基于组合测试的软件故障定位的自适应算法

周吴杰[1),2)] 张德平[4)] 徐宝文[2),3)]

[1)](东南大学计算机科学与工程学院 南京 210096)

[2)](南京大学软件新技术国家重点实验室 南京 210093)

[3)](南京大学计算机科学与技术系 南京 210093)

[4)](南京航空航天大学信息科学与技术学院 南京 210016)

摘 要 在研究了 Martínez 等人提出的组合测试错误定位模型的基础上,改进了他们用高维覆盖表来构造错误定位表的方法,并且针对安全值已知的待测系统提出了至多 t 维的错误交互定位的自适应算法,从而解决了 Martínez 等人提出的开问题之一,并进一步分析了自适应算法的性能,证明了定位错误交互需调用的测试过程数目是关于错误交互数与因素数多项式阶增长的,拓展了 Martínez 等人提出的针对 $t=2$ 的错误交互定位的自适应算法.

关键词 组合测试;覆盖表;错误定位表;自适应算法

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.01509

An Adaptive Algorithm of Locating Fault Interactions Based Combinatorial Testing

ZHOU Wu-Jie[1),2)] ZHANG De-Ping[4)] XU Bao-Wen[2),3)]

[1)](*Department of Computer Science & Engineering, Southeast University, Nanjing* 210096)

[2)](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

[3)](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

[4)](*College of Information Science and Technology, Nanjing University of Aeronautics & Astronautics, Nanjing* 210016)

Abstract In the paper, we study the model to locate fault interactions proposed by Martínez C et al, improve the conclusion of constructing error locating array based on the higher strength covering arrays. We propose an adaptive algorithm of locating the faulty interactions whose strength is at most t in the software under test with known safe values. The algorithm solves one of the open problems proposed by Martínez C et al. We analyze the performance of the algorithm that our algorithm performs a number of tests that is polynomial in $\log k$ and d, where k is the number of parameters in the system and d is an upper bound on the number of the faulty interactions whose strength is at most t.

Keywords combinatorial testing; covering array; error locating array; adaptive algorithm

1 引 言

随着计算机技术的飞速发展,软件与硬件系统越来越庞大,软件测试是保证这些系统的质量的一个关键手段.然而影响系统运行的因素有很多,如何来检测各个因素以及它们之间的相互作用对系统产生的影响,一般采取组合测试策略.对于小的软件系

收稿日期:2010-11-16;最终修改稿收到日期:2011-06-09.本课题得到国家自然科学基金(90818027,91018005)、国家"八六三"高技术研究专题项目与发展计划项目基金(2009AA01Z147)、国家"九七三"重点基础研究发展规划项目基金(2009CB320703)资助.**周吴杰**,男,1973 年生,博士研究生,讲师,主要从事软件测试技术、软件可靠性等方面的科研工作. E-mail: zhouwujie@seu.edu.cn. **张德平**,男,1973 年生,博士,讲师,主要从事软件测试技术、软件可靠性、数理统计等方面的教学和科研工作. **徐宝文**(通信作者),男,1961 年生,博士,教授,博士生导师,主要从事程序设计语言、软件工程、并行与网络软件等方面的教学与科研工作. E-mail: bwxu@nju.edu.cn.

统,其因素数很少时,可采取穷尽测试来测试各个因素交互,但对于大的软件系统,则穷尽测试是不可能的.例如,一个软件系统有 10 个因素,每个因素有 4 个不同的取值,则穷尽测试需要 $4^{10}=1048576$ 个测试用例.如果我们选择测试用例来测试所有因素的两两交互,或者 t 维交互,则测试用例会大大减少.上例中,两两交互测试至多只需 25 个测试用例.Kuhn 等人[1]的研究表明测试两两交互覆盖能发现大约 70%的错误,三维组合覆盖能发现 90%的错误.

人们应用组合测试策略进行软件测试已有很长时间,最早的组合测试是应用正交拉丁方和正交试验设计来对软件进行测试[2-3],后来贝尔实验室提出了基于贪心策略的组合测试用例生成算法,并开发了相应的测试工具[4].此后,美国喷气推进实验室的 Tung 等人以及 Colbourn 等人[5-8]在此基础上分别提出了改进的 TCG 方法和确定密度算法.2001 年,Lei 等人[9-11]提出了基于参数顺序扩展策略的组合测试用例生成算法.Kobayashi 和 Tsuchiya 等人[12]则在 2002 年提出用于生成两两组合测试用例集的基于递归的代数方法.2003 年,Cohen 等人[13-15]将模拟退火算法应用于变力度组合测试用例的生成.Shiba 和 Tsuchiya 等人[16]在 2004 年研究了遗传算法和蚁群算法在组合测试数据自动生成中的应用.

在组合测试中,由于测试失败预示着系统组件中存在错误,这就需要测试人员找出触发系统故障的错误交互.目前,对组合测试的结果进行调试和错误定位分析的研究还很少.2002 年 Zeller 和 Hildebrandt[17]提出 Δ-调试方法对待测系统中出现的故障进行调试以简化与孤立导致故障的环境或输入.2006 年,Yilmaz 等人[18]用分类树法来定位错误交互,系统的错误交互一般很难被精确地确定.2006 年徐宝文等人[19]提出了一种基于组合测试的软件故障调试方法,可把引发软件故障的错误交互锁定在很小的范围内,但他们的方法仅对因素个数较少的情况可用.2008 年 Colbourn 和 McClary[20]提出了 (d,t)错误定位表及错误侦测表的概念,用这些表来对组合交互错误进行定位.随后 Martínez 等人[21-22]提出了一般的错误定位表,在此模型下提出了自适应算法来定位错误交互,但他们只处理了二维组合覆盖,即在软件故障是由二维错误交互引发的假设下,给出错误交互定位的自适应算法,对于一般的情形并没有给出相应的解决办法,而是作为开问题提出来.因此,当利用组合测试策略来对系统进行测试时,某些因素的组合会导致系统发生故障,如何定位和侦测哪些因素组合会导致系统故障,是组合测试研究中一个亟待解决的问题.

本文在研究 Martínez 等人[21-22]提出的组合测试错误定位表模型的基础上,改进了他们提出的用高维覆盖表来构造错误定位表的方法,然后针对安全值已知时定位二维错误交互的自适应算法没法处理高维错误交互的不足,提出了具有安全值已知时至多 t 维的错误交互定位的自适应算法,从而解决了 Martínez 等人提出的开问题之一,并进一步分析了自适应算法的性能,得到结论:定位错误交互需调用的测试过程数目是关于待测系统中错误交互数与因素数的多项式阶增长的,推广了 Martínez 等人提出的针对 $t=2$ 的错误交互定位自适应算法.

