基于错误交互集的组合测试软件故障定位方法

王建峰,魏长安,盛云龙,姜守达
(哈尔滨工业大学自动化测试与控制系,黑龙江哈尔滨 150001)

摘 要: 提出了一种基于错误交互集的组合测试软件故障定位方法.根据组合测试数据的执行结果,生成可能的错误交互集,通过对集合中全部交互进行定性分析,以有效的避免不同测试用例覆盖的相同错误交互被重复定位的情况,减少所需附加测试用例的数目.提出了基于错误密度的测试用例分析方法,将已有测试结果作为先验知识,提高错误定位的效率.最后,经过算法效率分析及实验验证,本文算法能够在保证准确定位错误交互的基础上,有效减少所需附加测试用例的数目.

关键词: 组合测试; 软件故障定位; 错误交互集; 错误密度
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2014)06-1173-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.06.021

Locating Errors in Combinatorial Testing Using Set of Possible Faulty Interactions

WANG Jian-feng, WEI Chang-an, SHENG Yun-long JIANG Shou-da
(*Automatic test and control institute, Harbin institute of technology, Harbin, Heilongjiang* 150001, *China*)

Abstract: In this paper, we present an algorithm for locating errors in combinatorial testing, based on the set of possible faulty interactions (SPFI) constructed according to the result of combinatorial testing. By introducing the error density of each test case or interaction, we can analyze every test case which caused faults of the software under test, and then the additional test cases are generated to locate the faulty interaction. By constructing the set of possible faulty interactions, the same faulty interactions covered by different test case is avoid to be located repeatedly. And all of the faulty interactions can be located exactly when SPFI is empty. The final empirical results show that the number of test cases needed to locate all faulty interactions can be effectively reduced.

Key words: combinatorial testing; locating errors; set of possible faulty interactions; error density

1 引言

组合测试[1]是一种科学有效的软件测试方法.该方法能够快速有效地检测软件系统中各个因素之间的相互作用对系统产生的影响[2].根据生成测试数据的覆盖程度的不同可分为单因素覆盖、两两组合覆盖、三三组合覆盖及高维多因素组合覆盖.

在组合测试中,当测试用例引发软件错误时,需要测试人员找出触发系统故障的错误交互,如何快速准确的定位当前测试用例中的错误交互成为组合测试中的一个重要问题.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[3]、故障调试法[4]和错误定位表[5~8]等.利用分类树法,系统错误一般很难被精确地确定[5];故障调试定位法通过增加附加测试用例,能够对一些较小型系统错误进行较为精确的定位,但是在系统因素数目众多的情况下,触发软件故障的可能模式数目呈指数增长,使得错误定位效率很低[6];错误定位表在某些特定假设下能够对所有软件交互错误都能精确定位,但是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,在实际组合测试中进行错误定位具有局限性.

作为上述方法的重要补充,2009年,Martinez等人在一般错误定位表的基础上提出了自适应算法来定位两两组合覆盖错误交互[7].2011年,周吴杰等人推广了原有的自适应算法,提出了具有安全值已知时 t 维错误交互定位的自适应算法[9],能够根据当前错误定位的结果自动生成附加的测试用例,并证明定位错误交互所需的

收稿日期:2013-01-23;修回日期:2013-04-01;责任编辑:梅志强
基金项目:中央高校基本科研业务费专项资金资助(No.HIT.NSRIF.2012014)

测试数据是关于错误交互数与因素数的多项式增长的.但是,由于在自适应算法中对于多个测试用例覆盖的相同错误交互,没有采取相应的处理措施,造成了重复定位.并且在每完成一次错误定位后,均需要构造禁忌覆盖数组,即重新生成测试数据,影响了测试效率.

本文结合自适应算法和故障调试定位法的思想,提出了一种基于错误交互集的组合测试错误定位方法,根据组合测试数据的执行结果,生成可能的错误交互集(Set of Possible Faulty Interaction,SPFI),并引入了错误密度的概念,对每一个引发软件故障的测试用例进行分析,进而生成附加测试用例进行重新测试.

