一种最小测试用例集生成方法

聂长海 徐宝文

(东南大学计算机科学与工程系 南京 210096)

(江苏省软件质量研究所 南京 210096)

摘 要 测试用例的数量和质量决定软件测试的成本和有效性.该文提出了一种生成最小测试用例集的方法,该方法首先充分考虑测试目标中各个测试需求之间的相互关系,将满足测试需求的所有可用测试用例进行划分,根据划分的结果生成一个测试用例集,然后利用启发式算法、贪心算法或整数规划方法来消除冗余,对这个测试用例集进行进一步的简化.这种方法与已有方法相比,优点在于它可以生成满足所有测试需求的最小测试用例集.

关键词 软件测试;测试用例;测试用例集约简

中图法分类号 TP311

A Minimal Test Suite Generation Method

NIE Chang-Hai XU Bao-Wen

(*Department of Computer Science and Engineering, Southeast University, Nanjing* 210096)

(*Jiangsu Institute of Software Quality, Nanjing* 210096)

Abstract The cost and effectiveness of software testing is determined by the quantity and quality of the test suite. This paper proposes a minimal test suite generation method. It gives a partition to the set of all the applicable test cases at first on basis of the interrelations among the testing requirements, and then generates a test suite by the partition, at last a minimal test suite is obtained by reduction with the methods of integer programming, heuristic algorithm or greedy algorithm. Compared with the existed methods, our method has better property that can generate the minimal test suite to test all the test requirements sufficiently.

Keywords software testing; test case; test suite reduction

1 引 言

在进行软件测试的时候,首先必须根据软件需求分析、设计说明和编码等确定测试目标,即测试需求的集合,根据这些测试需求可以精心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.

对于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一般数量比较大,而且可能有比较大的冗余,即这组测试用例的某些子集也能满足所有的测试需求.由于运行和维护这些测试用例以及回归测试将耗费大量的时间、人力和物力,软件测试的成本太大,因此人们一直在研究如何设计出一组有效、数量少又能充分满足所有测试需求的测试用例集,从而在保证和提高软件测试质量的同时,降低软件测试的成本.

收稿日期:2002-09-26;修改稿收到日期:2003-04-06.本课题得到国家自然科学基金(60073012)、江苏省科技攻关项目(BE2001025)、江苏省三三三人才基金、武汉大学软件工程国家重点实验室开放基金、江苏省计算机信息处理技术重点实验室(苏州大学)开发基金资助.**聂长海**,男,1971 年生,讲师,博士研究生,主要从事软件工程和软件测试技术、模糊信息处理、神经网络等方面的教学与科研工作.**徐宝文**,男,1961 年生,博士,教授,主要研究领域为程序设计语言、软件工程、并行与网络软件等.E-mail:bwxu@seu.edu.cn.

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、一些启发式算法或整数规划等方法来进行精简,去掉一些冗余的测试用例[1~3]. 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,因最初选定的测试用例集的不同而不同,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.

我们提出的最小测试用例集的生成方法首先对测试目标中的所有测试需求进行整体优化. 由于每个测试需求对应一个可用测试用例集,这个集合中的每个测试用例都能满足该测试需求. 如果我们一开始在选择测试用例时尽可能地考虑在这些集合的交集中选择,即每选择一个测试用例,保证它尽可能多地满足各个测试需求,这样得到的测试用例集不仅能满足测试目标,而且数量少. 所以,对测试目标中的所有测试需求进行整体优化,实际上就是将满足测试目标的所有可用测试用例组成的集合依据各个测试需求之间的相互关系进行划分,其中的每一个等价类是能同时满足某一个或多个测试需求的测试用例组成的子集,然后利用贪心算法、一些启发式算法或整数规划方法,就可以直接生成一个最小测试用例集,这个测试用例集是满足这个测试目标中所有测试需求的不能再小的测试用例集.

本文第 2 节介绍涉及到的有关概念、符号和术语;第 3 节介绍与这个问题相关的现有的研究方法和结果,并分析了存在的问题;第 4 节介绍我们提出的方法;第 5 节通过一个实例把提出的方法与相关方法进行比较;最后是对全文的总结.