本文第 2 节介绍基本的组合测试模型、故障定位模型以及相关结论,并对 Martínez 等人的用高维覆盖表来构造错误定位表的方法进行推广,使得这个方法在实践中能够得到实用;第 3 节我们分析 Martínez 等人提出的在安全值已知的待测系统中的定位二维错误交互的自适应算法,提出在此待测系统中的至多 t 维的错误交互定位的自适应算法;第 4 节我们分析自适应算法的性能,得到结论:定位错误交互需调用测试过程数目是关于待测系统中错误交互数与因素数的多项式阶增长的,并对 Martínez 等人提出的针对 $t=2$ 的错误交互定位自适应算法进行推广;最后总结全文并讨论未来可进行研究的方向.

2 组合测试故障定位模型

为了研究基于组合测试的软件故障与侦测技术,这里先给出一些记号和形式化定义.

假设影响待测系统(Software Under Test, SUT)的因素共有 k 个,因素 i 有 $v_i(1\leq i\leq k)$个可能的取值,用 $0,1,\cdots,v_i-1$ 表示.我们用记号 $[0,v_i-1]$表示集合$\{0,1,\cdots,v_i-1\}$.假设这些参数的取值是相互独立的,即某个参数的具体取值不会影响其它参数的取值或存在性.

定义 1. 设 k 维向量 $\boldsymbol{T}=(T_1,T_2,\cdots,T_k)$,其中 $T_i\in[0,v_i-1]$,$i=1,2,\cdots,k$,则称这个 k 维向量 $\boldsymbol{T}$ 为第 i 个因素取值为 T_i的测试用例.

定义交互的概念如下.

定义 2. 设集合 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,$

$(i_t,a_{i_t})\}$，其中因素 i_j 互不相同，$a_{i_j}\in\{0,1,\cdots,v_{i_j}-1\}(j=1,2,\cdots,t)$，则称这个集合 I 为一个 t 维交互. 称集合 $f_I=\{i_1,i_2,\cdots,i_t\}$ 为交互 I 对应的因素集，一维交互 $\{(i_1,a_{i_1})\}$ 也称为顶点，简记为 (i_1,a_{i_1}).

定义 3.　设一条测试用例 $\boldsymbol{T}$ 的第 i_j 个因素取值是 $a_{i_j}(j=1,2,\cdots,t)$，即 $\boldsymbol{T}(i_j)=a_{i_j}$，则称这条测试用例覆盖了 t 维交互 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$.

这样，一条测试用例覆盖了 $\binom{k}{t}$ 个 t 维交互. 为了产生满足一定维数的交互覆盖要求的测试用例集，定义覆盖表如下.

定义 4.　如果 A 是一个 $n\times k$ 表，表中第 i 列元素都取自 $[0,v_i-1]$，且满足：每个可能的 t 维交互都被表中某一行所对应的测试用例所覆盖，即对任意的 t 维交互 $\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$，至少存在一行 r，使得 $A[r,i_j]=a_{i_j}$，$j=1,2,\cdots,t$，则称 A 是一个 t 维混合覆盖表，记为 $MCA(n;t,(v_1,v_2,\cdots,v_k))$. t 称为覆盖表的强度. 给定 t 和 $v_1,v_2,\cdots,v_k$，称使得 $MCA(n;t,(v_1,v_2,\cdots,v_k))$ 存在的最小的整数 n 为混合覆盖表数，记为 $MCAN(t,(v_1,v_2,\cdots,v_k))$，当定义中的 $v_1=v_2=\cdots=v_k=v$ 时，我们简记为 $CA(n;t,k,v)$ 和 $CAN(t,k,v)$.

在不引起混淆的情况下覆盖表或混合覆盖表统称为覆盖表.

一般人们用覆盖表或混合覆盖表来产生测试用例，表中每列对应一个因素，k 表示待测系统的因素数，v_i 表示每个因素可能取的取值个数，每一行表示一个测试用例，t 维覆盖表产生的测试用例集能覆盖到 k 个因素中任意 t 个因素所有可能的取值组合，二维覆盖表产生的测试用例集称为两两组合测试用例集，人们希望在不降低测试标准情况下来产生尽可能少的测试用例.

例如，一个打印系统有两种类型打印机 P_1，P_2，分别用 0，1 表示；打印的文件格式有 3 种 JPEG，PDF，PS，分别用 0，1，2 表示；颜色有黑白与彩色两种，用 0，1 表示；文件大小划分为小于等于 50 MB，大于 50 MB 且小于 500 MB 和大于等于 500 MB，也分别用 0，1，2 表示[22].

如果对因素的所有可能取值组合进行测试，则需要 $2\times3\times2\times3=36$ 个测试用例，若只考虑任意两个因素之间的交互作用，则只需要 9 条测试用例即可，如表 1 所示.

表 1　打印系统的两两组合测试用例集

测试用例	打印机	文件格式	颜色	文件大小	输出结果
1	0	0	0	0	pass
2	0	0	1	1	fail
3	1	0	1	2	fail
4	1	1	1	0	pass
5	1	1	0	1	pass
6	0	1	0	2	fail
7	0	2	0	0	fail
8	0	2	1	1	pass
9	1	2	1	2	pass

假设每条测试用例运行时只有两个结果：通过或失败，假设一个运行失败的测试用例至少包含一个交互错误，否则就运行通过，并且假设若一个交互是错误的，则所有包含此交互的测试用例运行都是失败的.

运行表 1 中的测试用例，得到输出结果列在表末，对于那些失败的测试用例，到底是哪个取值组合出了问题？为此，必须对覆盖表做新的研究，最好在测试完之后就能知道哪些是错误交互，这在没有额外的假设下很难做到，为此 Colbourn 和 McClary[20] 提出了错误定位表及错误侦测表的概念，后来 Martínez 等人[21-22] 在此基础上提出了错误定位表的概念(ELAs). 他们提出的错误定位模型是把待测系统(SUT)中的错误交互看成超图的边，从而构成一个错误交互超图来进行定位. 为了简化起见，这里采用交互集合的语言来介绍错误定位表模型.

假设某个 s 维交互导致错误，则所有包含这个 s 维交互的其它交互也会导致错误，为此只记录极小的错误交互，即其任何的真子集都不再是错误交互. 设某个待测系统(SUT)有 d 个极小错误交互，用 Π 表示所有的极小错误交互构成的集合，则 Π 中错误交互都不可能互相包含. 如果 Π 中每个错误交互都有 t 个元素，则记为 Π_t，如果每个错误交互至多有 t 个元素，记为 $\Pi_{\bar{t}}$. 以后若不特别说明，错误交互集都是指极小的错误交互集. 如果一条测试用例没有覆盖 Π 中任何错误交互，则称这条测试用例避开了 Π.