2 组合测试软件错误定位模型

对具有 k 个参数的待测系统(Software Under Test,SUT),假设这些参数分别有 $v_1,v_2,\cdots,v_k$ 个可能取值,即,对于任意因素 $i(1\leqslant i\leqslant k)$有 v_i 个可能取值,用0,1,$\cdots,v_i-1$ 来表示,用记号$[0,v_i-1]$表示集合$\{0,1,\cdots,v_i-1\}$,并假设各个参数之间取值均相互独立.参照当前常用的组合测试故障定位模型,给出如下定义[7].

定义 1 (交互,交互集,顶点)设集合 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$,其中各个因素 i_j 互不相同,$a_{i_j}\in[0,v_{i_j}-1](j=1,2,\cdots,t)$,则称这个集合 I 为一个 t 维交互,定义 $E_I=\{i_1,i_2,\cdots,i_t\}$为交互 I 所对应的全部因素所组成集合.定义集合 $H_t=\{I\mid I=\{(i_1,a_{i_1})\},(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}\}$为当前系统的全部 t 维交互集.一维交互$\{(i_j,a_{i_j})\}$也称为顶点,简记为(i_j,a_{i_j}).

定义 2 (测试数据)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 的测试数据.

若一条测试数据 $\boldsymbol{T}$ 的第 i_j 个取值为 $a_{i_j}(j=1,2,\cdots,t)$,即 $\boldsymbol{T}(i_j)=a_{i_j}$,则称这条测试数据覆盖了 t 维交互 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$,记为 $\boldsymbol{T}\supseteq I$.定义符号 $H_{T,t}$表示测试数据 $\boldsymbol{T}$ 所覆盖的全部 t 维交互所组成的集合.

定义 3 (覆盖数组,混合覆盖数组)设 $\boldsymbol{A}$ 是一个 $n\times k$ 矩阵,矩阵中第 i 列元素都取自$[0,v_i-1]$,矩阵的每一行对应一条测试数据,且满足:每个可能的 t 维交互都被表中的某一行所对应的测试数据所覆盖,即对于任意的 t 维交互 $I=\{(i_1,a_{i_1}),(i_2,a_{i_2}),\cdots,(i_t,a_{i_t})\}$,至少存在一行 r,使得 $\boldsymbol{A}(r,i_j)=a_{i_j}(j=1,2,\cdots,t)$,则称 $\boldsymbol{A}$ 是一个 t 维混合覆盖数组,记为 MCA$(n;t,(v_1,v_2,\cdots,v_k))$,$t$ 称为混合覆盖数组的强度,使得 MCA$(n;t,(v_1,v_2,\cdots,v_k))$存在的最小整数 n 称为混合覆盖数,当 $v_1=v_2=,\cdots,=v_k=v$ 时,称矩阵 $\boldsymbol{A}$ 为覆盖数组,记为 CA$(n;t,k,v)$.

在记录 CA 或者 MCA 时,可以把一些具有相同值域的项合并,并省略参数的个数 k,在这种情况下,一个 MCA$(n;t,(v_1,v_2,\cdots,v_k))$可以表示为 MCA$(N;t,S_1^{p_1},S_2^{p_2},\cdots,S_r^{p_r})$,其中 $k=\sum_{i=1}^{r}p_i$.

当一条测试数据引发了软件故障,如何精确定位引发软件错误的具体交互,是组合测试中进行软件故障定位分析的重要问题.假设对于任意测试数据 $\boldsymbol{T}$,在运行时只有两个可能结果:正确或错误,记为 $\boldsymbol{T}\rightarrow$True(SUT)和 $\boldsymbol{T}\rightarrow$False(SUT).

定义 4 (错误交互,极小错误交互)若存在某一交互 I,对于任意测试用例 $\boldsymbol{T}$,若 $\boldsymbol{T}\supseteq I$,必满足 $\boldsymbol{T}\rightarrow$False(SUT),则称 I 为该待测系统的一个错误交互.若对于错误交互 I,其任意真子集均不是错误交互,则称 I 为系统的一个极小错误交互,记为:$I\rightarrow$Min$F$(SUT).

系统的全部极小错误交互所组成的集合称为极小错误交互集,记为 Π.特别的,若 Π 中每个错误交互均有 t 个元素,则该集合为 t 维极小错误交互集,记为 Π_t;若 Π 中每个错误交互所包含元素数均不大于 t 个,则该错误交互集记为 Π_t.本文中若不进行特别说明,提到的错误交互集都是指极小错误交互集.若测试用例 $\boldsymbol{T}$ 没有覆盖 Π 中的任何错误交互,则称这条测试用例避开了 Π.

