利用函数影响力的相似程序间测试用例重用与生成

钱忠胜,宋 佳,俞情媛,成轶伟,孙志旺
(江西财经大学信息管理学院,江西南昌 330013)

摘 要: 在回归测试过程中,用例重用是一项很重要的工作,其充分利用软件升级变更前的已有资源,提高测试的效率. 从已有研究来看,回归测试的研究大部分侧重于用例优化方面,少部分提到利用程序升级变更前后的相似性来重用测试用例以提高用例生成效率. 针对回归测试用例重用问题,提出一种重用变更前相似程序的测试信息,并通过设计的适应度函数为变更后程序进化生成新用例的方法. 该方法利用构建的函数调用图进行程序相似部分的检测,并根据函数影响力设计适应度函数来调整个体的适应度值,保留适应度值高的优秀个体;再通过重用变更前相似部分的用例,以及进化生成的变更后的部分用例,来构成回归测试中新程序的用例. 实验结果表明,在目标路径覆盖率上,对于中小规模和大规模工业程序,本文方法比经典方法分别可提高 8% 和 17%.

关键词: 测试用例; 程序相似性; 函数影响力; 关键函数; 回归测试
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112(2022)07-1696-12
电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20210953

Test Case Reuse and Generation Between Similar Programs Based on Function Influence

QIAN Zhong-sheng, SONG Jia, YU Qing-yuan, CHENG Yi-wei, SUN Zhi-wang
(*School of Information Management, Jiangxi University of Finance & Economics, Nanchang, Jiangxi* 330013, *China*)

Abstract: In the process of regression testing, the reuse of test cases is a critical activity. It makes full use of the resources before updating the software to improve the efficiency of software testing. According to the existing results, most of them on regression testing focus on the test case prioritization, and a few mention the use of similarity before and after the programs upgrade changes to reuse test cases to improve the efficiency of test case generation. Aiming at the test case reuse in regression testing, this paper proposes an approach to reuse test data of similar programs before the change, and employs the fitness function designed to evolve to generate new test cases for the program after the change. It utilizes the function call graph constructed to detect similar parts of the programs. And according to the function influence, the fitness function is designed to adjust the fitness value of the individual, and excellent individuals with high fitness value are retained. Then, it reuses the test cases of the similar part before the change and the part of the test cases generated in the evolutionary process after the change, to form the test cases of the new program in the regression test. The experimental results show that, compared to the classical methods, the target path coverage rate of the proposed method is increased by 8% and 17% for small- and medium-sized, and large-sized industrial programs, respectively.

Key words: test case; program similarity; function influence; key function; regression test

1 引言

由于软件产业的飞速发展,人们对软件质量重要性的认识也逐渐增强. 软件测试在软件项目实施过程中的重要性日益突出. 寻找合适的测试用例是测试工作的重要任务,它可以提高软件的正确性、完整性、可靠性[1,2].

在测试中,回归测试是一项重要的测试活动,它是指对原来的程序进行变更后,重新对变更后的程序进行测试. 本文通过检测前后程序的相似部分,并重用相似部分的用例,提高回归测试的效率.

检测程序变更前后的相似性是本文回归测试中用例重用与生成的首要工作. 目前,不少学者提出程序相

收稿日期:2021-07-20;修回日期:2021-10-02;责任编辑:王天慧
基金项目:国家自然科学基金(No.61762041);江西省自然科学基金(No.20181BAB202009)

似性检测方法. 其中,有的通过分析相关的控制流图来检验图神经网络估计程序相似度的有效性[3];有的通过分语言行筛选方法将代码降维,并对其特征进行匹配,以便快速判定相似程序[4];有的借用循环神经网络、卷积神经网络等方法学习程序特征,以此来构建模型[5]. 本文从结构和功能上,通过构建函数调用图检测程序变更前后的相似部分,为生成所需用例做准备.

测试用例自动生成已经取得了诸多进展,但测试资源未得到充分利用,导致在测试过程中仍存在一些重复工作. 测试用例重用是指再次组织和运用以往测试过程中已生成的测试数据. 与重新生成用例相比,重用已有测试用例不仅可充分利用已有资源,而且可减少测试所需时间.

本文通过程序变更前后的相似性检测方法获取已有用例并重用,来提高回归测试效率,主要展开以下工作:

(1)利用函数调用图检测程序变更前后相似性,识别可进行用例重用的相似函数,并将该部分的用例重用到变更后程序的测试中;

(2)经相似性检测后,程序不相似部分则通过改进遗传算法生成测试数据,这里,将函数影响力作为适应度函数,优化遗传算法,筛选优秀用例.

2 函数关键性的判定与程序相似部分的检测

根据函数在程序中的重要程度,对函数关键性进行判定,并根据函数的调用情况检测程序的相似部分.

2.1 函数关键性的判定

不同的函数在程序中扮演着不同的角色,在程序中有一些函数对程序的正常运行起着决定性作用,本文将这些函数称为关键函数(Critical Function,CF). 通过建立函数调用图,对函数在程序中的层次进行识别,并记录函数的调用程度,以此来识别函数的关键性. 按照每个关键函数的关键性由大到小排列,形成的序列为关键函数序列(Critical Function Sequence,CFS).

为更好地说明函数关键性的判定方法,下面定义一些相关概念.

定义1 序列(Sequence)[6]. 程序每个执行路径可看作一个序列,它是一个有序的事件列表,用S=<S_1,S_2,…,S_n>来表示. 序列中的每个元素S_i(1≤i≤n)由二元组(Flg;FName)组成,其中,Flg为一个事件的进出标识,即程序中的调用入口或出口,Flg="I"为调用入口,Flg="O"为调用出口;FName指函数名.

定义2 序列模式(Sequence Pattern,SP)[6]. 序列S由多个模式组成,此时的序列即序列模式,用SP=$\{P_1,P_2,\cdots,P_n\}$表示,其中,n是构成序列SP的模式数. 在每个模式下,函数的调用入口就像一扇门,从"I"进入此门,再从"O"出门,可知每个模式P的长度均为偶数.

定义3 压缩序列模式(Condensed Sequence Pattern,CSP)[6]. 在删除了序列模式SP中相邻的重复模式后,剩余的序列被称为压缩序列模式. 将CSP中的一级模式划分为不同级别的子模式,并将它们存储到压缩序列模式集SP_{csp}中.

在文献[6]模型构建方法的基础上,对函数调用关系进行预处理,过程如下.

(1)建立初始程序序列. 将函数地址抽象为函数名,并识别函数的进出调用,利用这2个程序特征对执行轨迹进行顺序建模.

(2)删除相邻重复模式. 为减少重复循环模式的时间和内存消耗,简化执行序列,在得到程序初始序列后,保留第一时间出现的重复序列中的一个模式,删除其他重复序列,可得到压缩序列模式. 具体过程如算法1所示.

