基于遗传算法的数据流测试用例自适应生成算法①

许 力, 陈江勇

(福建师范大学 网络安全与密码技术重点实验室, 福州 350007)

摘 要: 测试用例的设计是软件测试实施的首要环节, 对后期测试工作具有重要的指导作用, 也是提高质量软件的根本保证. 针对 Moheb R. Girgis 算法的不足, 通过引入分支函数和改进遗传算法中的自适应性, 提出一种改进的数据流测试用例的自动生成算法, 实验表明, 改进算法在收敛速度和覆盖率等关键性能上都有较明显提高.

关键词: 遗传算法; 数据流; 测试用例; 自动化测试

Data Flow Test-Cases Adaptive Generation Algorithm Based on Genetic Algorithm

XU Li, CHEN Jiang-Yong

(Fujian Normal University, Key Lab of Network Security and Cryptology, Fuzhou 350007, China)

Abstract: The design of test-cases is one of the most important parts of software testing, which play an important role in guiding the post-testing and also is the fundamental guarantee of quality software. For the shortcoming of method raised by Moheb R. Girgis, an improved genetic algorithm for the automatic generation of data flow test-cases was proposed by introducing the branch functions and adaptive genetic strategies. Experiments show that the improved algorithm has a more increase in the performance of convergence rate and coverage rate.

Key words: genetic algorithms; data-flow; test-cases; automated testing

1 引言

软件测试的难题之一就是在满足一定覆盖准则的前提下进行测试用例的自动生成. 其中测试覆盖准则作为判断测试停止的重要依据主要有: 基于控制流的测试覆盖准则和基于数据流的测试覆盖准则. 目前, 国内外学者针对控制流覆盖准则相继提出了各种测试用例自动生成算法[1-5]; 然而基于数据流测试覆盖准则的测试用例自动生成算法研究的相对较少, 针对数据流测试覆盖准则中的全使用路径准则, Moheb R. Girgis 提出了采用遗传算法来进行数据流测试用例的自动生成[6](Automatic testcase generation algorithm for data flow, 简记为 ATGAFDF) , 然而其遗传算法的适应度设计过于简单并不能十分有效区别出种群中优秀的个体, 使得算法的收敛速度缓慢, 同时, 其采用了固定的交叉率和变异率使得算法容易陷入局部最优解. 基于此本文在其遗传算法的基础上提出一种改进的数据流测试用例自动生成算法(Improved automatic testcase generation algorithm for data flow, 简记为 IATGAFDF).

2 数据流测试基本概念

数据流测试与传统单纯基于控制流图的结构化测试的区别在于:传统的结构化测试方法基本上是从纯数学的角度来分析的, 这种测试方法无法对程序进行静态分析而且相应的测试覆盖准则也难以保证测试的有效性和充分性. 而数据流测试则是利用了变量之间的关系, 通过定义-使用路径得到一系列的测试指标用于衡量测试的覆盖率. 程序中使用变量的方式有: 给变量赋常量值、通过其他多个变量产生另一个变量的值、使用变量来存取外部数据. 定义-使用测试主要通过将控制流图的节点和特定的变量相关联并根据相应覆盖准则来产生测试用例. 其中根据变量定义与使用位置的不同, 可以将定义-使用对分成三类: 函数内定义-使用对, 函数间的定义-使用对, 类级别定义-使用对[7]. 而测试准则是一套测试标准, 主要作为测试人员

① 基金项目:福建省高校产学合作科技重大项目(2011H6008);福建省自然科学基金(2011J5148)
收稿时间:2012-12-11;收到修改稿时间:2013-01-29

选择测试路径集合的依据. 根据测试用例覆盖定义-使用对情况的不同可以定义出 6 种数据流测试准则, 包括全定义、全使用等路径测试准则[8].

图 1 是三个数中求最大值的程序. 通过数据流测试建模分析方法可知这段程序有三个变量, 且有 15 个定义-使用对. 显然, 通过人工直接设计测试用例的方式难以完全覆盖这 15 个定义-使用对. 一种可行的方法是采用随机算法产生随机的 a,b,c 的值, 然后判断定义-使用对的覆盖情况. 然而随机算法往往盲目性较大, 难以在短时间产生满足条件的测试用例. 本文就是主要设计一种改进的数据流测试用例生成自动生成算法 IATGAFDF 来快速生成所需测试用例.

```
int getMax(int a, int b, int c){
    if(a>b&&a>c) return a;
    else if(b>a&&b>c) return b;
    else if(c>a&&c>b) return c;
}
```

图 1 getMax 代码

3 IATGAFDF算法的设计与实现

3.1 相关 Web 页面获取 3.1 IATGAFDF 算法的编码策略

使用 IATGAFDF 算法求解问题时, 首先须将目标解的实际表示与遗传算法中染色体二进制位串之间建立一种一对一的映射关系, 这个过程称为编码.

