软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(9):2691−2712 [doi: 10.13328/j.cnki.jos.005984]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向关键字流图的相似程序间测试用例的重用*

钱忠胜, 宋 涛

(江西财经大学 信息管理学院,江西 南昌 330013)

通讯作者: 钱忠胜, E-mail: changesme@163.com

摘 要: 软件测试是软件开发中重要的一环,能有效地提高软件的可靠性和质量.而测试用例的重用可减少软件测试的工作量,提升测试的效率.提出一种面向关键字流图的相似程序间测试用例的重用方法,该方法将程序已经生成的测试数据重用到与之相似的程序中.可见,探究测试用例重用的前期工作是判定程序的相似性.对于程序相似性的判定,给出根据关键字流图相似性比较的方法:首先,将程序代码中的关键字存储在流图所对应的节点中,构建关键字流图;接下来,利用动态规划算法查找待测程序关键字流图的最大公共子图;最后,根据最大公共子图距离算法计算程序的相似度.较高相似程度的程序可用到测试用例重用的方法中.在利用遗传算法生成测试用例时,引用相似程序中适应度较高的测试用例,使种群在进行进化操作过程中不断与这些用例进行交叉,加快用例的生成效率.实验表明:将测试用例重用在相似程序的测试生成中,与传统方法相比,在覆盖率和平均进化代数等方面均有明显优势.

关键词: 关键字流图;程序相似性;遗传算法;测试用例重用;最大公共子图

中图法分类号: TP311

中文引用格式: 钱忠胜,宋涛.面向关键字流图的相似程序间测试用例的重用.软件学报,2021,32(9):2691−2712. http://www.jos.org.cn/1000-9825/5984.htm

英文引用格式: Qian ZS, Song T. Reuse of test cases between similar programs based on keyword flow graph. Ruan Jian Xue Bao/Journal of Software, 2021,32(9):2691−2712 (in Chinese). http://www.jos.org.cn/1000-9825/5984.htm

Reuse of Test Cases Between Similar Programs Based on Keyword Flow Graph

QIAN Zhong-Sheng, SONG Tao

(School of Information Management, Jiangxi University of Finance and Economics, Nanchang 330013, China)

Abstract: Software testing is an important part of software development, which can effectively improve software reliability and its quality. The reuse of test cases can improve the efficiency of testing and reduce the workload of software testing. Therefore, an approach to test case reuse between similar programs based on keyword flow graph is proposed. The method reuses test data generated by programs to their similar programs. Thus, the first step in exploring reuse of test cases is to determine similarities between programs. To determine the similarity of programs, a comparison method of similarity based on keyword flow graph is given. Firstly, the keywords in the program code are stored in the nodes corresponding to the flow graph to construct the keyword flow graph. Then, the maximum common sub-graph of keyword flow graph of the maximum programs tested is searched by using dynamic programming algorithm. Finally, the similarity of the programs is computed according to the common sub-graph distance algorithm. Programs with a high similarity can be used with test case reuse methods. In utilizing genetic algorithm to generate test cases, the test cases with high fitness of similar programs are selected. To speed up the efficiency of test case generation, the population intersects with these test cases continuously in the process of evolution. Experiments show that the reuse of test cases in test generation of similar programs has obvious advantages over traditional methods in terms of coverage and average evolution times.

Key words: keyword flow graph; program similarity; genetic algorithm; test case reuse; maximum common sub-graph

* 基金项目: 国家自然科学基金(61762041); 江西省自然科学基金(20181BAB202009); 江西省教育厅科技重点项目(GJJ180250)

Foundation item: National Natural Science Foundation of China (61762041); Jiangxi Provincial Natural Science Foundation of China (20181BAB202009); Key Project of Science and Technology of Jiangxi Provincial Department of Education of China (GJJ180250)

收稿时间: 2019-01-03; 修改时间: 2019-07-09, 2019-10-28; 采用时间: 2019-12-02

软件测试是为了发现程序错误,以提高程序质量的一个过程[1,2].软件测试贯穿软件开发的整个过程,是软件开发不可或缺的一个环节.通常在软件生命周期中,30%~40%的时间和精力花在软件测试上[3].研究表明:在软件测试中,毫无限制地对所有程序进行验证,将会花费维护费用的 50%[4,5].而软件测试的重用在提高软件测试质量、缩短测试周期和改善测试人员的经验不足等方面,均起着十分重要的作用.目前,软件测试重用的研究已成为软件测试工程化研究的热点之一[6].

软件测试重用技术是指在新的软件测试工作中重复使用已有的测试资源,其目的是充分利用之前软件测试中的经验和成果,增强测试的效率和可靠性[7].测试用例的重用是指测试工程师在执行回归测试或者一项新的测试工作时,通过直接调用或修改已经生成的测试用例,并将它们运用测试的过程,而不用每次面对新的测试工作都需要从头开始设计测试用例.测试用例作为软件测试的核心内容,它的重用是整个软件测试重用的关键环节[8].

测试用例重用的研究主要包含两个方面:可重用测试用例的生成和可重用测试用例的管理[9].本文主要研究将程序已有测试用例使用到相似程序测试用例的生成过程中,故检测待测程序之间的相似度是研究用例重用的前提.

程序代码相似性的研究技术基本成熟,主要运用于计算机教学和软件剽窃检测等领域[10].最长公共子串和 Levenshtein 距离等程序相似性度量算法重点考察源代码之间的相似程度,而基于图的结构特征的相似性分析方法使用图的结构来反映程序功能特征的相似性分析方法[9].图的结构特征是较为高级的语法特征,主要包括控制流图和数据流图两种:控制流图反映了程序的逻辑结构,数据流图反映了程序的数据流转关系.

可见,程序相似性的判定可以从序列和图等不同的特征进行.本文研究程序间相似性的目的是研究相似程序间测试用例重用的方法,以此提高测试用例的生成效率,减少软件测试的工作量.主要做了以下工作.

1) 对于待比较相似性的程序,构建它们的关键字流图.比较流图节点中的关键字是否相同,具有相同关键字的节点构成公共关键字流图子图;
2) 程序的关键字流图和关键字流图最大公共子图构建完成后,利用最大公共子图距离方法比较待测程序的相似程度,相似程度较高的程序可用于测试用例的重用;
3) 测试用例的重用是将程序已有的测试用例共享于相似程序.采用遗传算法来完成测试用例的重用,将相似程序已经生成的测试用例引用到种群进化的过程中,种群的其他个体通过向这些测试用例学习加快进化速度,完成测试用例的重用.

本文第 1 节分析程序相似性及测试用例重用的相关工作.第 2 节研究相似程序的判定,主要涉及关键字流图构建方法、关键字流图最大公共子图的查找以及根据最大公共子图距离求得待测程序的相似度.第 3 节提出相似程序间测试用例重用的方法,并阐明将遗传算法用于测试用例重用的原因;实验多方面将本文方法与传统方法在测试用例生成中的效率进行对比,根据实验结果给出本文存在的不足以及有效性分析.第 4 节总结全文,并提出下一步的研究工作.

1 相关工作

程序相似性的制定标准是一项重要工作,近年来,不少学者从语义结构和图等不同方面探究程序的相似性.

Kwon 等人[11]利用子图作为检测恶意程序的特征,因为同一个恶意程序家族中的程序会共享子图.子图的查找则需要程序样本中抽取 API 调用建立分层级行为依赖图,用它们寻找子图.Wang 等人[12]提出了使用控制流图和数据流图特征对二进制程序进行相似性比较和同源分析的方法,实验对象采用微软不同版本的动态链接库验证算法的有效性.该算法比较适用于同一公司发布的同源良性程序相似度的比较.

1965 年,Levenshtein[13]提出 Levenshtein 距离算法,该算法常用于字符串的比较,计算两个字符串差异的大小.源字符串通过添加、修改、删除等操作变化到目标字符串,其最少的改变次数看作是 Levenshtein 距离.Levenshtein 距离越小,字符串的相似度越大.Wang 等人[14]提出两种基于双向比较的最长公共子串算法,该算法将动态规划算法(LCSstrDP)与后缀数组算法(LCSstrSA)相结合,有效地解决了动态规划算法计算速度较慢的

问题,却增加了算法的复杂度,计算效率不高.Rong 等人[15]为了解决程序需要从两个给定集合中找到所有类似的字符串对的问题,提出一种在单个操作符中支持不同相似阈值的字符串相似性连接的方法,就判定程序相似性阈值的设定,设计了新的索引技术,针对不同的程序设计了不同判定程序相似的阈值,使得判定程序相似性更加准确,但复杂度较高.

Mc.Cabe[16]提出了圈复杂度的结构度量技术,该技术在流图基础上计算它的圈度,很多时候需要与其他特征结合进行程序结构的度量.陈浩与王广南等人[10]提出了基于程序依赖图的动态胎记技术来检测程序的相似性,该方法首先对流图的公共子图进行胎记标记,通过比较最大公共子图距离来判断程序相似性.这兼顾了局部特征和整体特征,但公共子图同构判定的局限性比较大.

分析程序相似性的目的是为了探究相似程序间测试用例重用的有效性,而测试用例的生成是用例重用的前提,只有保存足够的已生成的测试用例,才能完成它们的重用.近年来,国内外开展了许多关于重用测试用例和测试用例的生成等方面的研究工作.

Mayrhauser 等人[17]通过领域建模来构造可重用的测试用例,其构造包括脚本化、命令模版和参数值选择这 3 个部分,并开发出基于领域建模的测试用例生成工具 Sleuth.Vouffo-Feudjio 等人[18]提出了基于 TTCN 的测试模式,并探讨了该模式的重用规则,包括怎样在不同产品之间的水平重用和不同版本之间的纵向重用.

巩敦卫等人[19]就提高回归测试中测试数据的生成效率,提出一种新的回归测试数据进化生成方法.该方法利用已有的测试数据穿越路径与目标路径的相似程度,选择相似程度较高的测试数据作为初始化种群的部分个体.进一步,根据已有测试数据穿越路径与目标路径的对比,确定个体实施遗传操作的位置.姜淑娟等人[20]提出了基于模式组合的粒子群优化测试用例生成方法,通过新设定的交叉算子,将个体中的含有模式的部分组合成新个体,利用遗传算法生成测试用例的过程中,种群中其他个体均可向该适应度较高的新个体进行学习,加快种群进化速度,提高测试用例生成效率.