2　基本概念和术语

用 R 和 T 分别表示所有的测试需求集和测试用例集,并假定这两个集合都是非空有限集. 对于测试需求集 R 中每个测试需求 $r(r\in R)$,在测试用例集 T 中都存在测试用例 $t(t\in T)$ 满足这个测试需求. 用 m 和 n 分别表示测试需求集 R 和测试用例集 T 的基数,即 $|R|=m$,$|T|=n$.

测试用例 $t\in T$ 与测试需求 $r\in R$ 的满足关系用从 R 到 T 的二元关系 $S(R,T)$ 来表示,即 $S(R,T)=\{(t,r)\in T\times R$:测试用例 t 满足测试需求 $r\}$,其中 $S(R,T)$ 在不引起混淆的情况下可以简写为 S. 对于任何测试用例 $t\in T$ 或测试用例集 $T'(\neq\varnothing)\subseteq T$,我们用 $Req(t)$ 和 $Req(T')$ 分别表示所有被测试用例 t 和测试用例集 T' 所满足的测试需求所组成的集合. 类似的,$Test(r)$ 和 $Test(R')$ 分别表示所有满足测试需求 $r\in R$ 和测试需求集 $R'(\neq\varnothing)\subseteq R$ 的测试用例组成的集合.

定义 1.　若 $T_1\subseteq T$,且 $Req(T_1)=R$,则称 T_1 为关系 S 的代表集;如果对于任意代表集 T_2 有 $|T_1|\leq|T_2|$,则称 T_1 是 S 的优化代表集.

用 $REP(S)$ 和 $OPT\text{-}REP(S)$ 分别表示关系 S 的所有代表集和优化代表集组成的集合. 显然,T 是 S 的一个代表集,S 的优化代表集一定存在. 虽然它的优化代表集可以不唯一,但基数一定是相同的.

在测试用例集中有两类测试用例,一类是必不可少的测试用例,一类是冗余的测试用例[1,4].

定义 2.　对于测试用例 $t\in T$,如果 $Req(T\backslash\{t\})\neq R$,则称 t 是不可缺少的. 用 $ESS(S)$ 表示 S 中所有必不可少的测试用例组成的集合,在不引起混淆的情况下可简写为 ESS. 相反地,如果 $Req(T\backslash\{t\})=R$,则称 t 是冗余的. 对于测试用例 $t_{1\text{-}1}\in T$,如果存在 $t\in T$ 使得 $Req(t_{1\text{-}1})\subseteq Req(t)$,则称测试用例 $t_{1\text{-}1}$ 是 1-1 冗余的.

相应地,我们给出最小测试用例集的概念如下.

定义 3.　测试用例集 T 是满足测试需求集 R 的最小测试用例集,如果 T 的任意真子集 T_1 都不能实现对测试需求集 R 的充分测试,即 $Req(T)=R$,但对 $\forall T_1\subset T$,$Req(T_1)\neq R$.

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,不妨设 $R=\{r_1,r_2,\cdots,r_m\}$. 人们一般都是根据各个测试需求,产生相应的 n 个测试用例,然后在这些测试用例形成的测试用例集 T 的基础上运用各种简化算法,得到简化的测试用例集. 目前,对于测试用例集 T 的简化算法主要是贪心算法、一些启发式算法和整数规划的方法.

贪心算法为了把测试需求集 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 $O(mn\cdot\min(m,n))$[5].

Harrold[6]等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,这种方法首先将测试需求 $r_1,r_2,\cdots,r_m$ 分成 $R_1,R_2,\cdots,R_d$. 这里 $R_i(i=1,2,\cdots,d)$ 表示所有正好可以被 T 中 i 个测试用例满足的测试需求. 其中 $d=\max\{|Test(r)|;r\in R\}$ 表示一个测试需求最多可以被 d 个测试用例满足. 如果 $i<j$,这种方法认为满足 R_i 中的测试需求的测试用例比满足 R_j 中测试需求的测试用例"重要",所以这种方法首先挑选出满足 R_1 中测试需求的测试用例,然后将被这些测试用例满足的测试需求从 R 中去掉,再考虑 R_2 中未被满足的测试需求,从 $Test(R_2)$ 中选择测试用例使之尽可能多地满足剩下的测试需求,直到 R_2 中的测试需求全部被满足,然后 $R_3,R_4\cdots$ 直到 R_d. 这种方法最坏的时间复杂度是 $O(m(m+n)d)$.