定义 5.　给定 t 维交互 I 与错误交互集 Π_t，如果存在覆盖这个交互 I 的一条测试用例 T 避开了 $\Pi_t\backslash\{I\}$，则称这个 t 维交互 I 关于 Π_t 是可定位的，并称此测试用例 T 定位了交互 I. 如果每个 t 维交互关于 Π_t 都是可定位的，则称 Π_t 是可定位的.

类似地，可以给出如下定义.

定义 6.　给定 $s(s\leq t)$ 维交互 I 与错误交互集 $\Pi_{\bar{t}}$，设 I 包含的 $\Pi_{\bar{t}}$ 中的极小错误交互集合为 Γ_I，如果存在覆盖这个交互 I 的一条测试用例 T 避开了 $\Pi_{\bar{t}}\backslash\Gamma_I$，则称这个 s 维交互 I 关于 $\Pi_{\bar{t}}$ 是可定位的，并

称此测试用例 T 定位了交互 I. 如果每个 $s(s\leq t)$ 维交互关于 Π_t 都是可定位的,则称 Π_t 是可定位的.

由定义 5 与定义 6 知,组合测试方法只能定位可定位的错误交互集 Π,如一个待测系统(SUT)中有 3 个因素,每个因素取值都是二元的,如图 1 所示,类型 1 中交互集{{(2,0),(3,1)}{(2,1},(3,1)}}就是不可定位的,因为交互{(1,0),(3,1)}不可定位;类型 2 中交互集{{(2,0),(3,1)}{(2,1},(3,1)}}也是不可定位的,因为交互{(1,0),(3,1)}不可定位.

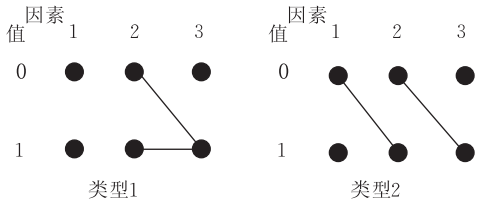

图 1 两种不同类型的错误交互

对于错误交互集 Π 是否可定位,判断起来比较复杂,如果对于任意一个因素 i,都至少存在一个顶点(i,s_i),错误交互集 Π 没有覆盖这个顶点,则错误交互集 Π 是可定位的.

定义 7[22]. 如果一个待测系统(SUT)中的错误交互集为 Π,且对 $\forall i\in[1,k]$,都存在顶点(i,s_i),使得(i,s_i)没有被包含在任何交互 $I\in\Pi$ 中,则这个待测系统(SUT)具有安全值,并称$(s_1,s_2,\cdots,s_k)$为这个待测系统的安全值向量.

所以如果一个待测系统(SUT)具有安全值向量,则其错误交互集 Π_t 是 t 维可定位的[22].

对于可定位的错误交互集 Π,定义错误定位表如下.

定义 8[22]. 设待测系统(SUT)中,其错误交互集 Π_t 是可定位的,一个 $n\times k$ 表 A,其第 i 列元素取自$[0,v_i-1](i=1,2,\cdots,k)$,且满足:每个 t 维交互 I 都能被 A 的某一行对应的测试用例所定位,则称 A 是强度为 t 的错误定位表,记为 $ELA(n;t,(v_1,v_2,\cdots,v_k))$. 当定义中的 $v_1=v_2=\cdots=v_k=v$ 时,我们简记为 $ELA(n;t,k,v)$.

定义 9[22]. 设待测系统(SUT)中,其错误交互集 Π_t 是可定位的,一个 $n\times k$ 表 A,其第 i 列元素取自$[0,v_i-1](i=1,2,\cdots,k)$,且满足:每个 $s(s\leq t)$ 维交互 I 都能被 A 的某一行所对应的测试用例所定位,则称 A 是强度至多为 t 的错误定位表,记为 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$. 当定义中的$v_1=v_2=\cdots=v_k=v$ 时,我们简记为 $ELA(n;\bar{t},k,v)$.

显然,每个错误定位表 $ELA(n;t,(v_1,v_2,\cdots,v_k))(ELA(n;\bar{t},(v_1,v_2,\cdots,v_k)))$都是 $MCA(n;t,k,(v_1,v_2,\cdots,v_k))$.

在文献[22]中,Martínez 等人表明,当 Π 中错误交互数是 k 的高阶无穷小时,绝大多数待测系统(SUT)都有安全值. 利用错误定位表 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$,我们很容易确定错误交互集 Π_t. 运行错误定位表所形成的测试用例集,对所有通过的测试用例其包含的所有的交互都是正确的,只需从所有的 $s(s\leq t)$ 维交互形成的集合中划去包含在某个通过的测试用例中的那些交互,剩下的就是所寻找的错误交互集,再对这个集合中划去所有的非极小错误交互,剩下就是所求的 Π_t.

与错误定位表概念类似的一个概念是禁忌覆盖表.

定义 10[23]. 设待测系统(SUT)中,交互集 Π 是可定位的,一个 $n\times k$ 表 A,其第 i 列元素取自$[0,v_i-1](i=1,2,\cdots k)$,且满足:

(1) A 的每一行所对应的测试用例避开了 Π;

(2)每个不包含 Π 中交互的 t 维交互都能被 A 的某一行所覆盖,

则称 A 是强度为 t 的禁忌覆盖表,记为 $CAFE(n;t,(v_1,v_2,\cdots,v_k),\Pi)$,$\Pi$ 中的边称为禁忌边. 当定义中的 $v_1=v_2=\cdots=v_k=v$ 时,我们简记为 $CAFE(n;t,k,v)$.

前面打印机系统的例子中,假设系统中错误交互如图 2 所示.

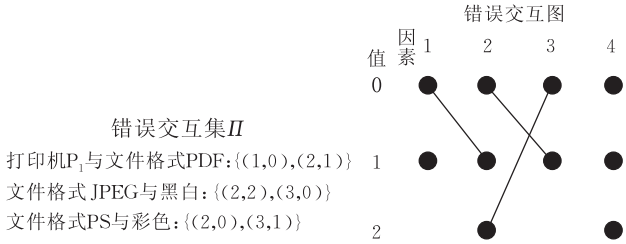

图 2 打印机系统的错误交互及错误交互图

我们可以根据这个错误交互集 Π 构造 $ELA(12;2,(2,3,2,3))$,如表 2 所示.

表 2 错误定位表 $ELA(12;2,(2,3,2,3))$

测试用例	打印机	文件格式	颜色	文件大小	输出结果
1	0	1	1	0	fail
2	1	0	1	0	fail
3	1	2	0	0	fail
4	0	0	0	0	pass
5	0	0	0	1	pass
6	1	0	0	2	pass
7	1	1	0	0	pass
8	1	1	1	1	pass
9	1	1	0	2	pass
10	1	2	1	0	pass
11	0	2	1	1	pass
12	0	2	1	2	pass

在 $ELA(12;2,(2,3,2,3))$ 表中可以看到每个错误交互都出现在不同行，每个非错误交互都出现在一个运行通过的测试用例中. 在表 2 中覆盖表不是一个错误定位表，因为交互{(1,1),(2,0)}不能被表中任何行所定位.

