第 36 卷 第 1 期 计 算 机 工 程 2010 年 1 月
Vol.36 No.1 Computer Engineering January 2010
·软件技术与数据库· 文章编号：1000—3428(2010)01—0045—04 文献标识码：A 中图分类号：TP311.52

基于冗余测试用例的最小测试用例集生成方法

郭晶晶，高建华
(上海师范大学计算机应用技术系，上海 200234)

摘 要：提出一种最小测试用例集的生成方法。保留在某个测试标准下冗余，但在其他测试标准下不冗余的测试用例，即在测试用例集简化期间通过有选择性地保留测试用例来生成一个测试用例集。与已有方法相比，该方法能在不影响测试组大小范围的情况下有效提高错误检测效率。

关键词：软件测试；测试标准；测试用例集最小化

Generation Method of Minimal Test Suite Based on Retaining Redundant Test Cases

GUO Jing-jing, GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234)

【Abstract】 This paper proposes a minimal test suite generation method. It presents a new method for test suite reduction that attempts to use addition coverage information of test cases to selectively keep some additional test cases in the reduced suites that are redundant with respect to the test criteria used for test suite minimization. It implements the method by modifying an existing heuristic for test suite minimization. Comparing with the existing methods, it can improve the fault detection capability of reduced test suites without severely affecting the extent of suite reduction.

【Key words】 software test; test criteria; test suite minimization

1 概述

在进行软件测试时，必须先根据软件需求分析、设计说明和编码等软件设计过程确定测试目标，即测试需求的集合，根据这些测试需求可以构造出一组测试用例。这组测试用例的数量和质量将决定软件测试的成本和有效性。

对每个测试需求一般都要产生相应的测试用例，以实现对这个测试需求的充分测试。这样产生的测试用例集数量比较大，而且可能有较大冗余，即这组测试用例的某些子集也能满足所有的测试需求。目前，人们解决这个问题的一般方法是：首先根据测试目标中的每个测试需求确定出相应的测试用例，所有这些测试用例组成初步的满足测试目标的测试用例集；然后针对这个测试用例集采用贪心算法、启发式算法或整数规划等方法进行精简，去掉一些冗余的测试用例[1]。这种方法的缺点在于它的效果取决于最初产生的测试用例集，因此，最初选定的测试用例集，不能从根本上实现根据测试目标对测试用例集的整体优化。

本文利用冗余测试用例[2]，运用多重测试标准，这些测试标准对应于多种类型的测试需求，不同的测试标准对于识别一个程序中运用不同结构和功能的测试用例是很有用的。因此，该方法的关键性步骤是：当一个测试用例 t 在测试覆盖标准 c 下满足了一些测试需求从而被选入简化用例集。

对由于 t 的选取而关于 c 冗余的测试用例 r，只要 r 在另外测试覆盖标准下也满足了其测试需求，就把 r 也选入到简化用例集。因此，必须有选择性地保留了这些关于一种测试标准时冗余，而去除关于另一些测试标准时不冗余的测试用例。

2 相关工作

2.1 测试用例最小化

测试用例最小化问题可以简单的描述为：给出一个测试用例集 $T=\{t_1, t_2,\cdots, t_m\}$，一系列测试需求 $R=\{r_1, r_2,\cdots, r_n\}$，这些测试需求必须得到满足从而提供预期的测试覆盖标准，即测试用例集 T 的子集 $\{T_1,T_2,\cdots,T_n\}$，任何一个属于 T 的子集 t_j 都满足 r_i，注意 i 不一定等于 j。

测试用例集最小化的定义：在原始用例集中，找到一个最小的测试用例子集，并能够提供跟原始测试用例一样的测试覆盖率。

2.2 测试用例最小化技术

4 个属性评价最小化技术为：(1)充分性(Adequacy)：最小化测试用例集必须能够保持与原始用例集相同的测试覆盖度；(2)精确性(Precise)：能最大限度剔除冗余用例，缩减用例集大小；(3)效益(Cost-effectiveness)：用于最小化的费用(即运行最小化算法得到最小化用例集的费用)应小于由于使用最小化用例集进行回归测试缩减下来的费用，即要求算法花费合理的代价(主要考虑时间效率)得到最小用例集合；(4)通用性(Generality)：使用于不同程序、不同的测试覆盖标准等。

