# 一种新的多路径覆盖测试数据进化生成方法

巩敦卫[1],张 岩[1,2]
(1.中国矿业大学信息与电气工程学院,江苏徐州 221116;2.牡丹江师范学院计算机科学与技术系,黑龙江牡丹江 157012)

**摘 要:** 提出一种新的用于多路径覆盖的测试数据生成方法.首先,将被测程序表示成一棵二叉树,对目标路径采用赫夫曼编码方法表示成二进制串;然后,采用遗传算法生成多个测试数据,设计的适应度函数综合考虑个体穿越的路径与每个目标路径的匹配程度.将提出的方法用于4个基准程序的路径覆盖测试数据生成,并与已有方法比较,结果表明本文方法计算量小,生成测试数据效率高.

**关键词:** 软件测试;路径覆盖;测试数据;遗传算法;赫夫曼编码
**中图分类号:** TP301 **文献标识码:** A **文章编号:** 0372-2112(2010)06-1299-06

# Novel Evolutionary Generation Approach to Test Data for Multiple Paths Coverage

GONG Dun-wei[1],ZHANG Yan[1,2]
(1. *School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, Jiangsu* 221116, *China*;
2. *Department of Computer Science and Technology, Mudanjiang Normal University, Mudanjiang, Heilongjiang* 157012, *China*)

**Abstract:** An approach to generating test data for multiple paths coverage is presented. First, the program under test is expressed as a binary tree, and the target paths are encoded into a binary string using Huffman coding; then, genetic algorithm is employed to generate multiple test data, and an individual's fitness is the degree of the traversed path matching the target paths. The proposed approach is applied to 4 benchmark programs, and compared with previous approaches. The results show that the proposed approach needs small amount of calculation and has high efficiency in generating test data.

**Key words:** software testing; path coverage; test data; genetic algorithms; Huffman coding

## 1 引言

软件测试是保障软件质量的重要手段[1],测试数据自动生成问题是软件测试的核心问题之一.路径覆盖测试是指在测试过程中,尽可能覆盖程序所有可行路径,许多软件测试问题都可以归结为路径覆盖测试的数据生成问题[2].

遗传算法(Genetic Algorithms, GA)是用于测试数据的自动生成的有效方法.目前已有许多研究成果,如Chen等采用多种群GA生成覆盖路径的测试数据[3],Sofokleous等基于GA建立了动态软件测试框架[4],Rajappa等利用GA并结合图论知识,生成软件的测试用例[5],Bouchachia将免疫GA用于测试数据生成中[6]等.这些成果极大地丰富了进化测试理论.

用GA生成覆盖路径的测试数据,需要将该问题转化为函数优化问题.适应度函数的设计对产生高性能的测试数据非常关键.但是,现有文献适应度函数大多是针对一条目标路径设计的,这将导致GA的一次运行,只能生成穿越一条路径的测试数据.因此,有必要提高生成测试数据的效率.为解决这一问题,Ahmed等首次提出基于GA的多路径测试数据进化生成方法,将测试数据生成问题转化为多目标优化问题,利用GA解决该问题时,分别计算进化个体对不同目标的满足程度,使得一次运行GA,生成分别穿越多条目标路径的多个测试数据[7],并用实验验证了该方法生成测试数据优于单路径生成方法.Ahmed方法的适应度函数由2部分组成,分别是层接近度和支距离,其中,层接近度指测试数据穿越的路径与每个目标路径不匹配的结点个数之和;支距离反映被穿越的路径与目标路径分支语句的前件还有多少距离,将支距离和层接近度之和作为个体相对于一个目标路径的适应值,将个体针对所有目标路径的适应值平均值作为个体的适应值.但是,当测试数据支

收稿日期:2009-05-15;修回日期:2009-12-25
基金项目:国家自然科学基金资助(No.60775044);江苏省"六大人才高峰"高层次人才项目(No.2008125);中国矿业大学青年科研基金资助项目(No.2008A034);江苏省"333高层次人才培养工程"项目;牡丹江师范学院青年学术骨干资助计划(No.G2008005)

距离较大时,层接近度与其相比几乎被忽略不计了,此时其作用得不到很好的体现;更重要的是当分支语句的前件比较复杂时,支距离的计算很复杂,导致测试数据的生成时间较长,效率不高.

