基于二叉树搜索空间缩减的测试数据生成*

邵 楠, 周雁舟, 惠文涛, 乔 辉
(信息工程大学, 郑州 450004)

摘 要: 为了减小适应度函数计算量,提高测试数据自动生成效率,提出一种基于二叉树表示的搜索空间数据缩减方法。利用二叉树编码,记录全空间中的覆盖路径和路径长度;将目标路径和测试路径长度进行对比,去除路径长度相差较大的路径;利用遗传算法生成测试数据并同已有两种方法进行比较。实验结果表明,在保证软件测试数据正确生成的情况下,该方法在进化代数和运行时间上有明显优势,生成测试数据效率高。

关键词: 软件测试; 二叉树; 测试数据; 路径覆盖; 空间缩减
中图分类号: TP311.56 **文献标志码**: A **文章编号**: 1001-3695(2014)01-0188-04
doi:10.3969/j.issn.1001-3695.2014.01.043

Test data generation based on binary tree search space reduction

SHAO Nan, ZHOU Yan-zhou, HUI Wen-tao, QIAO Hui
(*Information Engineering University*, *Zhengzhou* 450004, *China*)

Abstract: In order to reduce the calculations of fitness function and improve the data generating efficiency, this paper proposed a method of reducing data in whole space based on binary tree. The method used binary tree code to record the overlay path and the path length of each datum; compared the program paths with the target paths, left out the paths with larger path length difference; compared the data generated by genetic algorithm with other two methods. As is shown in the experiments, this method can improve the data generating efficiency, and has a great advantage in improving algebra and the running time in ensuring the generation of correct test data.

Key words: software testing; binary tree; test data; path coverage; space reduction

软件测试是保障软件质量的重要手段[1],需要消耗大量的人力和财力。自动测试数据生成是减少软件测试花费的一个重要手段,进一步提高自动测试数据生成效率成为近年来研究的热点问题。路径测试是测试数据自动生成问题领域中的难题之一,遗传算法在解决路径覆盖问题时有独特的优势。遗传算法作为一种启发式的搜索寻优算法,被国内外学者广泛研究并应用到基于路径测试数据自动生成方面。Ahmed 等人[2]将遗传算法应用到多路径测试数据生成领域,此方法将多路径覆盖测试的数据生成问题看成多目标优化问题,在测试数据生成方面有较大的优势;姚香娟[3]提出多路径覆盖测试数据生成问题数学模型,认为多路径覆盖测试问题可看做包含多个目标函数的优化问题,但又区别于一般多目标优化问题,较 Ahmed 方法有所提高;巩敦卫等人[4]利用哈夫曼编码(Huffman coding)将目标路径表示为二进制数,一次运行算法生成全部可行路径的测试数据。

由于在算法运算过程中,程序的输入空间与遗传算法的种群规模有很大的关系,可以得出:输入空间的大小可以在一定程度上反映算法搜索的效率。同时,在算法个体选择上要使用适应度函数[5],它是种群中个体优劣的量化反应。目前的适应度函数的计算量比较大,如果能够在计算适应度函数之前删除一些明显无法覆盖目标路径的输入数据,会大大加快算法的执行速度,提高测试数据生成的效率。Harman 等人[6]在基于搜索的测试数据问题中考虑了搜索空间缩减问题,随后在文献[7]提出了在基于搜索的测试数据生成中减少不相关数据的策略,通过实验证明了在输入变量中减少不相关数据策略对随机算法没有影响,但对于爬山算法、遗传算法等生成测试数据效率有明显提高。Korel[8]在提出动态改变生成测试数据时,利用变量影响路径走向固定数据的做法,其实就是一种搜索空间缩减的办法。张岩等人[9]提出确定目标路径与输入变量之间的关系,将可分目标路径分离出与部分分量相关的子路径,固定被穿越子路径对应的输入分量从而缩减搜索空间。

针对多路径覆盖和测试数据缩减问题,提出一种用二叉树表示程序路径的方法,全空间数据执行程序,一次执行可以覆盖多条路径,同时针对目标路径长度筛选与其相似长度的测试路径。这样避免了输入空间数据的重复运行,减少了适应度函数的计算量,在三角形分类程序和冒泡程序测试数据生成问题的应用中,与已有方法相对比,验证了所提方法的优越性。