设一程序 P 有 k 个输入变量, 第 i 个输入变量表示为 x_i, 其中 $1\leqslant i\leqslant k$, $D_i(1\leqslant i\leqslant k)$表示个变量 x_i 的取值集合, 叫做 x_i 的定义域 $(x_i \in D_i)$ 而 $(x_1,x_2,\cdots,x_k)\subset(D_1,D_2,\cdots,D_k)$ 表示一个向量, 该向量表示程序 P 的一个测试输入或者叫做测试用例.

1) 数值型变量编码

若变量 x_i 的取值集合 $D_i=[a_i,b_i]$, 精度要求为 d_i, 则 D_i 可以离散化成包含 $(b_i-a_i)\times 10^{d_i}$ 个元素的集合. 若 m_i 是满足条件 $(b_i-a_i)\times 10^{d_i}\leq 2^{m_i}-1$ 的最小整数, 则变量 x_i 可以用长度为 m_i 的二进制编码串 $s_i=l_m l_{m-1}\cdots l_2 l_1(l_j\in\{0,1\},1\leq j\leq m_i)$ 表示. 二进制编码串 s_i 到变量 x_i 的映射(即解码)可以用如公式(1)计算:

$$x_i=a_i+\frac{\sum_{i=1}^{m} l_{m_i}\cdot 2^{i-1}}{2^{m_i}-1}(b_i-a_i) \tag{1}$$

2) 非数值变量编码

布尔型变量: 无需进行编码, 因为其只有真和假两种可能的取值.

字符型变量: 字符型变量可以用其 ASCII 码换成二进制串进行编码.

字符串变量: 字符串可以看成是多个字符的连接, 故字符串编码用其所包含字符编码的连接. 也可以根据等价类划分或边界值分析等方法先将字符串进行分类, 然后用索引来代表字符串, 只对索引进行编码.

3.2 IATGAFDF 算法的适应度函数

在测试路径覆盖准则下, 评价一个测试用例(染色体)的好坏必须将测试数据输入到程序中执行并记录测试路径的覆盖率. 若根据数据流测试覆盖准则计算出的路径共有 n 条, H={h_1,h_2,……, h_n}, 而某个测试用例 t 覆盖了其中的 m 条测试路径, 则测试用例 t 的适应度函数可以用公式(2)计算:

$$fitness(t)=\frac{m}{n} \tag{2}$$

ATGAFDF 算法即采用的这种适应度计算公式, 并且其采用了全使用路径覆盖准则, 所以相应的适应度函数可以具体表示为公式(3).

$$fitness(t)=\frac{no.of\ \ def-use\ \ paths\ \ \text{cov}erd\ \ by\ \ t}{total\ \ no.of\ \ def-use\ \ paths} \tag{3}$$

公式的含义是一个测试用例(染色体)覆盖的测试路径条数越多其适应度值也就越大. 但是这种适应度函数的计算完全忽略了分支谓词对测试路径的影响. 随着分支取真值和取假值的不同, 测试用例覆盖的测试路径是决然不同的, 因此本文提出一种改进的适应度计算方法.

适应度函数计算的改进思想在于: 对于一个测试用例 t 来说, 其覆盖了一些测试路径, 而对于那些未能覆盖的测试路径来说, 主要是因为这些路径上的某个分支条件取值为假导致在测试路径上的后续语句得不到执行的机会, 否则对于一个顺序结构的程序来说, 每条语句肯定会按顺序依次被执行. 所以, 如果只是简单的将已经覆盖的测试路径数来计算适应度函数忽略了分支条件的取值对测试路径的贡献, 不仅降低了算法的收敛速度而且容易使算法陷入局部最优解中. 图 2 是某个程序片段的控制流图.

从图中可以看出在条件语句 if (a > b)取假的情况下, 包含节点 3、4、5、6 的所有路径都无法被覆盖. 假如对于两个测试用例 t_1,t_2, 他们覆盖相同的测试路径,

并且在节点 2 到节点 3 上的条件语句都为假, 即 a≤b. 但是对于测试用例 t_1 来说其 a 的值更接近 b 的值, 则我们认为测试用例 t_1 的适应度值更好, 因为其离 a>b 的条件期望值更近, 所以其更可能在染色体的交叉变异过程中产生满足条件语句为真的染色体, 从而覆盖包含节点 3、4、5、6 的测试路径. 因此, 在 IATGAFDF 算法中引入分支函数来改善相应的适应度计算.

