分析结果也说明了这一点.对于这样的系统,往往很难寻找一个合适的正整数 τ 来进行固定力度的组合测试:(1) 如果 τ 选取过大,则测试用例可能出现不必要的冗余,从而增加测试成本;(2) 若 τ 选取过小,则某些因素间的交互作用将难以在测试中得到完全的检测,从而降低错误检测能力.

为了解决上述问题, Cohen 等人^[1]在 2003 年提出了一种可变量度的组合测试方法.该方法在固定力度 τ 维组合测试的基础上,允许部分互不相交的因素子集中存在力度大于 τ 的交互作用,从而在一定程度上解决了上述问题.但是我们同时注意到,在 Cohen 等人提出的可变量度组合测试模型中,一方面仍然假设所有因素间均存在力度至少为 τ 的交互作用,另一方面则要求那些具有较高交互力度的因素子集互不相交.这样的限制可能会影响该模型的表达能力.

可见,在充分了解软件中因素间交互作用的基础上,有必要对现有的组合测试模型进行进一步改进.为此, Schroeder 等人^[4]提出一种基于程序输入输出关系的测试模型.在该模型中,程序的每个输出变量均依赖于若干个输入变量的交互作用.我们则

在该模型的基础上进一步扩展,提出一种新的可变量度组合测试模型^[5-6].这种新的可变量度组合测试模型可以更为准确地描述系统中的因素间交互作用,进而为测试用例集的设计和生成提供更好的指导.在本文中,一般称 Cohen 等人提出的方法为“狭义”可变量度组合测试方法,以便于新旧两种方法的区分.

在假设待测系统因素间交互关系已知的情况下,本文针对这种新的可变量度组合测试模型,提出了一系列基于 one-test-at-a-time 策略的可变量度组合测试用例生成算法,并通过实验对相关算法的性能进行分析和比较.实验结果表明,相对于已有的具备类似功能的测试用例生成算法和工具,本文提出的算法在测试用例集规模和算法运行时间上均具备一定优势.

2 模型及定义

假设影响待测软件系统(System Under Test, SUT)的因素共有 n 个,这些因素形成有限集合 $F = \{f_1, f_2, \dots, f_n\}$.其中因素 f_i 经过等价类划分等前期处理后共包含 a_i 个可选取值,不妨假设该因素的可取值集合为 $V_i = \{1, 2, \dots, a_i\} (1 \leq i \leq n)$.一般地,如果某个 SUT 中所有因素取值数量均相等,即 $a_1 = a_2 = \dots = a_n$,则称该 SUT 为单一水平系统;否则,称该 SUT 为混合水平系统.

定义 1. 称一个 n 元组 $test = (v_1, v_2, \dots, v_n)$ ($v_1 \in V_1, v_2 \in V_2, \dots, v_n \in V_n$) 为 SUT 的一条测试用例,相应地,称一个由多个这样的 n 元组所构成的集合为待测软件 SUT 的一个测试用例集.

2.1 已有组合测试模型

在组合测试中,需要考虑因素之间的交互作用,并根据因素间的交互作用设计测试用例集,以覆盖某些给定的因素取值组合.首先,对固定力度组合测试和“狭义”可变量度组合测试的相关概念进行简要介绍^[1,7].

定义 2. 设 $A = (a_{i,j})_{m \times n}$ 为一个 $m \times n$ 的矩阵,其第 j 列表示 SUT 的因素 f_j ,其中所有元素均取自集合 $V_j (j = 1, 2, \dots, n)$,即 $a_{i,j} \in V_j$.给定正整数 $\tau (1 \leq \tau \leq n)$,如果 A 中任意 τ 列,即第 $i_1, i_2, \dots, i_\tau$ 列均满足: $V_{i_1}, V_{i_2}, \dots, V_{i_\tau}$ 中元素的所有的 τ 元组

合均在这 τ 列所组成的子矩阵中至少出现一次,则称 $\mathbf{A}$ 为 τ 维组合覆盖表,记作 $\mathbf{CA}(m; \tau, F)$. 其中, τ 为 $\mathbf{CA}(m; \tau, F)$ 的组合覆盖力度.

显然,从 $\mathbf{CA}(m; \tau, F)$ 中任取 k 列 ($\tau \leq k \leq n$),其所构成的子矩阵的组合覆盖力度均为 τ . 因此,我们又称 τ 维组合覆盖表为固定力度组合覆盖表.

定义 3. 对于一个 τ 维组合覆盖表 $\mathbf{A}$,若 F 中存在 $t(t \geq 1)$ 个互不相交的子集 $F_i \subseteq F (i = 1, 2, \dots, t)$, 这些子集构成集合 C . 其中 $F_i (i = 1, 2, \dots, t)$ 包含 n_i 个因素,且这些因素对应的列所组成的 $m \times n_i$ 的子矩阵 $\mathbf{A}_i$ 满足 $\tau_i (\tau < \tau_i \leq n_i)$ 维组合覆盖的条件,即 $\mathbf{A}_i$ 的组合覆盖力度不等于 τ ,则称 $\mathbf{A}$ 为“狭义”可变力度的组合覆盖表,记作 $\mathbf{VSCA}(m; \tau, F, C)$.

定义 4. 使用固定力度组合覆盖表和“狭义”可变力度组合覆盖表设计测试用例集的方法,可分别称之为固定力度组合测试方法和“狭义”可变力度组合测试方法.

例如,对于一个如图 1 所示的面向对象系统进行类交互测试时,每次测试均需要从 4 个类体系结构(Class Cluster)中各选取一个子类组成一个具体的配置作为测试用例. 我们将每个类体系结构作为一个因素,其中的子类作为该因素的可选取值,从而有 $F = \{2^3 \times 3\}$ (表示 F 中共含 4 个因素,可选取值的数量依次为 2、2、2、3). 若采用固定力度二维组合测试,则可以找到一个规模为 6 的二维组合测试用例集 $\mathbf{CA}(6; 2, F)$,从而保证任意两个因素间的所有二元取值组合均至少出现一次,如表 1 所示. 在此基础上,进一步令 $C = \{\{ClassA, ClassB, ClassC\}\}$,则可得到如表 2 所示的可变力度组合测试用例集 $\mathbf{VSCA}(8; 2, F, C)$. 该测试用例集除保证二维组合覆盖外,还能保证 $ClassA, ClassB, ClassC$ 3 个因素间所有的三元取值组合均至少出现一次.

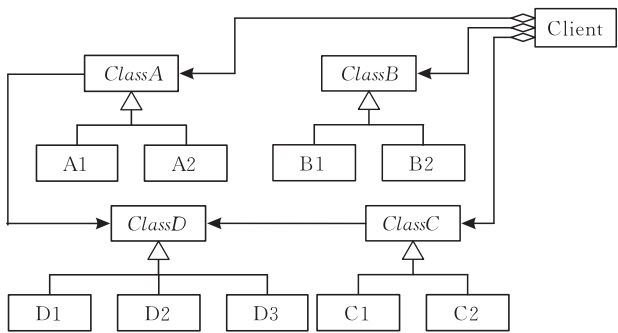


图 1 某面向对象系统的类图

表 1 $\mathbf{CA}(6; 2, F)$

ClassA	ClassB	ClassC	ClassD
A1	B1	C1	D1
A1	B1	C2	D2
A1	B2	C1	D3
A2	B1	C2	D3
A2	B2	C1	D2
A2	B2	C2	D1

表 2 $\mathbf{VSCA}(8; 2, F, C)$

ClassA	ClassB	ClassC	ClassD
A1	B1	C1	D1
A1	B1	C2	D2
A1	B2	C1	D3
A1	B2	C2	D1
A2	B1	C1	D2
A2	B1	C2	D3
A2	B2	C1	D1
A2	B2	C2	D2

2.2 新的可变力度组合测试模型

在上述定义的基础上,本小节对新的可变力度组合测试方法进行介绍^[5-7].

