可定位单个错误的错误定位表的生成方法

周吴杰[1] 张德平[2]
(东南大学计算机科学与工程学院 南京 210096)[1]
(南京航空航天大学信息科学与技术学院 南京 210016)[2]

摘 要 研究了组合测试错误定位表的结构。针对t维组合测试情形,在假设待测系统中只有一个强度小于等于t维的错误交互时,提出了一种新的构造这种特殊情形的错误定位表的方法。这种方法构造的错误定位表覆盖了所有的t维交互并且对任意两个t维交互,表中包含这两个t维交互的行的集合互不相同。最后提出了生成这种错误定位表的AETG-like算法。实验表明,用该方法构造出的错误定位表比用$t+1$维覆盖表构造的错误定位表其行数要少得多。

关键词 组合测试,错误定位表,AETG-like算法,覆盖表
中图法分类号 TP311 **文献标识码** A

Generation Approach of Error Locating Arrays to Locate One Interaction Fault

ZHOU Wu-jie[1] ZHANG De-ping[2]
(Department of Computer Science & Engineering, Southeast University, Nanjing 210096, China)[1]
(College of Information Science and Technology, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, China)[2]

Abstract Combinatorial testing is a practical approach to detect the faulty interactions among parameters or components in the system. Error locating arrays(ELAs) were defined by Mart'lnez C et al to detect and locate the faulty interactions in a system. We studied the structure of error locating arrays(ELAs), and proposed the new method to construct the special ELAs of locating one t-way fault interaction among the components. The special ELAs cover every t-way interaction and any two interactions appear in different rows. So the covering strength of the special ELAs is in the range of t and $t+1$. We proposed the AETG-like algorithm to generate the special ELAS. Experiments results show the size of the special ELAs is less than the size of $t+1$-way covering arrays that may be the ELAs.

Keywords Combinatorial testing, Error locating array, AETG-like algorithm, Covering array

1 引言

随着计算机技术的飞速发展,软件系统变得越来越庞大,软件测试成为保证这些系统质量的一个关键手段。然而,影响系统运行的因素有很多,一般采取组合测试策略检测各个因素之间的相互作用对系统产生的影响。组合测试作为一种重要的软件测试方法,因能以较少的测试用例来实现对被测试系统进行科学有效的测试而得到广泛的研究和应用。对于那些故障来源于软件系统中一些参数和参数之间的相互作用的系统,这种方法的效果尤其明显。Kuhn等的研究表明,测试两两组合覆盖能发现大约70%的错误,四维组合覆盖能发现90%的错误[1]。

在组合测试中,测试用例的运行失败预示系统组件中存在触发系统故障的错误交互,这就需要测试人员找出这些错误交互。目前基于组合测试的错误交互定位方法主要包括分类树法[2]、故障调试法[3]和错误定位表[4-6]等。利用分类树法,系统的错误模式(导致系统故障的是某些参数的取值组合)一般很难被精确地确定[7]。故障调试定位法通过增加附加测试用例数,能够对一些特殊的系统错误进行较为精确的定位,但对于一些大型软件系统,由于系统的交互因素数目众多,一个测试用例包含的模式数目呈指数增长,从而导致触发软件故障的可能模式数目呈指数增长,使得错误定位变得不适用[3]。在某些特殊假设下,错误定位表虽然对所有软件交互错误都能精确定位,但错误定位表的行数随系统中错误交互数的增加呈指数增长,并且对于较为大型软件系统其生成也是一项相当复杂且费时的任务,在实际组合测试中进行错误定位显得低效。因此,利用组合测试策略来对系统进行测试,发现某些因素的组合导致系统发生故障时,如何定位和侦测哪些因素组合会导致系统故障,一直是组合测试中一个亟待解决的重要课题。

本文研究了组合测试错误定位表的构造方法[4-6],这些构造方法都是用高维的覆盖表来生成错误定位表,所以生成的定位表的规模比较大。用这种定位表来指导测试时,测试花费比较高。例如对一些测试场景中,组件之间只有一个单因素或二维交互错误时,用三维覆盖表来定位交互错误就使得测试规模太大。针对这种特殊情形,本文提出了新的构造方

