一种改进的测试路径集生成算法①

姜姗姗[1]　赵中华[1]　张　波[2]　王启明[1]　(1.72671 部队 应用开发 山东 济南 250022;

2.72556 部队 71 分队 山东 济南 250002)

摘　要: 在对两种较为有效的分支测试路径集生成算法分析研究的基础上，提出了一种改进的测试路径集生成方法，并设计实现了一个面向分支覆盖的测试路径自动生成系统，通过实例验证后表明，该系统能有效生成分支测试路径集，并具有较高的测试覆盖率。

关键词: 测试覆盖; 分支测试; 非约束边; DD 图约简

Improved Method for Test Path Set Generation

JIANG Shan-Shan[1], ZHAO Zhong-Hua[1], ZHANG Bo[2], WANG Qi-Ming[1]

(1.The Application and Development Lab, PLA 72671, Jinan 250022; 2.The 71 Group, PLA 72556, Jinan 250002, China)

Abstract: An improved method for generating test path set is given based on then analysis of two effective algorithms of branch test paths generation. A system for generating test paths is designed and then validated with instances. The test results show that the system is effective in generating branch test paths, and reaches high test coverage.

Keywords: test coverage; branch testing; unconstrained arcs; DD-graph reduction

1　引言

测试覆盖是软件测试的一种度量方法，可以针对不同的测试需求，选择相应的测试覆盖标准。在软件白盒测试中，测试员往往会选择有一定效果且开销较小的覆盖准则，最常用的就是分支覆盖准则。分支覆盖是一个较为实用的覆盖标准，它度量的是程序代码段中整个 BOOL 表达式取值 true 和 false 在控制结构中是否被测试，分支测试为测试耗费和测试效力之间提供了一种有利的平衡[1]。

本文在对两种较为有效的分支测试路径集生成算法分析的基础上，给出了生成非约束边集的改进算法 Find-Exact-UE，将该算法所求得的非约束边集作为测试路径集生成的基础，然后结合对基于正向(逆向)广度优先生成树的测试路径生成方法的改造，实现了对整个测试路径集生成算法的改进，降低了算法的复杂度，减少不可行路径生成的路径概率。

2　基本概念

定义 1. DD 图[2](Decision-to-Decision Graph)：对于一个入口边(e0)和退出边(ek)不相同的有向图 G = (N, E), 若顶点集{ N - { $T(e_0)$, $H(e_k)$ }中的任一顶点 n 的入度 indegree(n) 和出度 outdegree(n) 之和大于 2，则该图为 DD 图。

定义 2. 支配关系：对于 DD 图 G = (N , E)，若从入口边 e_0 到边 e_j 的任何一条路径均包含边 e_i，则称边 e_i 支配边 e_j；根据 DD 图 G 中边的支配关系可以构造一棵支配树。

定义 3. 蕴含关系：对于 DD 图 G = (N, E)，若从边 e_i 到退出边 e_k 的任何一条路径均包含边 e_j，则称边 e_i 蕴含边 e_j；根据 DD 图中边的蕴含关系的定义，同样可以构造一棵蕴含树。

定义 4. 非约束边[3]：在 DD 图中对于边 e_i，如果 e_i 既没有支配其他的边也没有被其他边所蕴含，则称

① 收稿时间:2010-03-06;收到修改稿时间:2010-04-10

e_i 为非约束边。即支配树与蕴含树叶节点的交集即为非约束边集。

在 DD 图 G 中，覆盖所有非约束边的一组路径集必定能覆盖 DD 图的所有边；而且所有非约束边的集合是满足上述性质的最小集合。

3　现有算法分析及改进

3.1 现有算法分析

测试路径集的生成通常分两步进行：第一步是产生非约束边集；第二步产生一条能覆盖所选择的某条非约束边的路径。

Bertolino 等人在文献[3]中提出了基于非约束边的 FTPS(FIND-A-TEST-PATH-SET)。它将对 DD 图所有边的覆盖转化为对关键边的覆盖，这些关键边的集合即为非约束边集，是 DD 图边的子集。为 DD 图分别构造支配树和蕴含树，求两树叶节点的交集(本文称这种方法为精确算法)。在路径生成过程中，由某条非约束边向入口边 e_0 或退出边 e_k 搜索生成子路径。过程非常繁琐复杂，时空开销大，有时在同样的两条边之间可能重复递归的过程。