将程序的相似性应用在计算机教学和恶性程序检测等领域的研究比较多:最长公共子串算法[14]比较字符串的相似性,字符串的相似性完全取决于最长字符串的长度,具有片面性;Levenshtein 距离算法[13]比较适用于规模较小的程序相似性的比较;基于程序依赖图的动态胎记技术[10]来检测程序的相似性,需要公共子图的同构作为前提条件,局限性比较大.

在前人研究的基础上,本文提出面向关键字流图程序相似度的方法.该方法兼顾了源代码序列和程序功能结构相似度的比较两个方面.另外,借鉴巩敦卫和姜淑娟等人[19,20]的思想,提出了相似程序间测试数据的重用方法,通过相似程序间测试用例的共享实现用例重用.即:待测程序利用遗传算法生成测试用例,在种群进化阶段引入相似程序中已经生成的测试数据;在迭代过程,以一定概率向这些个体学习.与传统遗传算法种群个体之间相互学习的进化方式作对比,测试用例生成效率较高,证明了测试用例重用的有效性,而相似程序间测试用例重用的效果证明了判断相似程序方法的可行性.

2 程序相似性的判定

本文对程序相似性的判定结果应用到相似程序间的测试用例重用中,故程序相似度的判定不仅注重语义上的相似性,更加注重程序功能结构上的异同.本节提出关键字流图的构建方法,通过构建的关键字流图查找待测程序的关键字流图最大公共子图.最后,通过最大公共子图距离算法求得程序的相似度,程序相似性的比较如图 1 所示.本节给出判定程序相似性的步骤以及与程序相似性判定相关的定义.程序相似性判定的流程如下.

1) 判断待测程序能否实现测试用例的重用:首先观察测试程序所输入的参数是否受到个数限制,如果测试待测程序时输入的参数数量都是一定的,而且它们所限定的数量是不同的,意味着它们的测试用例无法直接共享,本文不作比较;
2) 关键字流图最大公共子图的构建:若待测程序之间能够实现测试用例的重用,则构建关键字流图,利用动态规划算法比较关键字流图中关键字的异同.若关键字相同,则该关键字所属的节点即属于公共流图子图,并对此节点进行标记,以此构建待测程序的关键字流图最大公共子图;

3) 相似度的判定:使用最大公共子图距离算法计算关键字流图子图距离,该距离的大小决定了程序的相似程度.

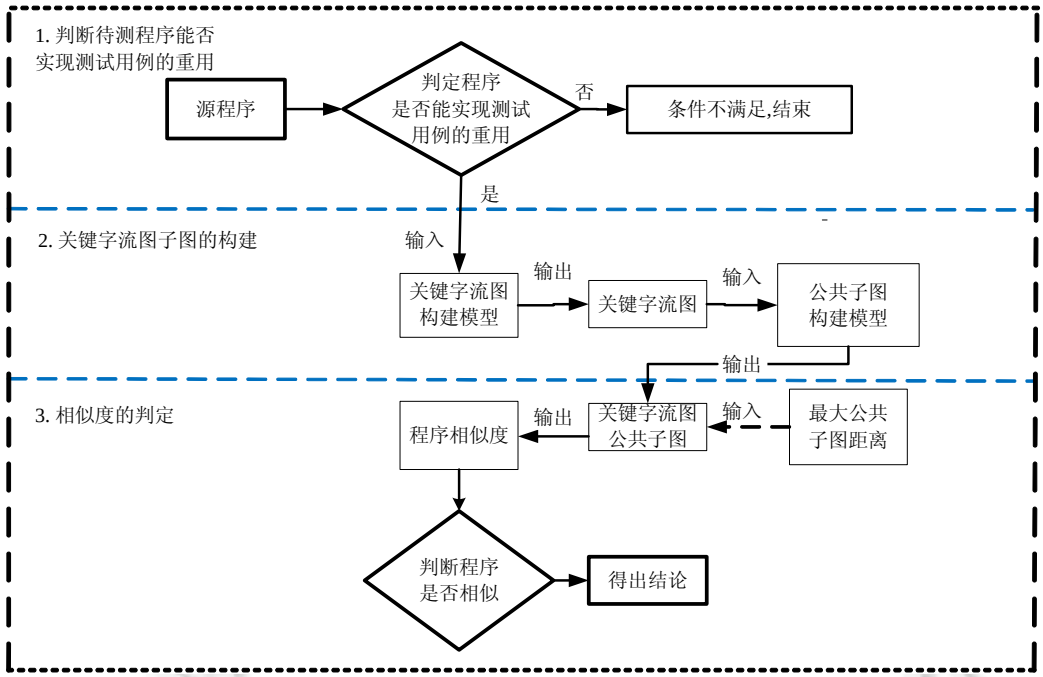


Fig.1 Determining program similarity

图 1 程序相似性的判定

2.1 相关定义

定义 1(关键字). 作为代码核心的标识符,用于表示一种数据类型或程序结构.像 Java 和 C 语言等编辑语言都有自定义的关键字.本文选择 Java 代码编写的程序作为实验对象,表 1 列出了 Java 语言中的关键字.

Table 1 Keywords of Java language

表 1 Java 语言的关键字

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	strictfp	short	static	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

定义 2(关键字流图). 关键字流图(keyword flow graph,简称 KFG)是一个五元组(*V*,*E*,*keyword*,*entry*,*exit*),其中:*V* 表示节点集,源程序的每个基本语句块都对应流图的相应节点;*E* 是边集,表示语句的流向;*keyword* 表示关键字集,存储着节点中的关键字,若源代码中无关键字,该节点储存字符 null;entry 和 exit 分别表示程序唯一的入口节点和出口节点.

定义 3(关键字流图公共子图). 节点集 *V* 中的每个节点均是关键字流图 G_1 中的节点,同样又是关键字流图 G_2 中的节点,那么节点集 *V* 在流图上构成的图即为流图 G_1 和 G_2 的公共子图.若存在节点集 *G*,且 G_1 和 G_2 不存在其他的子图的节点数大于 *G*,那么 *G* 是 G_1 和 G_2 的一个最大公共子图[8].

定义 4(前缀与后缀)[14]. 前缀 $Pref[S,i](1\leqslant i\leqslant L)$,指从字符串 S 的第 1 个字符开始至字符串的某个位置 i 结束的特殊子串,可记为 $S[1,i]$;后缀 $Suffix[S,i](1\leqslant i\leqslant L)$,指从字符串 S 的位置 i 开始至整个串末尾的一个特殊子串,可记为 $S[i,L]$.

根据定义 4,公共前缀(后缀)表示一个字符串既是字符串 S 的前缀(后缀),又是字符串 T 的前缀(后缀),那么该字符串是字符串 S 和 T 的公共前缀(后缀).

定义 5(最大公共子图距离). 给定两个非空图 G_1 和 G_2,以及它们的最大公共子图 $mcs(G_1,G_2)$,它们之间的距离[21,22]可表示为

$$D(G_1,G_2)=1-\frac{|mcs(G_1,G_2)|}{\max(|G_1|,|G_2|)} \tag{1}$$

其中,$|G_1|$和$|G_2|$分别表示 G_1,G_2 的节点数,$mcs(G_1,G_2)$表示最大公共子图的节点数.那么图 G_1 与 G_2 的相似度可以定义为

$$S(G_1,G_2)=(1-D(G_1,G_2))\times 100\%=\frac{|mcs(G_1,G_2)|}{\max(|G_1|,|G_2|)}\times 100\% \tag{2}$$

定义 6(组间变异和组内变异). 在统计学中,组间变异表示数据的各组平均数 $\bar{X}_i$ 与总平均数 $\bar{X}$ 之间的离均差平方和,记为 SS_{TR};而组内变异表示每组数据中的每个测量值 X_{ij} 与该组平均数 $\bar{X}_i$ 之间的离均差平方和,记为 SS_e.它们的公式分别表示为

$$SS_{TR}=\sum_{i=1}^{k}n_i(\bar{X}_i-\bar{X})^2,SS_e=\sum_{i=1}^{k}\sum_{j=1}^{n_i}n_i(X_{ij}-\bar{X}_i)^2 \tag{3}$$

其中,n_i 为第 i 组的组内数据个数,k 为组数.

定义 7(方差分析). 在统计学中,方差分析(analysis of variance,简称 ANOVA)是用于两个及两个以上样本均数差别的显著性检验,又称为“变异数分析”或“F 检验”.F 值为组间均方 MS_{TR} 与组内均方 MS_e 的比值,表示为

$$F=MS_{TR}/MS_e \tag{4}$$

其中,$MS_{TR}=SS_{TR}/k-1$,$MS_e=SS_e/N-k$.这里,N 为各组数据个数的和,k 为组数.

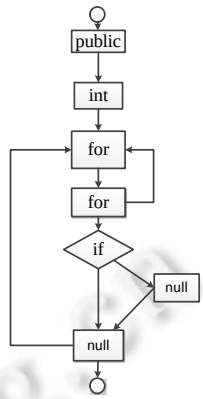
2.2 关键字流图的构建

关键字是代码核心的标识符,能够代表代码的结构或者数据类型.某些代码中的关键字相同,意味着它们的代码结构相同.源代码中,每行代码或者功能相近的若干行代码为一个基本块,每一个基本块构成一个节点.关键字流图的节点中存储了形成该节点基本块中的关键字,若基本块中无关键字,此节点存储字符串 null;若该行代码存在两个及以上的关键字,记录第 1 个关键字即可.构建关键字流图的步骤类似于控制流图的构造过程,已有很多研究,本节不再做具体说明.下面是冒泡排序的关键字流图构造示例.