到稿日期:2011-03-15 返修日期:2011-05-19
周吴杰(1973-),博士生,讲师,主要研究方向为软件测试技术、软件可靠性等,E-mail:zhouwujie@seu.edu.cn;张德平(1973-),博士,讲师,主要研究方向为软件测试技术、软件可靠性、数理统计等。

法来生成错误定位表。这种表的覆盖强度强于二维覆盖表，但弱于三维覆盖表。最后提出了生成这种特殊的错误定位表的 AETG-like 算法。试验结果表明，其生成的错误定位表的行数比三维覆盖表的行数要少得多。

本文第 2 节介绍了背景及相关研究工作，给出了基本的组合测试模型、错误定位表模型以及相关结论；第 3 节提出了特殊情形下的错误定位表的构造方法，并给出了类似于 AETG 算法的贪心算法；第 4 节通过实例分析得出错误定位表的规模要比 $t+1$ 维覆盖表的规模小得多；最后是总结并讨论了未来可进行研究的方向。

2 背景及相关研究

2.1 相关研究

目前，人们关于组合测试的研究主要集中在测试用例集的生成方面，即在满足给定组合覆盖标准的前提下，生成规模尽可能小的测试用例集，以节约测试成本。对于一些特殊情形，可用传统的代数方法来生成最优解[8,9]。对于一般的情形，由于生成最优的满足要求的测试用例集是 NP 完全的，因此人们用贪心算法或局部搜索方法来生成测试用例集[10-14]，这些方法不能保证得到最优解，但是生成测试用例集的时间相对较少。对于具体的应用场景，至少生成多少条测试用例才能满足组合覆盖需求，这个问题只有在某些特殊的情况下才能确定，对于一般情形，这仍然是个公开问题。

当软件交互错误被检测出后，如何利用运行测试用例所得的信息来进行故障定位，是软件调试和分析阶段艰巨的工作。目前，对组合测试的结果进行调试和分析的研究还很少。Yilmaz 等[2]采用分类树（classification tree）的方法分析触发系统故障的特征（fault characterization，触发被测系统错误的参数取值组合）。徐宝文等[3]首先假定“若某个模式是错误模式，则任意包含该模式的其他模式也将引发相同的错误”。在此前提条件下，他们通过分析执行失败的测试用例，生成一批与之类似的附加测试用例，并通过执行附加测试用例，逐步缩小故障模式集合 M 的范围，直到错误被精确地定位。Colbourn 和 McClary 提出了 (d,t) 错误定位表及错误侦测表的概念，用这些表来定位组合交互错误[4]。随后，Mart′lnez 等[5,6]提出了一般的错误定位表，并在此模型下提出了自适应算法来定位错误交互。

2.2 错误定位表及相关定义

为了更好地介绍基于组合测试的软件故障定位与侦测技术，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 $v_i(1\leqslant i\leqslant k)$ 个可能的取值，用 $0,1,\cdots,v_i-1$ 表示。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他参数的取值或存在性。

定义 1 设 k 维向量 $T=(T_1,T_2,\cdots,T_k)$，其中 $T_i\in[0,v_i-1]$，$i=1,2,\cdots,k$，则称这个 k 维向量 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 任取 t 个因素，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 互不相等，$a_{i_j}\in\{0,1,\cdots,v_{i_j}-1\}(i=1,2,\cdots,t)$，则称这个集合 I 为因素 i_j 取值为 a_{i_j} 的 t 维交互。称 $f=\{i_1,i_2,\cdots,i_t\}$ 为交互 I 对应的因素集，(i_1,a_{i_1}) 称为顶点。

定义 3 设一条测试用例 $T=(T_1,T_2,\cdots,T_k)$ 和 t 维交互 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$，满足 $T(i_j)=a_{i_j}(j=1,2,\cdots,t)$，则称这条测试用例 T 覆盖了这个 t 维交互。

由定义 3 可知，一条测试用例覆盖了 $\binom{k}{t}$ 个 t 维交互。为了产生交互测试用例，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 $\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$，至少存在一行 r，使得 $A[r,i_j]=a_{i_j}$，$j=1,2,\cdots,t$，则称表 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，将其简记为 $CA(n;t,k,v)$ 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。

一般地，人们用覆盖表或混合覆盖表来产生测试用例，表中每一列对应一个因素，k 表示因素数目，v_i 表示每个因素可能的取值个数，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，2 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的情况下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，即 JPEG，PDF，PS，分别用 0,1,2 表示，颜色及文件大小如表 1 所列。