定义 5 (定位)给定 t 维交互 I 与错误交互集 Π_t,若存在覆盖这个交互 I 的一条测试用例 $\boldsymbol{T}$ 避开了 $\Pi_t\setminus H_t$,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 $\boldsymbol{T}$ 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.

为详细说明错误交互集是否可定位的概念,以三个因素的待测系统为例,如图一所示,图中连线部分表示错误交互,即两种类型系统的错误交互集均为$\{\{(2,0),(3,0)\},\{(2,1),(3,0)\}\}$.其中类型 1 中错误交互集是不可定位的,因为交互$\{(1,0),(3,0)\}$是不可定位的,即不存在任何一个覆盖$\{(1,0),(3,0)\}$的测试用例避开了全部错误交互;而在类型 2 中,交互集就是可定位的,与类型 1 不同的是,测试用例$\{(1,0),(2,2),(3,0)\}$能够定位交互$\{(1,0),(3,0)\}$,并且其全部交互均可定位.

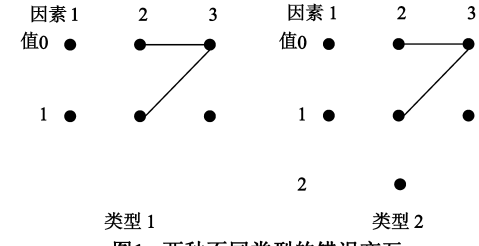

图1 两种不同类型的错误交互

目前,对于任一错误交互集 Π_t 是否可定位,尚没有一种简单有效的判断方法.已有的方法主要是根据系统是否具有安全值进行判断.安全值定义如下:

定义 6 (安全值)设当前 SUT 错误交互集为 Π_t,且对 $\forall i\in[1,k]$, $\exists(i,s_i)$ 没有被 Π_t 覆盖,则该 SUT 具有安全值,并称 $(s_1,s_2,\cdots,s_k)$ 为该 SUT 的安全值向量,其对应各个因素的取值称为该因素的安全点.

若一个待测系统具有安全值向量,则其错误交互集 Π_t 是可定位的[9].

3　安全值已知的组合错误定位算法

3.1　自适应算法

Martinez 等人在文献[7]中首次提出了一种自适应算法,用于在安全值已知的系统中进行组合错误定位,文献[9]对其进行了扩展,提出了能够定位 t 维错误交互的自适应算法.并且证明其所需的测试用例数是关于错误交互数多项式增长的.采用上述方法,定位错误交互集 Π_t 所需要的测试用例数为: $O(d^{t+1}\log k+d(\log k)^2)$.

所谓自适应,就是根据已有测试用例的错误定位结果来进行后面的测试用例选择.假设系统的安全值向量为 $(s_1,s_2,\cdots,s_k)$,已生成的 t 维覆盖数组为 $\mathrm{MCA}(n;t,(v_1,v_2,\cdots,v_k))$.对于每一条测试用例 $\boldsymbol{T}$,若 $\boldsymbol{T}\rightarrow$True(SUT),则继续执行下一条;否则调用错误定位方法对该测试用例进行分析.

错误定位方法的主要思想是将测试用例 $\boldsymbol{T}$ 划分成近似相等的 $t+1$ 个互不相交的集合,然后依次将每集合用安全值代替,则在新生成的 $t+1$ 个测试用例中,至少存在一个执行错误的测试用例,对该测试用例重复调用上述分组方法进入下一循环,直到其中非安全值的顶点个数不大于 t,把其对应因素的取值重新取成安全值来更新测试用例 $\boldsymbol{T}$,重复上述过程,这样得到一个错误交互集 Π_t,且 $\boldsymbol{T}$ 中再不会覆盖与 Π_t 中所有元素都相交的错误交互.然后再重新构造一个部分覆盖表来找出所有不属于 Π_t 且与 Π_t 中元素有交集的错误交互.

3.2　基于 SPFI 的组合测试错误定位方法

在自适应算法的错误定位过程中,若 $\mathrm{MCA}(n;t,(v_1,v_2,\cdots,v_k))$ 中多条不同的测用例覆盖同样的错误交互,则该交互会被重复定位,造成测试用例的重复增加;同时该算法仅对引发错误的测试用例进行分析处理,而对于执行正确的测试结果未加以利用,在生成部分覆盖表时,可能重复覆盖了已经测试验证过的正确交互,造成重复测试,影响了测试效率.