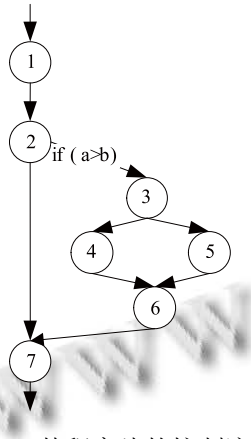

图 2　某程序片的控制流图

在控制流图中, 每个分支都可以用一个条件表达式来表示, 该条件表达式称为分支谓词, 其作用是描述了程序遍历该分支下语句的条件, 如判断语句“if(a>b)...”中分支谓词为“a>b”. 分支谓词的一般形式可表示为: A op B, 其中 $op \in \{<,\leq,>,\geq,==,!=\}$ 是关系运算符, A 和 B 是表达式. 为了便于计算适应度需要将分支谓词统一转化与 0 的关系表达式, 即 F rel 0, 其中 F 为分支函数, 分支函数的构造方法如表 1 所示.

表 1　分支函数的构造

分支谓词	分支函数	rel
A>B	B-A	<
A≥B	B-A	≥
A<B	A-B	<
A≤B	A-B	≤
A==B	abs(A-B)	==
A!=B	-abs(A-B)	<

由上表可知, 当分支谓词为真时, 则分支函数为负; 当分支谓词为假时, 则分支函数为正. 而覆盖某条路径, 则需满足该路径上的所有分支谓词应为真, 此时分支函数应为负. 由于分支函数直接构成了染色体适应度值的一部分, 不应该为负, 故可将分支函数修正为: 当分支谓词为真时, 则分支函数取零; 当分支谓词取假时, 分支函数即为计算出的分支函数值.

假如测试用例(染色体) t 已经覆盖的路径条数为 m, 总的需要覆盖的路径条数为 n, 在未覆盖的 $n-m$ 条路径中包含的被执行谓词数量为 p, 则适应度函数如公式(4)所示.

$$fitness(t)=(1-\alpha)\frac{n}{m}+a\cdot\frac{\sum_{i=1}^{p}\frac{1}{1+f_i(x)}}{p} \qquad (4)$$

其中 α 是分支谓词对适应度函数的影响权重因子, 取 0.5, $f_i(x)$ 是第 i 个分支的分支函数值.

3.3 IATGAFDF 算法的遗传策略

1) 选择策略

遗传算法中常用的染色体选择方法有轮盘赌转法、随机抽样法等, 其中以轮盘赌转法最为常见. IATGAFDF 算法即采用了轮盘赌转法.

2) 交叉策略

交叉运算主要作用是产生更好的新个体, 目前常用的交叉运算有单点交叉、两点交叉、均匀交叉等. 在 IATGAFDF 算法中采用的是单点交叉的方法.

3) 变异策略

变异操作增强了种群的多样性, 防止个体的早熟现象. 通常, 变异率 P_m 取值较小, 为 0.01 至 0.2 之间, 在 ATGAFDF 算法中取值为 0.15. 二进制编码中的变异方法有基本位变异、均匀变异、高斯变异和二元变异等. IATGAFDF 算法采用的是基本位变异方法.

3.4 IATGAFDF 算法的自适应策略

在遗传算法中交叉率 P_c 和变异率 P_m 设置的好坏直接影响着算法的收敛速度[9]. 交叉率 P_c 保证了物种的多样性, 若设置得太大, 容易破坏适应度高的染色体, 破坏了遗传算法的模式; 若设置的太小, 又不容易产生新的染色体, 导致搜索速度缓慢. 同样, 若变异率 P_m 设置得太小则难以产生新的染色体, 若设置得太大, 导致对染色体破坏太大, 以至于遗传算法退化成盲目的随机搜索算法. 一种可行的办法是通过多次试验找到合适的 P_c 和 P_m 值, 但是这样太繁琐, 针对不同问题难以找到最优解. 因此, IATGAFDF 算法采用了一种自适应的遗传策略, 根据染色体的适应度值动态设置 P_c 和 P_m 的值. 当种群中的染色体趋于局部最优时, 增加 P_m 和 P_c 的值, 当种群中的染色体较为离散时, 减少 P_m 和 P_c 的值. 对于高于平均适应度的染色体设置较小的 P_m 和 P_c 值, 使得优良物种得以保留; 而对于低于平均适应度的