算法1 重复模式的删除

输入: 序列模式(SP=$\{P_m,\cdots,P_n\}$)
输出: 压缩序列模式(CSP)

```
BEGIN
1  FOR P_i in SP={P_m,…,P_n}
2    IF (P_i.Flg == 'I')
3      FOR P_j in SP={P_m,…,P_n}
4        IF(P_j.Flg == 'O' &&P_j.FName == P_i.FName&&j>i)
5          SP_i←SP={P_i,…,P_j};   //识别程序序列模式
6          BREAK;
7        END IF
8      END FOR
9    END IF
10 END FOR
11 FOR each P in SP_i
12   IF adjacent repeated Q of P in SP_i  //若序列模式P存在相邻重复模式Q
13     Delete(Q);   //保留重复序列模式中第一个序列模式P,删除其他重复序列模式Q
14   END IF
15 END FOR
16 CSP←Combine each sequence pattern SP_i after condensing;
17 OUTPUT CSP;
END
```

(3)构建函数调用图(Function Call Diagram,FCD). 删除相邻重复模式后,利用序列模式级别来度量不同模式间的包含关系,而序列模式级别由模式中函数调入调出关系表示,由此构成函数调用图.

定义4 模式层级(Mode Level,ML). CSP层级关系中函数的层次级别,即调用期间所包含的其他函数

的出度和入度之差，且层级大的函数被层级小的函数调用.

在一个序列模式SP=$\{P_i,\cdots,P_j\}$中，P_i.Flg=I，P_j.Flg=O. 若一个序列模式的层次级别为ML，在另一个序列模式$SP_1=\{P_x,P_{x+1},\cdots,P_y\}$中，可找到$P_x$.Flg=$I$，$P_y$.Flg=$O$，则序列模式$SP_1$的层次级别是ML+$N$，其中，$N$为函数$P_a$.Flg=$I$和函数$P_b$.Flg=$O$在函数$P_i$和$P_x$间的函数个数，这里，$i<a<b\leqslant x$. 函数调用图的一个示例如图1所示.

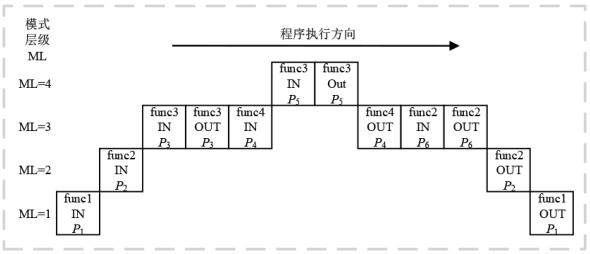


图1 函数调用图

例1 图1描述了长序列模式SP，它包含子模式P_1,P_2,P_3,P_4,P_5和P_6，可知SP=$\{P_1,P_2,\cdots,P_6\}$. 从序列模式的层次关系来看，若SP的序列模式层次为ML=1，则P_2,P_3,P_4,P_5和P_6的序列模式层次分别为ML=2，ML=3，ML=3，ML=4和ML=3. 例如，P_6的序列模式层次为ML(P_6)=1+5−3=3，其中，“1”表示序列模式本级；“5”表示函数func1和func2间的入度为5，即func2，func3，func4，func3和func2的函数调度标识是“I”；同理，“3”表示函数func1和func2间的出度为3，即func3，func3和func4的函数调用标识是“O”.

在程序执行序列模式集中，通过分析不同模式的函数可知，具有相同决策功能的函数越多，在不同情况下程序执行中函数调用的可能性越大. 可见，关键函数在程序执行中起着决定性作用.

设计一种基于函数调用图的检测算法来识别函数的关键性，主要分两步：首先，构建CSP的简单模式，消除重复模式，得出相应的函数层级关系；其次，统计在CSP中出现的有相同决策功能的函数，并计算它们在层级关系中出现的次数，根据函数的调用次数和层级来判定函数的关键性.

在CSP中的模式层级关系确定后，即可判定关键函数，具体过程如算法2所示.

在算法2中，第1~6行计算具有相同决策功能函数出现的次数，次数越多，说明函数越重要；第7行计算序列模式的层级，层级小的序列模式调用持续时间较长，这类调用被中断可能会影响整个程序的运行，由此看来层级越小的序列模式中函数的重要性越大；第8行根据函数的调用次数和序列模式层级将函数排序，即可得到根据函数的关键性从大到小排列的关键函数序列.

算法2 函数关键性的判定

输入: 函数调用图(FCD)

输出: 关键函数序列(CFS)

```
BEGIN
1  FOR Px, Py in SPcsp of FCD
2    IF (Px.CF == Py.CF)
3      Num = Count(Px.CF);
4      CFxy←Num;   //计算相同决策功能函数出现的次数
5    END IF
6  END FOR
7  Calculate(ML);   //计算序列模式层级
8  CFS←Sort(funcX);   //根据函数的调用次数和序列模式层级将函数排序
9  OUTPUT CFS;
END
```

例2 由图1可知，P_3和P_5是不相邻的相同模式，只需要归为一类序列模式，$SP_{csp}=\{P_1,P_2,P_3,P_4,P_6\}$，func1，func2，func3和func4在CSP中的个数分别为1，2，2和1. 则func2和func3在程序中是较为关键的函数.

根据函数的调用次数由多到少对函数进行排序，若函数的调用次数相同则比较函数的层级大小，层级越小就越重要，由此可得关键函数序列，且在该序列中，一个函数顺序越靠前，它在程序中就越关键.

2.2 程序相似部分的检测

由于程序相似部分的检测是回归测试中待测程序用例重用的前提，因此相似部分的检测是本文重要的工作之一. 从程序动态执行角度出发，在文献[6]提出的函数执行序列模式基础上进行改进，提出一种基于关键函数序列的程序相似部分检测方法. 由于大部分回归测试前后程序的变动不会太大，仅对比程序的关键函数序列不足以检测出回归测试前后函数的相似之处和细微变化，因此，在获得程序的关键函数序列后，进一步比较对应函数的关键字序列(即在程序的某一函数中，依据关键字出现的先后顺序所构成的序列). 注意，涉及的关键字有abstract，break，byte，case，catch，char，continue，default，else，extends，float，for，if，int，private，protected，public，return，static，switch，this，throw，try，void，while等.

通过比较关键函数的相似性来判定程序变更前后的相似性，重用变更前相似函数测试用例. 当已变更的程序重用已知用例后，无法完全满足自身测试需求时，则利用遗传算法筛选优秀个体，并进化种群，从而生成可用的测试用例. 程序相似部分检测过程如图2所示.

基于函数调用图的函数关键性判定及程序相似部分检测的具体过程如下.

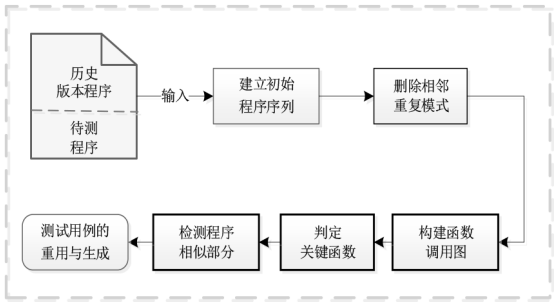


图2　程序相似部分检测

(1)将整个程序看作由多个函数组成,把函数地址转换为函数名,并识别函数的进出调用关系,根据执行轨迹的建模来描述整个程序的执行情况,即把程序动态执行过程转换为执行序列.

(2)对相邻的重复程序序列模式进行识别,利用重复模式删除算法,来减少程序的冗余执行序列,从而得到简化的程序序列. 具体过程如算法1所示.