如果采用合适的方法,针对所有目标路径,设计合适的适应度函数,经过较少的计算能比较不同进化个体的优劣,运行一次 GA,生成穿越多条路径的测试数据,将有可能提高生成测试数据的效率.本文采用赫夫曼编码(Huffman Coding)方法,将所有目标路径编码成二进制串,设计的适应度函数综合考虑穿越的路径与每个目标路径的匹配程度,一次运行 GA,生成穿越全部可行路径的测试数据.在 4 个基准测试程序路径覆盖测试数据生成问题中的应用,验证了所提方法的优越性.

## 2 目标路径的赫夫曼编码

### 2.1 赫夫曼编码

赫夫曼编码是由美国计算机科学家 Huffman 提出的一种编码方式,该编码能够使通常的数据传输数量降低到最少,已广泛应用于传真、图象压缩和计算机安全等领域.该编码源于一种树型结构.

赫夫曼树是一类带权路径长度最短的树[8].赫夫曼编码是利用赫夫曼树构造的一组最优前缀编码(其中任何一个字符的编码都不是其它字符编码的前缀).该编码的一种构造方法是:约定 0 表示左分支,1 表示右分支,则从根结点到叶结点的路径分支上的字符组成该叶结点的编码.图 1($a$)为权重分别为 3、2、1、1 的四个结点 $A$、$B$、$C$、$D$ 构成的赫夫曼树,其对应的编码如图 1($b$)所示.

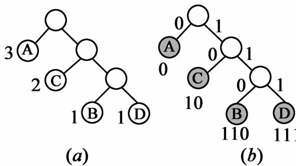



图1 赫夫曼树及其编码

### 2.2 目标路径的赫夫曼编码

被测程序一般有多条路径.在结构化程序设计中,程序通常由顺序、选择和循环三种结构组成.其中,选择和循环结构决定程序的不同分支走向,而选择结构又包括双分支和多分支两种情况,当然,多分支可以分解成多个双分支的组合;通过引入 $Z$ 路径覆盖[9],可把循环结构化成双分支选择结构.实际上 $Z$ 路径覆盖只是考虑了循环体被执行一次的情况,循环体执行多次时,可以展开成多个选择并列的情况.若把程序中每个分支语句的前件表示成一个结点,第一个分支语句的前件作为根结点,其两个分支中嵌套的选择语句前件分别表示成它的两个子结点,如果后面的分支语句与其是并列关系,则后续的分支结点画到其两棵子树上,这样就可以把一个程序表示成一棵二叉树,且从根结点到每个叶结点代表一条路径.

不同的程序员采用的编程方法不同,使得被测程序对应的二叉树形状不同,由于不考虑权重,不能称其为赫夫曼树,这里主要借鉴赫夫曼编码方法表示目标路径.把分支语句的假分支用 0 表示,真分支用 1 表示,得到从根结点到叶结点的路径编码也是前缀编码.

下面通过三角形分类程序说明路径的表示方法,其 $C$ 语言代码如图 2($a$)所示,图 2($b$)是该程序流程.程序的每个路径都可以用二叉树叶结点的赫夫曼编码表示.图 2($c$)表示的 4 个路径及编码分别是:非三角形1;非等边三角形01;等边三角形001;等腰三角形000.这样,所有路径可以编码表示为1 01 001 000.

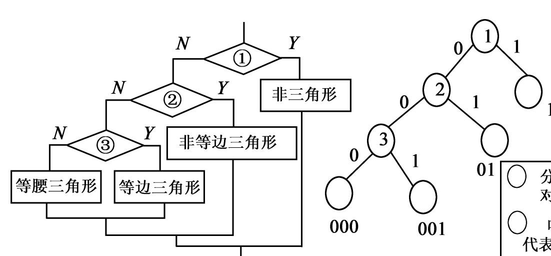
### 2.3 被测试程序的插桩

为了反映测试数据在程序执行中穿越的路径,需要在被测程序中插入一些变量,用变量的不同值代表执行分支语句的不同走向,变量的个数与路径编码的最大长度相同.可以用一个一维数组存放各分支语句的不同走向,数组类型用字符型或数值型.本文采用字符型.记 $s$ 为用于插桩的数组,其元素个数记为 $|s|$,与路径编码的最大长度相同.数组元素用来标记程序对应二叉树中某一层分支的走向,相同层的分支用相同下标的数组元素标识,数组各元素的初值赋为空字符'\0',插桩时的赋值如下:

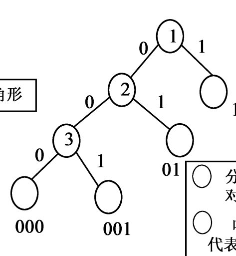
$$s[i]=\begin{cases}\text{'1' 执行真分支}\\\text{'0' 执行假分支}\end{cases},\quad i=0,1,\cdots,|s|-1 \tag{1}$$

```
void triangle(int a, int b, int c)
{if((a+b<=c)||(a+c<=b)||(b+c<=a))//分支①
    printf("Not_a_Triangle!\n");
  else
    if (a!=b&&b!=c&&c!=a)//分支②
        printf("Scalene!\n");
    else
        if (a==b&&b==c) //分支③
         printf("Equilateral!\n");
        else printf("Isosceles!\n");
}
```

($a$) 三角形分类源程序




($b$) 三角形分类程序流程图




($c$) 三角形分类程序的4个路径编码

```
char s[4]={'\0'};// 存放路径编码
void  triangle(int a, int b, int c)
{if((a+b<=c)||(a+c<=b)||(b+c<=a))//分支①
   {  s[0]= '1'; printf("Not_a_Triangle!\n");}
  else{s[0]='0';
      if (a!=b&&b!=c&&a!=c)  //分支②
        {  s[1]= '1'; printf("Scalene!\n");}
      else{s[1]= '0';
          if(a==b&&a==c) //分支③
          {s[2]= '1';printf("Equilateral!\n");}
          else{s[2]='0';printf("Isosceles!\n");}
          }}}
```

($d$) 插桩后的三角形分类程序

图2 三角形分类程序、流程图、路径编码及插桩后的程序

在程序分支语句的执行部分插桩变量赋值语句,比如,对于程序的第 $i$ 层分支语句,在条件为真的执行语句中插桩赋值语句 $s[i]$ = ‘1’,在条件为假的执行语句中插桩赋值语句 $s[i]$ = ‘0’. 三角形分类程序插桩后如图 2($d$)所示,带下划线部分为插桩的语句.

## 3　算法设计

### 3.1　适应度函数设计

由于目标路径用赫夫曼编码表示,因此,设计适应度函数与已有的工作有很大的区别. 思想是:考察 GA 产生的所有测试数据穿越的路径编码,逐一与每个目标路径编码子串对比,有完全相同的说明已经找到该子串对应路径的测试数据,就保存该测试数据,并把该子串从目标路径编码串中删除;通过比较测试数据穿越的路径编码与每个目标路径编码的匹配程度,计算该测试数据的适应值.

记第 $t$ 代进化种群的个体为 $x$,在不引起混淆的情况下,仍记解码后的测试数据为 $x$. 以 $x$ 为被测试程序的输入,穿越的路径记为字符串 $p(x)$,字符串的位数记为 $|p(x)|$. 设此时目标路径还有 $n(t)$ 个,记第 $j$ 条目标路径为 $p_j$,根据 $p(x)$ 与 $p_j$, $j=1,2,\cdots,n(t)$ 的匹配程度,计算该个体的适应值,记为 $f(x)$.

记 $|p_j|$ 为目标路径 $p_j$ 的编码位数,从左向右依次比较 $p(x)$ 与 $p_j$ 各位的编码. 记 $d_{jk}(p(x))$ 为反映 $p(x)$ 与 $p_j$ 第 $k$ 位编码是否相同的量,如果相同,则令 $d_{jk}(p(x))$ 为 1;否则,令 $d_{jk}(p(x))$ 为 0,这样最多比较 $\min(|p(x)|,|p_j|)$ 次.

考虑到按位比较时相同位的个数越多,说明个体越接近目标路径对应的测试数据. 因此,在编码按位比较的同时,用计数器 $m_j$ 标记 $p(x)$ 与 $p_j$ 相同位的个数. 由于在适应值计算之前找到测试数据的目标路径编码已经删除,所以 $m_j < \min(|p(x)|,|p_j|)$.