对于可定位的错误交互集，错误定位表规模有多大？Martínez 等人用更高强度的覆盖表来构造错误定位表.

假设待测系统(SUT)具有安全值向量，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d，则当 $t+d\leq k$ 时，每个 $MCA(n;t+d,(v_1,v_2,\cdots,v_k))$ 都是 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$；对于不具有安全值向量的待测系统，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且是可定位的，则当 $t(d+1)\leq k$ 时，每个 $MCA(n;t(d+1),(v_1,v_2,\cdots,v_k))$ 都是 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$[22].

这样可以利用高维的覆盖表来构造错误定位表，根据覆盖表的规模就可知错误定位表的规模的上界. 因为在固定强度 t 时，覆盖表的行数是关于因素数的对数增长的，所以错误定位表的行数也是关于因素数的对数增长的，即当待测系统具有安全值向量时，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d，则存在 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$ 满足 $n=O(d(v)^d\log k)$；对于不具有安全值向量的待测系统，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且可定位，则存在 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$ 满足 $n=O(d(v^t)^d\log k)$[22].

用高维覆盖表来构造错误定位表，其行数虽然关于因素数是对数增长的，但是关于错误数 d 却是指数增长的，这样当 d 比较大时，用这种方法构造的错误定位表来定位交互错误，测试用例存在了很大的冗余，尤其对于不具有安全值向量的待测系统，用 $MCA(n;t(d+1),(v_1,v_2,\cdots,v_k))$ 来构造 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$，即使 d 很小时，行数的规模也太大. 所以我们把这个结论改进为如下的定理.

定理 1.　设待测系统(SUT)中，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且是可定位的，设 $v=\min\{v_1,v_2,\cdots,v_k\}$，$\tilde{d}=\max\left\{\frac{t}{v}d,d\right\}$，则当 $t+\tilde{d}\leq k$ 时，每个 $MCA(n;t+\tilde{d},(v_1,v_2,\cdots,v_k))$ 都是 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$.

详细证明见附录.

有了这个改进的定理，我们显然有以下推论.

推论 1.　设待测系统(SUT)中，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且是可定位的，且 $t\leq v$，则当 $t+d\leq k$ 时，每个 $MCA(n;t+d,k,v)$ 都是 $ELA(n;\bar{t},k,v)$.

推论 2.　设待测系统(SUT)中，其错误交互集 $\Pi_{\bar{2}}$ 中错误交互数最多为 d 且是可定位的，且 $v_i\geq 2$ $(i=1,2,\cdots k)$，则当 $2+d\leq k$ 时，每个 $MCA(n;2+d,(v_1,v_2,\cdots,v_k))$ 都是 $ELA(n;\bar{2},(v_1,v_2,\cdots,v_k))$.

用定理 1 我们得到改进的定理 2.

定理 2.　设待测系统(SUT)中，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且可定位，设 $v=\min\{v_1,v_2,\cdots,v_k\}$，$\tilde{d}=\max\{td/v,d\}$，则存在 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$ 满足 $n=O(d(v)^{\tilde{d}}\log k)$.

从推论 2，定理 2 中我们知道，对于一般的待测系统，若已知错误交互最多是二维的，且错误交互数 d 很小时，用 $MCA(n;2+d,(v_1,v_2,\cdots,v_k))$ 可定位出交互错误，且其行数规模不是太大.

用错误定位表来定位交互错误，其优点是并行性，即所有的测试用例可同时执行，而其缺点是冗余的测试用例比较多，当每条测试用例执行时间很短且花费不高时可用此策略，否则我们可考虑自适应的策略.

3　具有已知安全值的待测系统的错误交互定位自适应算法

在这一节，我们考虑当一个 SUT 具有安全值时定位交互错误的自适应算法. 所谓自适应算法，就是根据前面选出的测试用例的执行结果来选取后面的测试用例. 根据 Martínez 等人的结论，可用更高维的覆盖表来构造具有安全值的错误定位表，但是需要预先知道错误交互的个数 d，并且表的行数 $n=O(d(v)^d\log k)$，所以如果错误交互数太大时，覆盖表很难构造出来，因为覆盖表的行数是关于 d 指数增长的. Martínez 等人对具有安全值 SUT 且错误交互集为 $\Pi_{\bar{2}}$ 的覆盖表提出了自适应算法，它不需要预先知道错误数，但要预先知道每个因素的安全值，并且所需的测试用例数是关于 d 多项式增长的，最坏情况下需要 $O(d(\log k)^2+d^2\log k)$ 条测试用例[22].

Martínez 等人提出的自适应算法简述如下：

假设一个带测系统(SUT)其安全值向量为 $\boldsymbol{s}=(s_1,s_2,\cdots,s_k)$，错误交互集为 $\Pi_{\bar{2}}$. 他们先用贪心密度算法建立一个 $MCA(n;2,(v_1,v_2,\cdots,v_k))A$[7]，然后对表中的每一行所对应的测试用例 T 执行测试，用过程 $TEST(T)$ 表示，其运行结果为通过或失败，

若通过,则这个测试用例所包含的所有1维2维交互都是正确的,对于失败的测试用例,调用一个过程 $LocateErrorInTest((s_1,s_2,\cdots,s_k),T,A=[1,k])$ 来确定被测试用例 T 所覆盖的错误交互.

过程 $LocateErrorInTest(s,T,A)$ 是递归调用的,每次调用时都假设 T 是失败的测试用例,过程输出的是测试用例集 T 所覆盖的因素在集合 A 中的错误交互集合,初始化 $A=[1,k]$,然后把 A 划分成近似相等的两个集合 A',A'',设 $T_i'=\begin{cases}T_i, & i\in A'\\ s_i, & i\notin A'\end{cases}$, $T_i''=\begin{cases}T_i, & i\in A''\\ s_i, & i\notin A''\end{cases}$,则递归调用过程 $LocateErrorInTest(s,T',A')$ 和 $LocateErrorInTest(s,T'',A'')$ 分别得到 E' 和 E'',再调用过程 $AcrossLocate(s,T,A',A'',E',E'')$,这个过程输出一个因素在 A' 中,另一个因素在 A'' 中的错误交互构成的集合,得到 E''',则 $LocateErrorInTest(s,T,A)$ 过程返回 $\Pi_{\bar{2}}=E'\cup E''\cup E'''$. 具体算法参见文献[22]中的算法2.