(3)利用函数关键性判定算法来识别不同层次上的函数分布规律,从而根据函数的出现次数以及层次关系来判定程序的关键函数. 具体过程如算法2所示.

(4)根据函数相似度的计算方法来检测程序之间的相似部分. 具体过程如算法3所示.

由2.1节得到函数调用图(FCD)后,通过进一步比较程序中函数同一层级关系对应的函数关键字,来识别程序间相似部分. 将待比较相似部分的程序的FCD中函数名作为节点,节点内包含该函数的关键字序列以及被调用次数(注意:不相同模式下具有相同决策功能的函数为同一种函数,统计时计算这种函数的调用次数总和即可).

假定有2个函数调用图D_{F1}和D_{F2},$D_{F1}=\{A_1,A_2,\cdots,A_m\}$,$D_{F2}=\{B_1,B_2,\cdots,B_n\}$,$m$和$n$分别为$D_{F1}$和$D_{F2}$中函数个数. 其中,$A_i=<a_1,a_2,\cdots,a_r>$和$B_j=<b_1,b_2,\cdots,b_s>$都是函数调用图$D_{F1}$和$D_{F2}$中的元素,代表对应函数的关键字序列,$1\leqslant i\leqslant m$,$1\leqslant j\leqslant n$,$a_u$和$b_v$分别为关键字序列$A_i$和$B_j$中的关键字,$1\leqslant u\leqslant r$,$1\leqslant v\leqslant s$.

根据函数调用图D_{F1}和D_{F2},得到在不同层级上的函数分布,并识别公共函数. 在此基础上,进一步查找对应函数的关键字形成每个函数对应的关键字序列,比较2个函数的关键字序列来判断程序的相似部分. 若待比较程序相应的函数关键字序列A_i和B_j相同,则认为这2个函数相似. 此处将函数调用图D_{F1}和D_{F2}的相似部分记为$\mathrm{Sim}=\{(A_i,B_j)\,|\,I(A_i,B_j),1\leqslant i\leqslant m,1\leqslant j\leqslant n\}$,其中,$I(A_i,B_j)$表示$D_{F1}$中第$i$个函数与$D_{F2}$中第$j$个函数相似.

基于函数调用图识别程序相似部分的详细过程如算法3所示.

算法3　程序相似部分的检测

输入: 待比较程序的函数调用图D_{F1}和D_{F2}

输出: 程序相似部分

```
BEGIN
1  Sim←∅;
2  Sequence(A_i,B_j)←SearchTheSameLevel(D_F1,D_F2);  //按照D_F1和D_F2中函数的层级关系,分别将同一层级的函数归类
3  Sort(Sequence(A_i,B_j));  //同一层级的函数按关键性从大到小排列
4  IF Length(A_i,B_j) == 0  //两个函数的关键字序列长度相同
5     Keywords←FindKeywords(Sequence(A_i,B_j));  //分别查找函数中的关键字
6     Compare(Keywords(A_i,B_j));  //依次比较D_F1和D_F2中相同层级内函数的关键字
7     IF Seq(A_i) == Seq(B_j)  //若两个函数关键字序列相同
8        Sim←RecordSimilarity(A_i,B_j);  //记录对应函数的相似部分
9     END IF
10 END IF
11 RETURN Sim;
END
```

3　相似程序间测试用例重用与生成

由于新程序未变更部分较多,利用历史版本程序的用例可覆盖新程序的大部分目标路径,未被用例覆盖的部分则需借用遗传算法将可重用旧版本程序的用例引入到种群进化中,通过适应度函数进行筛选,最终生成新程序的优秀用例.

3.1　基于函数影响力的适应度设计

利用遗传算法来实现回归测试中相似程序间测试用例的重用与生成,主要过程如下.

(1)在检测出程序的相似部分后,尝试利用与待测程序更相近的历史版本程序的测试用例去覆盖待测程序的目标路径,若生成的用例能满足回归测试需求,即可直接重用历史版本程序用例.

(2)若历史版本程序的用例无法完全满足覆盖待测程序目标路径的任务,则使用遗传算法进化生成满足测试需求的用例. 在使用遗传算法进行相似程序测试用例重用时,将历史版本程序中具有较高适应度值(即函数影响力(见下文定义11)越大,适应度值越高)的用例引入种群进化过程中,种群的其他个体也可学习并利用这些用例,这样即可更迅速地生成所需用例.

适应度函数在进化生成中发挥着重要作用,接下来介绍与设计的适应度函数有关的几个重要因素,并以图1中的函数调用图为例进行说明.

定义5　函数调用率(Function Call Rate,FCR). 具有相同决策功能的函数的调用次数F_{Call}占整个程序所有函数调用总次数P_{Call}的比率.

FCR可表示为

$$\mathrm{FCR}=\frac{F_{\mathrm{Call}}}{P_{\mathrm{Call}}} \tag{1}$$

例3 图1中所有函数调用的总次数为6,函数func1被调用了1次,由定义5可知,函数func1的调用率为1/6.

定义6 函数关键度(Function Criticality,FC). 指函数调用率与函数在程序中的模式层级比值.

函数关键度(FC)反映了函数在程序中被调用的频度以及函数在程序中被调用的持续程度. FC可表示为

$$\mathrm{FC}=\frac{\mathrm{FCR}}{\mathrm{ML}} \tag{2}$$

其中,ML为函数在函数关系调用图中的模式层级.

例4 图1中函数func1的层级为1,可得,函数func1的函数关键度为(1/6)/1=1/6.

程序变更后,函数一般有3种变更类型:增加、删除、修改. 这3种操作对回归测试有着不同程度的影响.

(1)增加:新版本程序添加了新函数.

(2)删除:新版本程序剔除了旧版本程序中的函数.

(3)修改:新版本程序更改了函数名或对函数内部进行了修复操作.

定义7 函数关联度(Function Relevance Degree,FRD). 该函数所关联的其他函数或者所调用的子函数数量.

例5 图1中函数func1的模式层级为1,其在程序执行期间分别调用了函数func2,func3,func4,由定义7可知,函数func1的关联度为3.

定义8 函数变更影响系数. 不同变更类型的函数变更个数与程序中函数变更总个数的比值.

函数变更影响系数λ_{F}可表示为

$$\lambda_{\mathrm{F}}=\frac{T_{\mathrm{F}}}{M_{\mathrm{F}}} \tag{3}$$

其中,T_{F}为以上3种函数变更类型的个数,M_{F}为变更函数总个数.

定义9 函数变更率(Function Modify Rate,FMR). 新版本程序在旧版本程序的基础上进行变更的程度.

FMR可表示为

$$\mathrm{FMR}=\lambda_{\mathrm{F}}\times\frac{H_{\mathrm{F}}}{N_{\mathrm{F}}}\times\mathrm{FRD} \tag{4}$$

其中,H_{F}为旧程序中有变更的函数个数;N_{F}为变更后新程序所拥有的函数总个数;FRD为函数关联度,与变更函数的关联度越大,受到函数变更的影响就越大.

定义10 函数穿越率(Function Traversal Rate,FTR). 测试用例对函数关键字的覆盖程度,即用例对函数进行测试时经过该函数中关键字的个数与该函数关键字总个数的比值.