为了加大不同编码位对应的匹配程度之间的区别,对不同的编码位设计了不同的权值,权值的大小取决于比较过程中的相同位的个数 $m_j$, $m_j$ 越大权值越大. 记比较到第 $k$ 位时相同位的个数为 $m_{jk}$,并记 $p(x)$ 与 $p_j$ 的匹配程度为 $f_j'(x)$,则 $f_j'(x)$ 可表示为:

$$f_j'(x) = \sum_{k=1}^{\min(|p(x)|,|p_j|)} m_{jk} \cdot d_{jk}(p(x)) \qquad (2)$$

式(2)表明,$p(x)$ 与 $p_j$ 的相同位数越多,二者越匹配,则 $f_j'(x)$ 越大.

不同的编码位对接近目标路径的贡献不同. 一般来讲,越是先比较的编码位,对接近目标路径的贡献越大,即从前向后比较连续相同的位数越多,个体越接近目标路径的测试数据. 因此,在计算 $x$ 的适应值时,为了增大进化个体间适应值的差别,尽量保留接近某个目标路径的测试数据,对式(2)进行如下修正:在 $p(x)$ 与 $p_j$ 从前向后按位比较过程中,记录从第一位开始连续相同的位数,记为 $c_j(p(x))$,并记修正后的匹配程度为 $f_j(x)$,则有:

$$f_j(x) = (c_j(p(x)) + 1) \cdot f_j'(x) \qquad (3)$$

式(3)保证了从前向后比较过程中,连续相同位数越多的路径,与目标路径的匹配程度越高. 比如,若第一位不相同,则 $c_j(p(x))=0$, $f_j(x)=f_j'(x)$;若第一位相同,第二位不同,则 $c_j(p(x))=1$, $f_j(x)=2f_j'(x)$;依此类推.

考虑 $p(x)$ 与此时所有目标路径的匹配程度,取其平均值作为个体 $x$ 的适应值 $f(x)$,则有:

$$f(x) = \frac{1}{n(t)} \sum_{j=1}^{n(t)} f_j(x) \qquad (4)$$

以三个数排序程序为例,假设进化到第 $t$ 代时目标路径编码为“<u>010</u> <u>100</u> <u>111</u>”,则目标路径数 $n(t)=3$,三个目标路径分别为 $p_1$ = “010”, $p_2$ = “100”, $p_3$ = “111”. 若第 1 个个体 $x_1$ 穿越的路径 $p(x_1)$ 为“101”,第 2 个个体 $x_2$ 穿越的路径 $p(x_2)$ 为“001”,从直观看来,第一个个体穿越路径“101”和目标路径“100”相同位数为 2,比较接近;而第二个个体穿越路径“001”和任何一个目标路径比较相同位数都未达到 2,所以,第一个个体优于第二个个体. 下面分别计算两个个体的适应值. 先计算 $x_1$ 的适应值:

首先用“101”与“010”比较,得到 $d_{11}(p(x_1))=0$, $d_{12}(p(x_1))=0$, $d_{13}(p(x_1))=0$,相同位的个数也为 0,即 $m_{11}=m_{12}=m_{13}=0$, $c_1(p(x_1))=0$. 则有

$$f_1'(x_1) = \sum_{k=1}^{3} m_{1k} \cdot d_{1k}(p(x_1)) = 0,$$

$$f_1(x_1) = (c_1(p(x_1)) + 1) \cdot f_1'(x_1) = 0$$

再用“101”与“100”比较,得到

$$f_2(x_1) = (c_2(p(x_1)) + 1) \cdot f_2'(x_1) = (2+1) \times (1\times 1 + 2\times 1 + 2\times 0) = 9$$

最后,用“101”与“111”比较,得到

$$f_3(x_1) = (c_3(p(x_1)) + 1) \cdot f_3'(x_1) = (1+1) \times (1\times 1 + 1\times 0 + 2\times 1) = 6$$

这样一来,个体 $x_1$ 的适应值为

$$f(x_1) = \frac{1}{3} \sum_{j=1}^{3} f_j(x_1) = \frac{1}{3} \cdot (0+9+6) = 5.0$$

采用同样的方法,可以得到个体 $x_2$ 的适应值为

$$f(x_2) = \frac{1}{3} \sum_{j=1}^{3} f_j(x_2) = \frac{1}{3} \cdot (2+1+1) \approx 1.333$$

比较个体的适应值,明显区分出了它们的优劣.

