基于符号零压缩二叉决策图的组合测试用例生成方法

黄钰尧 李凤英 常 亮 孟 瑜
(桂林电子科技大学广西可信软件重点实验室 广西 桂林 541004)

摘 要 组合测试是系统测试中一种非常有效的方法,能够在保证错误检出率的前提下采用较少的测试用例来测试系统。但是,组合测试用例集构造问题的复杂度是NP完全的。给出了一种基于符号零压缩二叉决策图(Zero-suppressed Binary Decision Diagram,ZBDD)的组合测试用例生成方法。该方法首先利用ZBDD的结构特性,对测试系统进行紧凑的符号表示。然后利用ZBDD的隐式操作,结合贪心算法的思想,不断地覆盖更多的组合并缩小未覆盖组合集合,生成2～4维覆盖强度的较小测试用例集。实验证明,所提方法不仅可行而且节点开销小。

关键词 组合测试,零压缩二叉决策图,覆盖强度,测试用例生成
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.01.045

Symbolic ZBDD-based Generation Algorithm for Combinatorial Testing

HUANG Yu-yao LI Feng-ying CHANG Liang MENG Yu
(Guangxi Key Lab of Trusted Software,Guilin University of Electronic Technology,Guilin,Guangxi 541004,China)

Abstract Combinatorial testing is an effective method in system testing,which can test system with fewer test cases under the premise of guaranteeing error detection rate. However,the complexity of the problem of constructing test cases is NP-complete,and many algorithms only get the suboptimal solution. This paper presented a method of generating test cases based on symbolic Zero-suppressed binary decision diagram(ZBDD). The method first uses the structure of ZBDD to perform a compact symbolic representation of the test system. Then,using implicit operations of ZBDD and combining with the idea of greedy algorithm,the method can cover more combinations and reduce the set of uncover combinations to generate smaller set of test cases. The method can satisfy 2 to 4 wise coverage. Experiments show that this method is feasible and has the characteristic of small node cost.

Keywords Combinatorial testing,Zero-suppressed binary decision diagram,T-wise coverage,Test case generation

1 引言

组合测试(Combinatorial Testing)是生成测试系统(Software Under Test,SUT)的测试用例集时在性能和代价上进行折中的一种方法。Kuhn等人的研究[1]表明,超过70%的软件故障是由一个到两个参数的相互作用而引发的。而且,人们用两两组合覆盖法对测试系统进行测试时发现了很多传统方法很难发现的错误[2]。在一个测试系统中有k个参数,而这k个参数分别对应各自的$v_1,v_2,\cdots,v_k$个取值,在传统方法下测试系统需要$\prod_{i=1}^{k}v_i$个测试用例,对于一些较为复杂的系统,这个测试用例集的规模太庞大而且需要消耗大量时间。组合测试的提出很好地解决了这个问题,通过两两组合、三三组合等,能够尽可能缩减测试用例集并尽可能覆盖更多的组合。举一个简单的例子,设待测系统中有4个参数,每个参数有3个取值,完全测试该系统需要81个测试用例,而两两组合覆盖测试仅需9个测试用例。目前人们已经提出很多数学方法、启发式搜索方法和各种贪心算法,但是每种算法都有局限性,并且都只在针对某个特殊问题时才具有一定优势。目前常用的算法有TConfig[3],SA[4],AETG[5,7],IPO[8],ACTS[9],PICT[10]以及基于有序二叉决策图的算法[11]等。

有序二叉决策图(Ordered Binary Decision Diagram,OBDD)及其扩展形式可以实现状态空间或者变量组合的隐式表示和搜索,是迄今为止最为有效的符号技术之一[12-13]。OBDD最初被开发用于处理布尔函数,这种数据模型经常出现在实际问题中,例如开关设备的组合(开/关)、故障组合和网络中的路径集合。近年有学者提出了基于OBDD的组合测试用例生成方法,证明了OBDD能够应用于组合测试领域,

到稿日期:2016-11-17 返修日期:2017-04-29 本文受广西自然科学基金项目(2016GXNSFAA380054),桂林电子科技大学研究生教育创新计划资助项目(YJCXS201541),广西高等学校高水平创新团队及卓越学者计划资助。
黄钰尧(1990－),男,硕士,主要研究方向为符号计算和软件测试,E-mail:334039026@qq.com(通信作者);**李凤英**(1974－),女,副教授,主要研究方向为Petri网、符号计算和软件测试等,E-mail:lfy@guet.edu.cn;**常 亮**(1980－),男,教授,主要研究方向为描述逻辑、语义Web服务等;**孟 瑜**(1976－),女,博士生,主要研究方向为符号计算和软件测试等。