染色体设置较高的 P_m 和 P_c 值, 使得对应的个体被淘汰出去. 这样既保证了物种的多样性又保证了算法的收敛速度. 经过上述改进 P_c 和 P_m 的计算公式如(5)如下:

$$P_c=\begin{cases}p_{c1}-\dfrac{(p_{c1}-p_{c2})(f'-f_{avg})}{f_{\max}-f_{avg}} & f'\geq f_{avg}\\ p_{c1} & f'\prec f_{avg}\end{cases}$$
$$P_m=\begin{cases}p_{m1}-\dfrac{(p_{m1}-p_{m2})(f_{\max}-f)}{f_{\max}-f_{avg}} & f\geq f_{avg}\\ p_{m1} & f\prec f_{avg}\end{cases} \quad (5)$$

其中 f_{avg} 为每一代群体的平均适应度值, f' 为要交叉的两个个体中较大的适应度值, $f_{\max}$ 为群体中最大的适应度值, f 为要变异个体的适应度值.

3.5 IATGAFDF 算法的结束条件

通常物种将根据遗传算法策略不断的繁衍, 直到最优解出现. 在 IATGAFDF 算法中需要记录已经覆盖的测试路径数, 当所有的测试路径都被覆盖时则说明已经找到了最优解, 算法结束. 但是某些测试路径上的语句节点可能难以覆盖或者无法覆盖, 因此一般需要为遗传算法为预先设定的最大进化代数作为终止条件, 通常可以取 100 代到 400 代.

4 实验对比

选取 10 个小于 50 行的函数为测试对象, 然后按照要求对程序进行插桩[10], 接着分别按照 ATGAFDF 和 IATGAFDF 算法策略产生测试用例, 最后以数据流测试准则中的全使用路径测试准则作为覆盖标准, 将 IATGAFDF 算法与 ATGAFDF 算法进行对比, 其中(5)式中的几个参数分别设计如下: $P_{c1}=0.9$, $P_{c2}=0.6$, $P_{m1}=0.1, P_{m2}=0.001$, 将 10 组测试结果取平均值其结果汇总如图 3 所示.

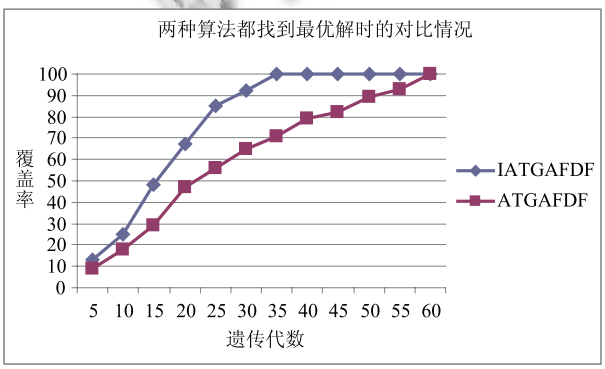

图 3 收敛速度对比

从图中可以看出, 在两种算法都未终止时, 相同的遗传代数 IATGAFDF 算法的测试覆盖率明显比 ATGAFDF 算法测试覆盖率高. 虽然遗传算法运行 60 代以后生成的测试用例集的覆盖率都达到了 100%, 但是明显 IATGAFDF 算法收敛速度比 ATGAFDF 算法快, 在遗传算法运行到 35 代左右, IATGAFDF 算法已覆盖所有的定义-使用对. 而 ATGAFDF 需要遗传到 60 代左右才完全覆盖目标路径.

为了进一步对比未能找到最优解时算法运行情况, 另外选取 10 个大于 100 行的函数作为测试对象并设置算法终止进化代数为 200 代. 整个测试结果汇总如图 4 所示.

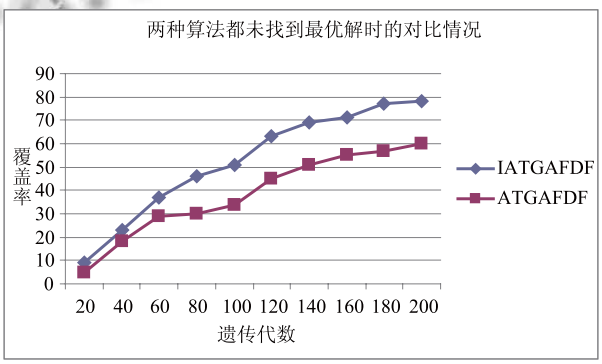

图 4 覆盖率对比

因为随着程序的复杂度的增加, 变量的定义-使用对将增加, 设计满足要求测试准则的测试用例将更加困难. 从图中可以看出, 两种算法运行终止时(遗传到 200 代)都未能找到最优解, 但是算法运行终止时 IATGAFDF 算法具有较高的测试覆盖率,达到了近 80%, 而 ATGAFDF 算法只有近 60%的覆盖率. 从实验结果可知无论是收敛速度或覆盖率 IATGAFDF 算法都具有较明显的优势.