表 1 打印机系统

打印机	文件格式	颜色	文件大小
0=P_1 1=P_2	0=JPEG 1=PDF 2=PS	0 =black & white 1= colour	0=(≤ 50) 1=(> 50 and < 500) 2=(≥ 500)

如果测试因素的所有可能取值组合，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 2 所列。

表 2 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

假设每条测试用例在运行时只有两个可能结果：通过或失败。假定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。进一步，假定若一个交互是错误的，则所有包含此错误交互的测试用例运行都是失败的。运行正确的测试用例所覆盖的所有交互都是正确的交互。运行表 2 中的测试用例，得到的输出结果列在表的最后一列。

对于那些失败的测试用例，如何才能精确定位是哪个取值组合出了问题？自适应的方法是根据测试信息来对各个交互导致的系统故障的可能性排序，然后生成一批附加的测试用例，来进一步对系统进行测试，以得到更精确的定位[2]。由于交互测试本身复杂，系统的交互呈爆炸增长，因此这种方法定位错误交互运行附加的测试用例比较多。非自适应的方法是测试阶段与调试阶段合二为一，最好在测试完之后就能知道哪些交互是错误交互，这在没有额外的假设下很难做到。基于此，Colbourn 和 McClary[4] 提出了(d,t)错误定位表及错误侦测表的概念；随后，Mart'lnez 等[5,6] 提出了一般的错误定位表概念。用错误定位表来指导测试，必须首先知道错误交互集合的结构或错误交互的维数与个数，且错误交互个数较多时，生成的错误定位表规模非常大。非自适应方法的优点是全自动的，测试完后所做的调试阶段的工作要少得多。在运行一条测试用例所需的时间很少时，这个方法简洁有效。本文研究特殊情形下的错误定位表的构造。

假设如果某个 s 维交互导致错误，则所有包含这个 s 维交互的其他交互也会导致错误。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。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，设 I 包含的 Π 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 $\Pi\backslash\Gamma_I$，则称这个 t 维交互 I 关于 Π 是可定位的，并称此测试用例 T 定位了交互 I。如果对每个 t 维交互关于 Π 都是可定位的，则称 Π 是可定位的。

对于可定位的错误交互集 Π，可定义错误定位表。

定义 6[6]（错误定位表 $ELA(n;t,(v_1,v_2,\cdots,v_k))$） 设待测系统(SUT)中，其错误交互集 Π 是可定位的，一个 $n\times k$ 表 A，其第 i 列元素取自$[0,v_i-1](i=1,2,\cdots,k)$，且满足每个 t 维交互 I 都能被 A 的某一行对应的测试用例所覆盖且避开了 $\Pi\backslash\{I\}$，则称 A 是强度为 t 的错误定位表，记为 $ELA(n;t,(v_1,v_2,\cdots,v_k))$。

对于可定位的 Π，错误定位表大小规模有多大？假设待测系统(SUT)的错误交互集 Π 中的错误交互数 $d<\min\{v_1,v_2,\cdots,v_k\}$，则当 $t+d\leqslant k$ 时，每个 $MCA(n;t+d,(v_1,v_2,\cdots,v_k))$都是 $ELA(n;t,(v_1,v_2,\cdots,v_k))$[4]。

对于覆盖表，我们能估计其行数的上界，所以对于错误交互数 $d<\min\{v_1,v_2,\cdots,v_k\}$ 的待测系统，当 $t+d\leqslant k$ 时，存在 $ELA(n;t,(v_1,v_2,\cdots,v_k))$满足 $n=O(d(v)^d\log k)$[6]。即当待测系统(SUT)中错误数比因素取值数少时，错误定位表的行数是因素数的对数阶。但是 Mart'lnez 等人用 $MCA(n;t+d,(v_1,v_2,\cdots,v_k))$来构造错误定位表，其行数还是存在很多冗余，如有些测试场景中系统中的交互错误较少；如只有一个 t 维交互错误，则我们可用可定位一个交互错误的定位表来生成测试用例，其覆盖的强度应介于 t 维与 $t+1$ 维之间。下一节将介绍这种新的构造。