图 2 中,图 2(b)是图 2(a)冒泡排序源码所对应的关键字流图.可以看出,关键字流图节点中存储了形成该节点的关键字.例如:生成第 2 个节点的基本块中含有关键字 int,该基本块在生成关键字流图过程中将该关键字储存在相应的节点上.待测程序的关键字流图构建完成,接下来比较待测程序关键字流图中关键字的异同,构建待测程序的关键字流图最大公共子图.图 3~图 5 分别给出了冒泡排序的控制流图、程序流程图和数据流图等几种比较常用的软件结构图与关键字流图的比较.通过对比发现:

- 关键字流图比较程序相似性,注重程序的结构和功能上的异同,符合功能结构相同的程序,其测试用例会以更大概率共享的理念;
- 控制流图能表达程序的结构,但不能展示图中每个节点的功能;
- 程序流程图能清晰地表达程序的流程及功能,但存在许多相似节点合并的现象;此外,用自然语言总结节点功能存在一定的主观性,会因个人用语不同而存在文字上差别较大的情况;
- 数据流图比较详细地阐述了程序主体及功能,也因此使其更加复杂,在比较程序相似性方面存在与程序流程图相同的缺点.

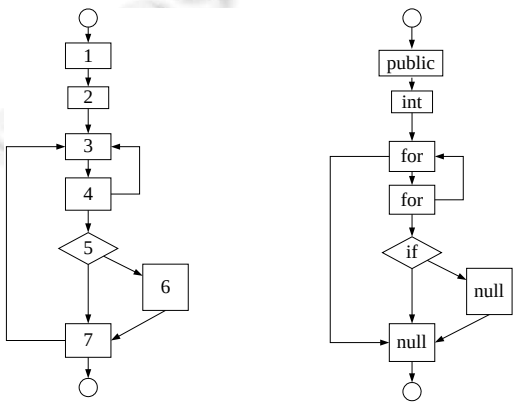
```
1   public void bubbleSort(int array[]) {
2       int t = 0
3       for (int i = 0; i < array.length - 1; i++)
4           for (int j = 0; j < array.length - 1 - i; j++)
5               if (array[j] > array[j + 1]) {
6                   t = array[j];
7                   array[j] = array[j + 1];
8                   array[j + 1] = t;
9               }
10      }
```



(a) 冒泡排序程序示例 (b) 关键字流图

Fig.2 Bubble sorting program and its keyword flow graph

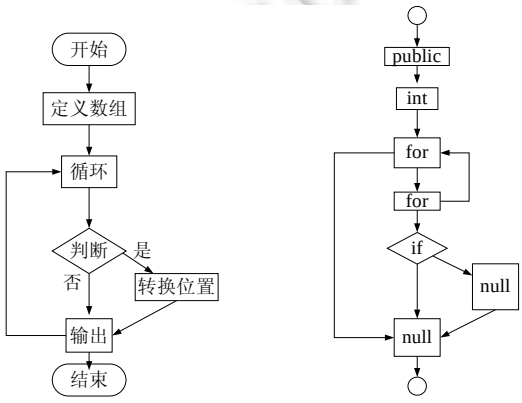
图 2 冒泡排序程序示例及其关键字流图



(a) 控制流图 (b) 关键字流图

Fig.3 Control flow graph and keyword flow graph of bubble sorting program

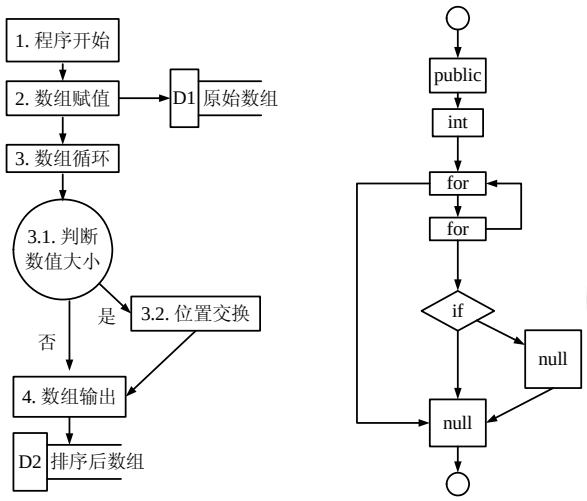
图 3 冒泡排序的控制流图及其关键字流图



(a) 程序流程图 (b) 关键字流图

Fig.4 Program flow chart and keyword flow graph of bubble sorting program

图 4 冒泡排序的程序流程图及其关键字流图



(a) 数据流图 (b) 关键字流图

Fig.5 Data flow diagram and keyword flow graph of bubble sorting program

图 5 冒泡排序的数据流图及其关键字流图

2.3 关键字流图最大公共子图的生成

待测程序根据第 2.2 节关键字流图的构造方法生成关键字流图,在关键字流图的基础上寻求它们的最大公共子图,最大公共子图的大小关系到待测程序的相似程度.动态规划算法[14]寻找公共子字符串是解决公共子字符串的经典算法之一,该算法能得到全局最优解.本节利用该方法求得相似程序的关键字流图最大公共子图.

在利用动态规划算法求解两个长度分别为 p,q 的字符串 S,T 的最长公共子串的问题之前,先给出求它们任意前缀子串对 $S[1,i],T[1,j]$的最长公共后缀的算法.该问题的递推关系如公式(5):

$$LCSuffix(S[1,i],T[1,j])=\begin{cases}LCSuffix(S[1,i-1],T[1,j-1])+S[i] & \text{if } (S[i]-T[j])\\ \text{""} & \text{otherwise}\end{cases}$$
$$\text{s.t. } i\leqslant p,1\leqslant j\leqslant q \tag{5}$$

其中,$LCSuffix(S[1,i],T[1,j])$表示前缀子串对 $S[1,i],T[1,j]$的最长公共后缀.

在字符串 S 和 T 所有前缀子串对的最长公共后缀中,长度最大的即为字符串 S 和 T 的最长公共子串,即:

$$LCS(S,T)=\max_{1\leqslant i\leqslant p,1\leqslant j\leqslant q}LCSuffic(S[1,i],T[1,j]) \tag{6}$$

其中,$LCS(S,T)$表示字符串 S 和 T 的最长公共子串.

关键字流图节点中的关键字可以看作由这些关键字组成的字符串中的一个字符,关键字流图节点中的关键字构成字符串,利用动态规划算法生成关键字流图最大公共子图,步骤如下.

1) 利用公式(5)和公式(6)求得最长公共子串;
2) null 字符代替两个关键字字符串中的最长公共子串.在进行关键字比较时遇到同为 null 时,需要考虑 null 所在节点边的情况,若来源于相同的关键字而且又指向相同的关键字,则将该节点计入最大公共子图中;
3) 判断最长公共子串的长度是否大于 0:若长度大于 0,重复步骤 1)和步骤 2);否则,结束.

例如,字符串 S=“public,int,if,null,for,if,null”,T=“public,int,for,for,if,null”,使用递推关系(见公式(5)和公式(6))求得字符串 S 和 T 所有前缀子串对的最长公共后缀,见表 2.

字符串 S 为快速排序程序关键字流图的节点中关键字的组合,T 字符串则代表了冒泡排序程序关键字流图中关键字的组合.从表 2 中可以看出:字符串 S 和 T 的公共子串为“for,if”和“public,int”,字符串 S 和 T 的最大公

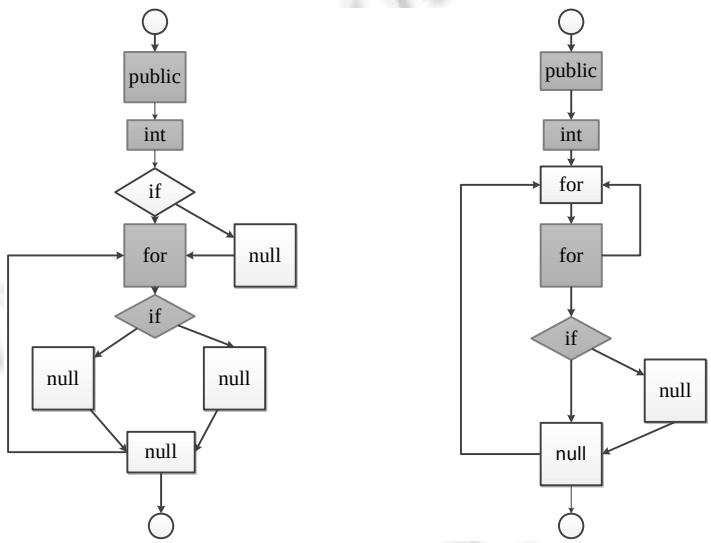
共子串为“public,int,for,if”.字符串 S 和 T 分别为快速排序和冒泡排序关键字流图节点存储信息组合而成.根据字符串 S 和 T 的最大公共子串构建关键字流图最大公共子图.

Table 2 Finding common keywords using dynamic programming algorithm

表 2 动态规划法查找公共关键字

T \ S	public	int	if	null	for	if	null
public	public	“.”	“.”	“.”	“.”	“.”	“.”
int	“.”	Public,int	“.”	“.”	“.”	“.”	“.”
for	“.”	“.”	“.”	“.”	for	“.”	“.”
for	“.”	“.”	“.”	“.”	for	“.”	“.”
if	“.”	“.”	“.”	“.”	“.”	for,if	“.”
null	“.”	“.”	“.”	“.”	“.”	“.”	“.”

图 6 是冒泡排序和快速排序关键字流图的最大公共子图(灰色部分).灰色部分节点存储了字符串 S 和 T 公共子串中的关键字,关键字流图的最大正是由 S 和 T 公共子串的关键字所在的节点构成.



(a) 快速排序关键字流图 (b) 冒泡排序关键字流图

Fig.6 Maximum common sub-graph of keyword flow graph (in gray)

图 6 关键字流图的最大公共子图(灰色部分)

2.4 程序相似性的判定

待测程序快速排序与冒泡排序的关键字流图最大公共子图已经生成,本节利用最大公共子图距离算法(见定义 5)计算程序相似程度.最大公共子图距离公式(公式(1))中,$|G_1|$和$|G_2|$分别表示程序快速排序和冒泡排序的节点数,$mcs|G_1,G_2|$表示最大公共子图的节点数.由图 6 知:冒泡排序与快速排序存储关键字的节点都为 5,关键字流图最大公共子图的节点为 4,故 $\max(|G_1|,|G_2|)=5$,$mcs|G_1,G_2|=4$,将具体数值带入公式,可得最大公共子图距离 $D(G_1,G_2)=0.2$.公式(2)将程序相似度定义为 $1-D(G_1,G_2)$,已知冒泡排序和快速排序的最大公共子图距离 0.2,故两程序的相似度 $S(G_1,G_2)=(1-D(G_1,G_2))\times100\%=80\%$.