假设因素集合 F 中的一组因素间存在交互作用,则可以将这些因素收集为 F 的一个子集 $r \subseteq F$,从而使得子集中所有 $|r|$ 个因素间存在一个力度为 $|r|$ 的交互作用(或称之为 $|r|$ 维交互作用). 对于一个给定的子集 r ,相应的组合测试用例集应覆盖 r 中因素间所有的 $|r|$ 元取值组合. 类似的,系统 SUT 中的每一个交互作用均可抽象为这样的一个子集. 假设 SUT 中共存在 t 个不同的交互作用,则可以构造一个由 F 的 t 个不同子集所组成的集合 $R = \{r_1, r_2, \dots, r_t\}$. 针对该 SUT 的组合测试用例集应覆盖 R 中所有子集所对应的因素间交互作用,即对于任一给定的 $r_k \in R (k = 1, 2, \dots, t)$,测试用例集必须覆盖 r_k 中因素间所有的 $|r_k|$ 元取值组合. 例如,在对一个具有 n 个因素的系统进行固定力度二维组合测试时,测试用例集就必须覆盖 $R = \{\{f_i, f_j\} \mid f_i, f_j \in F, 1 \leq i < j \leq n\}$ 中的所有 $|R| = n \times (n - 1) / 2$ 个二维相互作用.

定义 5. 称上述集合 R 为待测系统 SUT 的因素交互关系,简称交互关系;而 R 中的每一个元素 $r_k \in R (k = 1, 2, \dots, t)$ 均为 SUT 的一个因素组合覆盖需求,简称覆盖需求.

本文中,我们不妨假设:(1)覆盖需求 $r_i = \{f_{i,1}, f_{i,2}, \dots, f_{i,n_i}\} \in R (i = 1, 2, \dots, t)$ 中含有 n_i 个因素且有 $n_i > 1$;(2)因素集合 F 中的给定的两个因素 $f_i, f_j \in F (1 \leq i < j \leq n)$ 之间存在交互作用,当且仅当存在一个覆盖需求 $r \in R$ 使得 $f_i, f_j \in r$;(3) R 中任意

两个覆盖需求 $r_{i1}, r_{i2} \in R (i_1 \neq i_2)$ 之间不存在包含关系, 否则若有 $r_{i1} \subseteq r_{i2}$, 则可直接从 R 中删除 r_{i1} 而不影响测试用例集的组合覆盖能力; (4) 对于 F 中任意因素 $f_i \in F (1 \leq i \leq n)$, 至少存在一个覆盖需求 $r \in R$ 使得 $f_i \in r$, 否则说明 f_i 不与 F 中的其它任何因素发生交互作用, 不属于组合测试所考虑的范围。

定义 6. 设 $A = (a_{ij})_{m \times n}$ 为一 $m \times n$ 的矩阵, 其第 j 列表示 SUT 的参数 f_j , 其中所有元素均取自集合 $V_j (j = 1, 2, \dots, n)$, 即 $a_{ij} \in V_j (i = 1, 2, \dots, m)$. 给定 SUT 的一个组合覆盖需求 $r_k \in R$, 由 r_k 中的 n_k 个因素所对应的列可组成一个 $m \times n_k$ 的子矩阵 A_k , 若该 n_k 个因素的所有 n_k 元取值组合在 A_k 中均至少出现一次, 则称 A 满足覆盖需求 r_k . 若任取 $r_k \in R (k = 1, 2, \dots, t)$, 均有 A 满足 r_k , 则称 A 为满足交互关系 R 的可变力度组合覆盖表, 并记为 $VCA(m; F, R)$.

结合定义 5 可知, 对于交互关系 R 中的一个覆盖需求 $r_k \in R (k = 1, 2, \dots, t)$, r_k 中因素间所有的 $|r_k|$ 元取值组合可以构成集合:

$$CombSet_k = \{ (v_{k,1}, v_{k,2}, \dots, v_{k,n_k}) \mid v_{k,1} \in V_{k,1}, \\ v_{k,2} \in V_{k,2}, \dots, v_{k,n_k} \in V_{k,n_k} \}.$$

相应的, R 中所有覆盖需求所对应的组合可以构成集合:

$$CombSet = \bigcup_{k=1}^t CombSet_k.$$

显然, 一个满足 R 的可变力度组合覆盖表必须覆盖集合 $CombSet$ 中的所有组合。

定义 7. 使用可变力度组合覆盖表设计测试用例集的方法, 称之为可变力度组合测试方法。

不同于传统的固定力度维组合测试, 在可变力度组合测试中, 测试用例集不需要覆盖任意 τ 个因素之间的取值组合, 仅需满足交互关系 R 中所有 t 个覆盖需求即可. 而相对于“狭义”可变力度组合测试, 新的可变力度组合测试方法可以更为准确地描述系统中因素间的交互关系. 例如, 对于图 1 所示的面向对象系统, 因素交互关系可描述为 $R = \{r_1, r_2, r_3\}$, 其中 $r_1 = \{ClassA, ClassB, ClassC\}$, $r_2 = \{ClassA, ClassD\}$, $r_3 = \{ClassC, ClassD\}$. 若使用“狭义”可变力度组合测试模型, 则存在如下可能: (1) 令 $\tau = 2$ 且 $C = \{r_1\}$, 则 R 中所有覆盖需求均可满足, 但原本不属于 R 的覆盖需求 $\{ClassB, ClassD\}$ 也会被该模型包含在内; (2) 令 $\tau = 1$ 且 $R = \{r_1, r_2, r_3\}$, 则不存在冗余的覆盖需求, 但 r_1 与 r_2 间、 r_1 与 r_3 间以及 r_2 与 r_3 间均存在交集, 不符合

“狭义”可变力度组合测试模型的要求。

进行可变力度组合测试的一个重要前提是获取待测系统的因素交互关系, 这可以通过文档分析、静态代码分析等方式进行, 本文对此不做赘述. 假设已通过上述某种方式准确获取因素交互关系, 我们重点考虑如何在此基础上生成规模尽可能小的可变力度组合测试用例集, 以便在满足组合覆盖标准的前提下, 尽可能节约测试成本。

定理 1. 给定因素集合 F , 交互关系 R 以及正整数 $m < (a_1 \times a_2 \times \dots \times a_n)$, 生成一个规模为 m , 且满足交互关系 R 的可变力度组合测试用例集 $VCA(m; F, R)$ 是一个 NP-C 问题。

证明. 首先需证明该问题属于 NP 类, 然后若存在一个已知的 NP-C 问题可在多项式时间内转化为该问题, 或已知问题是该问题的一种特殊情况, 即可证明该问题为 NP-C 问题。

首先, 给定一个测试用例集 T , 判断其是否满足 R 中所有覆盖需求, 需依次检查 T 中的 m 条测试用例. 而在检查每一条测试用例时, 又需依次检查 R 中每个覆盖需求所对应的因素取值组合. 因此, 其时间复杂度不超过 $O(|m|^2 \times \sum_{k=1}^t |r_k|)$, 即多项式时间可判定, 说明该问题属于 NP 类。

其次, 已知固定力度二维组合测试用例集的生成问题是 NP-C 问题^[8-9]. 如上文所述, 二维组合测试用例集的生成问题是该问题在 $R = \{\{f_i, f_j\} \mid f_i, f_j \in F, 1 \leq i < j \leq n\}$ 时的一种特殊情况. 因此, 对于给定的 F, R 以及 m , 生成规模 m 的可变力度组合测试用例集是一个 NP-C 问题. 证毕。

由定理 1 可知, 我们暂时无法使用多项式时间的算法生成最优的可变力度组合测试用例集. 因此, 本文主要研究如何使用启发式算法生成近似最优的可变力度组合测试用例集。

3 相关工作及存在的问题

除 Cohen 等人提出的“狭义”可变力度组合测试方法外, 近年来人们所做的其它一些工作也与可变力度组合测试方法相关. 例如, Schroeder 等人^[4,10]提出的程序输入输出关系测试可看作是可变力度组合测试的一种特殊应用场景. 而开源测试用例生成工具 Test Vector Generator (TVG)^①以及

① <http://sourceforge.net/projects/tvg/>

微软 Czerwinka 开发的工具 PICT^[11]也可以用于可变力度组合测试用例集的生成.