FTR可表示为

$$\mathrm{FTR}=\frac{N_{\mathrm{CK}}}{N_K} \tag{5}$$

其中,N_{CK}是在测试时穿越函数内关键字的个数,N_K是函数中关键字节点总个数.

例6 若变更后的程序将图1中函数func1做修改(注意:由于此处仅对函数内部进行修改,故不改变原函数调用图),增加了一个if判断,其他关键字没有变动,由上面的定义可知,函数func1的变更率为$\lambda_{\mathrm{F}}\times H_{\mathrm{F}}/N_{\mathrm{F}}\times\mathrm{FRD}_{\mathrm{func1}}=1\times 1/6\times 3=1/2$. 另外,在计算函数func1的穿越率时,由于新增了一个if关键词,需将要经过的关键字个数加1.

FC反映了函数在程序中被调用的频度及函数被调用的持续程度. 函数被调用的次数越多,且在函数调用图中的模式层次越靠下,该函数对整个程序影响就越大,在回归测试中对该函数的测试也更重要. FMR表示新版本程序在旧版本程序的基础上进行变更的程度,变更程度越小,重用旧版本程序的测试用例就越有效. FTR检测重用旧版本程序的测试用例时对函数关键字节点的覆盖情况,反映了对旧版本程序用例的重用程度.

在测试用例进化生成过程中,赋予FC以30%的初始权重,FC值越大的函数对整个程序的影响越大,若此类函数存在问题,很有可能影响整个程序的正常执行,应使生成的测试用例能尽快覆盖此类函数,体现了关键函数在程序中被重视程度. 另外,分别赋予FMR及FTR各35%的初始权重. 程序更新后,虽然存在改动的部分目标路径在重用历史程序的用例后仍未被覆盖,但大部分目标路径能通过FMR与FTR来表示重用情况. FMR越小,表明新程序变更较小,则重用历史用例效果越明显;FTR越大,表明测试用例对目标路径的覆盖程度越大. 这两个因素都在一定程度上体现了用例的重用程度,对加快回归测试中新版本程序用例的进化生成有着更直接的促进作用,给予这两个因素更大的权重. 特别地,重用后未覆盖部分需进化生成另外的测试用例. 因此,在进化生成的过程中需重点关注程序的变更部分以及重用后未被覆盖的部分.

定义11 函数影响力(Function Influence,FI). 函数在程序中的重要程度,即函数需要被测试的轻重缓急程度.

FI由FC,FMR和FTR共同决定,可表示为

$$\mathrm{FI}=\omega_1\times\mathrm{FC}+\omega_2\times\mathrm{FMR}+\omega_3\times\mathrm{FTR} \tag{6}$$

其中,ω_1=30%,ω_2=35%以及ω_3=35%为3种测试影响因素的初始权重.

先用初始权重进行实验,通过观察实验结果对权

重进行适当调整,得出可取得较好测试数据生成效果的值(将函数影响力FI作为适应度函数,对适应度函数权重的详细分析见4.1.3节).

3.2 测试用例的重用与生成

本小节提出基于函数影响力的进化测试生成方法.在测试数据生成过程中,以相似程序间用例的重用为方向进行遗传算法的具体实现.从函数的关键程度、函数的变更程度及测试数据对函数的覆盖程度等方面来设计用例进化生成的适应度,从而加快变更后程序测试用例的生成.

根据3.1节设计的适应度函数(即函数影响力)给出一种基于函数影响力的测试进化生成方法,具体实现过程如算法4所示.

算法4　基于函数影响力的测试进化生成

输入: 种群规模popSize,个体individual,染色体长度chroSize,进化代数ET,交叉概率pc,变异概率pm,可重用个体reusablePop,目标路径集P,目标路径覆盖集path,最大进化代数MT

输出: 覆盖目标路径集P的测试数据集testset

```
BEGIN
1  SetParameters(popSize,pc,pm,P,MT);  //设置各个参数值
2  Initialize(popSize,chroSize,reusablePop);  //初始化种群
3  ET←0;  //进化代数
4  Select(reusablePop);  //选择可重用个体
5  FOR(ET ≤ MT || path ≠ P)  //若进化代数大于最大进化代数或目标路径已全被覆盖,算法终止
6      ET←ET + 1;
7      Cal_fitness(individual);  //根据式(6)计算个体适应度
8      newPop←Crossover(popSize,chroSize,reusablePop,
                        individualList,pc);  //交叉操作
9      newPop←Mutate(popSize,chroSize,newPop,pm);  //变异操作
10     IF(p ∈ P is covered)
11         testset←individual;  //若路径覆盖集中不含目标路径p,则将该个体加入测试数据集
12     END IF
13 END FOR
14 OUTPUT testset;
END
```

算法4利用改进后的遗传算法来进化生成测试用例.在该算法中,第1行和第2行设置各个参数并初始化种群;第3~13行首先选择可重用个体,再根据式(6)计算个体适应度值,然后在交叉、变异过程中,引入历史版本程序可重用的测试用例,并判断新种群中是否有个体覆盖了目标路径,若目标路径覆盖集合中没有该目标路径,则将该个体加入测试数据集;最后,判断是否满足算法终止条件,即种群是否达到最大进化代数或目标路径已全部被覆盖,若满足则终止算法,否则继续执行种群进化过程.

若重用历史版本程序的测试用例,可能会出现新版本程序的部分目标路径未被覆盖的情况,此时需要利用遗传算法来进化生成新的测试用例去覆盖待测程序未被覆盖的目标路径.图3详细描述了2.2节图2中用例重用与生成的进化过程.

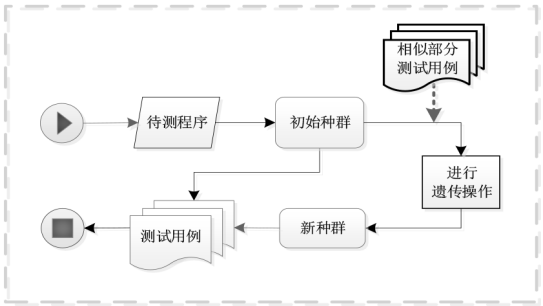


图3　测试用例的重用与进化生成

由图3可知,在种群进化过程中引入相似部分测试用例,将基于函数影响力设计的适应度函数作为衡量种群个体优劣的标准,保留适应度函数值较高的用例,与已知相似程序的测试用例共同形成新的用例集来满足测试需求.

依据关键函数的重要程度对该集合中的测试用例进行排序,排在前面的可优先进行重用.

4 实验设计与分析

为验证本文方法的有效性,分别针对基准程序和工业程序设计对比实验展开分析.实验环境:操作系统为Win10,内存为4 GB,主频为2.3 GHz,系统类型为X64,编程语言为Java.

4.1 实验过程与结果分析

本文在传统遗传法基础上根据函数影响力设计适应度函数,以提高测试效率.下面,先简要介绍随机法、传统遗传法及其他几种经典方法,再分别将本文方法与上述方法进行对比分析.

(1) 随机法:随机产生输入数据生成待测程序的测试用例.

(2) 传统遗传法:采用分支距离与层接近度结合的方法设计适应度函数,并根据该适应度函数进化生成测试用例.

(3) 文献[7]方法:将否定选择策略融入遗传算法,自动生成覆盖目标路径的测试数据.

(4) 文献[8]方法:对基本的细菌觅食算法进行改进,并应用于测试用例的自动生成.