### 3.2　算法终止条件和步骤

进化过程中,若所有目标路径的测试数据都找到,则终止进化;若进化一定代数后,还有某一(些)路径没

有找到,认为该路径是不可行路径,结束进化.

采用 GA 生成测试数据的步骤如下:

**步骤 1**　对算法所需的控制参数赋值,对被测试程序的路径编码,插桩被测试的程序;

**步骤 2**　初始化种群;

**步骤 3**　对进化个体解码,执行被测试程序;

**步骤 4**　判断是否有完全匹配的路径,若有,保存与该进化个体对应的测试数据,从目标路径编码中删除该被匹配的路径对应的编码;

**步骤 5**　判断是否满足终止准则,若是,转步骤 8;

**步骤 6**　根据式(4)计算进化个体适应值;

**步骤 7**　实施选择、交叉和变异操作,生成子代进化种群,转步骤 3;

**步骤 8**　停止进化,解码进化个体,输出测试数据.

### 3.3　算法分析

(1)不计算支距离,计算量至少减半

现有文献中适应值计算都考虑支距离,支距离的计算需要针对每一个简单谓词计算,假设一个目标路径有 $h$ 个结点,每个结点中至少有一个简单谓词,那么支距离运算需要计算 $h$ 个,然后求和,如果每个结点都是复合谓词,则计算量成倍增长;当程序中存在 flag 现象时,适应值不能有效指导进化,从而变成随机搜索[10];更重要的是当某个结点谓词对应的支距离较大,会造成对其它结点支距离被忽略不计的情况,这样极大影响适应值评价个体的真实效果.

(2)层接近度的计算量达到最小

本文方法路径的表示相当于把每个分支语句块作为一个结点,路径长度与分支语句的执行次数相同,是所有方法中最短的,从而计算层接近度比较的次数最少.其它文献对路径的表示方法各有不同,如有的按照路径经过的语句编号表示路径;有的按照分支语句及其后面的真假分支表示路径[7],这些表示路径的方法都明显比本文方法长,导致层接近度计算量加大.

(3)适应值的设计合理

本文层接近度计算不仅考虑编码比较时所有相同位的个数与位置,还考虑了从前向后连续相同位的个数,对个体真实评价起到了重要作用.保证无论针对哪个目标路径来说,较好的个体都会有较高的适应值,有利于指导种群的后续进化.

由此可知,从理论上分析,本文算法优于 Ahmed 方法及其它方法,下面通过实验验证方法的效率.

## 4　在基准程序测试中的应用

为了验证本文方法的性能,选择了三角形分类、求最大最小值、冒泡排序[7],以及三个数排序 4 个基准程序.它们的结构各不相同,如表 1($a$)所列.目前,运行一次 GA 找到多个路径的测试数据除了文献[7]中的 Ahmed 方法以外,还没有发现其它方法,该方法已验证优于单路径方法.每组实验在相同参数、相同的初始种群下使用本文方法和 Ahmed 方法各运行 100 次,比较两算法在找到覆盖相同目标路径测试数据的运行时间、进化代数等指标.所有程序均用 C 语言编写,在VC++ 6.0 环境下运行,使用机器主频是 2.80GHz,内存为 2GB,实验参数设置如表 1($b$)所列.