基于关键字流图判定程序相似度的伪代码见算法 1.

算法 1. 程序相似度的判定.

输入:待测程序 *program*1,program2;

输出:待测程序相似度 *similarity*.

begin

1 *keywords*←*findKeywords*(*program*1,*program*2); //查找待测程序源代码基本块中关键字

2 *KFGs*←*buildKFGs*(*keywords*); //分别构建待测程序的关键字流图

3 *subgraphs*←*findSubgraphs*(*KFGs*); //利用动态规划算法寻找关键字流图的最大公共子图

4 *Knum*,*Snum*←*nodeNumber*(*KFGs*,*Subgraphs*); //计算每个关键字流图的节点数,最大公共子图节点数

5 *similarity*←*calculateSimilarity*(*Knum*,*Snum*); //计算待测程序相似度

6 **return** similarity;

end

本节采用自适应的方法进行实验来选取程序是否相似的阈值[23],实验对象为已知功能相近的程序或者不相近的程序,实验方法为本节所提的程序相似性检测方法.表 3 和表 4 分别给出了本文方法验证部分基础程序及系统中常用功能程序的相似性实验结果.

Table 3 Testing of the similarity between basic programs (%)

表 3 基础程序间相似性的检测 (%)

	插值查找	二分查找	折半查找	顺序查找	基数排序	简单选择	冒泡排序	希尔排序
插值查找	100	71	75	71	56	50	50	43
二分查找	71	100	75	62	43	37	37	43
折半查找	75	75	100	100	43	37	37	43
顺序查找	71	62	100	100	43	37	37	43
基数排序	56	43	43	43	100	71	71	86
简单选择	50	37	37	37	71	100	100	67
冒泡排序	50	37	37	37	71	100	100	67
希尔排序	43	43	43	43	86	67	67	100

Table 4 Testing of similarity between commonly-used programs in the system (%)

表 4 系统常用功能程序间相似性的检测 (%)

	用户登录 *b*	登录验证	信息下载	身份验证 *b*
用户登录 *a*	84	44	42	38
身份验证 *a*	37	55	43	92
数据处理	32	36	38	47

表 3 考察了多个查找算法和排序算法两种不同功能的程序.从表中实验的数据可以看出:功能相近程序的相似性皆超过 70%;而不同功能的程序即使程序规模相近,其相似度也在 70%以下.本文将 70%设定为程序是否相似的阈值.

表 4 中选用系统中常用的功能程序考察其相似度.用户登录 *a* 与用户登录 *b* 是两种不同脚本的登录算法,登录验证为用户登录时后台的验证算法.用户登录、身份验证、数据处理和信息下载为同一系统下不同功能的算法.表 4 中的实验结果再次验证了功能相同的程序间相似性较高,而功能不同的不相似程序的相似度均在 70%以下,证明本文将判定程序是否相似的阈值设定在 70%的合理性(在判定规模较小的程序时存在一定的偶然性,应将特例排除).

2.5 用关键字流图比较程序相似性的优势

公共关键字流图子图比较程序相似性具有以下优势.

1) 将程序转化成关键字流图,关键字是代码的核心标识符,关键字相同的节点意味着生成该节点的代码语句结构相同.通过比较关键字流图来判定程序的相似性,解决了两个程序功能相同但源代码差异较大的问题;

2) 本文是在关键字流图最大公共子图的基础上比较程序的相似度.相对于最长公共子串比较相似性,公共关键字流图子图更大程度比较了两个程序,避免了最长公共子串比较程序只取决于最长公共子串大小片面性的问题;

3) 关键字流图的节点代表程序中的基本块,用关键字流图比较程序的相似性比较方便.而不在源代码上直接比较关键字是否相似,因为从源代码上比较关键字是以行代码为单位,容易造成程序代码的书写格式不同带来的误差.

例如,图 7 为冒泡排序另一种编写方式的部分代码,由于编程习惯不同,有些程序员会把第 2 行~第 4 行代码写成“int *t*, *i*, *j*;”,两种书写方式功能完全相同.第 2 行~第 4 行代码的作用是定义整型变量,属于一个基本块,会生成关键字流图的一个节点.而基于源代码进行关键字比较时,则会因为不同的书写格式造成不同的结果.

```
1   public void bubbleSort(int array[·]) {
2       int t=0;
3       int i=0;
4       int j=0;
5       for (i<array.length−1; i++)
6           for (j<array.length−1−i; j++)
```

Fig.7 Partial source code of bubble sorting program

图 7 冒泡排序部分源代码

2.6 程序相似性检验插件的原型

插件服务于软件项目主体,有效地扩展和完善了寄主软件的功能[24].基于本文第 2.2 节~第 2.4 节程序相似度比较的方法,开发了一个检测程序相似程度的插件,如图 8 所示.此插件的制作,方便了程序相似性的比较.下一节探究相似程序间测试用例重用的问题,其前期工作就是需要判断程序之间的相似程度,此插件将减少了判断程序相似性的工作.

插件的开发选用 Java 作为编辑语言,开发环境为 MyEclipse 2010.计算机配置为 Windows(Intel(R) Core (TM) CPU i5-6500,3.20GHz,8.00GB RAM,64 位操作系统.图 8 为检测程序相似程度的插件运行界面.

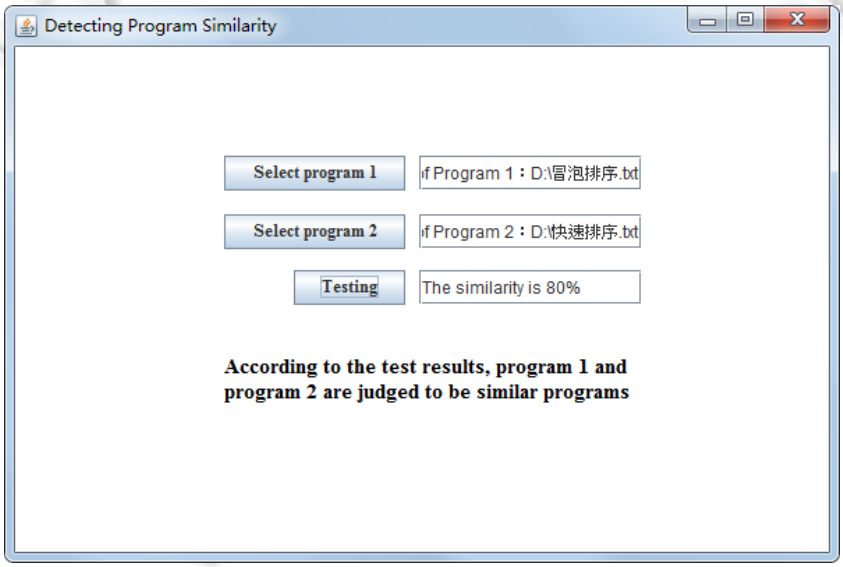


Fig.8 Plug-in for detecting program similarity

图 8 检测程序相似性的插件

按钮“Select program 1”和按钮“Select program 2”是两个选择待测程序的按钮,测试程序的相似度时,分别点击这两个按钮选择待测程序所在的文件,实验选择的待测程序为冒泡排序与快速排序.点击“Testing”按钮,将以百分数的形式显示两个待测程序的相似度.

通过检测,冒泡排序与快速排序的相似度为 80%,大于设定的阈值,可以判定冒泡排序和快速排序为相似程

序,并被设定为下一节研究相似程序间测试用例重用选择的实验对象.

3 基于遗传算法的相似程序间测试用例的重用

本节利用遗传算法完成测试用例的生成,实现测试用例的重用.遗传算法作为一种基于自然选择原理和自然遗传机制的通用搜索算法[25,26],通过进化过程获得的信息自行组织搜索,适应度大的个体具有较高的生存概率,并获得更适应环境的基因结构.在进化的遗传操作中,适应度较高的个体以更大概率将更优秀的基因遗传给下一代.

遗传算法具有良好的可扩展性,容易与其他算法相结合,也可以调节遗传操作和适应度函数等方式提升算法的效率.本节在种群进化过程上引入适应度较高的个体来提高测试用例生成的效率.其基本思路(重用模型如图 9 所示)如下.

1) 假设一个程序适应度较高的测试用例已经生成,将这些测试用例应用到其相似程序的测试中.实验采用遗传算法来完成测试用例的重用;

2) 利用遗传算法测试待测程序时,将重用的测试用例引用到遗传操作的种群进化中,种群的其他个体通过向这些测试用例学习,目的是加快种群进化速度,提高测试用例的生成效率.

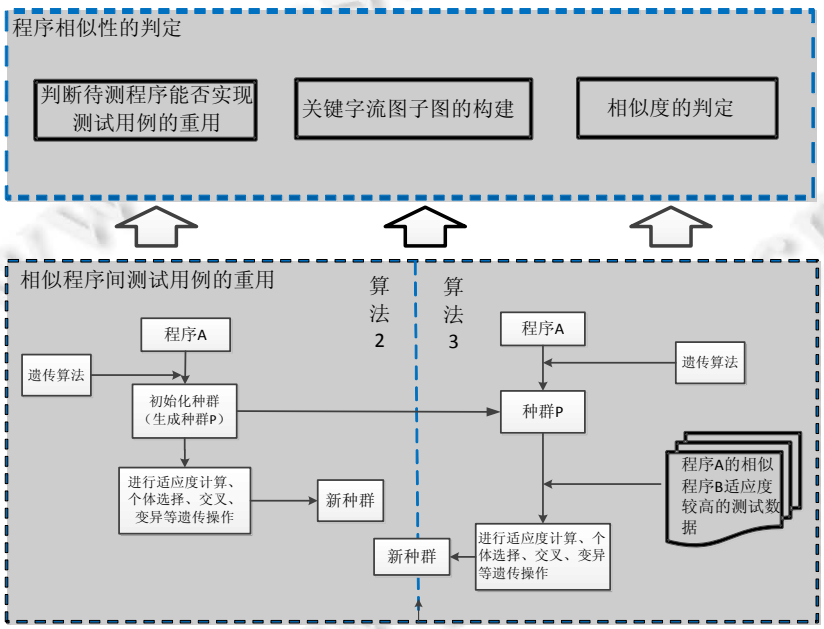


Fig.9 Reuse model of test cases for similar programs

图 9 相似程序间测试用例重用模型

3.1 适应度函数

遗传算法具有种群初始化、个体评价、选择运算、交叉运算、变异运算、进化终止条件判断等步骤.遗传算法的初始种群通常采用随机的方式生成,个体评价通过相应的适应度函数计算种群中各个个体的适应度值,选择运算就是将适应度较高的个体进行交叉变异操作,产生的个体组成下一代种群.重复选择进化的过程,直至终止条件满足,算法结束.