针对 FTPS 算法的局限性，毛澄映等人在文献[1]中补充了一种近似非约束边集求解算法 Find-SemiUE(本文称之为近似算法)。该算法不需要构造支配树和蕴含树，节省了空间开销，编程实现也简单许多。但所找到的非约束边集只对基本程序结构构成的程序来说非常接近精确结果，而对 break，goto 等语句引起非正常跳转较多的程序，找到的结果往往不够精确。在路径生成的过程中，构造两棵生成树并在树上搜索路径的生成方法虽然能有效地避免繁琐的递归过程，但是若 DD 图中存在永远都不能在同一路径上出现的非约束边，算法没有在路径生成之前就充分考虑非约束边之间的不相容信息，这样生成的有些路径是不可行的，还需要重新生成。

3.2 算法改进的实现流程

针对上述两种算法各自存在的局限性，本文对算法改进遵循以下的思路：

(1) 在非约束边集生成过程中不再利用支配树和蕴含树，而是直接对原 DD 图进行搜索和约简，找到符合条件的边就加入非约束边集；为避免近似算法的不准确性，在搜索过程中需要充分利用非约束边的性质对其进行分类。

(2) 在路径生成过程中对 DD 图进行广度优先搜索构造两棵生成树(正向和逆向)并在树上搜索路径；在路径生成之前利用非约束边之间的不相容信息，尽量排除不相容的非约束边在同一条路径中出现的情况，提高路径的可行概率。

图 1 是对图 4-1 的示例 DD 图 G 子结构划分情况。其中，子结构 1 和 2 的情况相同，都是循环中嵌套分支，且某一分支又嵌套分支。对该类子结构按照如下的方式进行处理：①在子结构中不断搜索头节点是分支节点的边 e，且 e 的所有分支的头节点相同；②将 e 的所有分支化简为一条边 e′，且 e′ 与 e 直接相连构成顺序结构，e′ 的头节点是所有分支的汇合点；③将 e′ 与 e 合并为一条边，该边的头节点是 e′ 的头节点，该边的尾节点是 e 的尾节点；④找到 e 的逆邻接边，若满足①中的约束条件，则重复步骤②、③直到超出子结构的范围；否则重复①直至找到满足条件的边。

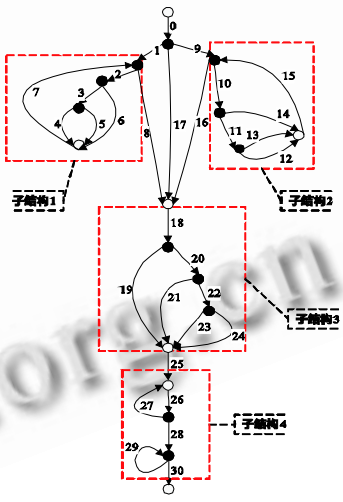

图 1　示例 DD 图 G 的子结构划分情况

子结构 1 所找到的非约束边有：4，5，6；子结构 2 所找到的非约束边有：12，13，14。

子结构 3 满足条件分支中又嵌套 if-else 条件分支的情况，没有嵌套在循环结构中。处理过程与子结构 1 和 2 的类似，只是最终的结果是一条顺序边。子结构 3 所找到的非约束边有：23，24，21，19。

子结构 4 中包含了可直接确认为非约束边的基本结构：自环边与反向边，这两种边可直接将其加入非约束边集，然后从原图中删除。所找到的非约束边有：29，27。

所有子结构处理完毕后，实现了各个子结构的约简，得到的图形如图 2 所示。

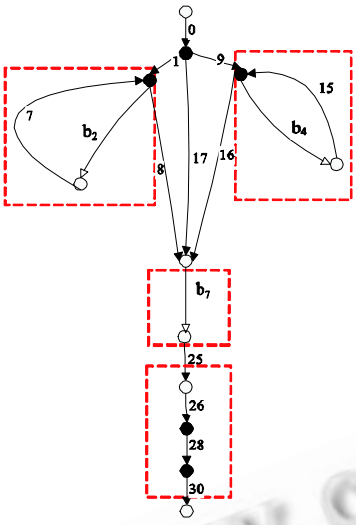

图 2　子结构处理完毕后得到的图形

此时得到的图形不是 DD 图，为进一步得到其余的非约束边且不会引入冗余的边，需要将该图形转化为 DD 图后再进行非约束边的求解。

在图 2 中寻找并去除出度和入度均为 1 的节点，该节点的入向边和出向边合并为一条边；如果合并后得到的边的头节点又满足出度和入度都为 1，继续执行以上操作，直至图中不存在此类节点。将得到的图形去除自环边后得到的图形进行判断，可以发现只有边 17 满足头节点入度 ≥ 2,且尾节点出度 ≥ 2，并且边 17 没有经过任何合并或约简操作，因而可以确定边 17 是非约束边。