在输入输出关系测试的模型中, X 表示所有输入变量的集合, Y 表示所有输出变量的集合; $D(x_i)$ 表示输入变量 x_i 的取值的集合, $T_{all} = D(x_1) \times D(x_2) \times \cdots \times D(x_n)$ 表示输入变量的所有取值组合. 在程序中, 每一个输出变量依赖于一个或多个输入变量. 用 $X(y)$ 表示与输出变量 y 有关的输入变量的集合, 用 $T(y)$ 表示只与输出变量 y 有关的输入变量的值组合, 该集合包含 $X(y)$ 中所有输入变量的取值组合. 在输入输出关系测试中, 需要生成一个规模尽可能小的测试用例集 $T \subseteq T_{all}$, 使得对于任意 $y \in Y$, $X(y)$ 中的所有组合均在 T 中出现至少一次. 显然, 若对于每一个 $y \in Y$, 都可将 $X(y)$ 对应于可变力度组合测试模型中的一个覆盖需求, 则集合 $\{X(y) | y \in Y\}$ 恰好对应于可变力度组合测试模型中的交互关系 R . 相应的, 输入输出关系测试中测试用例集的生成问题也与可变力度组合测试用例集的生成问题等价, 即输入输出关系测试中最小测试用例集的生成也是一个 NP-C 问题^[10].

为解决输入输出关系测试中测试用例集的生成问题, Schroeder 等人^[4,10]提出了 3 种测试用例集生成算法. 其中一种算法使用蛮力搜索的方式寻找最小测试用例集, 显然对于此类 NP-Hard 问题是不适用的. 另外两种近似算法分别名为 Union 和 Greedy. 其中 Union 的时间性能较好, 最坏时间复杂度不超过 $O(\sum_{k=1}^t (m \times n \times |CombSet_k|))$, 但所生成的测试用例集往往规模较大; 而 Greedy 可生成相对较小的测试用例集, 但时间复杂度高达 $O(m \times (\sum_{k=1}^t |CombSet_k|) \times (\prod_{i=1}^n a_i))$.

除上述算法外, 还有一些测试用例辅助生成工具也可用于可变力度组合测试用例集的生成, 如 TVG 和 PICT 等. 其中 TVG 内嵌一个非确定性算法, 除可用于生成固定力度组合测试用例集外, 也可针对给定的输入输出关系生成相应的测试用例集. TVG 的主要缺陷在于所生成的测试用例集规模偏大, 这一点将在后续的实验中给予说明. PICT 主要用于生成固定力度组合测试用例集, 但通过编辑输入文件中的“Sub-Model”域, 该工具也可生成可变力度的组合测试用例集. PICT 的主要问题是在生成可变力度组合测试用例集时时间性能不稳定, 远差于其在生成固定力度和“狭义”可变力度组合测试

用例集时的时间性能, 这一点也将在后续的实验中加以说明.

此外, 本课题组在组合测试领域的研究中, 也曾对因素间交互关系的特殊性加以考虑. 一方面, 针对交互作用仅存在于相邻因素之间的情况, 我们曾提出相邻因素组合测试方法并给出了可生成最小测试用例集的多项式时间算法^[12], 另一方面, 针对新的可变力度组合测试模型, 我们已经给出了 ReqOrder、ParaOrder 等测试用例集生成算法^[5-6], 其最坏时间复杂度分别为 $O(\sum_{k=1}^t (m \times |CombSet_k| \times |r_k|))$ 和 $O(\sum_{k=1}^t (m \times a_i \times |CombSet_k| \times \max_{1 \leq k \leq t} \{|r_k|\}))$.

4 基于 One-test-at-a-time 策略的算法

在固定力度组合测试用例集的生成中, one-test-at-a-time 策略由于其简单、有效、便于扩展等特点, 成为应用最普遍的方法之一^[13-14]. 相应的, 在可变力度组合测试用例集的生成中, 这一策略也得到了应用. 例如, 在 Schroeder 等人^[10]提出的输入输出关系测试及相应的测试用例集生成方法中, Greedy 算法即采用了该策略. 但由于该算法的时间性能较差, 因此有必要对基于 one-test-at-a-time 策略的可变力度组合测试用例集生成方法进行进一步的研究.

首先介绍使用 one-test-at-a-time 策略生成可变力度组合测试用例集的基本流程. 首先需根据待测系统 SUT 的因素集合 F 和交互关系 R 生成集合 $CombSet$, 并使其包含所有需要被测试用例集覆盖的因素间取值组合. 算法开始时, 初始化一个包含 0 条测试用例的空测试用例集 T . 此后, 每次均从测试用例全集 $T_{all} = V_1 \times V_2 \times \cdots \times V_n$ 中选择一条测试用例 $test$, 并将其加入测试用例集 T , 如此反复直至集合 $CombSet$ 中所有的组合均被测试用例集 T 覆盖为止. 该过程与固定力度组合测试用例集生成中 one-test-at-a-time 策略的流程基本相似, 具体可见算法 1. 此后, 测试用例集生成问题即转化为如何生成每一条测试用例这一相对简单的问题.

算法 1. One-test-at-a-time 策略基本框架.

输入: 因素集合 F , 因素交互关系 R

输出: 可变力度组合测试用例集 T

1. Initialize $T[0 \cdots 0][1 \cdots n]$; // 初始化一个 $0 \times n$ 的矩阵 T 作为测试用例集
2. 根据 F 和 R 生成集合 $CombSet$;
3. $UncovCombSet := CombSet$;
4. While ($UncovCombSet \neq \emptyset$)

- 5. 生成一条测试用例 *test*, 将其加入测试用例集 **T**;
- 6. 更新 *UncovCombSet*, 删除其中已被 *test* 覆盖的组合;
- 7. End While

为了生成尽可能小的可变力度组合测试用例集, 一种显而易见的方法就是使用贪心算法, 即每步均选择一条最优的测试用例, 使其最大限度地覆盖集合 *UncovCombSet* 中的组合. Schroeder^[10] 提出的 Greedy 算法即是如此. 但可以证明, 在 one-test-at-a-time 策略中生成一条最优测试用例是 NP-Hard 问题. 这一结论不仅在固定力度组合测试用例集的生成中成立^[15], 在可变力度组合测试用例集的生成中也是如此. 因此, Greedy 算法在生成每一条测试用例时, 都需要遍历规模为 $O(\prod_{i=1}^n a_i)$ 的整个解空间. 时间性能较差, 难以适用于较大规模的问题.

定理 2. 给定因素集合 *F*, 交互关系 *R*, 组合集合 *CombSet* 的一个真子集 *UncovCombSet* 以及一个正整数 $m < |R|$, 在使用 one-test-at-a-time 策略生成可变力度组合测试用例集的过程中, 生成一条测试用例 *test* 使其覆盖 *UncovCombSet* 中恰好 *m* 个组合是一个 NP-C 问题.