并且它相比于传统方法具有一定的优势。

零压缩二叉决策图(ZBDD)[14]是为了弥补 OBDD 处理组合集合问题时存在的不足而提出的,它是一种特殊类型的二叉决策图,使用 ZBDD 表示和处理组合集合问题,能进一步降低空间需求,从而更好地解决组合集合问题,是求解组合集合问题的有效工具。鉴于此,文中讨论了基于符号 ZBDD 的组合测试用例生成方法,给出了测试系统、未覆盖组合集合和测试用例的 ZBDD 描述,设计了生成测试用例的符号方法。根据该方法可以高效地求解出一个测试系统在给定覆盖强度下的测试用例集,并结合实例验证了基于符号 ZBDD 的测试用例生成方法的正确性和可行性。

2 零压缩二叉决策图

下面将基于组合集合表示问题的一个例子对 OBDD 与 ZBDD 各自的特点进行介绍。

设集合 $A=\{a_1,a_2,\cdots,a_n\}$,从 A 中不考虑次序地取出任意 $r(0\leqslant r\leqslant n)$ 个元素,所得到的集合称为集合中元素的一个组合。若集合 A 中的每个元素均对应一个布尔变量 x_i,则组合集合的元素可以用 n 维布尔向量 $x=(x_1,x_2,\cdots,x_n)$ 来表示,其中 $x_i\in\{0,1\}(i=1,2,3,\cdots,n)$ 称为向量元素。每一个向量元素 x_i 对应集合 A 中的一个元素 a_i,且当 $x_i\neq x_j$ 时,有 $a_i\neq a_j$。当 $x_i=0$ 时,表示对应的元素 a_i 不包含在组合中;否则,表示对应的元素包含在组合中。由此可见,组合集合可以用一个 n 元布尔函数 $f(x_1,x_2,x_3,\cdots,x_n)$ 来表示。函数 f 的值表示该变量对应的 A 中元素的组合是否属于该组合集合,称布尔函数 f 为该组合集合的特征函数。由于 OBDD 是布尔函数的一种紧凑规范表示,基于 OBDD 可以有效完成布尔表达式的各种逻辑操作。因此,可以用 OBDD 来表示组合集合的特征函数。

表示组合集合的 OBDD 有如下特点:

1)OBDD 中的节点数远小于组合集合中包含的元素个数。

2)基于 OBDD 的组合集合上的操作时间与表示该组合集合的 OBDD 中的节点数目成正比。

3)OBDD 表示依赖于变量的个数。当变量个数不同时,通过相同的组合集合会得到全然不同的 OBDD。

OBDD 与 ZBDD 的不同主要集中在第三点,以下例子可以说明。

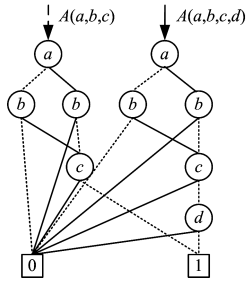

图 1 组合集合的 OBDD 表示

Fig. 1 OBDD express of combinatorial set

如图 1 所示,在不考虑不相关元素 c 和 d 的情况下,$A(a,b,c)=f(x_1,x_2,x_3)=x_1'x_2x_3'+x_1x_2'x_3'$ 和 $A(a,b,c,d)=f(x_1,x_2,x_3,x_4)=x_1'x_2x_3'x_4'+x_1x_2'x_3'x_4'$ 是组合集合$\{a,b\}$在不同域上的特征函数。但由于变量个数不同,表示相同组合集合的 OBDD 为两个不同的 OBDD。

之所以出现上述情况,是因为不相关的变量在集合和布尔函数中的含义是不同的。在集合中,如果某变量不出现,则表明该集合中不包括该元素,该变量取值为 0;而在布尔函数中,如果某变量不出现,则表明该变量取值对函数值没有影响,即函数值不依赖于该变量,该变量可以取 0 或 1。由此可见,OBDD 中的删除规则对于不相关变量的影响不是很大,在图 1 中,不相关的变量 c 和 d 仍出现在 OBDD 中。