对于一般的错误交互集 $\Pi_{\bar{t}}$,如果 $t>2$ 时,则上述的算法很难推广,因为当把集合 A 划分成几个近似相等的集合时,因素在各个集合之间的错误交互很难处理,所以对于一般的安全值已知的待测系统,Martínez 等人没有解决,而是作为开问题提出来,这一节我们主要解决这个问题.

设一个待测系统(SUT)具有安全值向量,其安全值向量仍设为 $\boldsymbol{s}=(s_1,s_2,\cdots,s_k)$,其错误交互集 $\Pi_{\bar{t}}$ 是可定位的,我们先用贪心密度算法建立一个 $MCA(n;t,(v_1,v_2,\cdots,v_k))A$[8],然后对表中的每一行所对应的测试用例 T 执行测试,用过程 $TEST(T)$ 表示,其运行结果通过或失败,对失败的测试用例,我们调用过程 $LocateErrorInTest(s,T,t)$ 来得到测试用例 T 中所覆盖的错误交互集. 我们的算法工作如下:先初始化集合 $A=[1,k]\backslash\{i\mid T_i=s_i\}$,然后把集合 A 划分成近似相等的 $t+1$ 个集合 $A_1,A_2,\cdots,A_{t+1}$,使得 $A=\bigcup_{i=1}^{t+1}A_i$,且 $A_i\cap A_j=\varnothing(i,j=1,2,\cdots,t+1)$,则在 $A\backslash A_i,i=1,2,\cdots,t+1$ 这 $t+1$ 个集合中,至少有一个集合,包含有错误交互的因素集,把它赋值给 A,进入下一个循环,直到 $|A|\leq t$,我们就调用辅助过程 $LocateAux(s,T,A)$,这个辅助过程确定因素在 A 中的强度小于等于 t 的所有错误交互. 然后我们把 A 对应因素的取值重新取成安全值来更新测试用例 T,再重复以上过程,这样得到一个错误交互集 Π,且 T 中再不会覆盖与 Π 中所有错误交互都不相交的错误交互,然后再执行一个 $LocateAcrossError(s,T,\Pi)$ 过程,这个过程主要确定被测试用例 T 覆盖且与前面得到的错误交互相交的其它错误交互. 设集合 B 是 Π 中所有错误交互所对应的因素集的并,即设 $\Pi=\{I_1,I_2,\cdots,I_l\}$,第 i 个交互 $I_i=\{(i_1,T_{i_1}),(i_2,T_{i_2}),\cdots,(i_s,T_{i_s})\}$ 对应的因素集合为 $f_{I_i}=\{i_1,i_2,\cdots,i_s\}$,则集合 $B=\bigcup_{i=1}^{l}f_{I_i}$,设 $B=\{l_1,l_2,\cdots,l_y\}$. 为了找出所有的不属于 Π 且与 Π 中错误交互有交集的错误交互,我们构造一个行数尽可能少的带有禁忌边的部分禁忌覆盖表 $PCAFE(m;t,y,2,\Gamma)$ P,这是一个二元的部分禁忌覆盖表,其中 $\Gamma=\{\{(y_1,1),(y_2,1),\cdots,(y_s,1)\}\mid\{(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),\cdots,(l_{y_s},T_{l_{y_s}})\}\in\Pi,s\leq t\}$ 表示禁忌边集合,则此覆盖表的每一行都避开了 Γ,且不属于 Γ 的任何交互 $\{(y_1,1),(y_2,1),\cdots,(y_t,1)\}$ 都被 P 的某一行所覆盖. 这个表是存在的,因为对任何一个不属于 Γ 的 t 维交互 $\{(y_1,1),(y_2,1),\cdots,(y_t,1)\}$,其对应的被 T 覆盖的 t 维交互 $\{(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),\cdots,(l_{y_t},T_{l_{y_t}})\}$ 不包含 Π 中的任何交互,一定存在一个每个 y_i 位取值为1,其余取值为0的 y 维二元向量覆盖它. 然后根据这个覆盖表的每一行来构造一个测试用例 T',如果此行第 y_i 列取值为1,此列所对应的 B 中的因素是 l_{y_i},则测试用例 T' 中的第 l_{y_i} 个因素取值为 $T_{l_{y_i}}$,否则 T' 中的第 l_{y_i} 个因素取值都为 $s_{l_{y_i}}$. 由于每一行都避开了 Γ,即对任意的 $\{(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),\cdots,(l_{y_s},T_{l_{y_s}})\}\in\Pi$,测试用例 T' 没有覆盖这个前面已寻找出来的错误交互,另外对任意不属于 Γ 的 $\{(y_1,1),(y_2,1),\cdots,(y_t,1)\}$,都存在某一行覆盖它,即对任意不包含 Π 中的任何交互的交互 $\{(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),\cdots,(l_{y_t},T_{l_{y_t}})\}$,都存在某个如上构造的测试用例 T' 覆盖它,也就是说,任意的与 Π 中某个错误交互有交集的都能被某一行所覆盖. 然后再递归调用过程,则可得到与 Π 中某个错误交互有交集的错误交互.

具体算法如下.

算法1. 定位交互错误的自适应算法.

输入:因素数目 k 及因素取值向量 $(v_1,v_2,\cdots,v_k)$,交互强度 t 以及安全值向量 $\boldsymbol{s}=(s_1,s_2,\cdots,s_k)$