(5) 文献[9]方法:依据不可达路径节点出现的概率计算个体穿越度,进而调整个体适应度函数的路径覆盖测试数据生成.

（6）文献[10]方法：在运用遗传算法生成测试数据时，利用神经网络模拟计算个体适应度值，对适应度值较高的个体进行插桩验证，从而获得精确的适应度值.

特别说明：(1)本文选取基准程序、中小规模工业程序和大规模工业程序作为待测程序，且在它们历史版本信息已知的基础上进行回归测试实验；(2)为了更好地体现新版本程序与历史版本程序间的变更程度，定义程序的“变更率”(即程序变更部分的代码行数占变更前总代码行数的比例)这一特征.

4.1.1 对基准程序的实验

基准程序具有结构多样性、运算各异性以及数据类型丰富性的特点，因而常被用于测试领域. 这里选取几种开源程序作为实验对象，信息见表1.

表1 基准程序信息

程序编号	待测程序名称	函数个数	代码行数	目标路径数	主要变更类型	变更率
PG1	三角形判别算法	3	70	40	增加	21.4%
PG2	时间差算法	4	53	21	删除	18.9%
PG3	24点算法	8	278	76	修改	19.4%
PG4	九宫格算法	4	170	37	修改	15.8%

注：PG1来源于https://www.cnblogs.com/cmt/p/14553189.html；PG2来源于http://www.zuidaima.com/share/3014318511737856.htm；PG3来源于http://www.zuidaima.com/share/1853602774846464.htm；PG4来源于https://blog.csdn.net/sihai12345/article/details/69053939

程序PG1为三角形判别算法. 程序PG2计算给定2个时间的差值. 程序PG3为24点算法，根据已知的4个整数(必须使用每个数字，且仅可使用一次)，通过加、减、乘、除这4个运算符及括号，组成一个运算表达式，使其计算结果为24. 程序PG4为经典的九宫格算法，将1到9这9个数不可重复地放到一个3×3的表格中，要求表格中的同行、同列及对角线的所有数字相加等于15.

下面分别利用随机法、传统遗传法及本文方法对表1中程序进行测试. 参数设置包括初始种群为100，交叉概率为0.8，变异概率为0.15，最大进化代数为200，独立运行次数为50. 实验结果见表2.

表2 基准程序上的结果对比

程序编号		PG1	PG2	PG3	PG4
随机法	平均进化代数	124.3	87.2	156.5	200.0
	测试用例数量	1 036	867	3 531	4 835
	覆盖率/%	100	100	100	75
传统遗传法	平均进化代数	83.6	56.3	146.1	200.0
	测试用例数量	671	925	3 872	2 761
	覆盖率/%	100	100	100	92
本文方法	平均进化代数	49.0	49.4	14.7	99.3
	测试用例数量	472	573	265	1 273
	覆盖率/%	100	100	100	100

由表2可知，本文方法通过重用待测程序历史版本的用例，在平均进化代数、用例生成数量及覆盖率方面均优于随机法和传统遗传法.

在测试用例数量方面，本文方法均少于另外2种方法，如程序PG3，生成测试用例数量明显较少，这表明本文方法生成的用例具有较高质量.

在覆盖率方面，本文方法在程序PG1，PG2，PG3，PG4上的覆盖率均为100%. 特别地，在程序PG4上，相较于随机法和传统遗传法，本文方法的覆盖率分别提高约25%和8%，提升效果较为明显. 可见，本文方法在覆盖率方面效果更优.

为更直观地表示不同算法在几个基准程序上的进化代数分布情况，这里采用箱型图进行展示，如图4所示.

从图4的不同算法在几个程序上的进化代数分布情况来看，本文方法的进化代数更少. 例如，对于待测程序PG3，本文方法的进化代数远低于随机法和传统遗传法，这是由于本文方法重用已有测试用例覆盖目标路径，只需再生成新用例去覆盖有变更的路径，而随机法和传统遗传法需要重新生成所有用例.

在基准程序上，本文方法对传统遗传法的改进具有积极效果，在降低平均进化代数和减少用例生成数量的同时，还能提升目标路径的覆盖率.

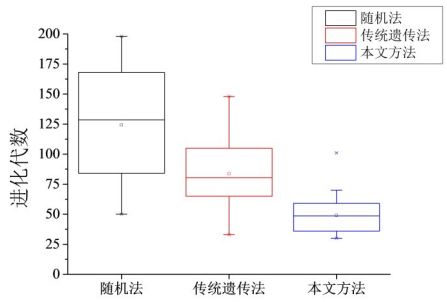
4.1.2 对工业程序的实验

下面针对工业程序进一步验证本文方法的有效性. 因本文策略与文献[7]方法均是对路径覆盖测试数据生成的改进，且实验目的及环境相近，故选择与该文献中方法进行比较，参数设置成与基准程序的一致.

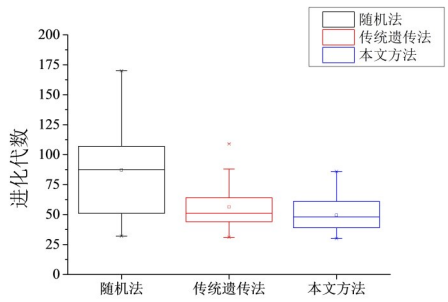
这里选取Replace，Space，Gzip，Sed_Part和Flex_Part这5种工业程序(来源于http://sir.unl.edu/portal/index.php)实验. 表3分别列出了这5种待测程序的基本信息.

Replace主要实现模式匹配和替换；Space是针对数组定义语言的解释器；Gzip，Sed_Part和Flex_Part均为Unix工具(Sed_Part和Flex_Part分别是大规模工业程序Sed和Flex的一部分).

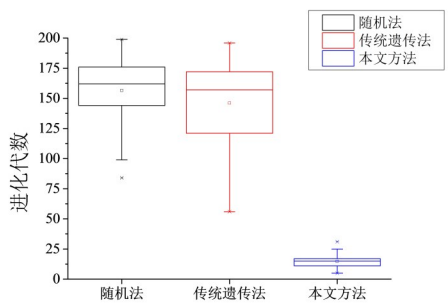
对每个待测程序选择部分函数进行实验，其实验



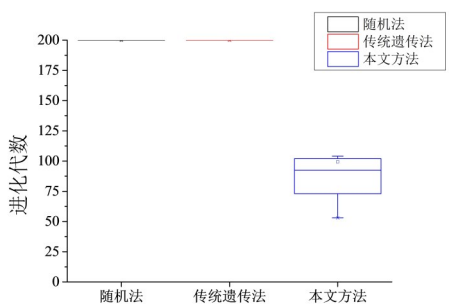
(*a*)　不同算法在程序PG1上的进化代数分布情况



(*b*)　不同算法在程序PG2上的进化代数分布情况



(*c*)　不同算法在程序PG3上的进化代数分布情况



(*d*)　不同算法在程序PG4上的进化代数分布情况

图4　3种算法在4个程序上的进化代数对比

结果如表4所示.

由表4可知,本文方法的平均进化代数及测试用例数量均比文献[7]方法少,路径覆盖率有所提高,这表明本文方法加快了进化过程.