在贪心算法和 Harrold 等提出的启发式算法的基础上,人们又提出一种将这两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,把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地满足未被满足的测试需求的测试用例,并把相应的测试需求从 R 中去掉;直到所有的测试需求都被满足,即 $R=\varnothing$ 时结束. 这种算法最坏的时间复杂度是 $O(mn+\min(m,n)nk)$,这里 k 表示一个测试用例最多能满足的测试需求的数量.

Chen 和 Lau[1,7]在以上算法的基础上提出一个新的测试用例集简化的启发式算法,在这个算法中不仅结合了贪心算法、Harrold 等提出的启发式算法的优点,而且充分考虑了剔除 1-1 冗余策略. 这个算法首先循环地执行:从 T 中挑选出必不可少的测试用例,从 R 中去掉相应的测试需求;然后在 T 中剔除 1-1 冗余测试用例(这一步可能产生新的必不可少测试用例);最后如果 $R\neq\varnothing$,即 R 中还有未被满足的测试需求时,利用贪心算法从 T 中选择测试用例满足剩下的测试需求. 这个算法的最坏时间复杂度是 $O(\min(m,n)(m+n^2k))$[4~6].

以上这 4 种算法各有特点,已经证实任何一种算法都不比其他算法优越. Chen 和 Lau 在文献[2]中用仿真方法研究这 4 种算法的性能,给实际使用这几个算法提供了一些参考和指导,不过这些方法不能保证产生的结果是最优的. Lee 和 Chung 提出了一种新的测试用例选择方法,可以保证选出的测试用例集是原测试用例集最优的简化. 这种方法把原问题转化为整数规划问题,利用整数规划方法求出最优解[3].

这几种算法都是对测试用例集的简化策略,而测试用例集是根据测试目标中的测试需求来确定的,所以决定算法实际效果的除了算法本身外就是它们所处理的测试用例集. 例如我们考虑一个最简单的例子,假如测试需求集 R 中有两个测试需求 r_1,r_2,即 $R=\{r_1,r_2\}$. 设所有能满足测试需求 r_i 的测试用例组成的集合为 T_i,对应测试需求 r_i 选择测试用例 t_i,其中 $t_i\in T_i(i=1,2)$. 如果 $T_1\cap T_2=\varnothing$ 或者选择的测试用例 t_1 只满足 r_1,测试用例 t_2 只满足 r_2,即 $t_1\in T_1-T_2$,而 $t_2\in T_2-T_1$,那么相应的测试用例集 $T=\{t_1,t_2\}$ 利用任何算法简化后仍然还是它自身. 而如果 $T_1\cap T_2\neq\varnothing$,且 T 中任何一个测试用例能同时满足两个测试需求,即 $t_1\in T_1\cap T_2$ 或者 $t_2\in T_1\cap T_2$,那么利用上述算法,这个测试用例集就可以简化为只剩下一个测试用例 t_1 或者 t_2.

由此可见,如果我们只着眼于测试用例集的简化算法研究,而不考虑最初的测试用例集的确定,所取得的效果将很有限. 基于以上研究,我们提出了一种新的最简测试用例集生成方法,该方法充分考虑了最初的测试用例集的作用,研究了如何根据测试目标中的测试需求来产生相应的最初测试用例集,证明了在此测试用例集基础上利用上述算法进行精简,可以产生最好的测试用例集. 特别是在此基础上应用 Lee 和 Chung 在文献[3]中提出的简化方法可以产生针对测试目标的最优的测试用例集.

人们可能比较关心测试用例集的简化,是否会降低它的错误检测能力. 对于这个问题,一些研究者研究表明测试用例集的简化不会对它的错误检测能力产生影响[9,10],而另外有些研究者却发现冗余的测试用例可以用来帮助提高测试用例集的错误检测能力[11]. 总之,测试用例的简化并不一定是以牺牲其错误检测能力为代价的. 所以,在这里我们只关心如何生成数量最少的、能满足测试目标中的所有测试需求的测试用例集.

4 最小测试用例集生成方法

我们提出一种最小测试用例集的生成方法,它充分考虑了测试目标中的各个测试需求之间的相互关系,给出了一种根据测试需求生成最小测试用例集的方法.