证明. 首先, 该问题属于 NP 类, 即多项式时间可判定. 具体来说, 判断一条测试用例 *test* 覆盖集合 *UncovCombSet* 中组合的数量, 需检查交互关系 *R* 中所有的覆盖需求, 以判断该覆盖需求所对应的组合是否被包含在 *UncovCombSet* 中. 该过程的时间复杂度不超过 $O(\sum_{k=1}^t (|r_k| \times |CombSet_k|))$.

其次, 在固定力度二维组合测试用例集生成问题中, 已知上述结论成立^[15]. 而正如上文所述, 固定力度二维组合测试可看作是可变力度组合测试在 $R = \{\{f_i, f_j\} | f_i, f_j \in F, i \neq j\}$ 时的一种特殊情况. 因此, 对于给定的 *F*、*R*、*UncovCombSet* 以及 *m*, 生

成一条测试用例使其覆盖 *UncovCombSet* 中恰好 *m* 个组合是一个 NP-C 问题. 证毕.

由上述结论可知, 类似 Greedy 算法这样的贪心策略在处理大规模问题时是不可行的. 因此, 我们在使用 one-test-at-a-time 策略时, 只能考虑使用多项式时间的算法生成近似最优的测试用例.

4.1 按照覆盖需求顺序生成单条测试用例

针对一个给定的组合集合 *UncovCombSet* 生成一条测试用例, 其最好情况是: 对于交互关系中的每一个覆盖需求, 均覆盖一个因素取值组合, 且该组合属于 *UncovCombSet*. 为此, 可以考虑将各个因素按照覆盖需求的顺序逐批次依次赋值.

我们注意到, 在生成这样一条测试用例时, 不同的覆盖需求的优先级也是不同的. 在直观上, 如果一个覆盖需求 $r_k \in R (1 \leq k \leq t)$ 所对应的组合集合 *CombSet_k* 中有大量的组合尚未被已有测试用例覆盖, 则在生成一条新的测试用例时, 该覆盖需求应被优先考虑, 以保证所生成的测试用例能覆盖 *CombSet_k* 中至少一个组合. 例如, 考虑图 2 所示的极端情况: 交互关系 *R* 包含两个覆盖需求 $r_1 = \{f_1, f_2\}$ 和 $r_2 = \{f_1, f_3\}$, 当 one-test-at-a-time 策略运行至某一步时, 集合 *CombSet₁* 和 *CombSet₂* 中尚未被已有测试用例覆盖的组合分别构成集合 *UncovCombSet₁* = $\{(f_1 = 1, f_2 = 1), (f_1 = 1, f_2 = 2)\}$ 和 *UncovCombSet₂* = $\emptyset$. 此时, 若优先为 *r₂* 中的因素赋值, 则在没有导向的情况下, 因素 *f₁* 和 *f₃* 的取值可随机选定, 且 $f_1 = 2$ 的概率为 50%. 而一旦 *f₁* 被赋值为 2, 当前所生成的这条测试用例就无法再覆盖 *UncovCombSet₁* 中的任何组合, 即对组合覆盖率的提高没有任何贡献, 是一条冗余的测试用例. 反之, 若优先为 *r₁* 中的因素赋值, 可以从 *UncovCombSet₁* 中选择组合 $(f_1 = 1, f_2 = 1)$ 或 $(f_1 = 1, f_2 = 2)$, 可保证覆盖至少一个组合.

SUT:	假设 $Evaluate(r_1) > Evaluate(r_2)$	假设 $Evaluate(r_1) > Evaluate(r_2)$
$F = \{f_1, f_2, f_3\}$	处理顺序: $r_1 \rightarrow r_2$	处理顺序: $r_2 \rightarrow r_1$
$R = \{r_1 = \{f_1, f_2\}, r_2 = \{f_1, f_3\}\}$	test1: 1 1 $\rightarrow$ 1 1 1 //覆盖 (1, 1)	test1: 1 $\rightarrow$ 1 1 1 //覆盖 (1, 1)
$UncovCombSet_1 = \{(1, 1), (1, 2)\}$	test2: 1 2 $\rightarrow$ 1 2 2 //覆盖 (1, 2)	test2: 2 $\rightarrow$ 1 2 2 //无覆盖
$UncovCombSet_2 = \emptyset$	测试用例集规模为 2	test3: 1 $\rightarrow$ 2 2 2 //覆盖 (1, 2)
		测试用例集规模为 3

图 2 覆盖需求的优先级及其效果(*r₂* 中的因素随机取值)

为了度量覆盖需求之间的这种优先级, 有必要定义一个覆盖需求度量函数. 由图 2 可知, 覆盖需求 *r_k* 的度量函数值应该与集合 *CombSet_k* 中未覆盖组合的数量成正比. 因此, 不妨以 *CombSet_k* 中未覆盖

组合的数量与 $\max_{1 \leq k \leq t} |CombSet_k|$ 的比值作为该函数的值. 显然, 该函数的值域为 $[0, 1]$.

此外, 我们注意到交互关系中不同的覆盖需求之间可能存在交集. 这样, 在为某个覆盖需求 *r_k* 中

的因素赋值时,其中的某些因素可能在处理其它覆盖需求时已被赋过值. 相应的,集合 $CombSet_k$ 中可被选择的未覆盖组合也会受到限制,只有部分满足已赋值因素取值条件的组合可被统计并用于计算上述度量函数. 例如,图 2 中的两个覆盖需求 r_1 和 r_2 间即存在交集,若优先处理 r_2 并选择 $(f_1=2, f_3=1)$,则在处理覆盖需求 r_1 时, $CombSet_1$ 中只有那些满足条件 $f_1=2$ 的因素取值组合可能被选择. 针对这一情况,有必要对度量函数进行调整,具体如下文所示.

进一步地,在极端情况下,若当前覆盖需求 r_k 中的所有因素均已被赋值,则说明 $CombSet_k$ 已存在一个组合被当前测试用例所覆盖. 此时可分两种情况讨论:如果该组合此前尚未被已有测试用例覆盖,说明当前测试用例覆盖了一个新的组合,度量函数值应达到最大值 1;反之,若该组合此前已被已有测试用例覆盖,则说明当前测试用例对于 r_k 没有贡献,度量函数值为最小值 0.

综合上述分析,我们可以针对覆盖需求 $r_k \in R$ ($1 \leq k \leq t$) 定义度量函数:

$$ReqEvaluate(r_k) = \frac{num_k}{(\max_{1 \leq k \leq t} |CombSet_k|)^{(n_k - p_k)/n_k}},$$

其中, num_k 表示集合 $CombSet_k$ 中未被已有测试用例覆盖、且满足已赋值因素取值条件的组合的数量, p_k 表示 r_k 中已赋值因素的数量,即 $p_k = |r_k| - |r'_k|$. 若 $p_k = n_k$,表示 r_k 中所有因素均已被赋值. 定义覆盖需求度量函数后,即可根据该函数的值来确定生成当前测试用例时各个覆盖需求处理的顺序.

如上文所述,一条测试用例对于某个覆盖需求的贡献可用该覆盖需求的度量函数值来度量:当该覆盖需求所含的因素均已被赋值后,若有新的组合被当前测试用例覆盖,则对应的度量函数值为 1;反之则为 0. 由此可见,在生成测试用例时,我们应使每个覆盖需求度量函数的值都尽可能的大. 相应的,可以将所有覆盖需求度量函数值之和,定义为针对交互关系 R 的一个全局度量函数:

$$GlobalEvaluate(R) = \sum_{k=1}^t ReqEvaluate(r_k),$$

在生成测试用例时,保证全局度量函数的值尽可能大即可.