1 目标路径的二叉树编码

1.1 二叉树编码

在计算机科学中,树是一种重要的非线性数据结构,直观地看,它是数据元素(在树中称为节点)按分支关系组织起来的结构。二叉树是每个节点最多有两个子树的有序树,通常子树被称为左子树(left subtree)和右子树(right subtree)。逻辑

收稿日期: 2013-04-14; **修回日期**: 2013-05-26 **基金项目**: 武器装备预研重点基金资助项目(9140A15060311JB5201)

作者简介:邵楠(1988-),男,山西运城人,硕士研究生,主要研究方向为软件测试、测试数据自动生成(we713we@163.com);周雁舟(1971-),男,副研究员,博士,主要研究方向为软件可靠性、软件测试;惠文涛(1990-),男,硕士研究生,主要研究方向为软件测试、数据挖掘;乔辉(1988-),男,硕士研究生,主要研究方向为软件可靠性、数据挖掘.

上二叉树有五种基本形态,如图 1 所示。

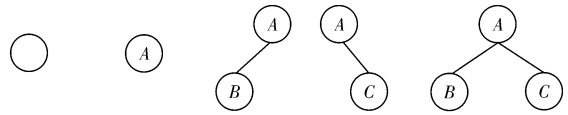
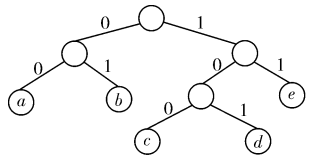


图 1　二叉树五种基本形态

由于二叉树只分左右两个子树,可以用 0,1 对二叉树进行编码。其编码方法是:用二叉树的叶子节点表示字符,对二叉树中每个分支节点的左、右分支分别用 0 和 1 进行编码,从树的根节点到叶子节点,沿途路径上所有 0 和 1 组成的编码序列作为该叶子节点所代表字符的二进制编码。图 2 中分别给出了 a、b、c、d 和 e 五个叶子节点的表示方法。



(a)一棵编码二叉树

$a:00$
$b:01$
$c:100$
$d:101$
$e:11$

(b)二进制编码

图 2　二叉树的编码

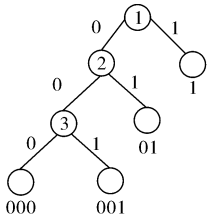
结构化程序由顺序结构、选择结构和循环结构组成。其中顺序结构和选择结构较易使用二叉树表示出来,循环结构可使用 Z 路径覆盖方法[10],可以看做多个分支结构并列,如果后面分支和前分支属于并列关系,可以将后一分支放到前一分支的子路径上。这样,可以将结构化程序转换为用二叉树表示的路径。

这里使用二叉树来表示许多文献使用过的基准程序(三角形分类程序),Koral[11] 和傅博[12] 都将此程序作为检验路径的范例。这是因为三角形分类程序含有清晰而又复杂的逻辑,且在较大范围的整数作为输入,只有少量输入组合能够满足某些特定分支。这些特性能检验测试效率,因此三角形分类程序被广泛地应用在软件测试研究中。

图 3 中,(a)为三角形分类源代码,1、2、3 个节点是分支谓词;由图 3(b)可以得出非三角形路径编码为 1,非等边三角形路径编码为 01,等边三角形路径编码为 001,等腰三角形路径编码为 000。

```
void triangle(int a,int b,int c)
  {if((a+b<c)&&(b+c<a)&&(a+c<b))
     printf("这不是一个普通三角形!");
   else
     if((a!=b)&&(a!=c)&&(b!=c))
       printf("这不是一个等边三角形!");
     else
       if((a==b)&&(a==c)
         printf("这是一个等边三角形!");
       else
         printf("这是一个等腰三角形!");
  }
```

(a)　三角形分类程序



(b)　二叉树表示的三角形路径编码

图 3　示例

1.2　被测试程序的插桩

对测试程序进行插桩的目的是为了记录被测数据在程序中运行的路径。这里,使用一个二维数组来记录被测数据运行路径,将数组插入源程序的一些关键路径,当程序运行到该语句时,对数组进行赋值,从而表明测试数据的运行路径。本文使用与巩敦卫[4] 不同的路径表示方法,用字符型数组 $s[i][j]$ 来记录第 i 个输入数据的第 j 个分支的走向。其中,i 值为被测数据的数量,j 的大小是二叉树的深度。使用 $s[i][0]$ 来记录当前数据的路径长度。二维数组的赋值为空字符'\0',插桩时第一个输入数据的赋值如下:

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

下面程序对三角形分类程序进行插桩,数组 $s[i][j]$ 不仅记录了输入数据的路径还记录了该路径的长度,为后一步缩减输入数据空间做好了准备。以下程序是对图 3 三角形分类程序进行了插桩。

```
char s[10][5] ={'\0'};//存放路径编码,假设有 10 个输入数据
int i =0;
void triangle( int a,  int b,  int c)
  {if( ( a + b < c) &&( b + c < a) &&( a + c < b) )
    {s[ i][1] ='1'; s[ i][0] =1;  printf( "这不是一个三角形!" ); }
    else {s[ i][1] ='0';  s[ i][0] =1;
      if( ( a! = b) &&( a! = c) &&( b! = c) )
        {s[ i][2] ='1'; s[ i][0] ++; printf( "这不是一个等三角形!" ); }
        else { s[ i][2] ='0'; s[ i][0] ++;
        if( ( a == b) &&( a == c) )
          {s[ i][3] ='1'; s[ i][0] ++; printf( "这是一个等边三角形!" ); }
        else
          {s[ i][3] ='0'; s[ i][0] ++; printf( "这是一个等腰三角形!" ); }
  }}
   i ++; //开始输入第二个数据
  }
```

2　基于二叉树搜索空间缩减的测试数据自动生成

2.1　搜索空间缩减思想

在复杂软件路径覆盖测试数据生成问题中,由于出现大规模路径覆盖,输入空间的缩减有现实意义。

对于测试数据的生成问题,目前通常有如下步骤:

a)构建程序目标路径图,可以帮助测试者选择目标路径;

b)选择目标路径——最重要的路径应该是要考虑的目标路径;

c)构建适应度函数——适应度函数是用来计算实际测试数据穿过的实际路径和目标路径的距离;

d)程序插桩——在每一段程序代码中插桩来记录程序的执行情况;

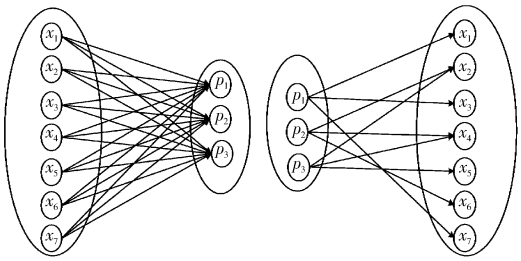
e)测试数据形成和执行插桩程序——初始化测试数据,进化过程选择最好的数据来实现目标路径。

假设被测程序为 P,输入变量为 $(x_1,x_2,x_3,\cdots,x_n)$,m 个目标路径分别为 $p_1,p_2,p_3,\cdots,p_m$。这里 m 和 n 都是比较大的数,不失一般性。$x_i(1<i<n,m<n)$ 为输入数据,运行程序后覆盖的测试路径为 $p(x_i)$,穿越 $p_j(1<j<m)$ 路径的输入数据为 x^*。若 x_i 和 x^* 相等,则 $p(x_i)$ 和 p_j 相同。考虑到有 m 个目标路径,则至少需要 m 个数据分别穿越目标路径。

在传统的利用遗传算法生成测试数据方法中,$(x_1,x_2,x_3,\cdots,x_n)$ 输入数据分别作为初始化数据,运行被测程序,通过适应度函数计算 $f(x_1)$,$f(x_2)$,$f(x_3)$,$\cdots$,$f(x_n)$ 选择下一代的数据。m 个目标路径则需要计算 $m(f(x_1),f(x_2),f(x_3),\cdots,f(x_n))$ 次适应度函数。