研究测试用例最小化技术，充分性是前提，精确性和效益是关键，通用性是目的。在使用以上评价体系时，本文基

基金项目：国家自然科学基金资助项目(60673067)
作者简介：郭晶晶(1988－)，女，硕士研究生，主研方向：软件测试；高建华，教授、博士
收稿日期：2009-06-04 **E-mail**：jguo_happy@163.com

于以下假设：测试用例的测试运行代价是一个常量，与是否使用测试用例缩减技术无关。

3 现有方法及其存在问题分析

对于一个待测软件系统，如果它的测试目标是由 m 个测试需求组成的集合 R，可设 $R=\{r_1, r_2,\cdots, r_m\}$。一般人们根据各个测试需求，产生相应的 n 个测试用例 $T=\{t_1, t_2,\cdots, t_n\}$。目前，对于测试用例集 T 的简化算法主要有贪心算法、启发式算法和整数规划的方法。

贪心算法[3]为了把测试需求集 R 对应的测试用例集 T 进行精简，每次从 T 中挑选一个测试用例，使之能最多地满足所有未被满足的测试需求，然后在测试需求集中去掉已经被满足的测试需求，直到所有测试需求都被满足，停止从 T 中挑选测试用例。这时挑选出来的测试用例组成的集合就是精简后的测试用例集合。贪心算法的最坏时间复杂度是 $O(mn\cdot\min(m, n))$。

文献[4]提出一种根据测试用例的重要性来选择测试用例的启发式方法(H 算法)。该算法将测试需求 $r_1, r_2,\cdots, r_m$ 分别划分到集合 $R_1, R_2,\cdots, R_d$ 中。其中，$R_i(i=1, 2,\cdots, d)$包含了所有正好可以被 T 中 i 条测试用例满足的测试需求。如果 $i<j$，则 H 算法认为满足 R_i 中测试需求的测试用例比满足 R_j 中测试需求的测试用例要“重要”。因此，H 算法首先选出满足 R_1 中测试需求的测试用例，然后再考虑 R_2，使用贪心算法选择测试用例，直到 R_2 中的测试需求全部被满足。依次处理 $R_3, R_4,\cdots, R_d$。该算法的最坏时间复杂度为 $O(m(m+n)d)$。

在贪心算法和文献[4]提出的启发式算法的基础上，文献[5]提出一种将这 2 种方法进行有机结合的新方法。这种方法首先选出必不可少的重要测试用例，将这些测试用例所满足的测试需求从集合 R 中去掉，然后利用贪心算法选出能最多的满足未被满足的测试需求的测试用例，并将相应的测试需求从 R 中去掉，这样直到所有的测试需求都被满足，即 $R=A$ 时结束。这种算法最坏的时间复杂度是 $O(mn+\min(m, n)nk)$，其中，k 表示一个测试用例最多能满足的测试需求数量。

文献[6]提出的测试用例选择方法把测试用例选择问题转化为整数规划问题，利用整数规划方法求出最优解，理论上可获得满足测试需求集 R 的最小测试用例集，但其计算复杂程度高，运算开销呈指数级增长。

以上这 4 种最小测试用例集生成方法算法各有特点，目前通过实验可证明，任何一种算法都不比其他算法更具优越性。各算法的主要特点及优缺点的比较分析如表 1 所示。

表 1 最小测试用例集生成方法的比较

方法	主要特点	优点	缺点
贪心算法	(1)用于解决最优化问题 (2)通过一系列贪心选择得到一个问题的解 (3)模仿人脑思考方式只针对当前情况选择最优解 (4)所求问题会简化为小规模的子问题	(1)编程简单 (2)容易理解 (3)运行效率高 (4)空间复杂度低 (5)可快速获得小范围内的最优解	(1)不能着眼于全局 (2)存在资源超标隐患
启发式算法	(1)在可接受的费用内寻找最好的解的技术 (2)解决问题时强调“满意”，不去苛求最优性和最优解 (3)求解问题时通过迭代过程实现	(1)计算步骤简单，易于实施 (2)避免陷入局部最优解 (3)易于将定量分析与定性分析结合	要快速收敛于最优解则对初始路径的设定有一定要求 不一定能保证所得解的可行性和最优性 无法阐述所得解同最优解的近似程度
整数规划	要求一部分或全部决策变量必须取整数值	理论上可获得最优解	(1)计算复杂度高 (2)运算开销大