利用改进后的算法最终找到的非约束边集 UE′ ={4, 5, 6, 12, 13, 14, 17, 19, 21, 23, 24, 27, 29}。利用文献[1]中给出的方法，求得精确的非约束边集 UE，通过比较可知 UE′ = UE，即新算法找到 DD 图 G 的非约束边集等于精确结果。

下面给出了算法的具体描述

算法 3.1: Find-Exact-UE (G).

输入: DD 图 G =(N , E).

输出: 非约束边集 UE′ .

```
{  UE′ = null ; k=1; t=1; Eprim = E ;
对 G 进行广度优先搜索，将生成树分别用邻接表和逆邻接表存储;
/*以下代码是用于寻找原 DD 图中的自环边并加入 UE′ ，然后将边从原图中去除*/
   for ( each e ∈ E)
   if ( H(e) = = T (e) )
   { UE′ = UE′ ∪{ e} ; E = E - { e} ; }
   /*以下代码是用于在得到的图中寻找反向边并删除；同时其所有分支边的源、目标节点均相同的边 */
  for ( each e ∈ E )
  {  ex = e ; E0 = null ; E1 = null ; E2 = null ;
   { if ( outdegree ( H(ex) ) >= 2 )
    {  E0 = { ex 的所有分支边}
      /*以下代码是用于去除反向边*/
      for (each e0′ ∈E0 )
      if ( (H(e0′ ) = = T ( ex )) && (T(e0′ ) = = H( ex )) )
       { UE′ = UE′ ∪{ e0′ } ; E = E - { e0 ′ } ;
                outdegree(H( ex ) ) -- ;
indegree( T(ex) ) -- ; }
     /* 以下代码是用于去除并列分支边*/
     while (ex 的任意两条不同分支边 e′ 和 e′ ′ 都满足 T(e′ ) = T(e′ ′ ))
      {  E1 = {ex 的所有分支边};
  UE′ = ( UE′ ∪ E1 ) － E2;
 E = ( E － E1 ) ∪ { ak } ;E2 = E2 ∪ { ak } ;
 H(ak) = H(e′ ) ;  T(ak) = H(ex) ;
/* 以下代码是用于判断如果 ak 尾节点的出度、入度均为 1，则将其入边和出边合并*/
     if ( (outdegree ( T (ak) ) = = 1) && (indegree ( T (ak) ) = = 1))
     { V =(V -{ T(ak)}); E = ( E －{ ex , ak })∪{ ak′ };
        T(ak′ ) = T( ex );  H(ak′ ) = H (ak );
        ex = 逆邻接表中 ak′ 的逆邻接边 ; }
        k++;  ex = 逆邻接表中 ak 的逆邻接边 ;
      }} } }
/*以下代码是用于对头节点或尾节点的出度、入度都为 1 的边进行约简*/
 while (E 中存在两条边 e 和 e′ ,且满足 e ≠ e′ , T(e′ ) = H( e ) ,
 indegree ( H (e) ) = 1, outdegree (T(e′ )) = 1 )
 {  V =V －{H (e);
    T(ck ) = T(e) ;  H(ck ) = H(e′ ) ;
```

```
  E = (E － { e , e′ }) + { ck } ; k++;  }
/*以下代码是用于对约简后的图去除自环边*/
 for ( each e ∈E )
  if ( H(e) = = T (e) )   E = E - { e} ;
/*以下代码是用于找到其余非约束边，且符合能够被加入 UE′ 条件的必须是原 DD 图中的边*/
 for ( each e ∈E )
 if (( indegree( H(e) ) >= 2 ) && (outdegree( T(e) ) >= 2) && (e ∈Eprim) )
 { UE′ = UE′ ∪{ e} ; E = E - { e} ; }
 return UE′ ; }
```