输出:至多 t 维的错误交互集 $\Pi_{\bar{t}}$

begin 用贪心密度算法生成混合覆盖表 $MCA(n;t,(v_1,v_2,\cdots,v_k))$ A

初始化 $\Pi_{\bar{t}}=\varnothing$,

for $i=1$ to n

```
        对覆盖表的第 i 行所对应的测试用例 T 运行测
        试过程 TEST(T)
        if TEST(T)=fail
          Π_i =Π_i ∪LocateErrorInTest(s,T,t)
        endif
      endfor
    在 Π_i 中删除所有的非极小的错误交互,仍设为 Π_i ;
    return Π_i
end
procedure LocateErrorInTest(s,T,t)
begin
    Π=BasicLocateErrorInTest(s,T,t)
    Π=LocateAcrossError(s,T,Π)
end
procedure BasicLocateErrorInTest(s,T,t)
begin
    初始化错误交互集 Π=∅
    T′=T
    while TEST(T′)=fail
      初始化集合 A=[1,k],
      for i=1 to k
        if T′_i=s_i
          A=A\{i}
        endif
      endfor
      T″=T′
      while (|A|>t)
        划分集合 A 为 t+1 个近似相等的集合 A_1,
        A_2,…,A_{t+1},
        for j=1 to t+1
          for i=1 to k
            if  i∈A_j
                T‴_i=s_i
            else
                T‴_i=T″_i
            endif          //定义T‴
          endfor
          if TEST(T‴)=fail
            A=A\A_j
            T″=T‴
            break
          endif
        endfor
      endwhile
      Π=Π∪LocateAux(s,T″,A)
                  //调用辅助过程 LocateAux(s,T″,A)
      B={i| ∃ I∈LocateAux(s,T″,A),使得
          (i,T′_i)∈I}
      for i=1 to k
        if i∈B
          T′_i=s_i    // 更新 T′
        endif
      endfor
    endwhile
    return Π
end
procedure LocateAux(s,T,A)
  //测试因素是 A 的子集的所有交互是否是错误交互
begin
      初始化 Π=∅
      设 A={i_1,i_2,…,i_t}
      for j=1 to t
        T′=T
        T′_{i_j}=s_{i_j}
        if TEST(T′)=fail
          Π=Π∪LocateAux(s,T′,A\{i_j})
        endif
      endfor
      if Π=∅
        Π={{(i_1,T_{i_1}),(i_2,T_{i_2}),…,(i_t,T_{i_t})}}
      endif
      return Π
end
procedure LocateAcrossError(s,T,Π)
    //定位被 T 覆盖且与Π 的交互有交集的错误交互
begin
  设 Π={I_1,I_2,…,I_l}
  设第 i 个交互 I_i={(i_1,T_{i_1}),(i_2,T_{i_2}),…,(i_s,T_{i_s})}
  集合 f_i={i_1,i_2,…,i_s}
                  //记第 i 个交互所对应的因素集合
  设集合 B=⋃_{i=1}^{l} f_i={l_1,l_2,…,l_y}
  Γ={{(y_1,1),(y_2,1),…,(y_s,1)}|
     {(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),…,(l_{y_s},T_{l_{y_s}})}∈Π,s≤t},
  用贪心算法构造部分覆盖表 PCAFE(m;t,y,2,Γ) P
     //对每个{(l_{y_1},T_{l_{y_1}}),(l_{y_2},T_{l_{y_2}}),…,(l_{y_s},T_{l_{y_s}})}∈Π,
       每一行都不覆盖{(y_1,1),(y_2,1),…,(y_s,1)}
     //且任取覆盖表的任意 t 列,若其对应的 t 维向
       量(1,1,…,1)不在 Γ 中,则存在某一行覆盖了
       这个 t 维向量(1,1,…,1)
    for j=1 to m
       T′=T
      for u=1 to y
        if P[j,u]=1
          T′_{l_u}=T_{l_u}
        else T′_{l_u}=s_{l_u}   //改变测试用例 T 在第 l_u 个
                                因素上取值为安全值
        endif
      endfor
        if TEST(T′)=fail
```

```
        Π=Π∪BasicLocateErrorInTest(s,T′,t)
        Π=LocateAcrossError(s,T,Π)
        break
        endif
    endfor
  return Π
end
```

对算法 1 作如下几点说明：

(1) 在算法 1 中，对于构造部分带禁忌边的覆盖表，可以用带约束的集成 SAT 的 AETG 工具来构造[24]，一般在每一行中错误交互数较少，所以也可用贪心算法等其它方法构造.

(2) 对于每次侦测到一个故障就调试使得错误消失的测试模型，则只需扫描覆盖表 A 的每一行，对出现 fail 的行调用基本的过程 $BasicLocateErrorInTest(s,T,t)$找出一个触发故障的错误交互就可以了. 并且这种调试模型可用 Δ-调试方法来调试[17].

(3) 若运行失败的测试用例 T 中有维数大于 t 的错误交互，算法 1 可能不能找出所有的维数小于等于 t 的错误交互，但对于维数大于 t 的错误交互，可在某次调用过程 $BasicLocateErrorInTest(s,T,t)$时侦测出来，在运行到 while 循环时，若在某次循环时，$TEST(T)=$fail，然后把集合 $A=[1,k]\backslash\{i\mid T_i=s_i\}$ 划分成 $t+1$ 个近似相等的集合 $A_1,A_2,\cdots,A_{t+1}$，若在 $A\backslash A_i,i=1,2,\cdots,t+1$ 这 $t+1$ 个集合中，没有任何一个集合包含有错误交互的因素集，则 T 中含有维数大于 t 的错误交互. 若要准确地定位出这个交互，则需要提高覆盖的维数 t，再来运行算法 1.

我们分析算法 1 所需调用测试 $TEST(\ )$的次数来度量算法 1 的性能，得到以下定理.

定理 3. 设待测系统中错误交互集 Π_t 的错误交互数 $|\Pi_t|=d$，则算法 1 调用 $TEST(\ )$的次数为 $O(d^{t+1}\log k+d(\log k)^2)$.

详细证明见附录.

注. 由定理 3，我们知道算法 1 调用 $TEST(\ )$的次数是关于错误交互数 d 多项式增长的，关于 t 是指数增长的，如果 t 比较大，需要调用附加的测试用例数有可能比较多，但 Kuhn 等人[3]的研究表明一般的待测系统的错误交互的维数小于等于 6，所以当 d 不大时，或运行失败的测试用例中包含的错误交互数不大时，算法 1 调用 $TEST(\ )$的次数并不多.

4 结论和未来的工作

在这篇文章中我们推广了 Martínez 等人[22]用高维覆盖表来构造错误定位表的方法，改进了 Martínez 等人[22]提出的在组合测试中怎样定位至多 t 维的交互错误的自适应算法，解决了 Martínez 等人[22]留下的开问题之一，即安全值已知的待测系统中，构造了自适应算法来定位错误交互集 Π_t 的错误交互，对于另一个开问题，即一般待测系统的错误定位仍然没有解决. 所以我们未来的工作一方面研究对一般待测系统的错误定位构造自适应算法，对具有安全值但安全值未知时构造算法来计算安全值向量；另一方面，对安全值已知的待测系统，我们研究生成测试用例的非自适应算法，即构造错误定位表来定位错误. 虽然自适应算法调用 $TEST(\ )$次数一般比非自适应算法调用的次数少很多，但是错误定位的自适应算法执行时并行程度不够. 研究针对安全值已知的待测系统构造的错误定位表与组群测试(group testing)里的池设计(pooling design)之间的关系[25-26]. 研究直接构造错误定位表 ELA 的非自适应算法或者对具有安全值但安全值未知的待测系统构造多阶段的非自适应算法.