4 最小测试用例集生成方法

本文提出一种最小测试用例集的生成方法，即保留冗余测试用例法，它充分考虑了不同测试标准对最小测试用例集的影响，通过保留在一种测试标准下冗余但在另一种测试标准下不冗余的测试用例来生成最小测试用例集的方法。

保留冗余测试用例法的主要思想如下：(1)测试用例集 T 中的每一个测试用例根据一个测试标准 C 被选入最小测试用例集。(2)再用其他的测试标准选取冗余的测试用例，这些测试用例关于 C 是冗余的，但关于其他标准是不冗余的。

ReduceWithSelectiveRedundancy 算法[2]给出了保留冗余测试用例法的具体步骤，算法中输入部分是测试用例的一个 T 集合以及被每一个测试用例按照至少 2 个不同测试标准所满足的测试需求。输出部分是一个简化后的测试用例集 RS，这个测试用例集满足所有被原来初始用例集 T 满足的测试需求。最初，RS 集是空的，并且把与每个标准有关的测试需求做出未标记状态。

该算法的具体步骤如下：

(1)初始化。所有需求均作未标记状态。同时对于每个测试用例算法保留了被这些用例满足的未做标记的测试需求的数目。初始化之后，算法中主要的循环开始一个接一个不断地递增选取测试用例到简化用例集，在基数有序增加的相关的测试用例集中循环考虑了对应于第一个标准 C_1 的未作标记的需求。

(2)用第一个测试标准选取下一个测试用例。所有在未标记的测试用例集中呈现的测试用例的当前基数被标识出来。算法中 selectTest 函数用来选取一个测试用例，这些用例满足最多未做标记的需求，这些需求的测试用例集是当前的基数，并把它加入到简化用例集中。在连接事件中，满足最多的未做标记的需求，这些需求中测试用例集是基数顺序增大的测试用例被选取。如果基数达到最大值，连接被任意破坏。对于每个测试标准，被选取的测试用例满足的未做标记的需求均做上标记。而且，把对于第一个测试标准时变的冗余的测试用例加入到一个冗余测试用例集中。

(3)从冗余的测试用例中做出选择。在关于 C_1 冗余的测试用例中，selectRedundentTests 函数被用来选取冗余测试用例。这些测试用例按照第 2 个标准的额外覆盖的递减顺序排列，并把它们加入到简化用例集中。最新满足的需求做上标记。算法循环地尝试运用剩下的测试标准选取冗余的测试用例。选取完冗余的测试用例后，重复进行第 2 步和第 3 步，直到所有的测试需求被标记。