3 可定位一个交互错误的错误定位表的构造

假设在待测系统中，系统的交互错误比较少，即没有交互错误或只有一个交互错误，且这个交互错误维数小于等于 t。这时，可以构造 $MCA(n;t+1,(v_1,v_2,\cdots,v_k))$作为错误定位表 $ELA(n;t,(v_1,v_2,\cdots,v_k))$来侦测和定位这个交互错误。然而，用这个覆盖表来指导测试，冗余比较多，因此寻找新的构造方法。

设系统中错误交互为 I^*，要把这个错误交互定位出来，只需对任意其他的 t 维交互 I，都至少存在一条运行通过的测试用例来覆盖这个交互，即存在一条运行正确的测试用例覆盖了 I，而没有覆盖 I^*。但是，不知道 I^* 在什么位置，所以我们假设：对任意的 t 维交互，设 I 对应的因素集合为 $f=\{i_1,i_2,\cdots,i_t\}$，则对任意的因素 $i(i\neq i_1,i_2,\cdots,i_t)$，都存在两个测试用例 T_1,T_2，使得这两个测试用例都覆盖 I，但在因素 i 上的取值互不相同。这样就能保证这两条测试用例中至少有一条是运行通过的测试用例，从而这个单个的错误交互 I^* 就是可定位的。所以我们有：

定理 1 假设待测系统中的错误交互数只有一个，且其维数$\leqslant t$，设 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：对任意的 t 维交互 I，其对应的因素集合为 $f=\{i_1,i_2,\cdots,i_t\}$，以及任意的因素 $i(i\notin f)$都存在两行 r_1,r_2，其对应的测试用例覆盖交互 I，且这两行在因素 i 处取值互不相同，即 $A[r_1,i]\neq A[r_2,i]$，则表 A 是可定位单个交互错误的错误定位表 $ELA(n;t,(v_1,v_2,\cdots,v_k))$。

假设待测系统中的错误交互数只有一个，且其维数$\leqslant t$时，由定理 1 可得到，如上构造的定位表肯定是一个 t 维覆盖表，且每个 t 交互都至少被覆盖了两次。所以它覆盖的比 t 维覆盖表强，比 $t+1$ 维覆盖表强度弱。因此定理 1 生成的错误定位表显然比用 $MCA(n;t+1,(v_1,v_2,\cdots,v_n))$来构造这种待测系统的错误定位表规模要小。并且我们有：

推论 1 假设待测系统中所有的因素取值都是二元情形时，定理 1 中构造的错误定位表就是 $t+1$ 维覆盖表 $CA(n;t+1,k,2)$。

当因素的取值越大时，定理 1 中构造的错误定位表的规模比 $t+1$ 维混合覆盖表规模就要小很多。

如果得到错误定位表，怎么得到错误交互呢？只需运行错误定位表所形成的测试用例集。所有通过的测试用例，其包含的所有交互都是正确的，只需从所有的 $s(s\leqslant t)$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 I^*。

如何生成定理 1 中的错误定位表，使得覆盖表的行数尽可能地少？AETG 算法是生成覆盖表的经典算法[10]，其生成覆盖表的规模比较优且时间效率高。我们设计的生成定理 1 中错误定位表的算法与经典的 AETG 算法类似，即先生成一批候选的测试用例，然后在这一批候选的测试用例中选取适应值函数取值最大的测试用例作为错误定位表的下一行。适应值函数取法既要考虑到被这条测试用例覆盖的未被覆盖的交互，也要考虑到已被覆盖的交互。

设已有的测试用例集合为 Γ，则有些交互未被 Γ 覆盖。有些交互虽然已被覆盖，但仍未满足定理 1 的条件，即至少存在一个与这个交互对应的因素集互异的因素，使得覆盖了这个交互的所有测试用例在这个因素处的取值是相同的，称这类交互为未完成覆盖的。设 I 是一个未完成覆盖的交互，其对应的因素集合为 $f=\{i_1,i_2,\cdots,i_t\}$，称其未完成因素集合为 $f^*(I)=\{i|i\notin f$ 且在 Γ 中任意覆盖了 I 的测试用例在 i 处的