参 考 文 献

[1] 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, 2002

[2] Mandl R. Orthogonal latin squares: An application of experimental design to compiler testing. Communications of the ACM, 1985, 28(10): 1054-1058

[3] Brownlie R, Prowse J, Phadke M. Robust testing of AT&T PMX/StarMail using OATS. AT&T Technical Journal, 1992, 71(3): 41-47

[4] Cohen D M, Dalal S R, Fredman M L et al. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 1997, 23(7): 437-444

[5] Tung Y W, Aldiwan W S. Automating test case generation for the new generation mission software system//Proceedings of the IEEE Arospace Conference, Big Sky, MT, USA, 2000: 431-437

[6] 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 (SE 2004). Innsbruck, Austria, 2004: 345-352

[7] Bryce R C, Colbourn C J. The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability, 2007, 17(3): 159-182

[8] Bryce R C, Colbourn C J. A density-based greedy algorithm for higher strength covering arrays. Software Testing, Verification and Reliability, 2009, 19(1): 37-53

[9] Tai K C, Lei Y. A test generation strategy for pairwise testing. IEEE Transactions on Software Engineering, 2002, 28(1): 109-111

[10] Lei Y, Kacker R, Kuhn D R, Okun V, Lawrence J. IPOG: A general strategy for t-way software testing//Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS2007). 2007: 549-556

[11] Lei Y, Kacker R, Kuhn D R, Okun V, Lawrence J. IPOG IPOG-D: Efficient test generation for multi-way combinatorial testing. Software Testing, Verification and Reliability, 2008, 18(3): 125-148

[12] Kobayashi N, Tsuchiya T, Kikuno T. A new method for constructing pair-wise covering designs for software testing. Information Processing Letters, 2002, 81(2): 85-91

[13] Cohen M B, Colbouns C J, Gibbons P B, Mugridge W B. Constructing test suites for interaction testing//Proceedings of the International Conference on Software Engineering (ICSE2003). Portland OR, 2003: 38-48

[14] Cohen M B, Colbouns C J, Collofello J S et al. Variable strength interaction testing of components//Proceedings of the International Computer Software and Applications Conference (COMPSAC2003). Dallas TX, 2003: 413-418

[15] Cohen M B, Colbouns C J, Ling A C H. Augmenting simulated annealing to build interaction test suites//Proceedings of 14th International Symposium on Software Reliability Engineering (ISSRE 2003). Denver Colorado, 2003: 394-405

[16] Shiba Toshiaki, Tsuchiya Tatsuhiro, Kikuno Tohru. Using artificial life techniques to generate test cases for combinatorial testing//Proceedings of the COMPSAC04. Hong Kong, China, 2004: 72-78

[17] Zeller A, Hildebrandt R. Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering, 2002, 28(2): 183-200

[18] Yilmaz C, Cohen M B, Porter M B. Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Transactions on Software Engineering, 2006, 32(1): 20-34

[19] Xu Bao-Wen, Nie Chang-Hai, Shi Liang, Chen Huo-Wang. A software failure debugging method based on combinatorial design approach for testing. Chinese Journal of Computers, 2006, 29(1): 132-138(in Chinese)
(徐宝文，聂长海，史亮，陈火旺. 一种基于组合测试的软件故障调试方法. 计算机学报，2006, 29(1): 132-138)

[20] Colbourn C J, McClary D W. Locating and detecting arrays for interaction faults. Journal of Combinatorial Optimization, 2008, 15(1): 17-48

[21] Martínez C, Moura L, Panario D, Stevens B. Algorithms to locate errors using covering arrays//Proceedings of the LATIN 2008 8th Latin American Theoretical Informatics. Lecture Notes in Computer Science 4957. Buzios, Brazil, 2008: 504-519

[22] Martínez C, Moura L, Panario D, Stevens B. Locating errors using ELAs, covering arrays and adaptive testing algorithms. SIAM Journal on Discrete Mathematics, 2009, 23(4): 1776-1799

[23] Danziger P, Mendelsohn E, Moura L, Stevens D: Covering arrays avoiding forbidden edges. Theoretical Computer Science, 2009, 410: 5403-5414

[24] Cohen M B, Dwyer M B, Shi Jiangfan. Interaction testing of highly-configurable systems in the presence of constraints// Proceedings of the International Symposium on Software Testing and Analysis. London, 2007: 129-139

[25] Du D Z, Hwang F K. Pooling Designs and Non-Adaptive Group Testing: Important Tools for DNA Sequencing. New York: World Scientific Publishing Company, 2006

[26] Cheng Y X, Du D Z. New constructions of one- and two-stage pooling designs. Journal of Computational Biology, 2008, 15(2): 195-205

附　录.

(1) 定理 1 的证明.

定理 1. 设待测系统(SUT)中，其错误交互集 $\Pi_{\bar{t}}$ 中错误交互数最多为 d 且是可定位的，设 $v=\min\{v_1,v_2,\cdots,v_k\}$，$\tilde{d}=\max\left\{\frac{t}{v}d,d\right\}$，则当 $t+\tilde{d}\leq k$ 时，每个 $MCA(n;t+\tilde{d},(v_1,v_2,\cdots,v_k))$ 都是 $ELA(n;\bar{t},(v_1,v_2,\cdots,v_k))$.

证明. 设 A 是一个 $MCA(n;t+\tilde{d},(v_1,v_2,\cdots,v_k))$，$I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_s,a_{i_s})\}$ 为任意 $s(s\leq t)$ 维交互，设 I 包含的 $\Pi_{\bar{t}}$ 中的极小错误交互集合为 Γ_I，即 $\Gamma_I=\{I'\mid I'\subseteq I, I'\in\Pi_{\bar{t}}\}$，并设 I 对应的因素集为 f_I，我们证明在 $MCA(n;t+\tilde{d},(v_1,v_2,\cdots,v_k))$ 中存在一行覆盖了 I，但避开了 $\Pi_{\bar{t}}\backslash\Gamma_I$. 设因素集 $C_0=\{i\mid i\in f_{I'},\forall I'\in\Pi_{\bar{t}}\}\backslash f_I$，记 $\Pi_0=\Pi_{\bar{t}}\backslash\Gamma_I$，则对集合 C_0，Π_0 做如下操作：

第 1 步. 在 C_0 中取一个因素 j_1，使得存在一个顶点 (j_1,a_{j_1}) 没有被 Π_0 中任何交互覆盖，若不存在这样的因素，则过程结束，取出顶点 (j_1,a_{j_1}) 后，在 Π_0 中删去因素集包含 j_1 的那些交互，记为 Π_1，即 $\Pi_1=\Pi_0\backslash\{I'\mid j_1\in f_{I'}\}$，更新 C_0 为 $C_1=C_0\bigcap\{i\mid i\in f_{I'},\forall I'\in\Pi_1\}$.

第 2 步. 同第 1 步一样，在 C_1 中取一个因素 j_2，使得存在一个顶点 (j_2,a_{j_2}) 没有被 Π_1 中任何交互覆盖，若不存在这样的因素，则过程结束，取出顶点 (j_2,a_{j_2}) 后，在 Π_1 中删去因素集包含 j_2 的那些交互，记为 Π_2，即 $\Pi_2=\Pi_1\backslash\{I'\mid j_2\in f_{I'}\}$，更新 C_2 为 $C_2=C_1\bigcap\{i\mid i\in f_{I'},\forall I'\in\Pi_2\}$.