设对某个程序的测试目标由 n 个测试需求 $r_1,r_2,\cdots,r_n$ 组成,相应每个测试需求 r_i 对应的所有可能的测试用例集为 T_i,即 T_i 中任一测试用例可以实

现对测试需求 r_i 的充分测试($1\leq i\leq n$). 其中 $T_i\neq\varnothing$,如果 $T_i=\varnothing$,则称对应的测试需求 r_i 是不可行的. 本文假定所有的测试需求都是可行的. 即 $T_i\neq\varnothing$,$i=1,2,\cdots,n$. 令 $T=T_1\cup T_2\cup\cdots\cup T_n$ 表示所有的可用测试用例集.

通常的方法是先从 $T_i(i=1,2,\cdots,n)$ 中选出相应的测试用例组成一个测试用例集,然后再应用第 3 节介绍的简化算法,对这个测试用例集进行简化. 这种方法的缺点是:在选择测试用例的时候,没有充分考虑各个测试需求之间的相互关系,从而使测试用例集的简化无法从根本上达到最优.

基于以上考虑,在选择测试用例之前,首先必须探明测试目标中各个测试需求之间的相互关系,实际上是对所有测试用例集 T 进行划分. 例如对于测试需求 r_1,r_2 所对应的测试用例集 T_1 和 T_2,可以划分为三个互不相交的部分:T_1-T_2,$T_1\cap T_2$,T_2-T_1. 其中 T_1-T_2 中的测试用例只能满足测试需求 r_1,T_2-T_1 中的测试用例只能满足测试需求 r_2,$T_1\cap T_2$ 中的测试用例既能满足 r_1,又能满足 r_2($T_1\cap T_2=\varnothing$ 时表示无法选择一个测试用例同时满足这两个测试需求;$T_1-T_2=\varnothing$ 时,$T_2-T_1=\varnothing$,$T_1\cap T_2=T_1=T_2$ 此时称两个测试需求等价).

再将测试需求 r_3 所对应的测试用例集 T_3 考虑进来,最多能得到如下 4 个互不相交的部分:$(T_1-T_2)\cap T_3$,$T_1\cap T_2\cap T_3$,$(T_2-T_1)\cap T_3$,$T_3-T_2-T_1$. 其中 $(T_1-T_2)\cap T_3\neq\varnothing$ 时,其中的测试用例能同时满足测试需求 r_1,r_3,但不能满足 r_2;$T_1\cap T_2\cap T_3\neq\varnothing$ 时,其中的测试用例能同时满足测试需求 r_1,r_2,r_3;$(T_2-T_1)\cap T_3\neq\varnothing$ 时,其中的测试用例能同时满足测试需求 r_2,r_3,但不能满足 r_1;$T_3-T_2-T_1\neq\varnothing$ 时,其中的测试用例只能满足测试需求 r_3.

以此类推,根据各个测试需求所对应的测试用例集,可以实现对整个测试用例集 T 的完全划分,从而可以明确各个测试需求之间的相互关系.

$T_1,T_2,\cdots,T_n$ 是 n 个测试需求对应的所有可用测试用例的集合,令 $T[i_1,i_2,\cdots,i_s]=T_{i1}\cap T_{i2}\cap\cdots\cap T_{is}$ 为 $T_{i1},T_{i2},\cdots,T_{is}$ 的交集,要求其中的每个元素一定同时属于这 s 个集合($1\leq s\leq n$),不可多也不可少. 例如 $T[1]$ 就表示所有只在 T_1 中的元素组成的集合,$T[1]$ 中的任何元素不会在 $T_2,\cdots,T_n$ 中任何一个集合中出现,即 $T[1]=T_1-T_2-\cdots-T_n$. $T[1,2]$ 表示所有只同时包含在 T_1,T_2 两个集合中的元素组成的集合,$T[1,2]=T_1\cap T_2-T_3-\cdots-T_n$. 而且 $T[1]\cap T[1,2]=\varnothing$.

定理 1. 设 $T_1,T_2,\cdots,T_n$ 是 n 个集合,令 $B=\{T[i_1,i_2,\cdots,i_s]\mid\{i_1,i_2,\cdots,i_s\}$ 是集合 $\{1,2,\cdots,n\}$ 的任意子集$\}$,且 $T=T_1\cup T_2\cup\cdots\cup T_n$,则 B 是全集 T 的一个划分,即该集合中任意两个集合互不相交,但它们的并集是全集 T.