取值都相同}。现在添加一条测试用例 T，则希望 T 能尽可能多地覆盖未被覆盖的交互，也要使得未完成覆盖的交互尽快地完成覆盖，即使得其对应的未完成因素集 f^* 中元素尽可能地减少。所以定义适应值函数如下。

$f(T)$=被 T 覆盖的未被覆盖的交互数+被 T 覆盖的未完成覆盖的交互其对应的未完成因素集合 f^* 中元素减少的数目和除以$(k-t)$

生成一条候选的测试用例方法如下：

首先定义因子对密度，对任意 $t-1$ 维交互$\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\}$定义其因子对密度为：

pairdensity$(\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\})$=未被覆盖的包含$\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\}$的交互数目+未完成覆盖的包含$\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\}$的交互其对应的未完成因素集 f^* 中的元素数目和除以$(k-t)$

选择前 $t-1$ 个因素 $i_1,i_2,\cdots,i_{t-1}$ 的取值 $a_{i_1},a_{i_2},\cdots,a_{i_{t-1}}$，使得交互$\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\}$的因子对密度 pairdensity$(\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{t-1},a_{i_{t-1}})\})$是最大的，其中 $i_1,i_2,\cdots,i_{t-1}$ 随机排序。然后对余下的因素 $i_t,i_{t+1},\cdots,i_k$ 随机排序，并依次对其取值。假设前面的 $j-1$ 个因素的取值已经确定，分别为 $a_{i_1},a_{i_2},\cdots,a_{i_{j-1}}$，在确定第 i_j 个因素的取值 a_{i_j} 时，对每个取值计算得分。计算方法与计算测试用例的适应值函数一样，定义如下。

设 a_{i_j} 与前面的因素取值 $a_{i_1},a_{i_2},\cdots,a_{i_{j-1}}$ 形成的交互为 $I_j=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_{j-1},a_{i_{j-1}}),(i_j,a_{i_j})\}$，记

Score(a_{i_j})=被 I_j 包含的未被覆盖的交互数+ 被 I_j 包含的未完成覆盖的交互其对应的未完成因素集合 f^* 中的元素减少的数目和除以$(k-t)$

确定第 i_j 个因素的取值为 a_{i_j}，使得 a_{i_j} 的得分 Score(a_{i_j}) 最大。

依此类推，直至所有因素的取值都被确定。具体见算法 1。

算法 1 AETG-like 算法生成错误定位表

输入：$t,(v_1,v_2,\cdots,v_k)$

输出：可定位一个交互错误的错误定位表 $ELA(n;t,(v_1,v_2,\cdots,v_k))$ A

```
begin
    生成所有的未被覆盖的 t 维交互集 Uncover，初始化为所有的 t 维交互集
    生成未完成覆盖的交互所对应的未完成因素集合 f*，初始化需计算的 f* 为空集
      初始化 A 为空表
    while(Uncover 不为空或至少有一个 f* 不为空时)
        对任意 t-1 维交互{(i1,a_i1),(i2,a_i2),…,(i_{t-1},a_{i_{t-1}})}，计算因子对密度 pairdensity({(i1,a_i1),(i2,a_i2),…,(i_{t-1},a_{i_{t-1}})})
        选择因素 i1,i2,…,i_{t-1} 以及因素的取值 a_i1,a_i2,…,a_{i_{t-1}}，使得 pairdensity({(i1,a_i1),(i2,a_i2),…,(i_{t-1},a_{i_{t-1}})})最大
        随机排序 i1,i2,…,i_{t-1}
        for t=1 to M //生成一条新的侯选测试用例
          其它参数随机排列为 i_t,i_3,…,i_k
          for j=t to k //此时前 j-1 个参数的取值已确定，分别为 a_{i1},a_{i2},…,a_{i_{j-1}}
              对第 j 个因素的每个取值计算得分函数 Score(a_{i_j})
              在第 j 个因素的所有取值中取出使得得分 Score(a_{i_j})最大的 a_{i_j} 作为参数 i_j 的取值
          endfor
        endfor
        从 M 条测试用例中选择一条测试用例 T，使得其适应值 f(T)最大，将其作为一行并到 A 的下面，得到新的表 A
        更新未被覆盖的 t 维交互集 Uncover
        重新计算所有的未完成覆盖的交互所对应的未完成因素集合 f*
    endwhile
    return A
end
```