本文方法是利用第 1 章提到的 $s[i][j]$ 二维数组来记录测试数据的覆盖路径及长度。输入数据 $(x_1,x_2,x_3,\cdots,x_n)$ 分别运行被测程序,$s[n][0]$ 记录 n 个输入数据的覆盖路径长度。m 个目标路径分别为 $p_1,p_2,p_3,\cdots,p_m$,其长度分别为 $L(1)$,$L(2)$,$L(3)$,$\cdots$,$L(m)$,p_i 路径长度为 $L(i)$,对 $L(i)$ 与 $s[n][0]$ 进行对比,找出长度相近的输入数据。这些数据对遗传算法的种群进行初始化。

用三角形分类程序来分析,假设有七个输入数据。传统方法是对每个目标路径所有数据都作为输入进行运算,如图 4(a)所示,七个输入数据时需要进行 21 次适应度函数计算,这里着重考虑的是输入数据,保证每个输入都能应用。本文方法是针对目标路径进行选择,如图 4(b)所示,假设每个路径分别有三个数的覆盖路径长度与当前路径相似,则不需要计算其他四个数据的适应度函数。



(a)传统方法 (b)本文方法

图 4 两种方法的对比

三角形分类程序路径长度的比较如下:首先输入空间的数据运行插桩后的程序,使用 $s[7][3]$ 记录;然后针对第 1 章提到的非三角形路径编码,普通三角形的编码为 1,非等边三角形路径编码为 01,等边三角形路径编码为 001,等腰三角形路径编码为 000。假设等边三角形和等腰三角形为目标路径,二者的路径长度为 3,在 s[7][0]中筛选路径长度为 2 和 3 的数据,删除路径为 1 的数据,从而得到输入数据,这些数据距离目标路径较近,距离覆盖目标路径的数据最近;最后利用遗传算法的变异和交叉运算来得到最后的测试数据。

2.2 遗传算法步骤

本节是在上文输入空间缩减的基础上,使用遗传算法解决路径覆盖测试数据生成问题。算法终止条件是:当找到能覆盖目标路径的数据或者达到最大进化代数时,算法结束。

这里使用的适应度函数为[7]

$$\text{fit}(t,i) = \text{approach\_level}(t,i) + \text{normalize}(\text{branch\_distance}(t,i))$$

$$\text{normalize}(d) = 1 - 1.001^{-d}$$

a)确定全空间输入数据范围,对被测程序进行静态分析,构建程序二叉树图,对被测程序进行插桩。

b)全空间输入数据执行被测程序,利用本文方法记录数据执行路径及路径长度。

c)利用遗传算法对数据进化,确定遗传算法的控制参数、种群大小,确定输入数据编码方式。

d)初始化种群,将目标路径长度相似数据赋值给种群。

e)判断是否有与目标路径完全吻合的路径,若有,保存该进化个体,转步骤 h)。

f)计算进化个体适应值,对种群进行筛选,保留适应度值较小的输入数据,删除不合理输入数据。适应度函数是遗传算法和具体问题的唯一接口,反映了被测程序实际路径同目标路径的距离。它是种群中个体优劣的一种量化反应。

g)进行选择、交叉和变异操作,生成新子代种群,转步骤 d)。

h)当达到进化代数时,停止进化,解码进化个体,输出测试数据。

2.3 算法性能分析

本文选用的进化算法为遗传算法。由文献[4]中对遗传算法的适应度函数的计算量分析可知,适应度函数的计算量很大。当程序较复杂时,大量输入数据运行被测程序的工作量很难接受。

假设遗传算法的种群大小为 m,当运行到 n 代时找到覆盖所有目标路径的测试数据。可得到需要运行的程序次数为 $m \times n$次,遗传算法中适应度函数的计算次数也为 $m \times n$ 次,算法的复杂度即为 $O(m \times n)$。下面分析本文方法算法的复杂度。

1)输入数据运行测试程序次数减少 按照原有方法,针对每个路径都有自己对应的适应度函数,假设有 t 个目标路径,平均运行 n 代找到测试数据,因此共需要运行 $t \times m \times n$ 次被测程序,算法的复杂度为 $O(t \times m \times n)$。运用二叉树表示方法,在每一代中只需要运行一次被测程序,需要运行 $m \times n$ 次,算法的复杂度为 $O(m \times n)$。复杂程序中存在多条路径,运行程序次数会大大缩减,测试数据生成效率也会有明显提高。