为了更加高效地表示组合集合,ZBDD 引入了不同于 OBDD 的化简规则,即 *pD*-删除规则[14],使 ZBDD 能够更简捷地表示组合集合,尤其是稀疏组合集合。

pD-删除规则:对于 OBDD 中的节点,如果它的 1-分支节点为终节点 0,则删除该节点,并将节点的父节点直接连接至该节点的 0-分支节点,如图 2 所示。与 OBDD 相比,ZBDD 并不删除那些 0-分支节点和 1-分支节点相同的节点,而是删除 1-分支节点为终节点 0 的节点。由此可见,在 ZBDD 中未出现变量的缺省含义与集合中未出现元素的含义一致,因此,*pD*-删除规则更有利于组合集合的表示。图 1 中所示的 $A(a,b,c)$ 与 $A(a,b,c,d)$ 有相同的 ZBDD 表示,$A(a,b,c)=A(a,b,c,d)=(a,b)$,如图 3 所示,即 ZBDD 结构的表示不依赖于变量个数。

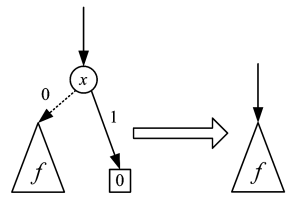

图 2 *pD*-删除规则

Fig. 2 *pD*-deleting rule

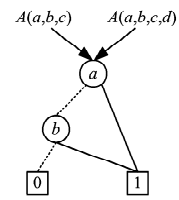

图 3 组合集合的 ZBDD 表示

Fig. 3 ZBDD express of combinatorial set

ZBDD 的特点如下:

1)在 ZBDD 中,不相关项的节点(从未在任何组合中出现)由 ZBDD*pD*-删除规则自动删除。在 OBDD 中,不相关的节点仍然保留,并且它们可能减少共享节点。

2)ZBDD 对于稀疏组合的表示和处理非常有效。例如,选择 1000 个项中的 10 个组合集合,ZBDD 表示比 OBDD 紧凑 100 倍。

3)从根节点到 1-终端节点的每条路径对应于集合中的每个组合,即 ZBDD 中的这种路径的数目等于集合中的组合的数目。在 OBDD 中,该属性不是总成立的。

4)在 ZBDD 中不存在等效节点时是最坏的情况，ZBDD 结构显式地存储组合中的所有项。也就是说，ZBDD 大小(的顺序)从不超过显式表示。如果更多的节点被共享，那么 ZBDD 将比线性列表更紧凑。

因此，ZBDD 更适合于表示组合集合，是求解组合集合问题的有效工具[15]。

3　测试系统的 ZBDD 表示

本节主要研究测试系统建模完成后如何使用 ZBDD 对其进行隐式表示及计算的问题。

例如，测试系统如表 1 所列。该测试系统的测试用例可以表示为类似($A1$，$B1$，$C1$，$D1$)的一个集合，共有$3*3*3*3=81$ 个这样的集合。

表 1　测试系统

Table 1　Test sytems

参数	A	B	C	D
参数值	$A1$	$B1$	$C1$	$D1$
	$A2$	$B2$	$C2$	$D2$
	$A3$	$B3$	$C3$	$C3$

文中使用 ZBDD 表示测试系统的方法是：每个变量表示一个参数值，每个参数对应的变量数就是它自身参数值的个数。假设在不对变量排序的情况下，参数 A 的 3 个变量 $A1$，$A2$，$A3$ 分别表示为 ZBDD 变量 x_0，x_1 和 x_2，参数 B 的 3 个变量表示为 x_3，x_4 和 x_5，以此类推。测试系统中的测试用例($A1$，$B1$，$C1$，$D1$)的 ZBDD 表示为($x_0x_3x_6x_9$)，如果将其转化为二进制就相当于 100 100 100 100，在 ZBDD 结构中表示一条路径。上例中所有的测试用例可以表示为如图 4 所示的 ZBDD。