注：算法 1 与经典的 AETG 算法类似，都是每次生成一条测试用例的贪心算法，每次在候选池中选择一条适应值最大的测试用例并入到测试用例集中。但我们的算法与经典的 AETG 算法最大的不同之处在于每次做贪心选择时既要考虑未被覆盖的交互数，也要考虑未完成覆盖的交互所对应的未完成因素数目，并且需对这两个要素取合适的权重。

4 实例分析

下面对不同的待测系统都假设系统中只有一个错误交互。试验主要研究 $t=2$ 的情形。表 3 中我们用算法 1 来生成可定位一个交互错误的错误定位表，然后用 AETG 算法来生成相应二维和三维的覆盖表，并比较它们的大小，记 ELA 表示可定位一个交互错误的错误定位表，CA(2)，CA(3)分别表示相应的二维和三维覆盖表。

表 3 可定位一个错误交互的错误定位表与 AETG 生成的覆盖表行数比较

类型	3^4	3^{13}	$6^15^14^63^82^3$	$3^{12}4^5$	$4^13^{39}2^{35}$	$5^34^43^12^2$	4^{40}	4^{30}
ELA 行数	22	42	110	66	103	83	128	105
CA(2)行数	9	18	38	25	28	31	45	41
CA(3)行数	27	71	202	125	246	176	286	252

表 4 中我们对算法 1 生成的错误定位表的行数与由 Charlie Colbourn 维护的覆盖表网址[15]上面的已知的覆盖表数的最好上界做了比较。待测系统的类型用如下符号表示：$a_1^{b_1}a_2^{b_2}\cdots a_s^{b_s}$ 表示待测系统共有 $b_1+b_2+\cdots+b_s$ 个因素，其中 b_1 个因素每个有 a_1 个取值，b_2 个因素每个有 a_2 个取值，…，b_s 个因素每个有 a_s 个取值。例如，$4^{12}7^5$ 表示待测系统有 17 个因素，其中 12 个因素有 4 个取值，其余 5 个因素每个有 7 个取值。

表 4 可定位一个错误交互的错误定位表与最好的覆盖表行数比较

类型	3^{20}	4^{16}	5^{24}	6^{18}	7^{16}	8^{18}	9^{20}	10^{20}
ELA 行数	55	78	139	166	206	278	360	431
CA(2)行数[22]	15	27	45	63	82	104	132	155
CA(3)行数[22]	60	124	245	609	637	960	1377	2263

在表 3 中，3 种表的行数数据都是我们编程运行得到的。从表 3 可看出，在每个因素取值较小时，这种特殊的错误定位表行数大致是三维覆盖表行数的一半，是二维覆盖表的行数的两倍多。随着因素个数的增加，ELAs 行数与 CA(3)行数比有逐渐减小的趋势。

在表 4 中，表 CA(2)，CA(3)的行数数据都来源于 Charlie Colbourn 维护的覆盖表网址[15]，这些是目前文献中构造相应的覆盖表所需的最少行数。从表 4 可看出，随着每个因素的

取值个数越来越大，ELAs行数与CA(3)行数比越来越小，即因素取值个数越大。用我们提出的方法来构造这种特殊的错误定位表，比用3维覆盖表来作为错误定位表其规模要小得多，定位这个单一的交互错误需要的测试用例数大大减少，从而节约了测试与调试阶段的成本与资源。

结束语 本文在Colbourn和McClary[4]及Mart′lnez等[5,6]提出的错误定位表的基础上研究了一类特殊的错误定位表，即可定位一个交互错误的错误定位表；研究了它们新的构造以及算法生成问题。所得到的错误定位表的覆盖强度在t维覆盖表与$t+1$维覆盖表之间，其行数比$t+1$维覆盖表行数要少得多。然而，对于实际的测试场景，待测系统中有无错误交互或有多少个错误交互都是未知的，这时我们生成的特殊的错误定位表可在加强交互测试充分性的同时用来帮助或减轻未来故障调试或定位阶段的工作。

对未来的工作，我们主要集中在以下几方面。

(1)对于待测系统中有多个错误交互时，研究如何设计错误定位表、这些错误定位表的规模、生成这些错误定位表的高效算法以及设计出的错误定位表定位错误交互的能力与准确性。