5 结论

本文在遗传算法的基础之上提出了改进的用数据流测试用例自动生成算法 IATGAFDF, 设计 IATGAFDF 算法时考虑到判定语句对测试路径覆盖的影响, 引入了分支函数的概念,同时考虑到固定的交叉率和变异率会影响算法的收敛性, 于是引入了自适应的遗传策略, 这样交叉率和变异率可以随着染色体的适应度值的不同进行动态调整. 实验表明, IATGAFDF 算法具有更快的收敛性和更高的覆盖率.

参考文献

1 Harman M, Mcminn P, Wegener J. The impact of input domain reduction on search based test data generation. Antonia Bertolinoed. Proc. of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. New York: ACM Press, 2007: 155−156.

2 Zheng L, Harman M, Hierons RM. Search algorithm for regression test case prioritization. IEEE Trans. on Software Engineering, 2007,33(4):225−228.

3 Berndt D, Fisher J, Johnson, et al. Breeding software testcases with genetic algorithm. Proc. of the 36th Hawaii International Conference on System Sciences. IEEE Press, 2002:1−4.

4 Bryce RC, Colbourn CJ. Constructing interaction test suites with greedy algorithm. The Proc. of ASE'05. Califormina, ACM Press, 2005. 124−136.

5 荚伟,谢军凯,奚红宇,等.遗传算法在软件测试数据生成中的应用.北京航空航天大学学报,1998,24(4):434−437.

6 Girgis MR. Automatic test data generation for data flow testing using a genetic algorithm. Journal of Universal Computer Science, 2005,11(6):898−915.

7 Rapps S, Weyuker EJ. Selecting software test data using data flow information. IEEE Trans. on Software Engineering, 1985,11(4):367−375.

8 Frankl PG, Weyuker EJ. An applicable family of data flow testing criteria. IEEE Trans. on Software Engineering, 1988: 1483−1498.

9 王小平,曹立明.遗传算法-理论、应用与软件实现.西安:西安交通大学出版社,2002:19−67.

10 AHmed SG, Harrold MJ. Using genetic algorithms to aid test-data generation for data-flow coverage. Proc. of the 14th Asia-Pacific Software Engineering Conference. Nagoya: IEEE Press, 2007: 41−48.

(上接第 84 页)

信息检测的误码率更低.

参考文献

1 姚钟涵,王慧琴,毛力.一种基于回声隐藏的数字音频水印算法.微计算机信息,2008,24:1−1.

2 Xu C, Wu J, Sun Q, Xin K.Applications of watermarking technology in audio signals. Journal Audio Engineering Society, October,1999,47(10):1995−2007.

3 Oh HO, Kim HW, Seok JW. Transparent and robust audio watermarking with a new echo embedding technique. Multimedia and Expo, 2001:317−320.

4 Huang DY, Yeo TY. Robust and inaudible multi-echo audio watermarking. IEEE Pacific Rim Conference on Multimedia, 2002. 615−622.

5 Kim HJ, Choi YH. A novel echo-hiding scheme with backward and forward kernels. Circuits and Systems for Video Technology, 2003,13:885−889.

6 赵朝阳,刘振华,王挺.数字音频信号的回声隐藏技术.计算机应用研究,2000,17:42−44.

7 殷凯,周辉.音频数字水印中回声隐藏技术的时域提取方法.计算机与数字工程,2006,34(5).

8 段柳云,宗节保,等.一种基于回声隐藏的倒谱提取优化算法.电子设计工程,2010,18(7).

9 Yan B, Sun SH, Lu ZM. Improved echo hiding using power cepstrum and simulated annealing based synchronization technique. The 2nd Int. Conf. on Machine learning and Cybernetics. Xi'an, China, 2003,2(5):2142−2147.

10 Bender W, Gruhl D, Morinoto N, et al. Techniques for data hiding. IBM Systems Journal, 1996,35(3/4):313−316.

11 唐升,侯榆青,克兢.一种基于短时能量自适应的回声隐藏算法.计算机应用,2008,28(11).