这样,我们就得到了在 one-test-at-a-time 策略中生成一条测试用例的一种可行方法. 在该方法中,每次均选择一个覆盖需求度量函数值最大的覆盖需求,不妨假设其为 r_k ($1 \leq k \leq t$),并保证该覆盖需求中至少含有一个尚未被赋值的因素. 选择 r_k 后,对于集合 $CombSet_k$ 中所有尚未被已有测试用例覆盖,

且满足已赋值因素取值条件的组合,均尝试将其填充入当前测试用例,并计算赋值后全局度量函数的值. 最后,选择一个使得全局度量函数值最大的组合,并最终按照该组合为 r_k 中的因素赋值. 该过程反复进行,直至 F 中所有的因素均在当前测试用例中被赋值. 具体过程可见算法 2.

算法 2. 按照覆盖需求顺序生成单条测试用例的算法(DA-RO).

输入: 因素集合 F , 因素交互关系 R , 剩余组合集合 $UncovCombSet$

输出: 测试用例 $test$

1. Initialize $test[1 \cdots n]$; //初始化一个长度为 n 的数组 $test$ 作为测试用例
2. $DealedReq := \emptyset$;
3. $DealedFactor := \emptyset$;
4. While ($DealedFactor \neq F$)
5. For $k := 1$ to t
6. If ($r_k \in R - DealedReq$) Then
 计算 $ReqEvaluate(r_k)$;
 End If
7. End For
8. 选择一个 $r_k \in R - DealedReq$, 使得 $ReqEvaluate(r_k)$ 最大;
9. For Each $comb \in CombSet_k$
10. If ($comb$ 满足 $test$ 中已赋值因素的取值条件) Then
11. 假设 r_k 中因素按照 $comb$ 赋值, 计算赋值后的 $GlobalEvaluate(R)$;
12. End If
13. End For
14. 选择一个 $comb \in CombSet_k$, 使得 $GlobalEvaluate(R)$ 最大;
15. 依据组合 $comb$ 对 r_k 中的因素进行赋值;
16. $DealedFactor := DealedFactor + r_k$;
17. $DealedReq := DealedReq + \{r_k\}$;
18. End While

下面分析算法 2 的时间性能. 首先,计算覆盖需求度量函数值所需的时间复杂度与覆盖需求、未覆盖组合的数量以及覆盖需求中已赋值因素的数量均有关,难以准确描述,只能得到一个上界 $O(|r_k| \times |CombSet_k|)$,相应的,计算全局度量函数值所需的时间复杂度上界为 $O(\sum_{k=1}^t (|r_k| \times |CombSet_k|))$. 在选择一个覆盖需求时,最多需要比较所有 t 个覆盖需求对应的度量函数值,其时间复杂度不超过 $O(\sum_{k=1}^t (|r_k| \times |CombSet_k|))$. 对所选覆盖需求 r_i ($i = 1, 2, \dots, t$) 中的因素进行赋值,需要针对集合 $CombSet_i$ 中所有尚未被覆盖、且满足已赋值因素取值条件的

组合分别计算交互关系度量函数值,其时间复杂度不超过 $O(|CombSet_i| \times \sum_{k=1}^t (|r_k| \times |CombSet_k|))$.

上述操作最多进行 $\min(t, |F|)$ 次后,即可保证 F 中所有的因素都在当前测试用例中被赋值. 因此,使用算法 2 生成一条测试用例所需的最坏时间复杂度为

$$O(\sum_{i=1}^t \sum_{k=1}^t (|r_k| \times |CombSet_k| \times |CombSet_i|)).$$

4.2 按照因素顺序生成单条测试用例

针对一个给定的组合集合 $UncovCombSet$ 生成一条测试用例,也可以考虑采取逐个因素依次赋值的方法进行. 如 AETG^[16]、TCG^[17]、DDA^[18] 等经典的固定力度组合测试用例集的生成算法即是如此. 本小节将尝试使用逐因素依次赋值的方法来生成一条测试用例,该方法也可看做是 DDA 算法在可变量组合测试用例集生成中的一种扩展.

采取逐因素依次赋值的方式生成测试用例,首先必须应确定因素处理顺序. 类似的,我们可以定义一个因素度量函数,并按照因素度量函数的值确定因素赋值的先后顺序. 与覆盖需求度量函数的意义类似,计算因素度量函数的意义在于优先处理那些涉及更多未覆盖组合的因素. 因此,可以考虑将因素度量函数定义为所有包含该因素的覆盖需求所对应的覆盖需求度量函数值之和,即

$$FactorEvaluate(f_i) = \sum_{k=1}^t ReqEvaluate'(r_k, f_i),$$

其中,函数 $ReqEvaluate'(r_k, f_i)$ 是覆盖需求度量函数 $ReqEvaluate(r_k)$ 的一个变种,其定义为

$$ReqEvaluate'(r_k, f_i) = \begin{cases} ReqEvaluate(r_k), & f_i \in r_k \\ 0, & f_i \notin r_k \end{cases}.$$

显然,一个因素若被包含于数量较多的覆盖需求,且这些覆盖需求都具有较高覆盖需求度量函数值,则该因素也就具有较高的因素度量函数值. 这样的因素在生成一条测试用例时也应尽早赋值,以尽可能提高测试用例对于未覆盖组合的贡献.

由此,可得到在 one-test-at-a-time 策略中生成一条测试用例的另外一种可行方法. 在该方法中,每次均选择一个因素度量函数值最大的因素,不妨假设其为 $f_i (1 \leq i \leq n)$. 选择因素 f_i 后,对于因素取值集合 V_i 中的所有 a_i 个取值,均尝试将其填充入当前测试用例,并计算进行该赋值后全局度量函数的值. 最后选择一个使得全局度量函数值最大的取值,并最终按照该取值为因素 f_i 赋值. 该过程反复进行,直至 F 中所有的因素均在当前测试用例中被赋值. 具体过程可见算法 3.

算法 3. 按照因素顺序生成单条测试用例的算法(DA-FO).

输入: 因素集合 F , 因素交互关系 R , 剩余组合集合 $UncovCombSet$

输出: 测试用例 $test$

1. Initialize $test[1 \cdots n]$; //初始化一个长度为 n 的数组 $test$ 作为测试用例
2. $DealedFactor := \emptyset$;
3. While ($DealedFactor \neq F$)
4. For $i := 1$ to n
5. If ($f_i \in F - DealedFactor$) Then
 计算 $FactorEvaluate(f_i)$;
 End If
6. End For
7. 从 $F - DealedFactor$ 中选择一个因素 f_i , 使得 $FactorEvaluate(f_i)$ 最大;
8. For Each $v \in V_i$
9. 假设 $test[i] := v$, 计算此时的 $GlobalEvaluate(R)$;
10. End For
11. 选择一个取值 v , 使得全局密度最大;
12. $test[i] := v$;
13. $DealedFactor := DealedFactor + \{f_i\}$;
14. End While

算法 3 在选择因素时,可能出现多个待选因素均具有最大度量值的情况. 在为给定因素选择赋值时,这一情况也可能出现. 与算法 2 类似,使用 First 策略选择其中下标最小的因素或取值即可.

下面分析算法 3 的时间性能. 在最坏情况下,每次为选择一个因素而计算 F 中所有 n 个因素的因素度量函数时,首先应计算 R 中所有 t 个覆盖需求所对应的覆盖需求度量函数,即选择一个因素所需的最坏

时间复杂度为 $O(\sum_{k=1}^t (|r_k| \times |CombSet_k|))$. 对所选

因素 f_i 进行赋值,需要针对 V_i 中所有 a_i 个可能取值分别计算全局度量函数的值,其时间复杂度为

$O(a_i \times \sum_{k=1}^t (|r_k| \times |CombSet_k|))$. 上述操作进行 $|F|$

次后,即可保证 F 中所有的因素都在当前测试用例中被赋值. 因此,使用算法 3 生成一条测试用例的最坏

时间复杂度为 $O(\sum_{i=1}^n \sum_{k=1}^t (a_i \times |r_k| \times |CombSet_k|))$.