**表 1　程序说明及参数设置**

($a$)程序说明

| 程序名 | 分支语句及个数 | 分支语句构成 |
|---|---|---|
| 三角形分类[7] | 3 个选择 | 构成 3 层选择嵌套 |
| 求最大最小值[7] | 1 个循环、2 个选择 | 循环中嵌套 2 个并列的选择 |
| 冒泡排序[7] | 2 个循环、1 个选择 | 构成 2 层循环嵌套,内层嵌套中有 1 个选择 |
| 三个数排序 | 3 个选择 | 构成选择并列关系 |

($b$)参数设置[7]

| 参数 | 选择方式 | 交叉方式 | 交叉概率 | 变异方式 | 变异概率 | 基因编码 |
|---|---|---|---|---|---|---|
| 取值 | 轮盘赌 | 单点 | 0.9 | 单点 | 0.3 | 二进制 |

### 4.1　在三角形分类程序中的实验

首先,针对三角形分类程序的不同数据范围,目标路径选择全部 4 条可行路径,设定找到全部路径为结束条件.实验时记录每次找到各条路径的进化代数,并求 100 次进化代数的平均值.由于每次运行的进化时间只有几毫秒,所以只是记录了 100 次的总进化时间.结果如表 2 所列.表中的代数比和时间比均通过本文方法/Ahmed 方法得到.

**表 2　三角形分类程序的实验结果**

| 数据范围 | 种群规模 | 平均进化代数 | | | 总进化时间($s$) | | |
|---|---|---|---|---|---|---|---|
| | | 本文方法 | Ahmed 方法 | 代数比(%) | 本文方法 | Ahmed 方法 | 时间比(%) |
| $[0,100]^3$ | 30 | 24.8 | 63.7 | 38.9 | 0.153 | 0.312 | 49.0 |
| $[0,200]^3$ | 50 | 35.4 | 74.6 | 47.5 | 0.165 | 0.461 | 35.8 |
| $[0,500]^3$ | 80 | 46.8 | 352.3 | 13.3 | 0.627 | 3.122 | 20.1 |
| $[0,1000]^3$ | 100 | 70.4 | 512.6 | 13.7 | 1.123 | 6.782 | 16.6 |
| $[0,2000]^3$ | 200 | 77.2 | 629.0 | 12.3 | 2.812 | 20.695 | 13.6 |

从表 2 可以看出:(1)随着输入数据范围的增大,两种方法找到目标路径的进化代数和进化时间增加,这是由于输入数据范围的增大使得搜索难度加大所致;(2)对于不同的输入数据范围,两种方法的代数比和时间比不同,代数比和时间比最大的分别是 47.5% 和 49.0%,说明找到覆盖目标路径的数据,本文方法最多只需要 Ahmed 方法约一半的进化代数和进化时间;(3)随着输入数据范围的增大,时间比越来越小,说明本文方法在大输入数据范围下的优越性更明显.

现在考察找到覆盖每一个目标路径的测试数据的进化代数.为此,以输入数据范围[0,1000]为例,记录了

100次找到覆盖每一个目标路径的测试数据的进化代数,并计算其平均值,如表3所列.

**表3 覆盖每个目标路径的测试数据的平均进化代数**

| | 非三角形 | 非等边三角形 | 等腰三角形 | 等边三角形 |
|---|---|---|---|---|
| 本文方法 | 1 | 1 | 4.9 | 70.4 |
| Ahmed 方法 | 1 | 1 | 5.7 | 512.6 |

从表3可以看出:(1)覆盖非三角形和非等边三角形路径的测试数据容易寻找,两种方法均在第1代找到这些数据;(2)覆盖等腰三角形的测试数据也在较少的进化代数找到,两个方法差别不大;(3)覆盖等边三角形的测试数据难以寻找,本文方法平均需要70.4代才找到该数据;而Ahmed方法平均需要512.6代才找到该数据.由此可以看出,对于难以覆盖路径的测试数据寻找,本文方法在进化代数方面明显优于Ahmed方法.

## 4.2 在其它3个基准测试程序中的实验

为了进一步验证本文方法的性能,对其它3个不同结构的基准测试程序,选择输入数据范围在$[0,1000]^3$的情况进行了实验.

进行方法比较时,三个被测程序选择的可行路径数、种群规模如表4所列.实验以找到覆盖全部目标路径的测试数据作为算法终止条件,记录总的进化时间和平均进化代数,结果如表4所列.