适应度函数是衡量种群个体优劣的标准[27],根据优胜劣汰的生存法则,通过淘汰生存能力低的个体实现种群的选择、进化.因此,适应度函数决定了种群的进化速度.合理的适应度函数能够全面提高种群个体的质量,有利于快速进化至最优解.适应度函数的设置是算法操作重要的一环.

分支距离法[28]和层接近度法[24]是两个设置个体适应度的经典方法.为了便于与主要参考文献作比较,实验选用了相同的函数(即分支距离法)设置个体的适应度.在程序的每个分支节点都插桩分支函数 f_i(插桩方式见图10),记录当前的测试用例与该分支的距离.当某分支被覆盖时,则将 f_i 设为0,若该目标路径共含有 m 个分支节点,其总的适应度函数值 FT 的计算见公式(7):

$$FT=\frac{\sum_{i=0}^{m-1}\frac{1}{f_i+1}}{m}\times 100\% \tag{7}$$

由适应函数的计算公式可以推算出,测试用例的适应度跟分支节点覆盖率成正比例关系.特别地,当程序的每个分支节点均被覆盖时,该测试用例的适应度为 1.

例如,对于数组 *int a*[·]={3,4,2}的测试用例测试冒泡排序,参考图 10 冒泡排序插桩后的伪代码.由于 $a[0]<a[1]$,则第 6 行的条件语句不满足,执行第 10 行的表达式,得 $f_2=|(3-4)|=1$.又由于 $a[1]>a[2]$,该条件语句满足,此时 $f_4=0$.则该测试用例与各个分支的距离为{0,0,1,0,0,0,0},此用例下分支节点个数 $m=7$,将数据带入适应度函数计算公式得 $FT\approx 0.87$.

```
1    void bubbleSort (int i; int j; int length; int a[·]; int count ){
2        for (i=0; i<length; i++){
3            count++; fcount_{t-1}=0;
4            for (j=0; j<length−1−i; j++){
5                count++; fcount_{t-1}=0;
6                if (a[j]>a[j+1]) {
7                    count++; fcount_{t-1}=0;
8                    a[j]=a[j+1]
9                }
10               count++; fcount_{t-1}=Math.abs(a[j]−a[j+1])
11   } } }
```

Fig.10 Pseudo-code of bubble sorting program after being instrumented

图 10 冒泡排序插桩后的伪代码

3.2 测试用例生成算法

测试用例生成算法采用 Java 语言编写,并在 MyEclipse 2010 中运行.计算机配置为 Windows(Intel(R) Core(TM) CPU i5-6500,3.20GHz,8.00GB RAM,64 位操作系统.算法的具体流程见算法 2(传统方法).

算法 2(传统方法).

输入:种群大小 *pop_size*,个体 *individual*,染色体长度 *chro_size*,进化代数 *gen_size*,交叉概率 *pc*,变异概率 *pm*;

输出:新种群.

begin

1 *initialize*(*pop_size*,*chro_size*); //初始化种群

2 **for** *i*←:*gen_size*

3 *individualSign_list*←*quicksort*(*pop_size*,*individual*);

4 *fitness_list*←*finess*(*pop_size*,*individual*,*individualSign_list*);

//第 3 步和第 4 步这两步种群个体测试冒泡排序,生成能表示节点距离的序列,计算个体适应度

5 *individual_list*←*select*¬(*pop_size*,*individual*,*fitness_list*,*elitism*);

//轮盘赌法选择个体

6 *pop*1_*list*←*crossover*1(*pop_size*,*chro_size*,*individual_list*,*pc*); //交叉产生新个体

7 *mutate*1(*pop_size*,*chro_size*,*pop*1_*list*,*pm*); //变异过程

end for

end

算法 2 是利用传统方法生成测试用例的遗传算法,采用随机方式初始化种群,轮盘赌法选择个体,以一定概

率进行交叉与变异等操作,生成新的种群;判断新种群中是否有个体满足目标路径被覆盖,若其被覆盖,记录种群的进化代数;判断是否满足算法的终止条件,即种群是否达到最大进化代数,若满足则终止算法,若不满足则循环种群进化的过程.

算法 3 将测试用例重用到待测程序的测试数据生成中,并作为对比实验检验测试用例的重用效果.该算法的初始种群采用算法 2 中的初始种群,以避免初始种群不同对实验结果造成的影响.在种群交叉进化的遗传操作过程中,算法 3 引入相似程序的测试用例作为个体交叉的对象,其他步骤与算法 2 一致.

算法 3(本文方法).

输入:种群大小 *pop_size*,个体 *individual*,染色体长度 *chro_size*,进化代数 *gen_size*,交叉概率 *pc*,变异概率 *pm*,引入的新个体 *shared_pop*;

输出: 新种群.

begin

1 *initialize*(*pop_size*,*chro_size*); //初始化种群

2 **for** *i*←:*gen_size*;

3 *individualSign_list*←*quicksort*(*pop_size*,*individual*);

4 *fitness_list*←*finess*(*pop_size*,*individual*,*individualSign_list*);
//第 3 步和第 4 步这两步种群个体测试冒泡排序,生成能表示节点距离的序列,计算个体适应度

5 *individual_list*←*select*¬(*pop_size*,*individual*,*fitness_list*,*elitism*);
//轮盘赌法选择个体

6 *pop2_list*←*crossover2*(*pop_size*,*chro_size*,*individual_list*,*shared_pop*,*pc*);
//与引入的个体交叉产生新的个体

7 *mutate2*(*pop_size*,*chro_size*,*pop2_list*,*pm*) //变异过程

end for

end

3.3 测试用例生成实验

选取第 3 节讨论的两个相似程序(即冒泡排序和快速排序)为实验对象,提出两个问题探究本文判断程序相似性方法的合理性以及相似程序间测试用例重用的有效性,收集实验结果加以分析.具体问题如下:

问题 1. 本文设计的关键字流图比较程序相似性,相对于其他方法比较程序相似性有什么优点?如何检验方法的有效性?

最长公共子串等方法只比较代码的相同程度,而忽略了源代码略有差别、但功能结构却相同的情况.提出利用关键字流图比较程序的相似性,将比较全部源代码简化为比较代表代码语句功能结构的关键字.

最大公共子图距离能够判断待测程序的相似程度.而通过遗传算法将程序已有的测试用例应用到相似程序的种群进化的遗传操作中,检验测试用例的生成效率来判断程序相似性判断的有效性.即:通过用例的共享,待测程序的测试效率明显提高,可证明两个程序相似性判断是正确的.

问题 2. 相似程序之间用例的共享能够减少多少测试工作量,提高多少测试效率?

检测关键字流图判别程序有效性的实验,也能完成用例的重用.我们结合姜淑娟等人[20]的思想,在初始化种群的环节中引入相似程序中已有的测试数据,种群其他的个体通过向这些测试用例的学习加速种群进化,主要通过 4 个评判标准来检验测试效率.评判标准见第 3.3.2 节.

我们采用对比实验来解答上面两个问题.实验的不同点主要在于种群进化的方式不同.

- 传统方法测试程序,初始种群随机生成,在初始种群中用轮盘赌法选择两个体进行遗传操作,轮盘赌法选中的两个个体以一定概率进行变异:若满足交叉条件,则两个个体进行交叉,进化成两个新个体.循环此操作,直到产生的新种群个体数与初始种群相同;
- 而利用本文方法测试程序,初始种群使用传统方法生成的种群,以排除初始种群不同对实验结果造成

的影响.引入相似程序已经生成的测试用例集,在种群进化的交叉部分,仍用轮盘赌法选择的个体与该用例集中的个体交叉,生成两个新的个体.其他操作与传统方法相同.

3.3.1 实验对象

第 2 节表明:冒泡排序和快速排序两个程序的相似度为 80%,判定为相似程序.实验对象仍选取此相似程序来探究相似程序间测试用例的重用问题.

实验假设快速排序的适应度较高的测试用例已知.研究测试用例的重用就是将已知的测试用例共享于与其相似的程序,故测试冒泡排序程序时,将引入快速排序已有的测试用例.测试用例有 32 个二进制的字符,其中,4 个二进制字符代表一个十进制的阿拉伯数字,意味着每个测试用例代表着长度为 8 的整数数组,整数大小在 0 到 15 之间.测试用例的生成采用遗传算法,种群的个数为 40.具体参数设置见表 5.

Table 5 Parameters setting in genetic algorithms

表 5 遗传算法中的参数设置

遗传操作	取值
选择策略	轮盘赌法选择
交叉策略	多点交叉
交叉概率	0.9
变异策略	单点变异
变异概率	0.1
最大进化代数	1 000
种群大小	40

3.3.2 评价标准

为了检验程序相似性检测方法的有效性及相似程序间测试用例重用的效果,并便于与丁蕊等人[28]提出的关键点路径法进行比较,同时也为了将遗传算法中随机因素对实验结果造成的影响降低到最小,我们将实验结果的 4 个评价标准(即覆盖率、平均进化代数、执行时间和方差分析)的定义如下.