表3　中小规模工业程序信息

程序编号	待测程序名称	函数个数	函数代码数	目标路径数	主要变更类型	变更率
PG5	Replace	3	98	10	修改	13.2%
PG6	Space	3	195	15	删除	26.1%
PG7	Gzip	2	239	47	增加	22.8%
PG8	Sed_Part	2	367	125	修改	17.3%
PG9	Flex_Part	2	1 103	233	修改	29.3%

表4　中小规模工业程序上的结果对比

程序编号	文献[7]方法			本文方法		
	平均进化代数	测试用例数量	覆盖率/%	平均进化代数	测试用例数量	覆盖率/%
PG5	4.0	232	100	3.8	139	100
PG6	32.0	2 678	100	27.5	1961	100
PG7	24.0	1956	100	15.8	853	100
PG8	200.0	13 564	97	200.0	9 892	98
PG9	200.0	14 521	86	200.0	10 516	94

注:文献[7]的数据来源于原文献

在平均进化代数方面,对于待测程序PG5,PG6和PG7(Replace,Space和Gzip),本文方法略少于文献[7]方法.程序PG8和PG9(Sed_Part和Flex_Part)进化代数达到实验设置的最大值,其原因是这2个程序存在无法覆盖的目标路径,在最大进化代数内不能覆盖所有目标路径.

在测试用例数量方面,在稍大程序PG8和PG9上,本文方法明显少于文献[7]方法.

在覆盖率方面,2种方法在程序PG5,PG6和PG7上,目标路径覆盖率均达到100%,而在程序PG8和PG9上均未完全覆盖目标路径.对于程序PG8和PG9,本文方法的覆盖率未能达到100%的主要原因是程序本身存在无法覆盖的路径.虽然待测程序PG8有较大变更,但用本文方法对其进行测试时,覆盖率还是有所提升,本文方法的目标路径覆盖率仍高于文献[7]方法.这进一步说明本文提出的方法用例生成效率更高,这是因为一方面引入了历史程序用例,另一方面利用函数影响力对适应度函数进行设计获得优秀个体.

以上仅为从中小规模程序中选择部分函数进行测试,接下来选取大规模工业程序Sed和Flex(均来源于http://sir.unl.edu/portal/index.php),以及Go(来源于https://blog.csdn.net/chongshangyunxiao321/article/details/50997404)进行实验,将本文方法与文献[8]方法、文献[9]方法、文献[10]方法进行对比.鉴于上述几种方法的算法类别、参数设置、程序设计语言、操作系统等都不一样,为比较的公平性和便捷性,此处,本文方法中的实验参数均设置成与相应对比方法一致.待测工业程序详细信息见表5.

表5 大规模工业程序信息

程序编号	待测程序名称	变量个数	代码行数	目标路径数	主要变更类型	变更率
PG10	Sed	20+	8 063	251	修改	27.4%
PG11	Flex	30+	11 783	327	增加	24.1%
PG12	Go	30+	28 547	100	修改	31.6%

注:大规模工业程序很少存在主要变更类型为“删除”的情况,因为在版本更新时,若删除部分较多,鉴于其规模且为保证其功能完整性,则往往会引起更多部分的修改或增加,使主要变更类型不是“删除”

Sed是Unix中一种非交互性文本编辑工具;Flex是Unix中词法分析生成器;Go是一种抽象策略性益智游戏软件. 对这3个待测程序进行测试,实验结果如表6所示. 为了更清晰地看出本文方法与其他几种方法在大规模工业程序上就平均进化代数方面的变化情况,这里增加了“平均进化代数之差百分比”(即本文方法与其他方法相比,平均进化代数增加或减少的比率)这一指标.

由表6可知,对于程序PG10和PG11(即Sed和Flex),本文方法的平均进化代数均比文献[8]方法要高,平均进化代数之差百分比分别为-17.0%和-54.5%(负数表示本文方法比文献[8]方法的平均进化代数相对增加),主要原因是文献[8]方法仅选取了部分可达路径作为目标路径,未考虑不可达路径,而本文方法在目标路径选取时未将不可达路径排除,这使得本文方法需实施更多的进化过程覆盖这些不可达路径. 另外,程序PG10和PG11对于它们的历史版本程序变更较大,可重用的测试用例数量较少,通过改进的适应度函数筛选优秀个体需更多的进化代数,从而导致平均进化代数高于文献[8],但在覆盖率方面分别提高了2%和17%. 相比于文献[9]方法,本文方法在平均进化代数方面分别减少了38.1%和55.4%,而在覆盖率方面均达到100%. 对于程序PG12,与文献[10]相比,本文方法的平均进化代数有所降低,平均进化代数之差百分比为44.7%,覆盖率达到了100%,提高了15%.

从实验结果来看,不论是在基准程序、中小规模工业程序,还是在大规模工业程序上,本文方法的测试用例生成效率优于其他对比方法. 具体地,在实现了目标路径全覆盖的情况下进化代数有一定程度的缩减. 这表明本文重用历史版本已知测试信息的有效性,以及运用函数影响力设计适应度函数的合理性.

4.1.3 适应度函数权重分析

根据以上实验结果可得出,不同待测程序的适应度函数影响因素的权重不同,具体结果如表7所示.

由表7可知,大部分被测程序,各适应度函数影响因素的权重与其初始权重相比没有较大变动. ω_1的范围在22%~30%之间,ω_2的范围在30%~39%之间,ω_3的范围在32%~44%之间,都在较小的范围内变动. 从表中的最大影响因素这一指标可看出,对于大部分被测程序,ω_3影响相对来说最大,ω_2次之,ω_1最小,即总体上函数穿越率影响最大,函数变更率影响次之,函数关键度影响最小.

4.2 显著性分析

从4.1节的实验结果可见,本文方法具有很大的改进效果. 为验证实验结果的可靠性,运用Z假设检验方法对实验结果进行显著性分析,设置显著性水平α=0.05,则Z_α=-1.96. 因在基准程序、中小规模工业程序与大规模工业程序上的实验对比指标存在差异,对基准

表6 大规模工业程序上的结果对比

待测程序	方法								平均进化代数之差百分比/%		
	文献[8]方法		文献[9]方法		文献[10]方法		本文方法		本文方法比文献[8]方法减少比率	本文方法比文献[9]方法减少比率	本文方法比文献[10]方法减少比率
PG10	平均进化代数	48 129.3	平均进化代数	1 398 272.4	—		平均进化代数	865 169.7	-17.0	38.1	—
	覆盖率/%	98	覆盖率/%	100			覆盖率/%	100			
PG11	平均进化代数	49 221.7	平均进化代数	6 132 570.6	平均进化代数	222 416.0	平均进化代数	2 732 528.1	-54.5	55.4	-11.3
	覆盖率/%	83	覆盖率/%	100	覆盖率/%	84	覆盖率/%	100			
PG12	—		—		平均进化代数	594 543.0	平均进化代数	329 067.8	—	—	44.7
					覆盖率/%	85	覆盖率/%	100			