针对上述问题,本文扩展了文献[9]自适应算法,提出了基于可能错误交互集(SPFI)的组合测试错误定位方法.算法的主要思想是生成可能的错误交互集 SPFI,然后对测试用例的执行结果进行分析,定位每条引发错误的测试用例所覆盖的错误交互.对于覆盖相同错误交互的测试用例,则不进行重复定位.

算法中,定义集合 M 为引发错误的测试用例组成的集合,初始值为空,而可能的错误交互集 SPFI 初始为系统的全部 t 维交互,在测试的过程中逐步缩小 SPFI 的规模.然后重点针对每个错误的测试用例,根据其所覆盖的可能错误交互数量进行依次分析,产生有效的附加测试用例,定位其覆盖的错误交互.最后,针对 SPFI 中的剩余交互,生成新的测试用例集进行重新测试,直至 SPFI = $\varnothing$,完成全部测试用例的定位过程.

算法的主要步骤如下:

步骤 1　初始化测试过程.初始化 SPFI = H_t, $\Pi_t=\varnothing$, $M=\varnothing$,利用已生成的 t 维覆盖数组 $\mathrm{MCA}(n;t,(v_1,v_2,\cdots,v_k))$ 对该 SUT 进行测试.

步骤 2　生成 SPFI.对于其中每一条测试用例 $\boldsymbol{T}$,若 $\boldsymbol{T}\rightarrow$True(SUT),则 SPFI = SPFI$\cap\overline{H_{T,t}}$,否则将 $\boldsymbol{T}$ 加入集合 M 中.当 $\mathrm{MCA}(n;t,(v_1,v_2,\cdots,v_k))$ 中全部测试用例执行完毕后,若 $M\neq\varnothing$,则必有 SPFI$\neq\varnothing$.

步骤 3　故障用例分析.依次选择集合 M 中的每条测试用例,定位其覆盖的错误交互并将该交互从 SPFI 中删除,移入 Π_t 中;对于已确定的正确交互则直接从 SPFI 中删除.重复该过程,直至 $M=\varnothing$;

步骤 4　重新生成测试数据集.若 SPFI = $\varnothing$,则完成错误定位过程;否则生成能够覆盖 SPFI 中全部交互的测试用例集 M',并且 $\forall\boldsymbol{T}'\in M'$,满足 $\Pi_t\cap H_{T',t}=\varnothing$,将 M' 赋值给 M,重复步骤 3.

3.3　故障用例分析方法

在故障用例分析过程中(3.2 节中步骤 3),本文借鉴了自适应算法的思想并进行较大改动,自适应算法中对测试用例 T 的分析采用随机分组方式,逐步缩小错误范围.本文算法中,采用已有的测试结果作为先验知识,分析各个交互所引发错误的可能性,尽可能将错误交互集中在较少的分组内,以加速错误定位过程.并且对于重复覆盖了相同错误交互的测试用例,不再进行重复定位,仅在最后利用 SPFI 对未定性交互进行处理,进一步减少了所需的测试用例数目.

对于 SPFI 中每个 t 维交互,采用错误密度来描述其引发软件错误的可能,错误密度定义如下:

定义 7　(错误密度)对于任一交互 I,对其错误密度 $q(I)$ 定义为引发软件故障的测试用例中,覆盖交互 I 的测试用例总数占全部错误用例的比值.设当前引发错误的测试用例集合为 M,且 $M\neq\varnothing$.对于 $\forall I\in$ SPFI,有:

$$q(I) = \frac{|\{\boldsymbol{T} \mid \boldsymbol{T} \in M, \text{且 } \boldsymbol{T} \supseteq I\}|}{|M|} \quad (1)$$

由定义可知,错误密度越大,该交互为错误交互的可能性就越大,相应的,对于 M 中任意一条测试用例 $\boldsymbol{T}$ 及相应因素集 E,$E \subseteq [1,k]$,定义 $Q(\boldsymbol{T},E)$为测试用例 $\boldsymbol{T}$ 在对应相应因素集 E 上的错误密度.

$$Q(\boldsymbol{T},E) = \sum [q(I)]^{\frac{1}{2}}, \quad I \in H_{T,t} \text{且} E_I \subseteq E \quad (2)$$

特别的,若 $E=[1,k]$,则式(2)可简写为:

$$Q(\boldsymbol{T}) = \sum [q(I)]^{\frac{1}{2}}, \quad I \in H_{T,t} \quad (3)$$

$Q(\boldsymbol{T})$称为测试用例 $\boldsymbol{T}$ 的错误密度.