1) 覆盖率:对程序运行 100 次,记录在最大进化代数 T=1000 内目标路径被覆盖的总次数,覆盖率为这 100 次中被覆盖次数的平均值,以百分比的形式表示;
2) 平均进化代数:目标路径被覆盖的每个程序运行 100 次,若该程序在最大进化代数 T=1000 内覆盖到了目标路径,记录此时程序的进化代数;若在最大进化代数内目标路径没有被覆盖,记录最大进化代数,平均进化代数即为这些记录值的平均值;
3) 执行时间:该评价指标记录两种方法(即传统方法与本文方法)的时间开销.传统方法生成测试用例的执行时间是测试数据或路径的选择与测试数据生成所消耗的时间总和;本文方法生成测试用例的时间除了测试数据或路径的选择与测试数据生成的时间以外,还包括判定程序相似性和测试用例重用花费的时间;
4) F 值:在方差分析中,每个被测程序运行 100 次,记录目标个体首次出现的进化代数,将传统方法与本文方法下获得的该数据作为两组样本来计算 F 统计量,即组间均方与组内均方的比值(见定义 6 和定义 7),该值能验证随机因素对实验结果造成的影响程度.

传统方法和本文方法均采用遗传算法测试同一程序,上面 4 种评价标准作为衡量两种方法测试效率优劣的指标,检验本文方法是否能够有效地提高测试用例的生成效率,完成测试用例的重用.

3.3.3 实验结果分析

测试冒泡排序来验证相似程序间测试用例重用的有效性,传统方法中的测试种群通过轮盘赌法选择个体进行交叉变异产生下一代,而作为对比实验的本文方法在种群进化中将引入相似程序快速排序适应度较高的测试用例.为了减少随机因素的影响,两种实验均独立运行 100 次,本文方法采用的初始种群为冒泡排序随机生成的初始种群.种群进化达到最大进化代数时算法终止,并记录 100 次实验共有的总时间.

表 6 表明:将快速排序适应度较高的测试用例运用到冒泡排序的测试中,测试用例的生成效率明显提高.其

中:在传统方法的 100 次独立实验中,在种群最大进化代数前只有 15 次能够覆盖目标路径,覆盖率为 15%,平均进化代数为 890.9,所用的总时间为 8.20s;采用本文方法的实验中的目标路径覆盖率为 100%,平均进化代数为 3.5,所用总时间为 2.74s.其 F 值为 160.36,经查表,组数为 2,每组数据量在 100 时,F 界值为 3.04,本文方法与传统方法的目标个体的进化代数作为两组数据所得 F 值远远大于 F 界值,意味着本文方法有效地改变了目标个体的平均进化代数,排除随机因素对实验造成的干扰.从实验结果可知,目标路径覆盖率、平均进化代数和时间 3 个衡量指标都能证明相似程序间测试用例重用效果明显.

Table 6 Evaluation result of testing bubble sorting program using traditional method and our method

表 6 传统方法和本文方法测试冒泡排序的评判结果

方法	覆盖率(%)	平均进化代数	执行时间(s)	F 值
传统方法	15	890.9	8.20	160.36
本文方法	100	3.5	2.74	160.36

为更加全面检测用例重用的效果,实验将改变种群大小,检验种群大小是否对实验结果造成较大影响.表 7 是不同种群大小下传统方法和本文方法在冒泡排序中的测试结果.

Table 7 Test results of bubble sorting program under different population sizes

表 7 不同种群大小下冒泡排序的测试结果

种群大小	评判标准	传统方法	本文方法
30	覆盖率(%)	9	100
	平均进化代数	895.3	5.3
	时间(s)	4.32	2.58
	F 值	158.97	
60	覆盖率(%)	22	100
	平均进化代数	863.0	4.1
	时间(s)	7.26	3.66
	F 值	149.98	
100	覆盖率(%)	26	100
	平均进化代数	853.1	3.3
	时间(s)	7.96	4.12
	F 值	153.37	

表 7 的实验结果验证了不同种群下对比实验的覆盖率和平均进化代数.实验结果显示:随着种群规模的增大,传统方法中的目标路径的覆盖率有增大的趋势,平均进化代数随之减少,F 值变化很小,但仍远大于 F 界值 3.04.本文方法下的实验结果表明:3 种种群大小下的目标路径覆盖率都为 100%,当种群的个体数量增加时,平均进化代数随之减少.该方法下的 100 次独立实验中目标路径皆被覆盖,平均进化代数远低于传统方法,测试效率比传统方法明显要高.表 8 是不同变异概率下测试冒泡排序的结果.

Table 8 Test results of bubble sorting program under different mutation probabilities

表 8 不同变异概率下冒泡排序的测试结果

变异概率	评判标准	传统方法	本文方法
0.1	覆盖率(%)	15	100
	平均进化代数	890.9	3.5
	时间(s)	8.33	2.74
	F 值	153.35	
0.2	覆盖率(%)	21	100
	平均进化代数	887.4	4.7
	时间(s)	8.66	2.70
	F 值	167.75	
0.3	覆盖率(%)	22	100
	平均进化代数	887.2	4.2
	时间(s)	8.56	2.56
	F 值	143.76	

由表 8 的实验结果可知:变异概率的改变,给实验结果带来了很小的变化.变异概率由 0.1 变化到 0.3 时,冒泡排序中目标路径的覆盖率提高了 7%,平均进化代数减少 3.7,F 值仍远大于 F 界值 3.04.本文方法测试结果表明:在 3 种变异概率下,其目标路径的覆盖率皆为 100%.对比两个实验的实验结果可知:3 种变异概率下使用本文方法的测试效率远远高于传统方法,相似程序检测时用例重用效果较好.

表 9 中的数据为本文方法和业界测试用例生成效率较高方法的各性能指标.为了检测相似程序间测试用例的重用效果,排除其他因素造成干扰,本实验遗传算法的各种参数和关键点路径法[20]中的参数保持一致,并且选择相同的目标路径.通过比较平均迭代次数可看出,本文方法实验中的种群能更早地覆盖目标路径.

Table 9 Performance comparison between our method and the key-point path method

表 9 本文方法与关键点路径法性能比较

方法	输入分量数	关键点理论路径数	平均迭代次数
关键点路径[20]	3	8	10.08
本文方法	3	8	5.3

3.4 进一步实验

为了证明面向关键字流图判断程序的相似性,并将测试用例重用到相似程序测试用例的生成中的效果较好,实验特别设计了基于控制流图、程序流程图及数据流图的方法来判断程序的相似性,并将相似度较高的程序作为实验对象检验测试用例的重用效果,以便于与我们基于关键字流图的方法作对比.实验按照与关键字流图相同的方法计算程序的相似性,例如:利用控制流图计算程序相似度时,首先将待测程序生成控制流图,接下来查找控制流图的最大公共子图.由于控制流图的节点中没有存储关键字等信息,故而查找公共节点时,考察有向边的数量与方向.若节点边的来向、去向和数量完全相同,则该节点属于公共子图中的节点.最后使用最大公共子图距离算法计算程序的相似度.详细过程可参考第 2 节.表 10 为待测程序的基本信息以及实验结果.

Table 10 Basic information and experimental results of the program to be tested

表 10 待测程序的基本信息与实验结果

程序名称	函数个数/代码行数	程序相似度(关键字流图/控制流图/程序流程图/数据流图)(%)	时间(s)	平均进化代数	覆盖率(%)
身份验证 登录验证	13/209 8/178	55/80/86/86	33.30	136.20	91
用户登录 登录验证	10/217 8/178	44/74/79/79	34.10	95.60	94
数据处理 信息下载	8/226 5/186	38/75/82/82	47.30	220.55	88

表 10 显示了 3 组待测程序的名称、子函数个数、代码行数以及基于关键字流图、控制流图、程序流程图与数据流图方法下的程序相似度.从数据中能够看出:在 3 组程序中,关键字流图下的程序相似度都低于 70%,而其他 3 种流图判定该程序的相似度时皆有较高的相似度(这样就超过了相似度设定的阈值,把本来不相似的程序也判定成相似了).实验结果显示了待测程序测试用例重用的效果(特别注意:表中的 3 个指标值是根据控制流图、程序流程图以及数据流图得到的结果,并且它们的测试过程一样,因此数据一致).从表中的 3 个评价指标能够看出:基于控制流图、程序流程图以及数据流图判定待测程序的相似度得到较高的值,其测试用例的重用在一定程度上确实降低了目标个体的平均进化代数,并且提高了其覆盖率,但均未达到 100%.结合第 3.3.3 节以及本文其他实验的结果,基于关键字流图相似程序间测试用例的重用的效果更好,其目标个体的覆盖率皆为 100%,而目标个体的平均进化代数也较为更低.

控制流图、程序流程图和数据流图判定程序相似性时只比较了程序的结构而没有考察图中节点的功能,而基于关键字流图比较程序相似性时,综合比较了程序的结构与功能.因此,判定程序的相似度更准确,其测试

用例具有更好的重用效果.

接下来,为了证明相似程序间测试数据是否可以相互引用,实验将测试冒泡排序已经生成的测试用例重用到快速排序的测试用例生成中.此外,进一步证明相似程序间测试用例重用的有效性.实验另外选取了两组典型基准程序和 5 组工业程序作为实验对象,分别将传统算法和本文方法利用到这些程序的测试用例生成中.实验对象的基本信息见表 11.

Table 11 Basic information of experimental subjects

表 11 实验对象的基本信息

程序编号	程序名称	函数个数	代码行数	程序相似度(%)	被测程序
PG1(https://www.cnblogs.com/morethink/p/8419151.html)	Bubble sorting	1	13	80	Quick sorting
PG2	Quick sorting	1	18		
PG3[2]	Sumday	1	22	75	Sumday
PG4	Test-interval	1	31		
PG5(https://blog.csdn.net/wwwliuzhi/article/details/78544093)	Triangle	1	15	74	E-triangle
PG6	E-triangle	1	20		
PG7(https://github.com/finnfu/stepcount)	Calculator1	18	286	73	Calculator1
PG8	Calculator2	20	341		
PG9(https://github.com/vencc/game)	Gobang_algorithm1	6	265	79	Gobang_algorithm1
PG10	Gobang_algorithm2	5	227		
PG11(明日科技编著:《java 项目开发实战入门》,吉林大学出版社,2017 年)	ABS_check1	13	209	97	ABS_check2
PG12	ABS_check2	15	228		
PG13[2]	tcas1	8	138	76	tcas1
PG14	tcas2	10	172		
PG15[2]	repalce1	21	516	82	repalce1
PG16	repalce2	18	446		
PG17	Calculator1	18	286	36	Calculator1
PG18	Gobang_algorithm1	6	265		
PG19	Calculator1	18	286	23	测试用例无法共享
PG20	ABS_check1	13	209		
PG21	Gobang_algorithm1	6	265	35	测试用例无法共享
PG22	ABS_check1	13	209		

说明:表中前 8 组中,同组程序出处来源相同.