注:文献[8],[9],[10]的数据均来源于原文献,且各自方法的实验程序不完全一样,故程序PG10和PG12在某些方法下的实验数据缺失

表7　适应度函数影响因素的权重分配

程序编号	影响因素权重			最大影响因素
	ω_1	ω_2	ω_3	
PG1	30%	35%	35%	ω_2, ω_3
PG2	30%	35%	35%	ω_2, ω_3
PG3	29%	30%	41%	ω_3
PG4	30%	38%	32%	ω_2
PG5	30%	35%	35%	ω_2, ω_3
PG6	30%	35%	35%	ω_2, ω_3
PG7	24%	36%	40%	ω_3
PG8	22%	34%	44%	ω_3
PG9	26%	31%	43%	ω_3
PG10	27%	39%	34%	ω_2
PG11	30%	37%	33%	ω_2
PG12	29%	34%	37%	ω_3

表8　Z检验验证结果

程序编号	统计量数值		检验结果
	Z_1	Z_2	
PG1	−4.09	−10.30	接受$H_1:\mu_1<\mu_2$和$H_1:\mu_1<\mu_3$
PG2	−4.54	−3.39	
PG3	−8.49	−9.75	
PG4	−5.83	−5.87	
PG5	−8.37	—	接受$H_1:\mu_1<\mu_2$
PG6	−9.66	—	
PG7	−8.59	—	
PG8	−2.69	—	
PG9	−4.55	—	
PG10	−9.94	−15.38	接受$H_1:\mu_1<\mu_2$和$H_1:\mu_1<\mu_3$
PG11	−18.15	−20.34	
PG12	−17.43	—	接受$H_1:\mu_1<\mu_2$

程序和中小规模工业程序实验建立假设,检验本文方法与对比方法的测试用例平均生成数量是否有显著差异;对大规模工业程序实验则建立假设,检验本文方法与对比方法的平均进化代数是否有显著差异.

以三角形判别算法为例,设随机变量X表示该算法进化生成的用例数量,它近似地服从正态分布. 比较本文方法与对比方法中的随机变量均值μ的大小,若μ越小,则说明对应的方法可更快地得到所需测试用例,该方法就越有效. 根据以上实验结果可知:本文方法、传统遗传法和随机算法的样本容量均为$n_1=n_2=n_3=50$,样本均值依次为$\bar{X}_1=472$,$\bar{X}_2=671$,$\bar{X}_3=1\,036$,样本方差$\sigma_1^2=52\,629$,$\sigma_2^2=65\,787$,$\sigma_3^2=97\,302$.

首先,建立原假设$H_0:\mu_1\geqslant\mu_2$和$H_0:\mu_1\geqslant\mu_3$,则对立假设$H_1:\mu_1<\mu_2$和$H_1:\mu_1<\mu_3$. 然后,计算统计量即

$$Z_1=\frac{\bar{X}_1-\bar{X}_2}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}}=\frac{472-671}{\sqrt{\frac{52629}{50}+\frac{65787}{50}}}\approx-4.09,$$

$$Z_2=\frac{\bar{X}_1-\bar{X}_3}{\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_3^2}{n_3}}}=\frac{472-1036}{\sqrt{\frac{52629}{50}+\frac{97302}{50}}}\approx-10.30,$$

最后,可得$Z_1\leqslant Z_\alpha$,$Z_2\leqslant Z_\alpha$. 故拒绝原假设$H_0:\mu_1\geqslant\mu_2$和$H_0:\mu_1\geqslant\mu_3$,接受对立假设$H_1:\mu_1<\mu_2$和$H_1:\mu_1<\mu_3$. 得出验证结论:在三角形判别程序中,本文方法与其他2种对比方法的结果具有显著差异,本文方法比另外2种对比方法的用例生成效率更高.

根据以上检验过程,对基准程序和工业程序的实验结果进行统计分析,检验结果如表8所示.

由表8可知,对于基准程序实验和中小规模工业程序实验,本文方法与随机法、传统遗传法及文献[7]方法相比,结果具有显著差异. 由此可知,本文方法生成的测试用例数量少于文献[7]方法. 本文的策略与文献[7]方法一样改进了遗传算法,同时本文策略重用已知相似程序的测试用例,这是本文方法在一定程度上优于文献[7]方法的部分原因. 对大规模工业程序中的待测程序PG10,本文方法与文献[8]方法和文献[9]方法相比,有显著差异,本文方法在保证完全覆盖待测程序目标路径的前提下,实验效果仍略优于文献[9]方法. 本文方法与文献[9]方法平均进化代数虽然均比文献[8]方法多,但是覆盖率均达到最佳. 对于待测程序PG11,本文方法的覆盖率达到100%,其平均进化代数少于文献[9]方法. 本文方法与文献[10]方法相比,有显著差异. 对于大规模待测程序PG12,本文方法平均进化代数少于文献[10]方法.

综上所述,不论在基准程序还是工业程序上,本文提出的基于函数影响力的相似程序间测试用例重用策略与其他对比方法相比均更有优势,同时也验证了本文策略能够提高测试数据生成效率.

5　相关工作

当前,程序相似性方面的研究逐渐深入. Gang等[11]提出了一种用于测量代码功能相似性的方法,其将编码控制流和数据流信息表示为紧凑矩阵,从语义矩阵中学习潜在特征,并执行二进制分类. Liu等[12]利用深度神经网络(Deep Neural Networks,DNN)提取二进制函数特征,并基于这些特征计算距离来检测二进制代码相似度. 汪洁等[13]先用数据流图表示程序运行时的行为或事件,再从数据流图中提取特征子图,将其表示为字符串替代图的匹配,并通过神经网络计算子图间相似性.

以上方法借助不同的程序表达方式考察程序的相似性,且很少运用在回归测试中. 回归测试前后的程序存在很多相同或相似函数,通过研究函数调用检测程

序修改前后的差异性是可行的.

近年来,在测试用例的重用以及用例生成方面的研究也较为广泛. Mu等[14]根据变更函数和相关函数确定要测试的变更路径,为待测试的变更路径选择相似函数调用路径,并对用例进行重用. 钱忠胜等[15]通过支持向量机回归模型预测适应度值并重用初始种群的优秀用例,生成测试数据. 王曙燕等[8]利用改进的细菌觅食来生成测试数据. 廖志伟等[16]利用蚁群算法实现多路径覆盖测试数据生成. 夏春艳等[7,9]、姚香娟等[10]、范书平等[17]通过遗传算法得到了满足路径覆盖的测试数据. 上面诸多启发式算法中,遗传算法具有良好的全局搜索能力,在测试用例生成方面普适性也较高,在求解路径覆盖测试数据自动生成问题中做适当改进可方便、有效地提高测试效率,得到了更广泛的应用.

从近几年在测试方面的研究来看,测试用例的重用策略对提升回归测试效率具有较高的可行性. 本文利用程序变更前后的相似性来提高程序变更后测试用例重用的效率.

有效的回归测试在不断变化的软件开发过程中扮演着重要的角色. Bajaj等[18]研究发现自然启发方法(如遗传算法)已被广泛应用于回归测试优化,其目的是尽可能使用最少时间,最大限度查找故障.

在回归测试中,变更后的新程序与前一版本的程序一般只存在较小差异. 本文通过构建函数调用图来对变更前后程序的相似部分进行检测,并将历史版本程序的用例重用于新版本中,这样,程序变更后未修改部分的目标路径就可先被覆盖;重用测试用例后,新版本程序变更部分的目标路径可能未被覆盖,利用函数影响力设计适应度函数来筛选进化过程中符合测试需求的种群个体,使生成的用例能够尽早地覆盖目标路径.