可见,在 $\sum_{i=1}^n a_i < \sum_{i=1}^n |CombSet_i|$ 的条件下,算法 3 的最坏时间复杂度低于算法 2 的最坏时间复杂度,而这一条件在大多数情况下是成立的.

5 实 验

为了检验算法的效果,我们基于算法 2、算法 3

分别实现了工具 DA-RO、DA-FO,并将它们与一些已有的算法和工具进行比较. 以下实验均在 2.66 GHz Pentium IV 处理器、1GB 内存的环境下进行.

5.1 可变力度组合测试用例集生成中的效果

首先检验算法在生成可变力度组合测试用例集时的效果. 我们将两种算法与 ReqOrder、ParaOrder、Union、Greedy、PICT、TVG 等算法和工具进行比较. 其中 PICT、TVG 均下载自网络, Union、Greedy 则按照文献[10]的描述实现. 对于算法 Greedy, 尽管 Cheng 等人^[19]针对其时间性能较差的弱点提出了一种解空间约简算法以提高其效率,但方法并非 Greedy 的变种或改进,而只是一种对算法输入进行预处理的方法. 原输入约简后可作为 Greedy 的输入,也同样可以作为其它测试用例集生成算法的输入. 考虑到该方法可能导致测试用例集规模增大,故我们在实现时不做考虑.

下面讨论实验的输入. 尽管文献[10,19]中已给出一些实验结果,但实验的输入数据却没有公开,因此我们需要设计实验方案. 首先,选择因素集合 $F = \{3^{10}\}$ 和 $F = \{2^3 \times 3^3 \times 4^3 \times 5\}$ 以分别代表单一水平

系统和混合水平系统;其次,从给定的集合中顺序选择给定数量的覆盖需求,从而建立交互关系 R (本次实验所使用的覆盖需求集合为附录中的 $ReqSet$). 出于两方面的原因,集合 $ReqSet$ 中所有覆盖需求所含因素的数量(即该覆盖需求的交互力度)均在 $2 \sim 4$ 之间: (1) Kuhn 等人^[2-3]曾指出,故障所对应的因素交互力度 (FTFI) 多数在 $2 \sim 4$ 之间,且 FTFI 值为 3 以上的故障数量极少; (2) 当交互关系 R 中出现极少数交互力度较大的覆盖需求时,测试用例集规模将主要取决于这些交互力度较大的覆盖需求,而算法性能本身的影响力则相对比较小.

表 3 和表 4 从测试用例集规模和算法运行的 CPU 时间这两个方面,给出了几种算法和工具在生成可变力度组合测试用例集时的效果. 由于 TVG 基于 GUI 界面,难以精确统计 CPU 时间,因此没有给出其耗时方面的数据. 对于 PICT,尽管相关文献说明它能处理任意数量的覆盖需求 (Sub-Models)^[11],但在实际运行中我们发现,当 $|R| > 2$ 且覆盖需求之间存在交集时,PICT 就已难以正常运行,在运行长达 1 h 的时间后仍然无法得出结果.

表 3 可变力度组合测试用例集的规模及算法耗时 ($F = \{3^{10}\}$)

R	DA-RO	DA-FO	ReqOrder	ParaOrder	Union	Greedy	TVG	PICT
2	81 (0.08 s)	81 (0.02 s)	81 (0.01 s)	81 (0.01 s)	162 (0.01 s)	81 (1.04 s)	81 (—)	81 (0.69 s)
3	81 (0.11 s)	81 (0.02 s)	81 (0.01 s)	81 (0.01 s)	242 (0.01 s)	81 (1.91 s)	84 (—)	—
10	86 (0.44 s)	84 (0.07 s)	99 (0.01 s)	96 (0.01 s)	503 (0.01 s)	93 (6.77 s)	86 (—)	—
20	95 (0.98 s)	99 (0.14 s)	128 (0.01 s)	105 (0.02 s)	858 (0.02 s)	91 (18.0 s)	105 (—)	—
30	116 (1.98 s)	120 (0.30 s)	157 (0.01 s)	111 (0.04 s)	1599 (0.04 s)	109 (34.0 s)	125 (—)	—
40	126 (3.02 s)	123 (0.44 s)	163 (0.01 s)	120 (0.05 s)	2057 (0.06 s)	111 (50.7 s)	135 (—)	—
50	135 (3.74 s)	135 (0.60 s)	172 (0.04 s)	132 (0.06 s)	2635 (0.10 s)	125 (70.2 s)	139 (—)	—
60	141 (4.96 s)	142 (0.77 s)	190 (0.05 s)	144 (0.08 s)	3257 (0.15 s)	141 (110 s)	150 (—)	—

表 4 可变力度组合测试用例集的规模及算法耗时 ($F = \{2^3 \times 3^3 \times 4^3 \times 5\}$)

R	DA-RO	DA-FO	ReqOrder	ParaOrder	Union	Greedy	TVG	PICT
2	64 (0.04 s)	64 (0.01 s)	64 (0.01 s)	64 (0.01 s)	104 (0.01 s)	64 (1.35 s)	64 (—)	64 (0.71 s)
3	144 (0.24 s)	144 (0.04 s)	144 (0.01 s)	144 (0.01 s)	248 (0.01 s)	144 (2.06 s)	144 (—)	—
10	144 (0.71 s)	144 (0.11 s)	144 (0.01 s)	144 (0.02 s)	505 (0.01 s)	144 (8.60 s)	144 (—)	—
20	160 (1.78 s)	160 (0.23 s)	166 (0.01 s)	161 (0.03 s)	929 (0.01 s)	160 (26.3 s)	161 (—)	—
30	165 (4.35 s)	175 (0.50 s)	204 (0.01 s)	179 (0.06 s)	1861 (0.06 s)	162 (66.8 s)	179 (—)	—
40	165 (5.67 s)	172 (0.65 s)	209 (0.02 s)	183 (0.11 s)	2244 (0.08 s)	167 (111 s)	181 (—)	—
50	182 (7.96 s)	186 (0.87 s)	229 (0.02 s)	200 (0.14 s)	2820 (0.13 s)	183 (161 s)	194 (—)	—
60	197 (11.15 s)	200 (1.22 s)	237 (0.05 s)	204 (0.16 s)	3587 (0.21 s)	197 (259 s)	209 (—)	—

通过表 3 和表 4 可以看出,在大多数情况下 Greedy 都可生成规模最小的测试用例集,但其运行时间也是所有算法(不考虑 TVG 和 PICT)中最长的. 在本文提出的算法中,DA-RO 共 10 次生成最小测试用例集,DA-FO 共 7 次生成最小测试用例集. 从总体上看,除 Greedy 外,DA-RO 在所有算法中效果最好,DA-FO 与 ParaOrder 效果相当,其次为 TVG 和 ReqOrder,而 Union 的效果最差. 在算法运

行时间方面,ReqOrder 用时最短,其次为 ParaOrder 和 Union,DA-FO 的时间性能略差于前三者,但优于 DA-RO,这与前文中对时间复杂度进行理论分析的结果是基本吻合的.

5.2 固定力度和“狭义”可变力度组合测试用例集生成中的效果

除检验算法在生成可变力度组合测试用例集时的效果外,我们还将检验它们在可变力度组合测试

的一种特例,即固定力度组合测试中的效果. 由于已有实验结果表明 Union 和 ReqOrder 的效果相对较差,生成测试用例集的规模过大,因此在后续的实验中将不再考虑该算法.

表 5 给出了 13 种算法和工具在生成固定力度

3 维组合测试用例集时的结果. 其中, AETG^[16]、GA^[20]、ACA^[20] 的数据来源于文献[20]; GA-N、IPO-N 的数据来源于文献[21]; IPO^[8] 的数据通过运行工具 TConfig^① 而得; Jenny 的数据则通过运行开源工具 Jenny^② 而得.