第 2 组基准程序的 PG3 被广泛应用于验不同测试数据生成方法的有效性[2],PG4 是测试两个时间之间间隔多少天的函数,测试该函数与 PG3 函数输入的参数类型皆为年月日的形式;PG5 函数验证输入的 3 个数是否能组成三角形,PG6 函数在此基础上验证三角形是否是等腰或者等边三角形;工业程序 PG7 和 PG8 摘选自 Github 网站不同作者编写的实现计算器功能的程序;PG7 和 PG8 实现的功能相同,但编写方式和使用的函数存在不同,故将 PG7 和 PG8 作为实验对象.

PG9 是五子棋游戏的赢棋算法,PG10 是在 PG9 基础上做了一定的删减,去掉了悔棋功能.PG11 是通讯录系统个人信息的校验算法.PG12 增加了办公室电话的校验功能.本节对 PG9 和 PG11 两个工业程序做了不同维护类型的修改,用来验证本文方法是否适应于回归测试.回归测试是确认修改程序是否引入新的错误或导致其他代码产生错误[29].

统计数据表明:回归测试占了软件测试总预算的 80%,软件维护阶段总费用 50%以上.尽管回归测试的代价如此之高,但它却是不可或缺的测试[30,31].回归测试中有效地利用已经生成的测试数据,近几年得到普遍的关注.程序 PG13 和 PG15 来源于 SIR 测试平台的西门子测试程序集[2],常作为程序测试的实验对象.PG14 和 PG16 是采用不同方法对 PG13 和 PG15 更新后的程序.该组程序能够验证相似程序间测试用例的重用效果,还能够验证本文方法是否适合程序的回归测试.

为了验证不相似程序之间测试用例的重用效果,实验还补充了 3 组实验(见表 11 的最后 3 组),将不相似程序 PG7,PG9 和 PG11 的适应度较高的测试用例相互引用.对于程序功能规模差距较大的程序,其测试用例共用的可能性较小.其中,程序 ABS_check1 的测试用例无法与程序 Calculator1 和 Gobang_algorithm1 的测试用例完

成重用.而程序 Calculator1 和 Gobang_algorithm1 的测试用例规模相近,为了检验其重用效果,实验将 Gobang_algorithm1 的测试用例处理成与 Calculator1 的测试用例长度一致.

新增实验仍采用传统算法和本文方法进行对比,传统方法测试程序的各种参数设置与第 3.3.1 节传统方法测试冒泡排序的参数一致.本文方法利用遗传算法生成测试用例的种群进化过程引用传统方法生成的测试用例,引用的个数为 10 个.实验种群大小均为 40 个,进化代数统一为 1 000 代,详细参数设置见表 12.

Table 12 Population size and maximum evolution times

表 12 种群规模及最大进化代数

程序编号	种群规模	最大进化代数
PG2	40	1 000
PG3	40	1 000
PG5	40	1 000
PG7	40	1 000
PG9	40	1 000
PG12	40	1 000
PG13	40	1 000
PG15	40	1 000
PG17	40	1 000

表 13 为传统遗传算法和本文方法测试程序的执行结果,可见,利用本文方法生成测试用例的效率更高.

Table 13 Results of testing under the traditional method and our method

表 13 传统方法和本文方法下程序测试的结果

被测程序编号	传统方法			本文方法			F 值
	平均迭代次数	目标路径覆盖率(%)	时间(s)	平均迭代次数	目标路径覆盖率(%)	时间(s)	
PG2	736.30	30	8.71	36.13	100	7.80	40.75
PG3	121.32	91	2.68	3.96	100	2.57	18.65
PG5	462.38	64	6.84	11.2	100	3.77	12.76
PG7	600.28	43	62.90	6.14	100	62.55	44.21
PG9	141.68	90	8.30	8.94	100	8.04	16.34
PG12	259.54	79	32.60	24.84	100	32.50	34.40
PG13	244.20	82	5.48	19.37	100	5.05	28.33
PG15	297.95	75	82.38	26.55	100	78.05	29.57
PG17	589.47	45	63.15	187.01	84	61.84	42.88

下面分别从 4 个评价标准对比测试用例的生成效率.

1) 本文方法测试程序的目标路径覆盖率为 100%(注意,PG17 对应的是不相似程序);而传统方法测试程序其目标路径覆盖率最高为 91%,测试 PG2 的路径覆盖率仅有 30%;
2) 平均迭代次数方面,本文方法测试程序的平均迭代次数均在 40 代以内;传统方法除测试 PG3 的实验以外,其他程序测试实验的平均迭代次数均在 130 代以上,甚至超过 600 代;
3) 用时方面,利用本文方法测试程序的执行时间皆少于传统方法;
4) F 值的大小表明随机因素对实验的影响.实验结果中的 F 值皆远大于 F 界值,则说明排除了随机因素对该实验的影响,证明了本文方法的有效性.

实验 PG17 的结果表明:虽然不相似程序之间测试用例的重用在一定程度上提高了目标个体的进化速度,但目标路径覆盖率有所降低,还是不如相似程序间测试用例的重用效果.

由测试 PG2 的实验结果可知,本文方法有效地提高了测试用例的生成效率.可证明测试冒泡排序可以引用快速排序已经生成的测试用例,测试快速排序也可以使用冒泡排序已经生成的测试用例.即,相似程序之间测试用例是通用的.观察 PG9,PG12,PG13 以及 PG15 的实验结果,本文方法的前 3 个评判标准都优于传统方法,这说明本文方法运用到回归测试是可行的.相似程序间测试用例的重用测试程序是将已经生成的测试用例重用到相似程序测试用例的生成过程.在种群进化的过程中,种群个体不断地向引入的优秀个体学习,实验证明,本文方法能够提高测试效率.本文方法的测试用例重用能够取得这么好的结果,其主要原因包含两个方面.

1) 利用遗传算法测试程序的过程中,引用了相似程序适应度较高的测试用例,引入的测试用例可以看作

是待测程序适应度比较高的测试用例,甚至某些测试用例作为目标个体参与了种群的进化,因此在种群迭代 1 000 次的情况下,种群进化出目标个体(覆盖率 100%)是大概率事件;

2) 容易陷入局部最优是遗传算法的弊端之一,实验发现:当种群进化到一定代数之后,种群基因种类较少,许多个体的基因相同或者非常相近,很大程度上阻碍了目标个体的生成;而外部个体的引入能够丰富种群基因,加速目标个体的进化.

3.5 问题回答

通过分析第 3.3.3 节和第 3.4 节的实验结果,对第 3.3 节开头中提出的两个问题做如下回答.

回答问题 1. 本文设计的关键字流图比较程序相似性,相对于其他方法比较程序相似性有什么优点?如何来检验方法的有效性?

第 2.5 节已经讨论过关键字流图比较程序相似性的优势,程序的相似程度可以通过最大公共子图距离进行判定.通过相似程序间测试用例的共享,实现数据用例的重用.可根据测试用例的重用效果证实关键字流图比较程序相似性方法的有效性.由表 6 传统方法与本文方法测试程序的评判结果,传统方法的覆盖率为 15%,平均进化代数为 890.9 代;而作为引入相似程序测试用例的本文方法的目标路径覆盖率为 100%,平均进化代数为 3.5.表 11 增加了两组基准程序和 5 组工业程序,本文方法测试程序的目标路径覆盖率皆为 100%(注意:PG17 除外,因其为不相似程序的测试),前 3 项评判标准皆优于传统测试方法,第 4 项评价指标能够排除实验的随机影响因素,从而证明了本文方法的有效性.由此可知:将相似程序(如,快速排序和冒泡排序)的测试用例相互引入到对方的测试中,测试效率显著提高.由此证实了本文程序相似性判定方法的有效性.

回答问题 2. 相似程序间用例的共享能够减少多少测试的工作量,提高多少测试效率?

从表 7 和表 8 可看出:在不同的种群规模和不同的变异概率下,本文方法目标路径的覆盖率都是 100%,4 项评价指标充分证明了本文方法优于传统方法.结合表 6 和表 13,在所有程序的测试中,本文方法下的目标路径覆盖率均为 100%(注意:PG17 除外,因其为不相似程序的测试),平均进化代数最高(即最差情况)也仅为 36.13;而传统方法目标路径覆盖率最大才达到 91%,平均进化代数除 PG3 以外均超过 130.可见,本文方法的测试效率显著高于传统方法.因为本文方法在进化遗传操作中引入了相似程序适应度较高的测试用例,种群在进化时随机地与这些个体进行交叉,其收敛速度明显高于传统方法.

3.6 有效性分析、存在的问题及其初步解决方案

影响实验效果的因素包括内部因素和外部因素.

1) 内部因素主要是遗传算法本身对实验结果造成的影响.种群初始化采用随机生成的方法,轮盘赌法选择个体进行遗传操作也具有一定的随机性,为了降低算法随机性带来的影响,进行 100 次独立的实验,实验结果取 100 次重复实验的平均值;

2) 外部因素包括程序本身对实验结果的影响.与实验选取的对象有关,本文方法测试不同组的程序,测试用例的重用效果略有差别.另外,实验发现:本文方法的实验结果受引入的测试用例的个数和适应度的影响,一定范围内引入的测试用例个数越多和采用较好适应度的测试用例往往能取到相对较好的结果,实验中引入的测试用例为传统方法生成的测试用例.

针对用例重用的问题,本文探究了相似程序间测试用例的重用,实验取得了较好的结果,证明了方法的有效性.然而,本文仍存在以下不足.

1) 本文方法在判定规模较小程序的相似性时会具有一定的偶然性,其准确性有待进一步提高;

2) 存在有些相似程序的用例长度不同的情况,测试数据无法共享,该条件有一定的局限性.