同理,$Q(\boldsymbol{T},E)$越大,则因素集 E 所包含错误交互的可能性就越大.在故障用例分析过程中,采用错误密度作为分组的依据,使得错误交互尽量集中在少数分组范围内.具体分析过程步骤如下:

步骤 1 将集合 M 中全部用例按其错误密度排序,并依次选择错误密度最大的测试用例进行分析,设当前所选测试用例为 $\boldsymbol{T}$,若 $\exists I \in \Pi_i$,满足 $\boldsymbol{T} \supseteq I$,则不对 $\boldsymbol{T}$ 进行任何处理,直接选择下一条测试用例;

步骤 2 设当前所选测试用例为 $\boldsymbol{T}=(t_1,t_2,\cdots,t_k)$,SUT 的安全值为 $\boldsymbol{S}=(s_1,s_2,\cdots,s_k)$,则初始化集合 $A=[1,k]\setminus\{i \mid t_i \neq s_i\}$,若 $|A|>t$,则将集合 A 划分成近似相等的 $t+1$ 个互不相交的集合 $A_1,A_2,\cdots,A_{t+1}$,分组时采用贪心算法的思想,在每增加一个分组时,使 $Q(\boldsymbol{T},\cup A_j)$均具有最大值,即错误交互尽量集中,继续下一步;若 $|A| \leqslant t$,则利用安全值逐一替换各因素,定位 A 中的极小错误交互,输出至 Π_i,结束分析过程;

步骤 3 利用每一个集合 A_j,将测试用例 $\boldsymbol{T}$ 相应位置上的取值替换成安全值,生成新的测试用例 $\boldsymbol{T}_j=(t_{j1},t_{j2},\cdots,t_{jk})$,$j \in [1,t+1]$,其中:

$$t_{ji} = \begin{cases} t_i, & \text{if} \quad i \in A_j \\ s_i, & \text{if} \quad i \notin A_j \end{cases}, \quad i \in [1,k] \quad (4)$$

步骤 4 顺序执行该 $t+1$ 条测试用例,并实时更新 SPFI.由于引发系统的错误交互最多为 t 维,可知上述 $t+1$ 个测试用例中,至少有一条包含错误交互的测试用例;

步骤 5 设执行过程中出现 $m(m \geqslant 1)$条错误,取全部 m 条故障的测试用例交集所对应的因素集 A',可知 A'即为 $A_1,A_2,\cdots,A_{t+1}$中的 $t+1-m$ 个分组,根据 A'生成新的测试用例 $\boldsymbol{T}'$,若 $\boldsymbol{T}' \rightarrow$False(SUT),则将 $\boldsymbol{T}'$赋值给 $\boldsymbol{T}$,返回步骤 2;若 $\boldsymbol{T}' \rightarrow$True(SUT)或 $A'=\varnothing$(即 $m=t+1$)任选 m 条故障测试用例中的一条赋值给 $\boldsymbol{T}$,返回步骤 2.

其中,在步骤 2 中,采用了贪心算法对集合 A 进行分组,使得错误交互更为集中,具体过程如下:

首先,遍历测试用例 $\boldsymbol{T}$ 在因素集 A 上的全部可能错误交互,并选择其中错误密度最大的交互 I,根据每个分组大小,将因素集 E_I 作为最先的 n 个分组的初始数据,其中

$$n = \left\lfloor \frac{t(t+1)}{|A|} \right\rfloor$$

若当前第 n 个分组数据未达到上限,则选择 A 中其他因素加入该组并使 $Q(\boldsymbol{T},\cup A_j)$具有最大值,其中 $j \in [1,n]$;之后每增加一个新的分组时,均遍历 A 中全部剩余因素,使得 $Q(\boldsymbol{T},\cup A_j)$最大,直至完成全部 $t+1$ 个分组.