表 5 固定力度 3 维组合测试用例集的规模

	DA-RO	DA-FO	ParaOrder	Greedy	TVG	PICT	AETG	GA	ACA	GA-N	IPO-N	IPO	Jenny
S_1	50	47	53	43	48	48	38	33	33	52	47	48	51
S_2	64	64	106	64	120	111	77	64	64	85	64	64	112
S_3	213	211	225	184	239	215	194	125	125	223	173	200	215
S_4	362	359	363	325	409	369	330	331	330	389	371	366	373
S_5	1592	1587	1624	1474	1949	1622	1473	1501	1496	1769	1502	1678	1572
S_6	242	237	225	220	269	241	218	218	218	336	199	239	236
S_7	119	116	108	106	133	119	114	108	106	120	113	120	130
S_8	365	369	377	388	429	368	377	360	361	373	368	464	397

注: $S_1: 3^6$; $S_2: 4^6$; $S_3: 5^6$; $S_4: 6^6$; $S_5: 10^6$; $S_6: 5^7$; $S_7: 5^2 4^2 3^2$; $S_8: 10^1 6^2 4^3 3^1$.

通过表 5 可以看出,相对于几种已有的专用于固定力度组合测试用例集生成的算法和工具(如 GA、ACA、AETG),本文提出的两种算法在性能上存在一定的差距. 而相对于几种可用于生成可变量度组合测试用例集的算法和工具,本文提出的算法除略差于 Greedy,但在大多数情况下均优于 ParaOrder、PICT 和 TVG. 在本次实验中,尽管 Greedy 可以生成规模很小的测试用例集,但其时间性能依然是其最大的缺陷,例如对于输入 S_5 , Greedy 运行所需的 CPU 时间已接近 1 h,而其它算法和工具基本都可在 10 s 之内得出结果.

最后,我们还对几种算法和工具在“狭义”可变量度组合测试用例集生成中的效果进行了检验. 本次实验比较了本文提出的两种算法、Greedy、PICT、TVG 以及 Cohen 等人^[1] 提出的专门用于生成“狭义”可变量度组合测试用例集的算法 SA(Simulated Annealing).

表 6 给出了几种算法和工具生成的“狭义”可变量度组合测试用例集的规模. 该实验所使用的输入以及算法 SA 的数据均来源于文献[1]. 对于其中的部分输入,由于 Greedy 算法未能在 1h 的规定 CPU 时间内运行完毕而没有得到实验结果.

表 6 “狭义”可变量度组合测试用例集的规模

	C	DA-RO	DA-FO	ParaOrder	Greedy	TVG	PICT	SA
VSCA($m; 2, 3^{15}, C$)	$\emptyset$	21	20	33	—	22	35	16
	CA(3, 3 ³)	28	29	27	—	27	81	27
	CA(3, 3 ³) ²	28	29	33	—	30	729	27
	CA(3, 3 ³) ³	28	30	33	—	30	785	27
	CA(3, 3 ⁴)	32	34	27	—	35	105	27
	CA(3, 3 ⁵)	40	42	48	—	41	131	33
	CA(3, 3 ⁴), CA(3, 3 ⁵), CA(3, 3 ⁶)	46	46	49	—	53	1376	34
	CA(3, 3 ⁶)	46	46	53	—	48	146	34
	CA(3, 3 ⁷)	53	53	54	—	54	154	41
	CA(3, 3 ⁹)	60	60	62	—	62	177	50
VSCA($m; 2, 4^3 5^3 6^2, C$)	CA(3, 3 ¹⁵)	70	78	82	—	81	83	67
	$\emptyset$	41	40	40	44	44	43	36
	CA(3, 4 ³)	64	64	64	67	67	384	64
	CA(3, 4 ³ 5 ²)	131	132	140	119	132	781	100
	CA(3, 5 ³)	125	125	125	126	125	750	125
	CA(3, 4 ³), CA(3, 5 ³)	125	125	129	126	125	8000	125
	CA(3, 4 ³ 5 ³ 6 ¹)	207	211	220	209	237	1266	171
	CA(3, 5 ¹ 6 ²)	180	180	180	181	180	900	180
VSCA($m; 2, 3^{20} 10^2, C$)	CA(3, 4 ³ 5 ³ 6 ²)	256	261	264	258	302	261	214
	$\emptyset$	100	100	100	—	101	100	100
	CA(3, 3 ²⁰)	100	105	119	—	103	940	100
	CA(3, 3 ²⁰ 10 ²)	401	409	445	—	423	423	304

① <http://www.site.uottawa.ca/~awilliam/>
② <http://burtleburtle.net/bob/math/jenny.html>

通过表 6 可以看出,使用模拟退火算法的 SA 对于所有输入均可生成最小的测试用例集.若不考虑 SA 所得的结果,则在其它算法和工具中,DA-RO 共 17 次生成了最小测试用例集,DA-FO 共 12 次生成了最小测试用例集,均明显优于其它算法和工具.我们还注意到,此前在测试用例集规模方面表现优异的 Greedy 在本次试验中没有任何优势,且对于部分输入由于其较差的时间性能而没有在规定 CPU 时间内得到结果.

5.3 实验结论

通过以上共 3 组实验可以看出,本文提出的算法 DA-RO 和 DA-FO 在可变力度组合测试用例集、固定力度组合测试用例集以及“狭义”可变力度组合测试用例集的生成中均具有比较好的效果.其中 DA-RO 所生成测试用例集的规模普遍小于 DA-FO,但时间性能则不及后者.由于上文所述的几种算法均为启发式算法,都不能保证生成最优解,因此在实际的测试工作中,综合运用各种测试用例生成方法,取长补短,可以取得更好的效果.而本文提出的算法在各场景下均具有一定的优势,在组合测试、尤其是可变力度组合测试用例集的生成时,可成为一种较优的选择.

6 小 结

针对传统的固定力度组合测试方法以及 Cohen 等人提出的“狭义”可变力度组合测试方法中所存在的问题,本课题组提出一种新的可变力度组合测试方法.这种新的可变力度组合测试模型可以更为准确地描述待测系统中的因素间交互作用,进而可为测试用例集的设计和生成提供更好的指导.针对这一模型,在假设因素间交互关系已知的情况下,本文给出了两种基于 one-test-at-a-time 策略的可变力度组合测试用例集生成算法.实验表明,本文提出的算法除在可变力度组合测试用例集的生成中具有很好的效果外,在固定力度组合测试用例集以及“狭义”可变力度组合测试用例集的生成中同样具有一定的竞争力.

目前,组合测试领域的大多数工作关注于相对简单的固定力度组合测试模型.而事实上,不同类型组合测试模型的选择对于组合测试的应用起着至关重要的作用.因此,有必要深入研究组合测试的各种应用场景,总结更多、更详细的组合测试模型,指导人们迅速区分哪些软件系统或软件系统的哪些方面适合使用组合测试方法.此外,由于组合测试用例集

的生成问题是 NP-C 问题,而元启发式搜索方法是一种解决 NP-C 问题的常用方法,因此有必要对其在组合测试用例集生成中的应用进行进一步的研究和探索.