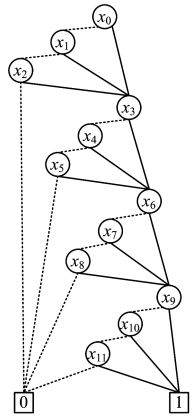

图 4　有效集合 Valid 的 ZBDD 表示

Fig. 4　ZBDD express of effective set Valid

图 4 中共有 81 条路径，对应着测试系统的 81 个测试用例，本例中称图 4 中的 ZBDD 为有效集 Valid。本算法中，如果测试强度为 2，还需对每一对参数组合的参数值之间的组合集合进行 ZBDD 表示，称为未覆盖集合 Uncover，如参数 A 与参数 B 的参数组合有集合{($A1$，$B1$)，($A1$，$B2$)，($A1$，$B3$)，($A2$，$B1$)，($A2$，$B2$)，($A2$，$B3$)，($A3$，$B1$)，($A3$，$B2$)，($A3$，$B3$)}，对应的 ZBDD 表示为{(x_0x_3)，(x_0x_4)，(x_0x_5)，(x_1x_3)，(x_1x_4)，(x_1x_5)，(x_2x_3)，(x_2x_4)，(x_2x_5)}，Uncover 的 ZBDD 表示如图 5 所示。同理可得其他参数之间的两两组合的表示。与 OBDD 的方法相比，本方法主要的区别之一就在于 Uncover 的表示，在 OBDD 表示中需要 Uncover 对参数进行两两组合得到集合的所有节点，而 ZBDD 表示仅需要两个参数所对应的参数值的少量节点即可。因此对于大型、Uncover 数量多的测试系统，ZBDD 能够减少更多的节点开销。

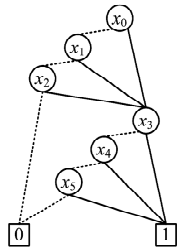

图 5　未覆盖集合 Uncover[1] 的 ZBDD 表示

Fig. 5　ZBDD express of uncovered set Uncover[1]

4　基于符号 ZBDD 的组合测试用例生成方法

通过将 ZBDD 数据结构的特点和贪心算法的思想相结合，设计出基于符号 ZBDD 的组合测试用例生成算法。本算法需要利用多个辅助 ZBDD，如有效集 Valid，未覆盖组合集合 Uncover[i]($i\in(1,2,\cdots,n)$，$n=C_k^t$，其中 k 表示参数的个数，t 表示覆盖强度)，收集集合 collected。其伪代码如下：

```
1. 输入：测试系统各个参数、参数值、限制关系
2. 输出：组合测试用例集
3. Valid = CreateInitValidZdd();
4. Valid = getConstrains();
5. for(i=1 to NumofUncover){
6.      Uncover[i++]=InitUncover();
7. }
8. do{
9. collected=Valid;
10. sort (Uncover[]);
11. for(i=1 to NumofUncover){
12.      if(Uncover[i]!=NULL){
13. tmpuncover = EnlargeProject(Uncover[i++]);
14.      if((tmpuncover∩collected)! =NULL)
15.        collected=tmpuncover∩collected;
16. }
17. }
18. chosen = chosePath(collected);
19. result=result ∪ chosen;
20.      for(i=1 to NumofUncover){
21.      if(Uncover[i]==NULL)
22.        Continue;
23.      tmpchosencover= chosenCover(chosen);
24.      chosencover=ShrinkProject(tmpchosencover);
25.      UncovZdd[i]=UncovZdd[i] − chosencover;
26. }
27. }while(RemainNumofUncover!= 0)
28. Output result
```

伪代码中第 3－6 行将有效集合 Valid 和未覆盖集合 Un-

cover[]进行初始化。然后开始进行迭代,每一次迭代需要执行如下步骤:在第 9 行,用有效集 Valid 初始化收集集合 collected,collected 的作用是在每一次迭代中收集那些能够覆盖尽可能多的新组合的测试用例;第 10 行对未覆盖集合 Uncover[]进行排序。这里结合了贪心算法的思想,希望每一次迭代后 collected 收集到的测试用例都尽可能多地覆盖新的组合,因此需要对 Uncover[]进行排序。本算法实际上对 Uncover[]进行了两轮排序,第一轮排序按照每一个 Uncover[i]中所剩余路径数量进行降序排序,Uncover[i]的每一条路径对应着相应的参数之间的一个参数值的组合,这样排序能够保证每次迭代时 collected 总是能够先照顾到剩余组合多的 Uncover[i]。第一轮排序后得到大量剩余组合一样多的 Uncover[i]连续排在一起,而且他们两两之间对应的第一个参数中有很多是相同的,例如,Uncover[1-6]在迭代一定次数后,它们对应的关系如表 2(Unci 为 Uncover[i]的缩写)所列,这样就会导致 collected 在收集时过多地考虑参数 A,使得 collected 收集到的测试用例集合在一定程度上忽略了参数{BC},{BD},{CD}之间的组合,因此需要第二轮排序,在剩余组合相同的情况下尽可能让相邻 Uncover[]的第一个参数不同或者不相邻。