**表4 其它3个基准测试程序的实验结果**

| 程序 | 种群规模 | 目标路径数 | 进化代数 | | | 进化时间(s) | | |
|---|---|---|---|---|---|---|---|---|
| | | | 本文方法 | Ahmed方法 | 代数比(%) | 本文方法 | Ahmed方法 | 时间比(%) |
| 三个数排序 | 100 | 7 | 23.8 | 46.9 | 50.8 | 0.467 | 0.928 | 50.3 |
| 冒泡排序 | 30 | 6 | 14.5 | 27.9 | 52.0 | 0.154 | 0.321 | 48.0 |
| 最大最小值 | 100 | 8 | 57.6 | 229.2 | 25.1 | 1.163 | 5.658 | 20.6 |

从表4可以看出对于所有的测试程序,本文方法的平均进化代数和进化时间都比Ahmed方法的少,其中,进化代数和进化时间方面比例最大的,本文方法分别是Ahmed方法的52.0%和50.3%,这说明,找到覆盖目标路径的测试数据,本文方法的进化时间和进化代数都约是Ahmed方法的一半,结果与三角形分类程序相符.这充分说明,对于4个基准测试程序,本文方法都优于Ahmed方法.

## 4.3 在复杂路径覆盖中的实验

前面的实验中基准程序的输入数据都是3个整数,选择的路径都是文献[7]中的路径,循环结构最多只运行2次,此外,在冒泡排序程序和求最大最小值程序中,只针对数组长度是3的情况.为了验证算法在路径结点更多、更复杂的情况下的性能,分别对冒泡排序程序和求最大最小值程序做了第三组实验.

首先考虑数组长度是5的情况.在冒泡排序程序的$2^{\frac{5\times(5-1)}{2}}=1024$个路径中选择5个可行路径,在求最大最小值程序的$2^{2\times(5-1)}=256$个路径中选择5个可行路径.两个程序的输入数据范围为$[0,1000]^5$,种群规模为30,最大运行代数为8000,计算找到每个目标路径的平均进化代数和100次的总进化时间,结果如表5所列.

**表5 数组长度为5的实验结果**

(*a*)求最大最小值程序平均进化代数和总进化时间

| | 路径1 | 路径2 | 路径3 | 路径4 | 路径5 | 总进化时间(s) |
|---|---|---|---|---|---|---|
| 本文方法 | 1 | 1 | 1.1 | 2.6 | 21.8 | 1.645 |
| Ahmed 方法 | 1 | 1 | 74.3 | 151.0 | 167.5<br>(9次失败) | 7.229 |

(*b*)冒泡排序程序平均进化代数和总进化时间

| | 路径1 | 路径2 | 路径3 | 路径4 | 路径5 | 总进化时间(s) |
|---|---|---|---|---|---|---|
| 本文方法 | 1.5 | 2.4 | 3.5 | 7.7 | 18.2 | 0.163 |
| Ahmed 方法 | 1.3 | 2.5 | 4.7 | 22.3 | 29.6 | 0.321 |

从表5可以看出,对于复杂路径,本文方法在进化代数和进化时间方面的性能也优于Ahmed方法,尤其是在最大值最小值程序中体现的更加明显,100次实验过程中,Ahmed方法有9次没有找到第5条目标路径.

继续将数组长度增加到10,这时,冒泡排序程序的路径长度为45,总路径数为$2^{45}$个;求最大最小值程序路径长度为18,总路径数为$2^{18}$个.分别选择5个可行路径进行实验,输入数据范围均为$[0,1000]^{10}$,种群规模为100,最大运行代数为50000,进化时间为每次运行时间平均值,其它内容与表5相同,结果如表6所列.

**表6 数组长度为10的实验结果**

(*a*)求最大最小值程序平均进化代数和平均进化时间

| | 路径1 | 路径2 | 路径3 | 路径4 | 路径5 | 进化时间(s) |
|---|---|---|---|---|---|---|
| 本文方法 | 2.5 | 8.1 | 9.6 | 18.5 | 37.1 | 0.027 |
| Ahmed 方法 | 27.3 | 212.7 | 227.2 | 6425.3 | 31454.9 | 7.724 |

(*b*)冒泡排序程序平均进化代数和平均进化时间

| | 路径1 | 路径2 | 路径3 | 路径4 | 路径5 | 进化时间(s) |
|---|---|---|---|---|---|---|
| 本文方法 | 42.8 | 147.6 | 222.5 | 341.8 | 430.4 | 0.414 |
| Ahmed 方法 | 1120.9 | 24402.6 | — | — | — | 20.548 |

从表6中求最大最小值程序可以看出,对于更加复杂的路径,本文方法在进化代数和进化时间上显著优于Ahmed方法.尤其是冒泡排序程序,本文方法能在平均430.4代找到分别覆盖5条目标路径的测试数据,平均需要的时间仅为0.414s,而Ahmed方法用20.548s到50000代停止进化时只能找到2条目标路径.这充分说明适应值函数的设计对测试数据生成的重要性.