ReduceWithSelectiveRedundancy 算法代码如下：

输入 T：原测试用例集
R：所有测试需求的集合，$\{r_1,r_2,\cdots,r_n\}$
T_i：测试用例集 T 的子集合，$i=1,2,\cdots,n$
T_i^c：分别覆盖了每个需求 r_i^c 的测试用例集
C_k：测试标准，$1\leqslant c\leqslant k$
输出 RS：精简后的测试用例集

```
Begin:                          //第 1 步：初始化
redundant=Φ,RS=Φ,curcard=0,
maxcard=max(T_i^1);
For each c,所有需求 r_i^c 不作标记;
For each t and c do
numUnmarked^c[t]=number of T_i^c containing t;
end for
loop   //循环开始
```

```
curcard=curcard+1;
//第 2 步：用第一个测试标准选取一个测试用例
while (存在 r_i^1 未作标记时)
list:=在 T_i^1 中的 curcard 未作标记；
nestTest=SelectTest(curcard,list,maxcard);
//嵌套调用 SelectTest 函数
RS=RS∪{nextTest};mayreduce=false;
For each T_i^1 包含 nestTest 满足 r_i^1 是未作标记的 do mark r_i^1
        For each t in T_i^1 do
          numUnmarked^1[t]= numUnmarked^1[t]-1;
          if numUnmarked^1[t]==0 and t∈RS then
          redundant=redundant∪{t};
end for
if cardinality of T_i^1==maxcard then mayreduce=TRUE;
end for
for each c (2≤c≤k) do
for each T_i^c containing nestTest s.t r_i^c is unmarked do mark r_i^c;
for each t in r_i^c do numUnmarked^c[t]= numUnmarked^c[t]-1;
      end for
end for
SelectRedundantTests(RS, redundant, numUnmarked^c:1≤c≤k, 2)
//第 3 步：从冗余的测试用例中作出选择
        redundant=Φ;
//嵌套调用 SelectRedundantTests 函数
        if mayreduce then maxcard=max cardinality of T_i^1 s.t r_i^1
is unmarked;
endwhile
until curcard==maxcard;
end ReduceWithSelectiveRedundancyHGS //循环结束
function SelectTest(size, list, maxcard)
// SelectTest 函数根据测试标准 1 选取测试用例到简化组
        for each t in list do
        count[t], testlist;
        if cardinality of testlist==1 then
            return the test in testlist;
else if size==maxcard then
          return any test in testlist;
else
          return SelectTest(size+1, testlist, maxcard)
endif
end SelsctTest
function SelectRedundantTests(RS, redundant, numUnmarked^i:
1≤I≤k, c)//SelectRedundantTests 函数从冗余测试用例中作出选择
addcoverage[t]=0;
for each t in redundant do addcoverage[t]= numUnmarked^c[t];
while(冗余用例 t 满足 addcoverage[t]＞0)
toadd; RS=RS∪{toadd}; redundant= redundant-{toadd};
redundantAgain=Φ;
for each m(c≤m≤k) do
    for each T_i^m containing toadd s.t. r_i^m is unmarked do mark r_i^m;
        for each t in T_i^m do
numUnmarked^m[t]= numUnmarked^m[t]-1;
if m==c and numUnmarked^m[t] ==0 and t ∈RS then
redundantAgain= redundantAgain∪{t};
endif
            endfor
        endfor
endfor
redundant=redundant- redundantAgain;
if c<k then
SelectRedundantTests(RS, redundant, numUnmarked^i: 1≤i≤k,
c+1);
Endif
addcoverage[t]=0 for all t;
for each t in redundant do addcoverage[t]= numUnmarked^c[t];
endwhile
end SelectRedundantTests
```

5 实例研究

本节通过一个实例演示了基于保留冗余测试用例的测试用例集约简方法的计算过程，该方法是在对现有的启发式算法改进基础之上进行的。

针对函数 Input()所对应的测试需求 r_1～r_4，本文给出该函数的程序控制流程如图 1 所示，其中，*S*1, *S*2, *S*3, *S*4, *S*5, *S*6, *S*7 分别代表该函数的语句。

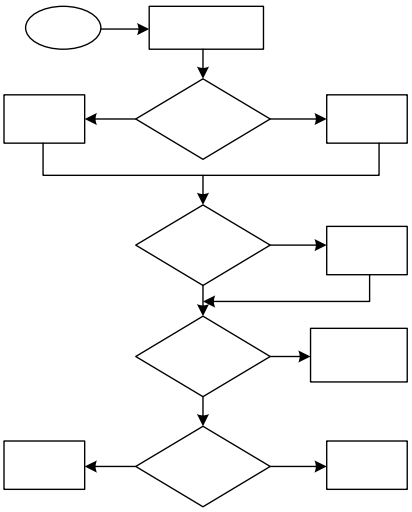

图 1　函数 Input()的控制流程

给出一组可以满足该测试需求的测试用例集 *T* 如下：

t_1: (*A*=1, *B*=1, *C*=-1, *D*=0)

t_2: (*A*=-1, *B*=-1, *C*=1, *D*=-1)

t_3: (*A*=-1, *B*=1, *C*=-1, *D*=0)

t_4: (*A*=-1, *B*=1, *C*=1, *D*=1)

t_5: (*A*=-1, *B*=-1, *C*=1, *D*=1)