6 总结

本文提出了一种改进的基于函数影响力的相似程序间测试用例重用进化策略. 实验结果表明,本文的方法在降低进化代数、减少测试用例数量以及提高路径覆盖率等方面改善效果明显,主要贡献如下.

(1)检测程序相似部分,为重用测试用例做准备. 将历史版本程序的测试用例运用到新程序的回归测试中,充分利用已有测试资源,减少不必要的用例生成;对于程序有变更的部分,继续利用历史版本程序的测试用例,借助遗传算法进化生成新的用例.

(2)根据函数影响力设计新的适应度函数,促进用例进化生成. 分析回归测试前后程序的变更情况,改进适应度函数,加速进化生成所需测试用例,使目标路径能被迅速覆盖,同时优化用例集,以满足后续回归测试需求.

本文的策略重点在于通过函数变更等信息改进适应度函数,并重用已知程序相似部分的用例,以提高测试效率. 回归测试中的历史版本程序与待测程序往往具有较高的相似性,符合重用的要求. 本文策略不仅适用于回归测试,理论上讲也可应用于其他具有相同或相似结构功能的相似程序间的测试,但其真实效果如何值得进一步研究.

参考文献

[1] DI NUCCI D, PANICHELLA A, ZAIDMAN A, et al. A test case prioritization genetic algorithm guided by the hypervolume indicator[J]. IEEE Transactions on Software Engineering, 2020, 46(6): 674-696.

[2] PEI Han-yu, YIN Bei-bei, XIE Min, et al. Dynamic random testing with test case clustering and distance-based parameter adjustment[J/OL]. Information and Software Technology, 2021, 131: 106470. https://doi.org/10.1016/j.infsof.2020.106470.

[3] NAIR A, ROY A, MEINKE K. funcGNN: A graph neural network approach to program similarity[C]//ESEM'20: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). New York, USA: ACM, 2020: 1-11.

[4] 李玫, 高庆, 马森, 等. 面向代码相似性检测的相似哈希改进方法[J]. 软件学报, 2021, 32(7): 2242-2259.
LI Mei, GAO Qing, MA Sen, et al. Enhanced simhash algorithm for code similarity detection[J]. Journal of Software, 2021, 32(7): 2242-2259. (in Chinese)

[5] 刘芳, 李戈, 胡星, 等. 基于深度学习的程序理解研究进展[J]. 计算机研究与发展, 2019, 56(8): 1605-1620.
LIU Fang, LI Ge, HU Xing, et al. Program comprehension based on deep learning[J]. Journal of Computer Research and Development, 2019, 56(8): 1605-1620. (in Chinese)

[6] ZHANG B, SHAN C, HUSSAIN M, et al. Software crucial functions ranking and detection in dynamic execution sequence patterns[J]. International Journal of Software Engineering and Knowledge Engineering, 2020, 30(5): 695-719.

[7] 夏春艳, 张岩, 万里, 等. 基于否定选择遗传算法的路径覆盖测试数据生成[J]. 电子学报, 2019, 47(12): 2630-2638.
XIA Chun-yan, ZHANG Yan, WAN Li, et al. Test data generation of path coverage based on negative selection genetic algorithm[J]. Acta Electronica Sinica, 2019, 47(12): 2630-2638. (in Chinese)

[8] 王曙燕, 王瑞, 孙家泽. 基于改进细菌觅食算法的测试用例生成方法[J]. 计算机应用, 2019, 39(3): 845-850.
WANG Shu-yan, WANG Rui, SUN Jia-ze. Test case generation method based on improved bacterial foraging optimization algorithm[J]. Journal of Computer Applications, 2019, 39(3): 845-850. (in Chinese)

[9] 夏春艳, 张岩, 宋丽. 基于节点概率的路径覆盖测试数据进化生成[J]. 软件学报, 2016, 27(4): 802-813.
XIA Chun-yan, ZHANG Yan, SONG Li. Evolutionary generation of test data for paths coverage based on node probability[J]. Journal of Software, 2016, 27(4): 802-813. (in Chinese)

[10] 姚香娟, 巩敦卫, 李彬. 融入神经网络的路径覆盖测试数据进化生成[J]. 软件学报, 2016, 27(4): 828-838.
YAO Xiang-juan, GONG Dun-wei, LI Bin. Evolutional test data generation for path coverage by integrating neural network[J]. Journal of Software, 2016, 27(4): 828-838. (in Chinese)

[11] ZHAO G, HUANG J. DeepSim: Deep learning code functional similarity[C]//ESEC/FSE 2018: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA: ACM, 2018: 141-151.

[12] LIU B C, HUO W, ZHANG C, et al. αDiff: cross-version binary code similarity detection with DNN[C]//ASE 2018: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. New York, USA: ACM, 2018: 667-678.

[13] 汪洁, 王长青. 子图相似性的恶意程序检测方法[J]. 软件学报, 2020, 31(11): 3436-3447.
WANG Jie, WANG Chang-qing. Malware detection method based on subgraph similarity[J]. Journal of Software, 2020, 31(11): 3436-3447. (in Chinese)

[14] MU Y M, GAO X X, SHEN M E. Research of reuse technology of test case based on function calling path[J]. Chinese Journal of Electronics, 2018, 27(4): 768-775.

[15] 钱忠胜, 俞情媛, 宋涛, 等. 基于支持向量机回归模型的测试用例生成与重用[J]. 电子学报, 2021,49(7): 1386-1391.
QIAN Zhong-sheng, YU Qing-yuan, SONG Tao, et al. Test case generation and reuse based on support vector machine regression model[J]. Acta Electronica Sinica, 2021, 49(7): 1386-1391. (in Chinese)

[16] 廖伟志, 夏小云, 贾小军. 基于蚁群算法的多路径覆盖测试数据生成[J]. 电子学报, 2020, 48 (7): 1330-1342.
LIAO Wei-zhi, XIA Xiao-yun, JIA Xiao-jun. Test data generation for multiple paths coverage based on ant colony algorithm [J]. Acta Electronica Sinica, 2020, 48 (7): 1330-1342. (in Chinese)

[17] 范书平, 张岩, 马宝英, 等. 基于均衡优化理论的路径覆盖测试数据进化生成[J]. 电子学报, 2020,48(7): 1303-1310.
FAN Shu-ping, ZHANG Yan, MA Bao-ying, et al. Evolutionary generation of test data for paths coverage based on balance optimization theory[J]. Acta Electronica Sinica, 2020,48(7): 1303-1310. (in Chinese)

[18] BAJAJ A, SANGWAN O P. A survey on regression testing using nature-inspired approaches[C]//Proceedings of the 2018 4th International Conference on Computing Communication and Automation(ICCCA). Greater Noida, India: IEEE, 2018: 1-5.

作者简介

钱忠胜　男, 1977 年 1 月出生, 江西鹰潭人. 2008年在上海大学获工学博士学位. 江西财经大学教授, 博士生导师. 主要研究方向为软件工程、机器学习、智能化软件等.
E-mail: changesme@163.com

宋　佳　女, 1997年6月出生, 四川眉山人. 江西财经大学信息管理学院硕士研究生. 主要研究方向为软件工程、智能推荐系统等.