算法中,采用已有的测试结果作为先验知识,计算各测试用例在相应因素集上的错误密度,作为故障用例分析过程中的分组依据,使得错误交互集中在较少的分组内,使得 m 值也随之增大,进而提高了故障定位速度.同时,对覆盖相同错误交互的测试用例不重复分析,利用 SPFI 定位其覆盖的其他交互,并利用错误密度对测试用例进行排序,优先分析可能覆盖错误交互较多的测试用例,以尽量减少故障用例分析的次数.

3.4 算法性能分析

本节对算法性能进行分析,主要分析算法所需增加的测试用例数目 N 及其他系统参数之间的关系.本文算法中,仅在故障用例分析和重新生成测试数据集的过程中增加了新的测试用例,因此,重点对这两个方法进行分析.设上述两个方法中所需增加的测试用例数目分别为 N_L 和 N_R,则有 $N=N_L+N_R$.

在故障用例分析过程中,将集合 A 分成 $t+1$ 个分组,执行不同的测试用例,当出现 m 个引发错误的测试用例,该 m 个错误测试用例的交集即为其中 $t+1-m$ 个分组全部对应因素.当 $m=1$ 或 $m=t+1$ 时,因为进入下一次循环中保留了原有 $t+1$ 个分组中的 t 个,在算法中出现 $t+1$ 条测试用例全部错误时,执行步骤与 $m=1$ 时完全相同,在进行算法效率分析时,取 $m \in [1,t]$.每进入下一次循环时候,集合 A 中选择了原有 $t+1$ 个分组中的 $t+1-m$ 个,此时的衰减速度为 $\frac{t+1-m}{t+1}$.

初始时,$|A|=k$,循环至 $|A| \leqslant t$,设循环次数为 r,因此,循环终止条件为:$(\frac{t+1-m}{t+1})^r k \leqslant t$,由于每次循环过程中,所增加的附加测试用例为 $t+1$ 个,因此,在完成单条测试用例错误定位过程中,所增加的附加测试用例数为:

$$N_L = \frac{\log k - \log t}{\log(t+1) - \log(t+1-m)}(t+1) \quad (5)$$

在最差情况下,式(5)中,取 $m=1$,表示当前错误交互的 t 维因素恰被分到 t 个不同的分组中,或者当前测试用例包含多个错误交互且覆盖全部 $t+1$ 个分组,

此时,每定位一个错误交互所需的附加测试用例数与文献[9]中的自适应法相同.因此,对于 M 中任意一条测试用例 $\boldsymbol{T}$,定位其覆盖的 d_t 条错误交互最多增加的测试用例数目为:

$$N_{L\max} = d_t \frac{\log k - \log t}{\log(t+1) - \log(t)}(t+1) \tag{6}$$

在重新生成测试数据集的过程中,对于 M 中任意一条测试用例 $\boldsymbol{T}$,定位其覆盖的 d_t 条错误交互后,有:

$$\left| \mathrm{SPFI} \cap H_{T,t} \right| \leqslant \binom{k}{t} - d_t$$

即在集合 SPFI 中,$\boldsymbol{T}$ 所覆盖的交互数最多为 $\binom{k}{t} - d_t$. d_t 条错误交互所对应因素个数最多为 $d_t t$ 个.生成新的组合覆盖表覆盖上述全部交互最多需要 $\binom{d_t t}{t}$ 条测试用例,即依次在 $d_t t$ 个因素中选取 t 个用安全值替换即可.根据 $d_t \leqslant d$ 可知,对于 M 中一条测试用例 $\boldsymbol{T}$,定位其全部错误交互,该方法中所需增加的测试用例条数不超过 $\frac{d^t t^t}{t!}$.

设 M 中测试用例数目最大为 w,在 MCA$(n;t,(v_1,v_2,\cdots,v_k))$中,任意 t 维交互平均被覆盖的次数为 R,则有 $w \leqslant dR$,又根据组合覆盖数目的行数 n 满足 $n = O(v^t \log k)$,则有 $R = O(\log k)$.对于覆盖相同错误交互的测试用例仅在重新生成测试数据集的过程中进行处理,因此:

$$\begin{aligned} N_L &\leqslant d\left[d_t \frac{\log k - \log t}{\log(t+1) - \log(t)}(t+1) \right] \\ &\leqslant d^2 \frac{\log k - \log t}{\log(t+1) - \log(t)}(t+1) \\ N_R &\leqslant dR \frac{d^t t^t}{t!} \\ N &= O\left(d^2 \frac{\log k - \log t}{\log(t+1) - \log(t)}(t+1) + d \log k \frac{d^t t^t}{t!} \right) \end{aligned}$$

由于在实际测试中,均有 $t > 1$,所以 $N = O(d^{t+1} \log k)$.