分支覆盖标准下测试用例 t_1～t_5 与测试需求 r_1～r_4 的满足关系如表 2 所示。在表中运用由 Harrold 发明的启发式算法同时采用分支覆盖标准给出测试用例集最小化的结果。给出一个测试用例集 *T* 和一系列测试需求 $r_1, r_2, \cdots, r_n$ 来对程序提供预期的测试覆盖，启发式算法认为 T 的子集合 $T_1, T_2, \cdots, T_n$ 就像任何一个属于测试用例集 T_i 都可被用来测试 r_i。

表 2　分支覆盖标准下测试用例 t_1～t_5 与测试需求 r_1～r_4 的满足关系

	r_1^T	r_1^F	r_2^T	r_2^F	r_3^T	r_3^F	r_4^T	r_4^F
t_1			X					
t_2		X		X	X	X		
t_3	X	X						X
t_4		X	X		X	X	X	
t_5		X		X	X		X	

分支覆盖的原理是设计若干个测试用例，使得程序中每个分支的取真和取假分支至少各执行一次。因此，分值覆盖就是考虑判定条件的取真分支和取假分支，下面就结合表 2 采用分支覆盖标准来选出一组满足全部测试需求的最小测试用例集。

首先看测试用例集 *T* 中的第 1 组测试用例 t_1: (*A*=1, *B*=1,

C=-1, D=0)，用 t_1 来验证一下所有的测试需求，把它所满足的取真和取假分支在表 2 中用“X”标记出来，然后再看测试用例 t_2, t_3, t_4, t_5，同样的方法在表 2 中用“X”做标记。最初，因为分支 r_1^T 和 r_4^F 仅被测试用例 t_1, t_2 满足，测试用例 t_1, t_2 分别被选入最小化用例集。接下来，所有被 t_1 和 t_2 满足的分支都被标记覆盖，结果，测试用例 t_2 关于分支覆盖变的冗余，因为它所有的分支已经被标记为覆盖了。现在，t_4 或 t_5 以被选取来覆盖剩余的分支 r_4^T，把 t_4 选入最小用例集，然后分支 r_4^T 被标记覆盖，根据分支覆盖导致 t_5 冗余。因为所有的测试需求被最小用例集{t_1, t_2, t_4}中的测试用例覆盖所以算法终止。注意到测试用例 t_3 在判定条件 C>0 的取假分支中暴露了被 0 除的错误不能被选入最小用例集。因此，测试组的错误检测效率由于测试用例集最小化而降低了。

下面运用以上的例子来提出本文的方法：有选择性地保留在一种测试标准下冗余，但在另一种测试标准下不冗余的测试用例。对以上的函数例子，再给出一个测试标准，即语句覆盖标准。通过表 3 把测试用例集 T 中的所有测试用例在该标准下的覆盖信息展示出来。新方法就是建立在分支覆盖与语句覆盖相结合的基础上。在每个测试用例 t_i 被选取之后通过插入如下检验修改启发式算法：如果任何测试用例 t_j 在分支覆盖下冗余但只要 t_j 在语句覆盖标准不冗余，也把 t_j 选入最小测试用例集中。

表 3　语句覆盖标准下测试用例 t_1～t_5 与测试需求 r_1～r_4 的满足关系

	x(2,4)	x(3,4)	x(3,5)	y(4,7)	A(1,r_1)	B(1,r_2)	C(1,r_3)	D(1,r_4)
t_1					X	X	X	
t_2				X	X	X	X	X
t_3	X				X	X	X	
t_4		X	X	X	X	X	X	X
t_5		X	X		X	X	X	X

语句覆盖的原理：设计若干测试用例，使程序中的每条语句至少执行一次。在刚才的启发式算法中，已经得出关于分支覆盖下冗余的测试用例为{t_3, t_5}，下面研究在语句覆盖标准下这两个测试用例中哪个可以保留。