证明. 令 $A=\cup T[i_1,i_2,\cdots,i_s]$,$\{i_1,i_2,\cdots,i_s\}$ 是集合 $\{1,2,\cdots,n\}$ 的子集. 由于 $T=T_1\cup T_2\cup\cdots\cup T_n$,且 $T[i_1,i_2,\cdots,i_s]=T_{i1}\cap T_{i2}\cap\cdots\cap T_{is}$,从而 $A\subseteq T$.

反之,对 $\forall\chi\in T$,一定有某个集合 T_{i_1} ($1\leq i_1\leq n$) 使得 $\chi\in T_{i_1}$. 这个元素 χ 要么只属于集合 T_{i_1},要么同时属于集合 T_{i_1} 和其他若干个集合,即一定有 $\chi\in T[i_1,i_2,\cdots,i_s]$,$1\leq s\leq n$,$s=1$ 时,$T[i_1,i_2,\cdots,i_s]$ 即为 $T[i_1]$. 所以 $\chi\in A$,由 χ 的任意性可得 $T\subseteq A$. 综上可知 $A=T$.

设 $T[i_1,i_2,\cdots,i_s]$,$T[j_1,j_2,\cdots,j_t]$($1\leq s\leq n$,$1\leq t\leq n$) 是 B 中任意两个不同集合,$\{i_1,i_2,\cdots,i_s\}\neq\{j_1,j_2,\cdots,j_t\}$. 则 $T[i_1,i_2,\cdots,i_s]\cap T[j_1,j_2,\cdots,j_t]=\varnothing$,否则存在 $\chi\in T[i_1,i_2,\cdots,i_s]\cap T[j_1,j_2,\cdots,j_t]$,那么 $\chi\in T[i_1,i_2,\cdots,i_s]$ 时,说明 χ 只同时属于 $T_{i_1},T_{i_2},\cdots,T_{i_s}$ 这 s 个集合,而 $\chi\in T[j_1,j_2,\cdots,j_t]$ 时,又说明 χ 只同时属于 $T_{j_1},T_{j_2},\cdots,T_{j_t}$ 这 t 个集合,这与条件相矛盾. 所以 B 中任意两个集合互不相交. 所以 B 是全集 T 的一个划分. 证毕.

为了得到所有可用测试用例集 T 的一个划分,可用一个链表来记录划分的结果,每个子集存放在每个结点. 只要保证这个链表中每个结点存放的测试用例集互不相交,它们的并集是 T,就得到了 T 的一个划分. 根据上面的讨论,我们依据以下步骤产生划分链表:首先将满足测试需求 r_1 的测试用例集 T_1 放在链表首部第一个结点内,然后考虑满足测试需求 r_2 的测试用例集 T_2 与 T_1 的关系. 如果 $T_1\cap T_2=\varnothing$ 时,将集合 T_2 及其对应测试需求的编号填入链表第一个结点后的结点内. 此时链表中有两个结点,第一个结点中存放的是只满足测试需求 r_1 的测试用例集 T_1,第二个结点中存放的是只满足测试需求 r_2 的测试用例集 T_2,没有同时满足 r_1 和 r_2 的测试用例. 如果 $T_1\cap T_2\neq\varnothing$,此时首先将链表中第一个结点中测试用例集 T_1 去掉 $T_1\cap T_2$ 中的元素,然后将 $T_1\cap T_2$ 填入链表的第二个结点,并把 r_1 和 r_2 的标号填入相应的域,以表明该结点中的测试用例集可同时满足这两个测试需求. 最后将 T_2 去掉 $T_1\cap T_2$ 中的元素,如果非空,把它填入第三个结点中. 这样链

表中就有 3 个结点,第一个结点中存放的是只满足测试需求 r_1 的测试用例集 $T_1-T_1\cap T_2$,第二个结点中存放的是同时满足 r_1 和 r_2 的测试用例集 $T_1\cap T_2$,第三个结点存放的是只满足测试需求 r_2 的测试用例集 $T_2-T_1\cap T_2$. 然后,再考虑测试需求 r_3 对应的测试用例集 T_3 与链表中各个结点中测试用例集的关系.