一直做下去，直到第 p 步过程结束，这时，我们取出了 p 个因素 $j_1,j_2,\cdots,j_p$，得到 Π_p,C_p，这时在 C_p 中所有因素 j，其对应的任何顶点 (j,a_j) 都被 Π_p 中某个交互覆盖，而 Π_p 中交互数 $\leq d-p$，设 C_p 中的因素数为 q，记为 $C_p=\{l_1,l_2,\cdots,l_q\}$，则有 $\sum_{j\in C_p}v_j\leq t(d-p)$. 记 $v=\min\{v_1,v_2,\cdots,v_k\}$，则有 $vq\leq t(d-p)$. 又因为 $\Pi_{\bar{t}}$ 可定位，所以存在一个覆盖交互 I 的测试用例 T 避开了 $\Pi_{\bar{t}}\backslash\Gamma_I$，设交互

$$\bar{I}=I\cup\{(l_1,T_{l_1}),(l_2,T_{l_2}),\cdots,(l_q,T_{l_q})\}\cup\{(j_1,a_{j_1}),(j_2,a_{j_2}),\cdots,(j_p,a_{j_p})\},$$

则$\bar{I}$的维数$=s+p+q\leq t+\frac{t(d-p)}{v}+p\leq t+\max\left\{\frac{t}{v}d,d\right\}=t+\tilde{d}$,因为 A 是一个 $MCA(n;t+\tilde{d},(v_1,v_2,\cdots,v_k))$所以 A 中存在一行覆盖了交互$\bar{I}$,则这一行定位了交互 I. 证毕.

(2)定理 3 的证明.

引理 1. 过程*BasicLocateErrorInTest*(s,T,t)中从集合$A=[1,k]$循环到$|A|\leq t$需调用 $TEST(\ )O\left((t+1)\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}\right)$次.

证明. 因为开始 $A=[1,k]$,每循环一次,就把集合 A 分成几乎相等的$t+1$ 个子集,然后保留 t 个,所以每循环一次集合 A 的元素个数就为原来的$\frac{t}{t+1}$,最后$|A|\leq t$,设循环次数为 r,则$\left(\frac{t}{t+1}\right)^r k\geq t$,所以 $r=\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}$,而每次得到一个新的集合$A$后,需调用$TEST(\ )$至多 $t+1$ 次,所以一共调用$(t+1)\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}$次.

引理 2. 过程 $LocateAux(s,T,A)$调用 $TEST(\ )$的次数$\leq t!$.

证明. 设过程 $LocateAux(s,T,A)$调用$TEST(\ )$的次数为 $D(t)$,而此过程至多 t 次递归调用过程 $LocateAux(s,T,A/\{i_j\})$,所以 $D(t)\leq tD(t-1)$,而 $D(1)=1$,所以 $D(t)\leq t!$.

引理 3. 设测试用例 T 中错误交互数为 d_T,则过程 $LocateAcrossError(s,T,\Pi)$ 中生成的部分禁忌覆盖表 $PCAFE(m;t,y,2,\Gamma)$ P 行数为 $O\left(\frac{t^t}{t!}d_T^t\right)$.

证明. 因为禁忌边集的边数$|\Gamma|\leq d_T$,而部分禁忌覆盖表的因素数 $y\leq td_T$,因为每一行至少覆盖一个$\{(y_1,1),(y_2,1),\cdots,(y_t,1)\}$,所以 $PCAFE(m;t,y,2,\Gamma)$ P 的行数为 $m\leq\binom{y}{t}\leq\binom{td_t}{t}\leq\frac{t^t}{t!}d_T^t$.

引理 4. 设测试用例 T 中错误交互数为 d_T,则过程 $LocateErrorInTest(s,T,t)$调用$TEST(\ )$的次数为 $O(d_T^{t+1}+d_T\log k)$.

证明. 所有的错误交互或者从过程 *BasicLocateErrorInTest*(s,T,t)得到,或者从过程 $LocateAcrossError(s,T,\Pi)$得到,由引理 1,2 可知,在过程 *BasicLocateErrorInTest*(s,T,t)中每得到一个错误交互最多需调用 $TEST(\ )(t+1)\cdot\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}+t!$次,由引理 3 可知,在*LocateAcrossError*(s,T,Π)中每得到一个错误交互最多需调用$TEST(\ )$ $\frac{t^t}{t!}d_T^t+(t+1)\cdot\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}+t!$次,所以过程 $LocateErrorInTest(s,T,t)$最多需调用 $TEST(\ )$的次数为 $\left(\frac{t^t}{t!}d_T^t+(t+1)\frac{\log k-\log t}{\log\left(\frac{t+1}{t}\right)}+t!\right)d_T$次,即 $O(d_T^{t+1}+d_T\log k)$次.

定理 3. 设待测系统中错误交互集 Π_T 的错误交互数$|\Pi_T|=d$,则算法 1 调用 $TEST(\ )$的次数为 $O(d^{t+1}\log k+d(\log k)^2)$.

证明. 因为混合覆盖表 $MCA(n;t,(v_1,v_2,\cdots,v_k))$ A 的行数为 $O(\log k)$,所以对每一行确定这一行覆盖的错误交互需调用$TEST(\ )$的次数为$O(d_T^{t+1}+d_T\log k)$,而$d_T\leq d$,所以算法 1 调用$TEST(\ )$的次数为 $O(d^{t+1}\log k+d(\log k)^2)$.

证毕.

ZHOU Wu-Jie, born in 1973, Ph. D. candidate, lecturer. His research interests include software testing and statistical testing etc.

ZHANG De-Ping, born in 1973, Ph. D., lecturer. His research interests include software testing, statistical testing and mathematical statistics.

XU Bao-Wen, born in 1961, Ph. D., professor, Ph. D. supervisor. His research interests include programming language, software engineering, parallel and network, acquisition technique on knowledge and information etc.

Background

This paper is supported by the National Natural Science Foundation of China under Grant Nos. 90818027, 91018005, the National High-Tech Research Subjects and Development Plan of China under Grant Nos. 2009AA01Z147, and the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320703. These projects mainly research on the follows fields: Combinatorial coverage method and its effectiveness for software testing, the existence and the generation algorithm for several minimal combinatorial covering table, some techniques for software fault diagnosis and debugging based on combinatorial testing, the application of the combinatorial testing in Web testing and software configuration testing. The authors have made some works on the test case generation algorithm and the applications. This paper focuses on the research of software fault diagnosis and debugging based on combinatorial testing.