表 2 第一轮排序结果

Table 2 The first-round sort results

排序结果	Unc3	Unc1	Unc2	Unc5	Unc6	Unc4
参数 1	A	A	A	B	C	B
参数 2	D	B	C	D	D	C
剩余组合	5	5	5	5	5	4

第 11—15 行是 collected 收集测试用例的迭代过程。在收集之前,需要将每一个 Uncover[i]进行放大投影(Enlarge Project)操作,目的在于:1)使放大投影后的 ZBDD 中除了当前 Uncover[i]对应的组合关系不变,其他参数全部为任意取值;2)使 collected 与放大投影后的 ZBDD 取交集后,collected 中只保留包含当前 Uncover[i]中组合的测试用例。在这部分中,collected 每次与当前 Uncover[i]放大投影后的 ZBDD 取交集后,collected 中的每个测试用例都覆盖了当前 Uncover[i]中的一个组合,当迭代完成后,collected 与 N 个 Uncover[i]都进行了交操作,这表示 collected 中的每一条测试用例都覆盖了 N 个组合,前面的排序就是为了使 N 尽可能的大。

第 20—25 行是对 Uncover[]进行更新的操作。其中第 23—24 行的作用是计算之前选中的测试用例 chosen 覆盖了 Uncover[i]中的哪一个组合。这里有一个 ShrinkProject 即缩小投影的操作,它的作用是将一条完整路径中对应 Uncover[i]的组合提取出来。第 25 行更新 Uncover[i],将已经被 chosen 覆盖的组合从 Uncover[i]中删除。

至此,一轮迭代结束,得到了一条覆盖尽可能多的组合的测试用例。

由于使用 ZBDD 表示来举例的篇幅过大,下面使用集合的形式并结合上文中的例子来具体描述算法。在表 1 的测试系统中有 4 个参数,每个参数有 3 个值,没有限制约束。通过算法第 3 行计算得出图 4 中的有效集 Valid,以 ZBDD 的形式存储。由于本例没有约束,跳过第 4 行。第 5—7 行初始化未覆盖组合集合 Uncover[i],在本例中,Uncover 共有 6 个,分别是参数组合{AB},{AC},{AD},{BC},{BD}和{CD}的组合集合,其中参数组合{AB}的参数值未覆盖组合集合 Uncover[1],如图 5 所示,在本例中其余 5 个 Uncover[]的初始化后的结构与 Uncover[1]相同。但随着算法迭代次数的增加,每个 Uncover[]更新的情况各不相同。算法进入迭代之后,先对 Uncover[i]进行排序,排序依据上文的解释,此处不再赘述。

排好序后须将每一个 Uncover[i]进行放大投影,本例中参数{AB}的参数值的组合集合存储时如图 5 的 ZBDD 所示,对应的 Uncover[1]表示成组合集合{($A1$,$B1$),($A1$,$B2$),($A1$,$B3$),($A2$,$B1$),($A2$,$B2$),($A2$,$B3$),($A3$,$B1$),($A3$,$B2$),($A3$,$B3$)},以下缩写为{AB}。

将其放大投影(Enlarge Project)的操作相当于集合运算:{AB}×{$C1$,$C2$,$C3$}×{$D1$,$D2$,$D3$},且记为 tmpuncover。而 collected 初始化之后相当于集合{$A1$,$A2$,$A3$}×{$B1$,$B2$,$B3$}×{$C1$,$C2$,$C3$}×{$D1$,$D2$,$D3$}。由于首次迭代时每个 tmpuncover 与 collected 相同,因此交集之后没有任何变化,只需从最后结果中任意选出一个组合作为第一个用例即可。本例选择($A2$,$B3$,$C3$,$D2$)作为第一个测试用例。根据此测试用例更新 Uncover[]。然后用不同组合对上述测试用例进行组合提取即缩小投影(Shrink Project),如:($A2$,$B3$,$C3$,$D2$)对{AB}提取的结果为($A2$,$B3$),那么对应于{AB}的 Uncover[1]更新后的组合集合为{($A1$,$B1$),($A1$,$B2$),($A1$,$B3$),($A2$,$B1$),($A2$,$B2$),($A3$,$B1$),($A3$,$B2$),($A3$,$B3$)};同理,根据测试用例($A2$,$B3$,$C3$,$D2$)更新每一个 Uncover[i]后,即可开始新一轮的迭代,随着迭代的进行,Uncover[i]中的组合数会越来越少,直到为零。当所有 Uncover[i]中的组合数都为零时,表明所得到的测试用例集已经覆盖了当前强度下的所有组合。算法结束。