4　工具设计、实现与验证

根据改进后的算法设计实现了测试路径集自动生成工具 AGTP，工具由三个功能模块组成，如图 3 所示：

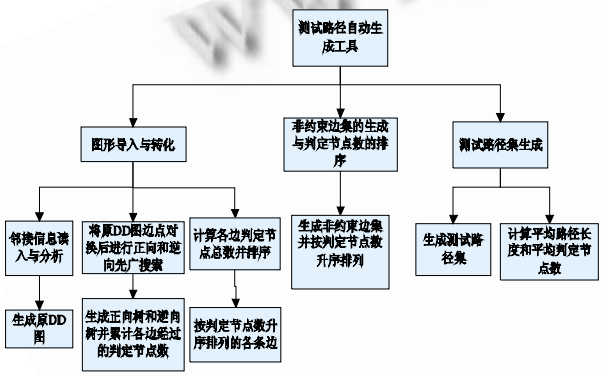

图 3　工具功能结构图

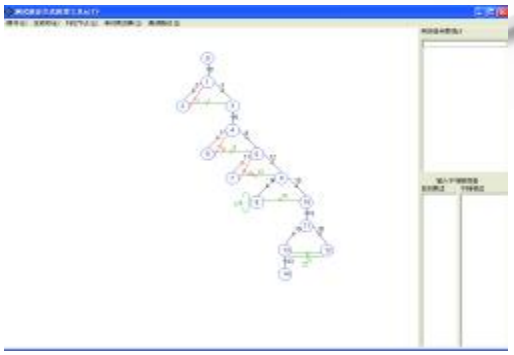

图 4　导入 DD 图后显示图形

使用工具前，测试人员已经获得程序源代码或者程序对应控制流图，将其转化为 DD 图，将得到的 DD 图转化为正确的输入信息存入单独建立的文本文件中。导入 DD 图的操作需要最先执行。选择所要处理的 DD 图输入文件，即可将输入数据转化为所需图形。图 4 显示的是成功导入 DD 图后的工具界面。各节点用圆圈表示，并用圆圈内的数字对其进行编号；各条边上的箭头指向表示方向，边的旁边以数字进行编号。为更清楚地区分不同类型的边，工具将反向边设置为红色，自环边与并列分支边为绿色，其他边为黑色。

计算各边的累计判定节点数，在右侧列出运算结果，并按判定节点数的升序进行排列；同时，左侧画板的原 DD 图上将非约束边加粗显示。如图 5。

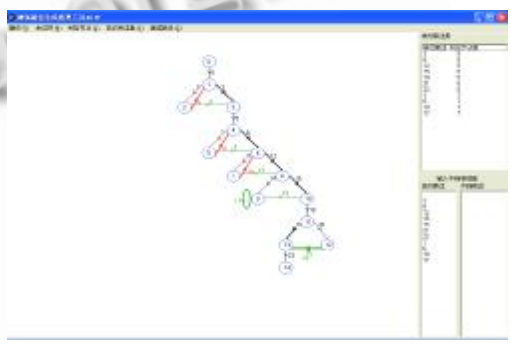

图 5　生成非约束边集后的结果及图形显示

根据前面分析得到的非约束边之间的不相容信息，在每条非约束边对应的位置输入不相容边的编号，若不存在则输入“-”。

图 6 显示输出结果，包含了测试路径集、平均路径长度和平均判定节点数的计算结果。

而若利用 FTPS 算法，最终得到的测试路径集 PS′ 共有 7 条路径，以 7 个测试用例达到 100%的分支覆盖率。

图 6　测试路径集生成结果界面

(下转第 109 页)

(上接第 101 页)

路径集 PS′ 中所含有的路径数为 7 条，平均路径长度为 10.43，平均判定节点数为 6.43；而利用工具 AGTP 求得的测试路径数为 4 条，平均路径长度为 11.25，平均判定节点数为 7.25。通过对比可以看出，工具 AGTP 在没有较大增加路径平均复杂程度的同时，可以以较少的测试用例达到了所需的分支覆盖目标，因而可以认为，该工具能够有效地生成可行测试路径集并达到了所需的覆盖目标。

参考文献

1 毛澄映.卢炎生.分支测试中测试路径用例的简化生成方法.计算机研究与发展, 2006,43(2):321—328.

2 Marre´ M, Bertolino A. Using Spanning Sets for Coverage Testing. IEEE Trans on Software Engineering, Nov. 2003,29(11):974—984.

3 Bertolino A, Marre´ M. Automatic generation of path covers based on the control flow analysis of computer programs. IEEE Trans Software Eng, 1994,20(12):885—899.