2)删除冗余数据,减少适应度函数计算量 输入数据运行被测程序后,根据上文介绍 $s[i][j]$ 二维数组会记录覆盖路径和长度,通过比较目标路径的长度可以选择那些和路径长度接近的输入数据作为遗传算法的输入数据,相当于在输入数据中进行了一次筛选。假设目标路径长度为 $L(i)$,在 $s[n][0]$ 搜索长度为 $L(i)$的路径,若存在长度相等的数据,直接计算适应度函数,适应度函数为最小值者为覆盖目标路径数据,此时算法计算复杂度为 $O(1)$;若未找到覆盖目标路径数据时,可以选择运行程序时路径长度为 $L(i-1)$,$L(i)$,$L(i+1)$的输入数据作为遗传算法的初始值,若无法覆盖初始种群,则选择路径长度为 $L(i-2)$和 $L(i+2)$的数据,依此类推覆盖全种群数据。这样就减少了冗余数据的适应度函数计算,提高了数据生产效率。

3 实验分析

为了验证本文方法的有效性,分别选择了具有代表性的基准程序:三角形分类程序(triangle)、冒泡排序程序(bubble sort)[13]来验证。将本文的基于二叉树的空间缩小的遗传算法与其他方法进行比较,遗传算法的种群大小、编码方式、交叉和变异概率不变,算法的终止条件是达到最大进化代数或者找出输入数据。

实验用机配置为:Intel Pentium 4 CPU 2.93 GHz,1 GB 内存,Windows XP 系统。被测程序均采用 Java 语言编写,在 Eclipse 环境下运行。遗传算法采用二进制编码方式,轮盘赌选择方式,交叉概率为 0.85,变异率为 0.05,单点交叉方式。

3.1 三角形分类程序

在此程序中,遗传算法的输入变量为 0 ~ 500,种群大小为 200,最大迭代次数为 1 000。采用 24 bit 二进制数对三个输入边分别编码。P1 为普通三角形,P2 为等腰三角形,P3 为等边三角形。表 1 中传统方法是指全空间的遗传算法。目前,在测试数据自动生成中比较不同方法性能主要从两方面来衡量:找到覆盖目标路径测试数据所需要的代数和运行时间。在不同方法中,迭代次数和运行时间越小则效率越高,方法的性能越好。

表 1 不同方法的性能

路径	平均迭代次数			平均运行时间/ms		
	随机法	传统方法	本文方法	随机法	传统方法	本文方法
P1	4	3	5	98	143	156
P2	170	25	16	436	124	142
P3	119	17	14	598	298	176

从表 1 可以看出,就路径 P1 而言,三种方法在迭代次数上差别不大,但运行时间上本文方法却较最简单的随机法花费时间长,原因是 P1 路径是覆盖普通三角形,此测试数据较易得到,而本文方法却需要输入空间中所有数据运行程序,因此要花费大量时间。但对于 P2 和 P3 两个比较难覆盖路径进行测试则可看到,无论是迭代次数还是运行时间,本文方法都较随机法和传统遗传算法有明显优势。

以覆盖 P3 路径为目标,测试不同种群大小下上述三种方法找到测试数据所需要的代数。

从图 5 中可以看出,随着种群数的不断增大,随机法生成覆盖路径数据的代数达到最低点后会继续增长,这是因为随机法是无规律的种群进化方法,种群较大时容易在无用数据间徘徊。传统的遗传算法和本文提出的二叉树空间缩减方法都能够在种群增大时生成测试数据代数减小,在小规模种群时两种方法性能相差无几,但随着种群数增大,本文方法较传统方法有明显优势。

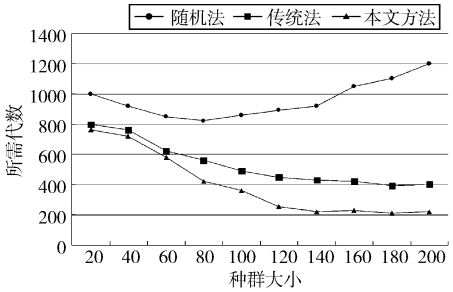


图 5　各种方法的对比

为进一步说明本文方法的优越性,三角形分类程序同文献[2,4]中的方法相比较(两种方法分别记为 Ahmed 方法和 GONG 方法),结果如表 2 所示。

表 2　三种不同方法的性能对比表

程序	方法	平均进化代数	平均进化时间/ms
P2	本文方法	6.4	0.53
	Gong	6.9	0.98
	Ahmed	12.2	2.11
P3	本文方法	10.4	1.34
	Gong	11.2	2.01
	Ahmed	18.8	4.32