5 实验结果

为了检测 ZBDD 算法生成测试用例的效果,将它与常用测试用例生成算法就生成测试用例数量方面进行比较(见表 3),并针对 OBDD 算法进行了节点开销以及时间花费的比较(见表 4、表 5)。从表 3 中可以看出,对于不同的测试系统,各个算法各有优势,而本文的 ZBDD 测试用例生成算法在 $S1$ 系统中得到最小测试用例集,在其他测试系统中测试用例集规模也与当前系统中最优的算法非常接近。在未列举出来的一些不常见的测试系统中 ZBDD 算法的表现同样出色。由此可知 ZBDD 测试用例生成算法在组合测试领域是可行的。从表 4 中可以看到,ZBDD 算法比 OBDD 算法节省了大量的节点开销,当 uncover 数量越多时,两者的差距越明显,在 $S4$ 和 $S5$ 系统中,OBDD 算法的节点数量已经超过百万级别,而本算法仍然维持在 3 万个节点以内。表 5 列出了 OBDD 算法和 ZBDD 算法的时间花费对比结果,结果表明 ZBDD 算法的耗时较多。

表 3　与各个常用算法生成的测试用例数量的比较

Table 3　Comparison of test case number generated by common algorithms

测试系统	ACTS	PICT	Jenny	FoCuS	ZBDD
*S*1	12	9	11	10	9
*S*2	21	18	18	20	20
*S*3	33	37	38	37	38
*S*4	28	27	28	30	28
*S*5	16	15	16	15	16
*S*6	11	10	12	10	10

表 4　OBDD 算法与 ZBDD 算法的节点开销比较

Table 4　Comparison of nodel overhead of OBDD algorithm and ZBDD algorithm

测试系统	Uncover 数量	OBDD	ZBDD
*S*1	6	119	66
*S*2	78	8755	794
*S*3	1830	993562	26102
*S*4	2775	—	28302
*S*5	4950	—	25141
*S*6	6	108	56

表 5　OBDD 算法与 ZBDD 算法的时间花费比较/s

Table 5　Comparison of time overhead of OBDD algorithm and ZBDD algorithm/s

测试系统	Uncover 数量	OBDD	ZBDD
*S*1	6	0.016	0.001
*S*2	78	0.032	0.047
*S*3	1830	2.108	5.556
*S*6	6	0.04	0.01

表 6　*S*6 测试系统经过 ZBDD 算法得出的测试用例集

Table 6　Test case generated by *S*6 test system through ZBDD algorithm

	OS	FlashVersion	CPU	Browser
1	XP	8	INTEL	FOX
2	XP	9	AMD	IE
3	OSX	10	AMD	FOX
4	LINUX	9	INTEL	FOX
5	LINUX	8	AMD	FOX
6	XP	10	INTEL	IE
7	OSX	8	INTEL	FOX
8	OSX	9	INTEL	FOX
9	LINUX	10	INTEL	FOX
10	XP	8	INTEL	IE

本文使用最常见的 6 个测试系统对程序算法进行测试：

*S*1：3^4（4 个参数，每个参数有 3 个取值）；

*S*2：3^{13}（13 个参数，每个参数有 3 个取值）；

*S*3：$4^{15}3^{17}2^{29}$（61 个参数，其中 15 个参数有 4 个取值，17 个参数有 3 个取值，29 个参数有两个取值）；

*S*4：$4^{1}3^{39}2^{35}$（75 个参数，其中 1 个参数有 4 个取值，39 个参数有 3 个取值，35 个参数有两个取值）；

*S*5：2^{100}（100 个参数，每个参数有 2 个取值）；

*S*6：$3^{2}2^{2}$（4 个参数，其中 2 个参数有 3 个取值，2 个参数有 2 个取值）。

*S*6 是一个简单的测试系统并且带有约束，它的 4 个参数分别是 OS，Browser，CPU 和 FlashVersion，对应的参数值分别是（XP，OS X，LINUX），（FOX，IE），（INEL，AMD）和（8，9，10）。约束：OS！＝“XP”＝＞Browser！＝“IE”。