上例中，当 t_1 和 t_2 被启发式算法选入最小测试用例集之后，t_3 关于分支覆盖标准被识别为是冗余的。但是在语句覆盖标准下，t_3 覆盖了测试需求与 x 有关的语句 3 和语句 4，即 x(3, 4)，这个既不被 t_1 满足，也不被 t_2 满足。因此，t_3 符合新方法的要求，t_3 被保留。接下来，t_4 或 t_5 可以被启发式算法选取来覆盖分支 r_4^T，t_4 被保留。这样，测试用例 t_5 关于分支覆盖和语句覆盖标准都变的冗余。因此，t_5 不被选取。到此，所有的取真和取假分支及语句都被标记为覆盖，即测试需求均得到满足，算法终止。计算出来的最小测试用例集为{t_1, t_2, t_3, t_4}，在判定条件 C>0 的取假分支中暴露了被 0 除的错误。

6　结束语

本文分析了测试用例集简化的现有方法的缺点，提出了在测试用例集简化期间通过有选择性的保留测试用例来生成最小化测试用例集的新方法。新方法认为在测试用例集最小化期间考虑多种测试标准比仅考虑一个标准更有效。该方法具有通用性，可以和多种现有的基于工作表的用例集最小化技术结合起来，而且具有开放性，可将该方法与 Hermdall 和 George 提出的基于模型测试的测试用例集简化技术合并获得从软件的正规规约说明中导出的测试需求。应用新方法可以提高软件的错误检测效率降低软件测试成本。

参考文献

[1] Lee J G, Chung C G. An Optimal Representative Set Selection Method[J]. Information and Software Technology, 2000, 42(1): 17-25.

[2] Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction[J]. IEEE Transactions on Software Engineering and Methodology, 2007, 33(2): 108-123.

[3] Johnson D S. Approximation Algorithms for Combinatorial Problems[J]. Journal of Computer and System Sciences, 1974, 9(3): 256-278.

[4] Harrold M J, Gupta R, Soffa M L. A Methodology for Controlling the Size of a Test Suite[J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285.

[5] Chen Tsong Yueh, Lau M F. Heuristics Towards the Optimization of the Size of a Test Suite[C]//Proc. of the 3rd International Conference on Software Quality Management. Seville, Espagne: [s. n.], 1995: 415-424.

[6] Chen Tsong Yueh, Lau M F. Dividing Strategies for the Optimization of a Test Suite[J]. Information Processing Letters, 1996, 60(3): 135-141.

编辑　金胡考

（上接第 44 页）

护注册表的目的。只要对所要保护的某些路径结合成相关的规则，就可以对不同的路径的键进行保护。本文实验所编写的程序在使用 Windows Mobile 6 操作系统的多普达 S1 智能手机上测试通过，可以实现对指定路径的注册表键值的保护，实现功能包括禁止打开、禁止修改、禁止删除、禁止添加内容等。

4　结束语

本文阐述了 Windows Mobile 系统中注册表保护的原理，并实现了 Windows Mobile 注册表保护的系统。实验证明，该系统性能良好，实用性强，提高了系统注册表的安全性，从而提高了 Windows Mobile 系统的安全性。

参考文献

[1] 刘彦博, 胡　砚, 马　骐. Windows Mobile 平台应用与开发[M]. 北京: 人民邮电出版社, 2006.

[2] 张建畅, 陶会荣, 王建超, 等. 基于 WinCE 的嵌入式系统注册表的研究[J]. 微计算机信息, 2008, 24(14): 44-46.

[3] 于代国, 孙建孟. Win32 钩子技术及应用[J]. 微计算机应用, 2004, 25(4): 508-511.

[4] Microsoft Corporation. Microsoft Mobile 6 SDK Help[EB/OL]. (2008-10-16). http://msdn.microsoft.com/en-us/library/bb158486.aspx.

[5] Murray J. Inside Microsoft Windows CE[M]. [S. l.]: Microsoft Press, 1998.

[6] Leman D. Spy: A Windows CE API Interceptor[J]. Dr Dobb's Journal, 2003, 28(10): 54-59.

编辑　任吉慧