参 考 文 献

- [1] Cohen M B, Gibbons P B, Mugridge W B, Colbouns C J, Collofello J S. Variable strength interaction testing of components//Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC2003). Dallas, TX, USA, 2003: 413-418
- [2] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing//Proceedings of the 27th NASA/IEEE Software Engineering Workshop. NASA Goddard Space Flight Center, Greenbelt, MD, USA, 2002: 91-95
- [3] Kuhn D R, Wallace D R. Software fault interaction and implication for software testing. IEEE Transactions on Software Engineering, 2004, 30(6): 1-4
- [4] Schroeder P J, Korel B. Black-box test reduction using input-output analysis//Proceedings of the International Symposium on Software Testing and Analysis (ISSTA2000). Portland, Oregon, USA, 2000: 21-22
- [5] Wang Ziyuan, Nie Changhai, Xu Baowen. Generating combinatorial test suite for interaction relationship//Proceedings of the 4th International Workshop on Software Quality Assurance (SOQUA2007). Croatia, 2007: 55-61
- [6] Wang Ziyuan, Xu Baowen, Nie Changhai. Greedy heuristic algorithms to generate variable strength combinatorial test suite//Proceedings of the 8th International Conference on Quality Software (QSIC2008). Oxford, UK, 2008: 155-160
- [7] Wang Zi-Yuan, Xu Bao-Wen, Nie Chang-Hai. Survey of combinatorial test generation. Journal of Frontiers of Computer Science and Technology, 2008, 2(6): 571-588(in Chinese)
(王子元, 徐宝文, 聂长海. 组合测试用例生成技术. 计算机科学与探索, 2008, 2(6): 571-588)
- [8] Lei Y, Tai K C. In-parameter-order: A test generation strategy for pairwise testing//Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE1998). Washington, USA, 1998: 254-261
- [9] Williams A W, Probert R L. Formulation of the interaction test coverage problem as an integer program//Proceedings of the 14th International Conference on the Testing of Communicating Systems. Berlin, Germany, 2002: 283-298
- [10] Schroeder P J. Black-box test reduction using input-output analysis[Ph. D. dissertation]. Chicago, IL, USA: Department of Computer Science, Illinois Institute of Technology, 2001
- [11] Czerwonka J. Pairwise testing in real world: Practical extensions to test case generator//Proceedings of the 24th Pacific Northwest Software Quality Conference. Portland, Oregon, USA, 2006: 419-430

- [12] Wang Zi-Yuan, Nie Chang-Hai, Xu Bao-Wen, Shi Liang. Optimal test suite generation methods for neighbor factors combinatorial testing. Chinese Journal of Computers, 2007, 30(2): 200-211(in Chinese)
(王子元, 聂长海, 徐宝文, 史亮. 相邻因素组合测试用例集的最优生成方法. 计算机学报, 2007, 30(2): 200-211)
- [13] Bryce R C, Colbourn C J. Constructing interaction test suites with greedy algorithms//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE2005). Long Beach, California, USA, 2005: 440-443
- [14] Bryce R C, Colbourn C J, Cohen M B. A framework of greedy methods for constructing interaction test suites//Proceedings of the 27th International Conference on Software Engineering (ICSE2005). St. Louis, Missouri, USA, 2005: 146-155
- [15] Bryce R C, Colbourn C J. The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability, 2007, 17(3): 159-182
- [16] Cohen D M, Dalal S R, Fredman M L, Patton G C. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 1997, 23(7): 437-444
- [17] Colbourn C J, Cohen M B, Turban R C. A deterministic density algorithm for pairwise interaction coverage//Proceedings of the IASTED International Conference on Software Engineering (SE2004). Innsbruck, Austria, 2004: 345-352
- [18] Tung Yu-Wen, Aldiwan W S. Automating test case generation for the new generation mission software system//Proceedings of the IEEE Aerospace Conference. Big Sky, Montana, USA, 2000: 431-437
- [19] Cheng C, Dumitrescu A, Schroeder P J. Generating small combinatorial test suites to cover input-output relationships//Proceedings of the 3rd International Conference on Quality Software (QSIC2003). Dallas, TX, USA, 2003: 76-82
- [20] Shiba Toshiaki, Tsuchiya Tatsuhiro, Kikuno Tohru. Using artificial life techniques to generate test cases for combinatorial testing//Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC2004). Volume 1, Hong Kong, China, 2004: 72-78
- [21] Nie Changhai, Xu Baowen, Shi Liang, Dong Guowei. Automatic test generation for N -way combinatorial testing//Proceedings of the 2nd International Workshop on Software Quality (SOQUA2005). Fair and Convention Center, Erfurt, German, 2005: 203-211

附 录

本文 5.1 节的实验中,所使用的两个因素集合分别为 $F = \{3^{10}\}$ 和 $F = \{2^3 \times 3^3 \times 4^3 \times 5\}$. 它们均含有 10 个因素,即 $F = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}\}$. 为了方便表述,我们不妨以各个因素的下标代表该因素,从而将因素集合标记为 $F = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. 在这一表示方式下,实验所使用的覆盖需求集合 *ReqSet* 如下:

$ReqSet = \{\{2, 3, 8, 9\}, \{1, 2, 3, 10\}, \{5, 6, 8, 9\}, \{1, 2, 4, 10\}, \{1, 4, 9\}, \{7, 8, 9\}, \{5, 10\}, \{2, 4, 5\}, \{1, 3, 7, 8\}, \{5, 7\}, \{3, 4, 5, 9\}, \{3, 4, 6\}, \{6, 7\}, \{1, 7, 9\}, \{9, 10\}, \{1, 6\}, \{2, 4, 6,$

$10\}, \{2, 7, 8, 10\}, \{1, 5\}, \{1, 3, 4\}, \{2, 4, 7, 10\}, \{3, 5, 8, 9\}, \{1, 3, 7, 10\}, \{1, 2, 8, 9\}, \{1, 4, 8, 10\}, \{4, 5, 8, 9\}, \{2, 6, 8, 10\}, \{2, 4, 7, 9\}, \{2, 3, 6\}, \{4, 5, 6, 8\}, \{1, 3, 8, 10\}, \{2, 3, 4\}, \{2, 3, 7\}, \{3, 6, 10\}, \{4, 7, 8\}, \{2, 3, 5, 8\}, \{3, 6, 9\}, \{1, 2, 7, 8\}, \{4, 6, 9\}, \{1, 2, 3, 9\}, \{3, 4, 10\}, \{2, 6, 9\}, \{2, 4, 6, 8\}, \{1, 2, 3, 8\}, \{3, 5, 6, 8\}, \{4, 6, 8, 10\}, \{1, 2, 8, 10\}, \{1, 2, 4, 7\}, \{2, 5, 9\}, \{2, 5, 6\}, \{1, 7, 8, 10\}, \{3, 7, 8, 10\}, \{3, 7, 9\}, \{3, 4, 7\}, \{3, 4, 8\}, \{2, 4, 8, 10\}, \{1, 3, 8, 9\}, \{1, 2, 7, 10\}, \{2, 4, 8, 9\}, \{1, 2, 4, 8\}\}.$



WANG Zi-Yuan, born in 1982, Ph. D.. His research interest is software testing.

QIAN Ju, born in 1980, Ph. D., associate professor. His research interest is software analysis.

CHEN Lin, born in 1979, Ph. D.. His research interest is software analysis.

XU Bao-Wen, born in 1961, Ph. D., professor, Ph. D. supervisor. His research interests include programming language, software engineering, etc.

Background

Combinatorial testing technique is widely used to detect failures caused by interactions among parameters. People usually pay their attention on the fault detection ability of combinatorial testing. Therefore, the most works about combinatorial testing focus on the problem of generating test suite. Besides the fixed strength combinatorial testing, the variable strength combinatorial testing is also required for some special applications. We focus on the problem of generating test suite for variable strength combinatorial testing,

and proposed some test generation algorithms in this paper. This work is based on existing results of our group in the filed of combinatorial testing, including test generation, fault location, test prioritization, and etc. This paper is mainly supported by a project of National Natural Science Foundation of China (90818027). This project mainly research on software testing techniques for the evolution of software quality. In such a project, combinatorial testing is one of methods to generate and optimize test cases.