结束语　本文研究了基于符号 ZBDD 的组合测试用例生成方法，给出了测试系统的 ZBDD 表示方法和基于 ZBDD 的组合测试用例生成算法。在此基础上进行了实验，通过与其他算法进行比较，验证了 ZBDD 算法的有效性。通过对比 OBDD 算法得出，ZBDD 算法的优势在于节点的开销大幅度地减少，虽然时间花费有一定的增加，但在合理的范围内。今后研究的重点是进一步开展基于符号 ZBDD 的组合测试用例生成算法的优化和改进，使其能够生成带权值参数的测试系统的测试用例集，并且在不改变节点开销和测试用例集生成规模的情况下尽量缩短时间。

参 考 文 献

[1] KUHN D R，REILLY M J. An Investigation of the Applicability of Design of Experiments to Software Testing[C]// Software Engineering Workshop，2002. NASA Goddard/IEEE，2002：91.

[2] DUNIETZL I S，MALLOWS C L，IANNINO A，et al. Applying Design of Experiments to Software Testing[C]// International Conference on Software Engineering. 1997：205-215.

[3] WILLAMS A W，PROBERT R L. A Practical Strategy for Testing Pair-wise Coverage of Network Interfaces[C]// International Symposium on Software Reliability Engineering，1996. IEEE，1996：246-254.

[4] COHEN M B，GIBBONS P B，MUGRIDGE W B，et al. Constructing Test Suites for Interaction Testing[C]// International Conference on Software Engineering. IEEE，2003：38-48.

[5] COHEN D M，DALAL S R，FREDMAN M L，et al. The AETG System：An Approach to Testing Based on Combinatorial Design [J]. IEEE Transactions on Software Engineering，1997，23(7)：437-444.

[6] COHEN D M，DALA S R，KAJLA A，et al. The Automatic Efficient Test Generator（AETG）system[C]// Proceeding of 5th International Symposium on Software Reliability Engineering，1994. 1994：303-309.

[7] COHEN D M，DALA S R，PARELIUS J，et al. The Combinatorial Design Approach to Automatic Test Generation[J]. IEEE Software，1996，13(5)：83-88.

[8] TAI K C，LIE Y. A Test Generation Strategy for Pairwise Testing[J]. IEEE Transactions on Software Engineering，2002，28(1)：109-111.

[9] KUHN R，LEI Y，KACKER R. Practical Combinatorial Testing：Beyond Pairwise[J]. It Professional，2008，10(3)：19-23.

[10] CZERWONKA J. Pairwise Testing in Real World Practical Extensions to Test Case Generators[OL]. http://core.ac.uk/display/20718771.

[11] SEGALL I，TZOREF-BRILL R，FARCHI E. Using binary decision diagrams for combinatorial test design[C]// International Symposium on Software Testing and Analysis（ISSTA 2011）. Toronto，Canada，2011：254-264.

[12] BRYAN R E. Symblic Boolean Manipulation with Ordered Binary Decision Diagram[J]. ACM Computing Surveys，1992，24(3)：293-318

[13] XU Z B, GU T L, ZHAO L Z. A Novel Symbolic ADD Algorithm for Maximum Flow in Networks [J]. Journal on Communications, 2005, 26(2): 1-8. (in Chinese)
徐周波, 古天龙, 赵岭忠. 网络最大流问题的一种新的符号 ADD 求解算法[J]. 通信学报, 2005, 26(2): 1-8.

[14] MINATO S. Zero-suppressed BDDs and Their Applications [J]. International Journal on Software Tools for Technology Transfer, 2001, 3(2): 156-170.

[15] LI F Y, GU T L, CHANG L, et al. Timed Petri and ZBDD Based Approach for Assembly Sequence Planning[J]. Computer Science, 2012, 39(2): 170-174. (in Chinese)
李凤英, 古天龙, 常亮, 等. 一种基于赋. Petri 网和 ZBDD 的装配序列规划方法[J]. 计算机科学, 2012, 39(2): 170-174.

[16] WANG Y, NIE C H, ZHANG S B. An Empirical Study of the Combinatorial Testing of Time-shifted TV Programs[J]. Computer Engineering and Science, 2015, 37(5): 958-966. (in Chinese)
王燕, 聂长海, 张少兵. 面向时移电视类业务的组合测试实践研究[J]. 计算机工程与科学, 2015, 37(5): 958-966.