依此类推,直到最后一个测试需求 r_n 对应的测试用例集 T_n. 因此这个队列的每个结点由 3 个域组成:一个用来存放某个测试用例子集,另一个存放这组测试用例能同时满足的测试需求所对应的标号集,最后一个域存放指针,即使用如下所示的结构:

```
Struct TestcaseNode
  { Requirement_Labels: set1; //测试需求对应的编号集
    Test_Set: set2; //同时满足上面测试需求的所有测试用例集
    Next: * TestcaseNode
  }
```

具体实现如算法 1. 从算法 1 最后得到的链表中,我们可以清楚地看到哪些测试需求可以同时被一个测试用例满足,一个测试用例最多可以同时满足哪些测试需求.

根据算法 1 得到的结果,应用贪心算法、一些启发式算法或优化算法便可确定一个最小的测试用例集. 在下一部分将通过一个例子来说明这个过程. 并与现有的方法进行比较,说明这种方法的有效性.

5 实例研究

本节通过一个简单的例子来说明我们方法的有效性. 假设一个系统 S 的测试目标由 5 个测试需求组成,即 $R=\{r_1,r_2,r_3,r_4,r_5\}$. 每个测试需求 r_i 对应的测试用例集为 $T_i(i=1,2,3,4,5)$. 首先应用我们提出的对所有可用测试用例集 T 进行划分的算法处理后得到这样一个子集族:

$\{T[1],T[1,2],T[1,2,4],T[1,3],T[2,3],T[3,4],T[4,5],T[4],T[5],T[3,4,5]\}$.

其中 $T[1]$表示只满足测试需求 r_1 的测试用例组成的集合,$T[1,2,4]$表示所有同时满足测试需求 r_1,r_2,r_4 的测试用例组成的集合,详细如表 1. 这个子集族中任意两个集合互不相交,并且

$T1=T[1]\cup T[1,2]\cup T[1,2,4]\cup T[1,3]$,

$T2=T[1,2]\cup T[1,2,4]\cup T[2,3]$,

$T3=T[1,3]\cup T[2,3]\cup T[3,4]\cup T[3,4,5]$,

$T4=T[1,2,4]\cup T[3,4]\cup T[4,5]\cup T[4]\cup T[3,4,5]$,

$T5=T[4,5]\cup T[5]\cup T[3,4,5]$.

算法 1. 对所有可用测试用例集 T 的划分.

```
//该算法产生一个链表队列,每个结点中包含一个测试用例子集及这个子集可以满足的测试需求的标号组成的集合,从而形成一个对所有测试用例组成的集合 T 的划分.
{ p,q,l: * TestcaseNode;
  p=new TestcaseNode;
  p.Requirement_Labels={1}; //初始化
  p.Test_Set =T1;
  p.next=Null;
  for i=2 to n do
    { q=p;
      while q≠Null do          //对 Ti进行划分
        { if q.Test_Set∩Ti≠∅ then
          {l=New TestcaseNode;
           l.Test_Set=q.Test_Set∩Ti;
           l.Requirement_Labels=
                        q.Requirement_Labels∪{i};
           Ti=Ti-l.Test_Set;
           q.Test_Set=q.Test_Set-l.Test_Set;
           l.next=q.next;
           q.next=l;
           q=l.next; }
        }
        if Ti≠∅ then
        { l=New TestcaseNode;
          l.Test_Set=Ti;
          l.Requirement_Labels={i};
          l.next=Null;
          q.next=l;}
    }
}
```

表 1 测试用例集与测试需求的满足关系表

	$T[1]$	$T[1,2]$	$T[1,2,4]$	$T[1,3]$	$T[2,3]$	$T[3,4]$	$T[4,5]$	$T[4]$	$T[5]$	$T[3,4,5]$
r_1	√	√	√	√						
r_2		√	√		√					
r_3				√	√	√				√
r_4			√			√	√	√		√
r_5							√		√	√

在这个子集族的基础上,应用第 3 部分提出的 5 种算法中的任何一种,可以得到测试用例应该在集合 $T[1,2,4]$与 $T[3,4,5]$ 或者在集合 $T[1,2]$ 与 $T[3,4,5]$中选取.这样就可以用两个测试用例实现对这个测试目标的 5 个测试需求的充分测试.这个结果是对这个测试目标的最优测试用例选择方案,而无法使用任何方法再对此进行优化.