从表 2 可以看出:a)在平均进化代数方面,本文方法代数只需 Ahmed 方法的一半,较 Gong 方法相当;b)在平均进化时间方面,本文方法对 Ahmed 方法有明显优势,约为 Gong 方法一半左右。这是因为 Gong 方法中适应度函数只考虑层接近度,在层接近度相近时适应度函数无法指导个体进化。而本文方法使用经典适应度函数计算避免了层接近度相近时出现的问题,同时利用输入空间缩减技术避免了计算浪费,节省了时间,从而更好地指导进化。

3.2　冒泡排序程序

利用本文方法对较复杂的冒泡排序程序进行分析验证。冒泡排序是假设有 N 个数据的数组,第一个数据比后一位数据大(升序,小则降序),依次比较 $N-1$ 次,最大(最小)数到最后,这样依次比较第二个数,直到排序结束。在冒泡排序程序中,也使用三个可执行目标路径进行算法验证。输入的范围为 0 ~ 500,种群规模为 200,最大运行代数为 50,找到覆盖目标路径的测试数据和达到最大运行代数停止运行。

从表 3 可以看出,对于复杂路径,利用本文的方法较传统方法在进化代数方面有明显优势。

表 3　冒泡排序程序平均进化代数

方法	路径 1	路径 2	路径 3
传统方法	2.9	24.4	27.3
本文方法	2.4	7.4	17.8

4　结束语

路径测试数据自动生成是程序结构测试的一个重要领域。本文利用二叉树图表示程序路径,提出用二维数组来记录数据覆盖路径及长度,在运行遗传算法中,删除和目标路径长度相差较大的输入数据,减少了不必要的适应度函数计算,使算法初始化数据较为理想,减少了遗传算法的进化代数,从而提高了测试数据的生成效率。本文分别对三角形分类程序和冒泡排序程序两个基准程序进行测试,并与其他方法进行比较,实验结果表明,本文所使用的方法在进化代数和运行时间方面有明显优势。需要指出,本文方法是基于程序二叉树图基础上进行的,下一步需要在程序自动转换二叉树图上进行研究,提高转换效率。

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.

[2] AHMED M A, HERMADI L. GA-based multiple paths test data generator[J]. Computer & Operations Research,2008,35(10):3107-3124.

[3] 姚香娟. 复杂软件测试数据进化生成理论及应用[D]. 徐州:中国矿业大学,2011.

[4] 巩敦卫,张岩. 一种新的多路径覆盖测试数据进化生成方法[J]. 电子学报,2010,38(6):1304-1329.

[5] ANDREA A. It does matter how you normalize the branch distance in search based software testing [C]//Proc of the 3rd International Conference on Software Testing, Verification and Validation. 2010:205-214.

[6] HARMAN M, HASSOUN Y, LAKHOTIA K, *et al*. The impact of input domain reduction on search-based test data generation[C]//Proc of ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York:ACM Press,2007:155-164.

[7] McMINN P, HARMAN M, LAKHOTIA I,*et al*. Input domain reduction through irrelevant variable removal and its effect on local, global and hybrid search-based structural test data generation[J]. IEEE Trans on Software Engineering,2012,23(4):453-477.

[8] KOREL B. Automated software test data generation [J]. IEEE Trans on Software Engineering,1990,16(8):870-879.

[9] 张岩,巩敦卫. 基于搜索空间自动缩减的路径覆盖测试数据进化生成[J]. 电子学报,2012,40(5):1011-1016.

[10] 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, 2006,12(6): 92-94.

[11] KORAL B. Automated software test data generator[J]. IEEE Trans on Software Engineering,1990,16(8):870-879.

[12] 傅博. 基于模拟退火遗传算法的软件测试数据自动生成[J]. 计算机工程与应用,2005,41(12):82-84.

[13] 史娇娇,姜淑娟. 基于遗传算法的动态可变参数的测试数据自动生成工具[J]. 计算机科学,2012,39(5):124-127.