4　仿真实验

为说明算法效果,本节将本文算法与其他组合测试软件故障分析方法进行比较.在目前较流行的算法中,分类树法很难精确完成错误定位;错误定位表法不需要进行附加测试,但是由于采用了高维覆盖表,其所需测试用例数远大于其他算法.因此,本文选用的比较对象主要包括自适应算法[9]和故障调试法[4].

表 1 为对于不同系统,利用已生成的组合覆盖数组进行测试,在不同的错误交互数目时算法所需的附加测试数量.

从表 1 可以看出,故障调试法所需的附加测试用例数目明显多于其他算法,这是由于故障调试法仅对当前已有模式进行分析,无需重新生成新的覆盖数组,因此造成所需测试的错误模式较多.而自适应算法则对错误的测试用例进行分析,在一定程度上减少了所需测试用例的数目.本文算法和自适应算法相比,所需测试用例数仍具有优势,这是因为本文算法采用了错误密度作为分组依据,多次计算不同交互的错误密度,使得错误交互尽量集中,降低了定位单条错误交互时所需测试用例数目,并且利用 SPFI 避免了重复交互的多次定位,减少了重复测试的次数.同时,在分组时,需要多次计算错误密度,导致本文算法运行速度有所降低,但是对于多数系统来说,算法所需时间均小于 1s,在实际应用过程中,对测试效率影响较小.

表 1　不同的错误交互数目时算法比较结果

SUT	d	故障调试法	测试用例数 自适应法	本文算法
MCA$(n;2,3^{12}4^5)$	2	114	89	26
	3	145	97	36
	5	204	154	58
CA$(n;2,6,4)$	2	21	21	16
	3	28	29	21
	5	47	39	33
MCA$(n;2,5^1 3^8 2^2)$	2	39	29	20
	3	47	31	28
	5	95	63	45
MCA$(n;2,5^2 4^2 3^2)$	2	19	14	11
	3	36	23	19
	5	53	41	36
MCA$(n;2,5^1 4^4 3^{11} 2^5)$	2	43	31	25
	3	78	58	37
	5	171	106	64
	10	386	223	140
CA$(n;2,10,10)$	2	46	41	23
	3	58	38	31
	5	121	70	53
	10	255	150	110

为进一步说明错误交互数目增加对算法结果的影响,选取典型系统 CA$(n;2,10,10)$,逐步增加其错误交互个数 d,分别选用上述三种方法进行定位,实验结果如图 2 所示.同时为说明在定位单个错误时,不同系统规模下算法的执行情况,选取当前 SUT,固定其每个参数为 10 个取值,不断增加参数个数,结果如图 3 所示.

由图 2 和图 3 中可以看出,随着系统规模增大和故障数目的增多,故障调试法所需的测试用例变化最为明显,而自适应算法和本文算法变化相对缓慢,这是由于故障分析法中,将所有可能引发错误的交互均进行测试,采用了一种类似于穷举的方式对故障进行定位.与自适应算法相比,本文算法在大多数情况下均具有

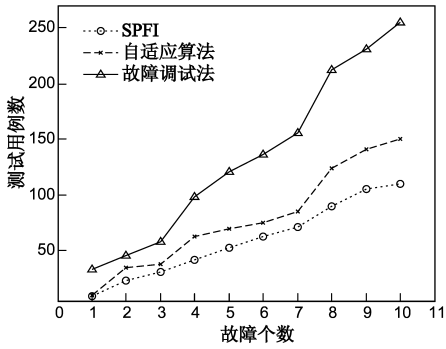

图2 错误交互数目对算法的影响

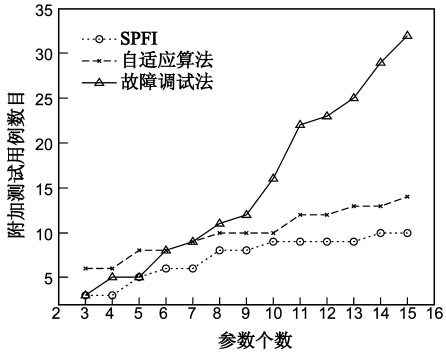

图3 系统规模对算法的影响

优势,所需测试用例数目减少了 10% ~ 30%.这是因为采用了错误密度作为分组的依据,并且利用 SPFI 作为交互是否完成定位的依据,避免了重复定位过程,减少了所需测试用例的数目.