(2)对于待测系统中有多个交互错误时，借鉴基于程序谱的方法[16]来对其中可能的交互错误的可能性排序，然后生成附加的测试用例，进行进一步的测试以便精确定位。

(3)对于规模较大的待测系统，错误交互数较多时，生成的错误定位表规模比较大，并且定位的准确性可能不高，这时应研究可定位这些错误的高效的自适应算法，即根据前面的测试用例的运行结果来生成下一条测试用例的算法，来定位与侦测这些交互错误。

参考文献

[1] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing[C]// Proceedings of the 27th NASA/IEEE Software Engineering Workshop. NASA Goddard Space Flight Center, 2002

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

[3] 徐宝文，聂长海，史亮，等. 一种基于组合测试的软件故障调试方法[J]. 计算机学报，2006，29(1)：132-138

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

[5] Mart′lnez C, Moura L, Panario D, et al. Algorithms to locate errors using covering arrays[C]// LATIN 2008. Lecture Notes in Computer Science 4957. Springer, 2008: 504-519

[6] Mart′lnez C, Moura L, Panario D, et al. Locating errors using ELAs, covering arrays and adaptive testing algorithms[J]. SIAM Journal on Discrete Mathematics, 2009, 23: 1776-1799

[7] 严俊，张健. 组合测试：原理和方法[J]. 软件学报，2009，20(6)：1393-1405

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

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

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

[11] Tung Y W, Aldiwan W S. Automating Test Case Generation for the New Generation Mission Software System[C]// Proc. IEEE Arospace Conf. 2000: 431-437

[12] Bryce R C, Colbourn C J. A Density-based Greedy Algorithm for Higher Strength Covering Arrays [J]. Software Testing, Verification and Reliability, 2009, 19: 37-53

[13] Lei Y, Kacker R, Kuhn D R, et al. IPOG IPOG-D: Efficient Test Generation for Multi-way Combinatorial Testing [J]. Software Testing, Verification and Reliability, 2008, 18(3): 125-148

[14] Cohen M B, Colbouns C J, Gibbons P B, et al. Constructing Test Suites for Interaction Testing [C] // Proceedings of the Intl. Conf. on Software Engineering (ICSE2003). Portland OR, 2003: 38-48

[15] Colbourn C J. Covering array tables [EB/OL]. http://www.public.asu.edu/ccolbou/src/tabby, Dec. 2010

[16] Abreu R, Zoeteweij P, Gemund A. Spectrum-based multiple fault localization[C]// Proc. ASE'09. 2009: 88-99

(上接第114页)

[3] Garcia-Molina H, Polyzois C A. Issues in Disaster Recovery[C]// IEEE Compcon. February 1990: 573-577

[4] Yonghong S, et al. TH-CDP: An Efficient Block Level Continuous Data Protection System[C]// IEEE International Conference on Networking, Architecture, and Storage. 2009

[5] Maohua L, Shibiao L, Tzi-cker C. Efficient Logging and Replication Techniques for Comprehensive Data Protection[C]// 24th IEEE Conference on Mass Storage Systems and Technologies. 2007

[6] Gibson G A. Network attached storage architecture [J]. Communications of the ACM, 2000, 43(11): 59-72

[7] Keeton K, Santos C, Beyer D, et al. Designing for disasters[C]// Proceedings of the 3th USENIX Conference on File and Storage Technologies. San Francisco, CA, USA, 2004: 59-72

[8] 王德军，王丽娜. 容灾技术研究[J]. 计算机工程，2005，31(3)：39-42

[9] Toigo J W. Disaster Recovery Planning: Strategies for Protecting Critical Information Assets (3rd edition) [M]. Prentice Hall PTR, 2002

[10] Stager R K. Method and System for Data Recovery in a Continuous Data Protection System[P]. US 2005/0188256. A1, 2005-08-04

[11] Zhu Ning-ning, Chiueh T-C. Portable and Efficient Continuous Data Protection for Network File Servers [A] // 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07) [C]. 2007

[12] Wang Li-na, Guo Chi. Building Hot Snapshot Copy Based on Windows File System[J]. Wuhan University Journal of Natural Sciences, 2006, 11(1): 1503-1506

[13] 万继光，吴龙树. 一种网络备份系统的性能分析和优化[J]. 小型微型计算机系统，2008(1)：150-15