如果不采用我们提出的方法,按照现有的方法,即首先根据每个测试需求产生相应的测试用例,在上面这个例子中由测试需求 r_i产生测试用例 $t_i(i=1,2,3,4,5)$形成测试用例集 $T=\{t_1,t_2,t_3,t_4,t_5\}$.这时对于 T 的简化结果主要取决于测试用例 t_1,t_2,t_3,t_4,t_5的取法.例如当 $t_1\in T[1],t_2\in T[2,3],t_3\in T[3,4],t_4\in T[3,4],t_5\in T[5]$,即 t_1从 $T[1]$中取,此时测试用例 t_1只能满足测试需求 r_1,t_2从 $T[2,3]$中取,它能同时满足测试需求 r_2,r_3,同样测试用例 t_3,t_4能同时满足测试需求 r_3,r_4,而 t_5只能满足测试需求 r_5.在这种情况下,应用前面第 3 节的几种简化算法,只能简化为$\{t_1,t_2,t_3,t_5\}$或者$\{t_1,t_2,t_4,t_5\}$,就不能再简化了;而在 $t_1\in T[1,2],t_2\in T[2,3],t_3\in T[3,4],t_4\in T[3,4],t_5\in T[5]$时,又可简化为$\{t_1,t_3,t_5\}$或者$\{t_1,t_4,t_5\}$;在 $t_1\in T[1,2,4],t_2\in T[2,3],t_3\in T[3,4],t_4\in T[3,4],t_5\in T[3,4,5]$时,这时 T 又可以简化为$\{t_1,t_5\}$.

综上所述,现有方法中应用贪心算法、一些启发式算法或优化算法等,不能从根本上产生最优的测试用例集,只能产生相对最优的测试用例集.

6　结束语

本文分析了测试用例集简化的现有方法的缺点,提出了一种根据测试目标中各个测试需求之间相互关系产生最小测试用例集的方法.应用这种方法可以生成由测试目标中所有测试需求所确定的固有最小测试用例集,从根本上解决了测试用例集的简化问题.从而能生成最少的测试用例,实现对测试目标的充分测试,最大限度地提高测试效率,降低软件测试成本.

参　考　文　献

1　Chen T Y, Lau M F. A new heuristic for test suite reduction. Information and Software Technology, 1998, 40(5/6): 347～354

2　Chen T Y, Lau M F. A simulation study on some heuristics for test suite reduction. Information and Software Technology, 1998, 40(13): 777～787

3　Lee J G, Chung C G. An optimal representative set selection method. Information and Software Technology, 2000, 42(1): 17～25

4　Chen T Y, Lau M F. Dividing strategies for the optimization of a test suite. Information Processing Letters, 1996, 60(3): 135～141

5　Johnson D S. Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences, 1974, 9(3): 256～278

6　Harrold M J, Gupta R, Soffa M L. A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270～285

7　Chen T Y, Lau M F. Heuristics towards the optimization of the size of a test suite. In: Proceedings of the 3rd International Conference on Software Quality Management, Seville, Espagne, 1995, 2: 415～424

8　Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage. In: Proceedings of ICSM'01, Florence, Italy, 2001, 11: 92～102

9　Wong W E, Horgan J R, London S *et al*. Effect of test set minimization on fault detection effectiveness. In: Proceeding of the 17th International Conference on Software Engineering, Seattle, Washington DC, 1995. 41～50

10　Wong W E, Horgan J R, Mathur A P *et al*. Test set size minimization and fault detection effectiveness: A case study in a space application. In: Proceeding of the 21st Annual International Computer Software and Application Conference of COMPSAC 97, Washington DC, 1997. 522～528

11　Chen T Y, Lau M F. On the completeness of a test suite reduction strategy. The Computer Journal, 1999, 42(5): 430～440

NIE Chang-Hai, born in 1971, Ph. D., lecturer. His research interests include software engineering and software testing, fuzzy information processing, neural network and etc.

XU Bao-Wen, born in 1961, Ph. D., professor and Ph. D. supervisor. His research interests include programming language, software engineering, concurrent and internet software.