[17] CHEN H, WANG S Y, PAN X Y. Pairwise Combinatorial Testing Suite Global Optimization and GenerationMethod[J]. Computer Engineering and Applications, 2012, 48(11): 32-36. (in Chinese)
陈皓, 王曙燕, 潘晓英. 成对组合测试数据的整体优化和生成方法[J]. 计算机工程与应用, 2012, 48(11): 32-36.

[18] GARGANTINI A. Efficient Combinatorial Test Generation based on Multivalued Decision Diagrams[M]// Hardware and Software: Verification and Testing. Springer International Publishing, 2014: 220-235.

[19] MINATO S I. Graph-Based Representations of Discrete Functions[M]// Representations of Discrete Functions. Springer US, 1996: 1-28.

[20] COEHN M B, GIBBONS P B, MUGRIDGE W B, et al. Constructing Test Suites for Interaction Testing[C]// International Conference on Software Engineering. IEEE, 2003: 38-48.

[21] NIE C, LEUNG H. A survey of combinatorial testing[J]. Acm Computing Surveys, 2011, 43(2): 33-63.

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

[23] SPICHKOVA M, ZAMANSKY A. A Human-Centred Framework for Combinatorial Test Design[C]// International Conference on Evaluation of Novel Approaches To Software Engineering. 2016.

[24] CHENG C, DUMITRESCU A, SCHROEDER P. Generating Small Combinatorial Test Suites to Cover Input-output Relationships[C]// International Conference on Quality Software. 2004: 76-82.

[25] KUHN D R, KACKER R N, LEI Y. Practical Combinatorial Testing[OL]. http://permanent. access. gpo. gov/gpo14815/get-pdf. pdf.

[26] SAHL A N, ALSARIERA Y A, ZAMLI K. Late Acceptance Hill Climbing Based Strategy for Addressing Constrain Within Combinatorial Test Data Generation[C]// The 7th edition of Annual Software Testing Conference, 2014. 2014.

[27] KUHN D R, KACKER R N, YU L. Measuring and specifying combinatorial coverage of test input configurations[J]. Innovations in Systems & Software Engineering, 2015, 12(4): 1-13.

(上接第 244 页)

[3] CHEN P, XIAO H, SHEN X, et al. DROP: Detecting Return-Oriented Programming Malicious Code[C]// Information Systems Security, International Conference (ICISS 2009). Kolkata, India, 2009: 163-177.

[4] DAVI L, SADEGHI A R, WINANDY M. ROPdefender: a detection tool to defend against return-oriented programming attacks [C]// ACM Symposium on Information Computer & Communication Security Cited on. 2011: 40-51.

[5] ONARLIOGLU K, BILGE L, LANZI A, et al. G-Free: defeating return-oriented programming through gadget-less binaries[C]// Twenty-Sixth Computer Security Applications Conference (ACSAC 2010). Austin, Texas, USA, 2010: 49-58.

[6] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Transparent ROP exploit mitigation using indirect branch tracing[C]// Usenix Conference on Security. 2013: 447-462.

[7] CHENG Y, ZHOU Z, YU M, et al. ROPecker: A Generic and Practical Approach For Defending Against ROP Attacks[C]// Network and Distributed System Security Symposium (NDSS14). 2014.

[8] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns[C]// ACM Conference on Computer and Communications Security (CCS 2010). Chicago, Illinois, USA, 2010: 559-572.

[9] BLETSCH T, JIANG X, FREEH V W, et al. Jump-oriented programming: a new class of code-reuse attack[C]// ACM Symposium on Information, Computer and Communications Security (ASIACCS 2011). Hong Kong, China, 2011: 303-307.

[10] CARLINI N, WAGNER D. ROP is still dangerous: breaking modern defenses[C]// USENIX Conference on Security Symposium. USENIX Association, 2014: 385-399.

[11] Windows ISV Software Security Defenses[EB/OL]. (2010-12-01) [2015-01-30]. https://msdn. microsoft. com/en-us/library/bb430720. aspx.

[12] SCHWARTZ E J, AVGERINOS T, BRUMLEY D. Q: exploit hardening made easy[C]// Usenix Conference on Security. USENIX Association, 2011: 25-25.