若判定为相似程序的测试用例长度不同,那么在遗传算法生成测试用例的过程中,种群个体不能与相似程序的测试用例进行交叉进化.若将引入个体的长度与进化种群个体的大小改成一致,则能够完成用例长度不同的程序间测试用例的重用.

本文就存在的不足设想如下初步解决方案.

1) 若待测程序的测试用例长度小于相似程序测试用例的长度,减小相似程序测试用例的长度,使待测程序与相似程序的测试用例长度相同,根据关键字流图最大公共子图,删减不属于最大公共子图节点对应测试用例的部分;
2) 若待测程序的测试用例长度大于相似程序测试用例的长度,增加相似程序测试用例的长度,使待测程序与相似程序的测试用例长度相同.找出相似程序关键字流图不存在关键字的节点所对应的位置,随机增添最大公共子图对应相似程序的测试数据,至待测程序与相似程序测试用例的长度相同.

4 总结及进一步研究

本文提出了一种面向关键字流图判断相似程序,并重用已知的测试用例来完成相似程序测试的方法.其主要贡献如下.

1) 提出一种面向关键字流图的程序相似性的比较方法.通过构建的关键字流图求关键字流图最大公共子图,利用最大公共子图距离算法计算程序的相似度.该方法兼顾比较程序的序列和功能结构的相似性.通过关键字流图判定相似的程序,程序规模相近,程序功能结构相似;
2) 提出一种基于相似程序程度间测试用例重用的方法.将程序已经生成的测试用例重用到相似程序的测试用例生成中,在遗传算法进化时,引入相似程序适应度较高的测试用例,种群个体以一定概率与引入的测试用例进行交叉变异,有利于加快种群的进化速度,提高测试用例的生成效率;
3) 开发了一个判定程序相似性的插件,该插件根据本文所提出的方法对程序相似性进行判定,用户选择放在两个待测程序的文件(每个文件内只放待测程序的代码),点击测试按钮运行插件.执行结束返回两个程序的相似度,并根据设定的阈值判定它们是否相似.

在第 3.6 节中分析了本文存在的不足,就测试待测程序需要输入测试数据受到个数限制而无法重用的问题,设想了初步的解决方案.在未来的工作中,进一步研究相似程序间测试用例重用的局限性,特别针对相似程序因测试数据数量受限而不能重用的问题,深入分析前面假设的可行性,以提高相似程序间测试用例重用的通用性与准确性.

References:

[1] Towey D, Dong Y, Sun CA, Chen TY. Metamorphic testing as a test case selection strategy. Science China Information Sciences, 2016,59(5):050108.
[2] Wu C, Gong DW, Yao XJ. Selection of paths for regression testing based on branch coverage. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):839−854 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4975.htm [doi: 10.13328/j.cnki.jos.004975]
[3] Li YY, Huang S, Hui ZW, Li LY. Test case retrieval method for different data model. Computer Science, 2017,44(11):221−225 (in Chinese with English abstract).
[4] Qiu XK, Li XD. A path-oriented tool supporting for testing. Chinese Journal of Electronics, 2004,32(S1):235−237 (in Chinese with English abstract).
[5] Beizer B. Software Testing Techniques. New York: Van No Strand Reinhold, 1990.
[6] Mu YM, Gao XX, Shen M. Research of reuse technology of test case based on function calling path. Chinese Journal of Electronics, 2018,27(4):768−775.
[7] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.004420]
[8] Shan JH, Jiang Y, Sun P. Research progress in software testing. Acta Scientiarum Naturalium Universitatis Pekinensis, 2005,41(1):134−145 (in Chinese with English abstract).
[9] Zhang J. Research on teat case reuse in software testing [Ph.D. Thesis]. Shanghai: Shanghai University, 2012 (in Chinese with English abstract).
[10] Chen H, Wang GN. Research for measuring softw are similarity based on graphs [MS. Thesis]. Changsha: Hunan University, 2009 (in Chinese with English abstract).

[11] Kwon J, Lee H. BinGragh: Discovering mutant malware using hierarchical semantic signatures. In: Proc. of the 7th Int'l Conf. on Malicious and Unwanted Software. Fajardo: IEEE, 2012. 104−111.

[12] Wang Z, Piercek, Mcfarling S. Bmat--A matching tool for stale profile propagation. Journal of Instruction Level Parallelism, 2000,2:1−6.

[13] Levenshtein V. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics-Doklady, 1966,10(8): 707−710.

[14] Wang KY, Kong SQ, Fu YS, Pan ZY, Ma WD, Zhao Q. Two longest common substring algorithms based on bi-directional comparison. Journal of Computer Research and Development, 2013,50(11):2444−2454 (in Chinese with English abstract).

[15] Rong C, Silva YN, Li C. String similarity join with different similarity thresholds based on novel indexing techniques. Frontiers of Computer Science, 2017,11(2):307−319.

[16] Mc.Cabe TJ. A complexity measure. IEEE Trans. on Software Engineering, 1976,22(4):308−320.

[17] Mayrhauser AV, Mraz R, Walls J, Ocken P. Domain based testing: increasing test case reuse. In: Proc. of the IEEE Int'l Conf. on Computer Design: VLSI in Computers & Processors. IEEE, 2002.

[18] Vouffo-Feudjio A, Schieferdecker I. Test patterns with TTCN-3. In: Proc. of the Int'l Workshop on Formal Approaches to Software Testing. 2004. 170−179.

[19] Gong DW, Ren LN. Evolutionary generation of regression test data. Chinese Journal of Computers, 2014,3(11):489−499 (in Chinese with English abstract).

[20] Jiang SJ, Wang LS, Xue M, Zhang YM, Yu Q, Yao HR. Test case generation based on combination of schema using particle swarm optimization. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):785−801 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4966.htm [doi: 10.13328/j.cnki.jos.004966]

[21] Bunke H. On a relation between graph edit distance and maximum common sub-graph. Pattern Recognition Letters, 1997,18: 689−694.

[22] Bunke H, Shearer K. A graph distance metric based on maximal common sub-graph. Pattern Recognition Letters, 1998,19: 255−259.

[23] Liu JN, Xing Q, Zhao WD. Program similarity detection algorithm. Computer and Digital Engineering, 2015,43(12):2145−2149 (in Chinese with English abstract).

[24] Qian ZS, Hong DF, Wang XJ. A plug-in test case generation method based on contact layer proximity and node probability coverage. Int'l Journal of Performability Engineering, 2017,13(8):1281−1292.

[25] Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning. MA: Addison-Wesley, 1989.

[26] Holland HJ. Adaptation in Natural and Artificial Systems. MIT Press, 1992.

[27] Qian ZS, Wang XJ. An approach to resource scheduling based on user expectation in cloud testing. Int'l Journal of Performability Engineering, 2017,13(8):1206−1218.

[28] Ding R, Dong HB, Zhang Y, Feng XB. Fast automatic generation method for software testing data based on key-point path. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):814−827 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4971.htm [doi: 10.13328/j.cnki.jos.004971]

[29] Muccini H, Dias M, Richardson D. Towards software architecture-based regression testing. ACM SIGSOFT Software Engineering Notes, 2005,30(4):1−7.

[30] Ding ZJ, Zhou ZX. Survey on Web service composition testing. Ruan Jian Xue Bao/Journal of Software, 2018,29(2):299−319 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5383.htm [doi: 10.13328/j.cnki. jos.005383]

[31] Mishra DB, Panda N, Mishra R, Acharya AA. Total fault exposing potential based test case prioritization using genetic algorithm. Int'l Journal of Information Technology, 2018,29(2):299−319.

附中文参考文献:

[2] 吴川,巩敦卫,姚香娟.基于分支覆盖的回归测试路径选择.软件学报,2016,27(4):938−854. http://www.jos.org.cn/1000-9825/4975.htm [doi: 10.13328/j.cnki.jos.004975]

[3] 李园园,黄松,惠战伟,李留义.面向不同数据模式的测试用例检索方法.计算机科学,2017,44(11):221−225.

[4] 邱晓康,李宣东.一个面向路径的软件测试辅助工具.电子学报,2004,32(S1):235−237.

[7] 陈翔,陈继红,鞠小林,顾庆.回归测试中的测试用例优先排序技术述评.软件学报,2013,24(8):1695−1712. http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP. J.1001.2013.004420]

[8] 单锦辉,姜瑛,孙萍.软件测试研究进展.北京大学学报(自然科学版),2005,41(1):134−145.

[9] 张娟.软件测试中测试用例复用的研究[博士学位论文].上海:上海大学,2012.

[10] 陈浩,王广南.基于系统调用依赖图的程序相似性研究[硕士学位论文].长沙:湖南大学,2009.

[14] 王开云,孔思淇,付云生,潘泽友,马卫东,赵强.两种基于双向比较的最长公共子串算法.计算机研究与发展,2013,50(11):2444−2454.

[19] 巩敦卫,任丽娜.回归测试数据进化生成.计算机学报,2014,37(3):489−499.

[20] 姜淑娟,王令赛,薛猛,张艳梅,于巧,姚惠冉.基于模式组合的粒子群优化测试用例生成方法.软件学报,2016,27(4):785−801. http://www.jos.org.cn/1000-9825/4966.htm [doi: 10.13328/j.cnki.jos.004966]

[23] 刘军娜,邢琪,赵卫东.程序相似度检测算法.计算机与数字工程,2015,43(12):2145−2149.

[28] 丁蕊,董红斌,张岩,冯宪彬.基于关键点路径的快速测试用例自动生成方法.软件学报,2016,27(4):814−827. http://www.jos.org.cn/1000-9825/4971.htm [doi: 10.13328/j.cnki.jos.004971]

[30] 丁志军,周泽霞.Web 服务组合测试综述.软件学报,2018,29(2):299−319. http://www.jos.org.cn/1000-9825/5383.htm [doi: 10.13328/j.cnki.jos.005383]

钱忠胜(1977－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为智能化软件,软件自动化,智能推荐算法,人工智能,大数据与数据挖掘.

宋涛(1993－),男,硕士生,主要研究领域为软件自动化,软件测试与验证,程序分析与理解.