本组实验验证了对解决长且复杂的路径覆盖测试数据生成问题,本文方法优越性是十分明显的.

# 5 结束语

寻求测试数据生成的有效方法,一直是软件测试

研究的重要内容之一.本文提出对目标路径采用赫夫曼编码,设计合适的适应度函数,是成功应用 GA 的关键.将设计的方法应用到 4 个典型的基准测试程序,与 Ahmed 方法的比较结果说明,本文方法在进化代数和进化时间方面均具有优越性.本文结果丰富了基于进化优化的软件测试理论和方法,提高了软件测试效率.

本文采用的编码方法在路径较长并且目标路径较多时,受到不同编程语言对字符串长度的限制,一个字符串不能存放全部目标路径的编码,此时,如何存放目标路径,是需要进一步研究的问题.另外,针对不同的被测试程序,自动生成路径编码,并自动插桩程序,以提高软件测试的自动化程度,也是需要进一步研究的问题.

**参考文献:**


[1] 邱晓康,李宣东.一个面向路径的软件测试辅助工具[J].电子学报,2004,32(12A):231 - 234.
Qiu Xiaokang, Li Xuandong. A path-oriented tool supporting for testing[J]. Acta Electronica Sinica, 2004, 32(12A): 231 - 234. (in Chinese)

[2] 单锦辉,王戟,齐治昌.面向路径的测试数据自动生成方法述评[J].电子学报,2004,32(1):109 - 113.
Shan Jinjui, Wang Ji, Qi Zhichang. Survey on path-wise automatic generation of test data[J]. Acta Electronica Sinica, 2004, 32(1): 109 - 113. (in Chinese)

[3] Chen Y, Zhong Y. Automatic path-oriented test data generation using a multi-population genetic algorithm[A]. Proceedings of the 4th International Conference on Natural Computation[C]. Jinan, China: IPICNC, 2008. 566 - 570.

[4] Sofokleous A A, Andreou A S. Automatic, evolutionary test data generation for dynamic software testing[J]. The Journal of Systems and Software, 2008, 81(11): 1883 - 1898.

[5] Rajappa V, Biradar A, Panda S. Efficient software test case generation using genetic algorithm based graph theory[A]. Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology[C]. Nagpur, India: IPICETET, 2008. 298 - 303.

[6] Bouchachia A. An immune genetic algorithm for software test data generation[A]. Proceedings of the 7th International Conference on Hybrid Intelligent Systems[C]. Washington, DC, USA: IP IC HIS, 2007. 84 - 89.

[7] Ahmed M A, Hermadi I. GA-based multiple paths test data generator[J]. Computer & Operations Research, 2008, 35(10): 3107 - 3124.

[8] 严蔚敏,吴伟民.数据结构(C 语言版)[M].北京:清华大学出版社,2007.

[9] 夏辉,宋昕,王理.基于 Z 路径覆盖的测试用例自动生成技术研究[J].现代电子技术,2006(6):92 - 94.
Xia Hui, Song Xin, Wang Li. Research of test case auto generating based on Z path coverage[J]. Modern Electronics Technique, 2006(6): 92 - 94. (in Chinese)

[10] David W Binkley. FlagRemover: A testability transfor-mation for transforming loop assigned flags[J]. ACM Transactions on Software Engineering and Method-ology, 2009, 2(3): 110 - 146.


**作者简介:**

**巩敦卫** 男,1970 年 3 月出生于江苏铜山,博士,教授,博士生导师,主要研究方向:基于搜索的软件工程、智能优化与控制.
E-mail: dwgong@vip.163.com

**张 岩** 女,1972 年 5 月出生于黑龙江集贤,副教授,博士研究生,主要研究方向:基于搜索的软件工程.
E-mail: zhangyancumt@126.com