5 结论

本文提出了一种基于 SPFI 的组合测试错误定位算法,利用组合测试的执行结果,生成可能引发软件错误的交互集合 SPFI,并通过对引发软件错误的测试用例进行逐条分析,定位其所覆盖的错误交互,并通过对 SPFI 中剩余全部交互进行组合覆盖定位,最终精确定位全部错误交互.经过算法效率分析及实验验证,与自适应算法相比,本文算法能够在保证准确定位错误交互的基础上,所需附加的测试用例数目减少了 10% ~ 30%.

下一步的研究工作主要集中在:(1)对于安全值未知的待测系统,如何快速有效的确定系统的安全值,进而快速定位软件错误;(2)研究新的组合测试方法,将错误定位过程和测试用例生成过程相结合,在生成组合测试用例的过程中,实时定位已有的软件错误,并避免生成包含已知错误的测试用例.

参考文献

[1] 严俊,张健.组合测试:原理与方法[J].软件学报,2009,20(6):1393 - 1405.
Yan J, Zhang J. Combinatorial testing: principles and methods [J]. Journal of Software, 2009, 20(6): 1393 - 1405. (in Chinese)

[2] 黄陇,等.参数配对及 *n*-way 组合覆盖算法研究[J].计算机学报,2012,35 (2):257 - 269.
Huang long, et al. Research on algorithm of parameter pair wise and *n*-way combinatorial coverage [J]. Chinese Journal of Computers, 2012, 35(2): 257 - 269. (in Chinese)

[3] 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.

[4] 徐宝文,聂长海,史亮,等.一种基于组合测试的软件调试方法[J].计算机学报,2006,29(1):132 - 138
XU Baowen, NIE Changhai, SHI Liang, et al. A software failure debugging method based on combinatorial design approach for testing[J]. Chinese Journal of Computers, 2006, 29(1): 132 - 138. (in Chinese)

[5] Matrtinez C, Moura L, et al. Algorithms to locat errors using covering arrays [A]. Proceedings of LATIN 2008 8th Latin American Theoretical Informatics [C]. Buzios, Brazil: Lecture Notes in Computer Science, 2008. 504 - 519.

[6] 周吴杰,张德平,徐宝文.基于部分覆盖表的错误交互定位方法[J].计算机学报,2011,34(6):1126 - 1136.
ZHOU Wujie, ZHANG Deping, XU Baowen. Locating error interactions based on partial covering array[J]. Chinese Journal of Computers, 2011, 34(6): 1126 - 1136. (in Chinese)

[7] Martinez C, Moura L, et al. Locating errors using ELAs, covering arrays and adaptive testing algorithms[J]. SIAM Journal on Discrete Mathematics, 2009, 23(4): 1776 - 1799.

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

[9] 周吴杰,张德平,徐宝文.基于组合测试的软件故障定位的自适应算法[J].计算机学报,2011,34(8):1509 - 1518.
ZHOU Wujie, ZHANG Deping, XU Baowen. An adaptive algorithm of locating fault interactions based combinatorial testing [J]. Chinese Journal of Computers, 2011, 34(8): 1509 - 1518. (in Chinese)

作者简介

王建峰 男,1984 年出生,哈尔滨工业大学自动化测试与控制系博士研究生.主要研究方向为软件测试技术,组合测试方法.
E-mail: hit08b901005@126.com

魏长安 男,1981 年出生,哈尔滨工业大学自动化测试与控制系讲师.主要研究方向为虚拟试验技术,自动测试技术等.
E-mail: nathanwei@163.com

盛云龙 男,1988 年出生,哈尔滨工业大学自动化测试与控制系硕士研究生.主要研究方向为软件自动测试技术.