软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):839−854 [doi: 10.13328/j.cnki.jos.004975]

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

基于分支覆盖的回归测试路径选择*

吴 川[1], 巩敦卫[2], 姚香娟[3]

[1](中国矿业大学 计算机科学与技术学院,江苏 徐州 221116)

[2](中国矿业大学 信息与电气工程学院,江苏 徐州 221116)

[3](中国矿业大学 理学院,江苏 徐州 221116)

通讯作者: 巩敦卫, E-mail: dwgong@vip.163.com

摘 要: 回归测试是迭代式软件开发的重要环节,测试数据生成是回归测试的前提.传统的回归测试方法,从已有的测试数据中选择部分测试数据,并生成一些新的测试数据,以验证程序的正确性.但是,该方法容易生成冗余的测试数据,从而降低了回归测试的效率.研究了回归测试的分支覆盖问题,通过利用已有测试数据的路径覆盖信息,并选择一定个数的路径,以覆盖所有的目标分支.首先,以若干路径形成的集合作为决策变量,以路径最少、覆盖的分支最多以及包含的未覆盖路径最少为目标,建立路径选择问题的 3 目标优化模型;然后,采用遗传算法求解上述模型时,设计了基于目标重要性的个体评价策略;最后,基于已有的测试数据与选择的路径之间的覆盖关系,确定需要生成的测试数据.将所提方法应用于 6 个基准工业程序测试中,并与其他方法比较.实验结果表明,采用该方法选择的路径,能够覆盖更多的分支,需要生成的测试数据更少,回归测试消耗的时间更短.

关键词: 回归测试;分支覆盖;路径选择;多目标优化;遗传算法

中图法分类号: TP311

中文引用格式: 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):839−854. http://www.jos.org.cn/1000-9825/4975.htm

英文引用格式: 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). http://www.jos.org.cn/1000-9825/4975.htm

Selection of Paths for Regression Testing Based on Branch Coverage

WU Chuan[1], GONG Dun-Wei[2], YAO Xiang-Juan[3]

[1](School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

[2](School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, China)

[3](College of Science, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: Regression testing is an important part of the iterative software development, and test data generation is the premise of regression testing. Traditional regression testing methods select a part of test data from existing ones, and generate a number of new test data, so as to verify the correctness of a program. However, these methods are prone to generate redundant test data, therefore reducing the efficiency of regression testing. This paper researches the problem of branch coverage for regression testing, makes full use of information on the path coverage from existing test data, and selects a certain number of paths to cover all the target branches. First, taking a set consisting of several paths as the decision variable, the least number of paths, the largest number of covered branches, and the least number of uncovered paths as the objectives, a 3-objective optimization model for the problem of selecting paths is established. Then,

* 基金项目: 国家自然科学基金(61375067, 61203304, 61573362); 江苏省自然科学基金(BK2012566); 中央高校基本科研业务专项基金(2012QNA41)

Foundation item: National Natural Science Foundation of China (61375067, 61203304, 61573362); Natural Science Foundation of Jiangsu Province of China (BK2012566); Fundamental Research Funds for the Central Universities (2012QNA41)

收稿时间: 2015-09-15; 修改时间: 2015-10-15; 采用时间: 2015-11-20; jos 在线出版时间: 2016-01-13

CNKI 网络优先出版: 2016-01-14 13:15:59, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1315.004.html

a strategy of evaluating an individual based on the importance of an objective is designed in solving the above model using genetic algorithm. Finally, test data required to generate is determined according to the relation between existing test data and the selected paths. The proposed method is applied to test six benchmark and industrial programs, and compared with other regression testing methods. The experimental results show that paths selected by the proposed method can cover more branches, with fewer test data required to generate and less time consumption for regression testing.

Key words: regression testing; branch coverage; path selection; multi-objective optimization; genetic algorithm

回归测试,是指验证软件维护过程中是否引入新的缺陷以及软件维护是否正确而进行的测试[1].目前,软件通常采用迭代式开发,软件的升级换代成为惯例,且升级换代的速度不断加快,使得回归测试的成本不断提升.这说明,研究合适的方法,提高回归测试的效率,对降低软件维护的成本是很有必要的.

在回归测试中,测试的目标主要是程序中被修改及其影响的元素.例如,程序中的被修改的语句、被修改语句所在的分支,以及被修改语句所在的路径等,均可视为测试的目标,相应的测试分别称为语句覆盖测试、分支覆盖测试以及路径覆盖测试.为了进行回归测试,需要从已有的测试数据集中选择部分测试数据,或者生成一些新的测试数据,以满足新增的测试需求.目前,研究者已经提出了多种测试数据集优化技术,包括:测试数据选择、约简、优先以及扩增等[2],以减少回归测试的成本.

选择性回归测试在软件维护范围较小或缺陷尚未全部修复时执行.与全面回归测试相比,选择性回归测试的执行频度更高,更需要控制测试的成本[3].对于选择性回归测试,最常用的方法为测试数据选择.该方法是在已有的测试数据集中,按照某种准则,选择部分测试数据,以减少测试的代价.选择性回归测试的过程如图 1 所示,当已选择的测试数据不能满足新增的测试需求时,还需要生成一些新的测试数据.此时,现有的基于测试数据选择的回归测试技术存在如下 3 个问题.

(1) 已有的测试数据选择研究工作,虽然有些方法利用了已有测试数据集,减少了测试数据集扩增的代价,但却没有充分利用已有测试数据的覆盖信息,使得需要生成的新的测试数据增多.

(2) 一个测试数据往往能够覆盖多个测试目标,但是,现有的测试数据进化生成方法产生了很多冗余的测试数据,从而增加了测试成本.

(3) 程序的变动使得选择的测试数据难以覆盖测试目标,降低了在后续测试中的利用率,从而需要生成更多的、新的测试数据.

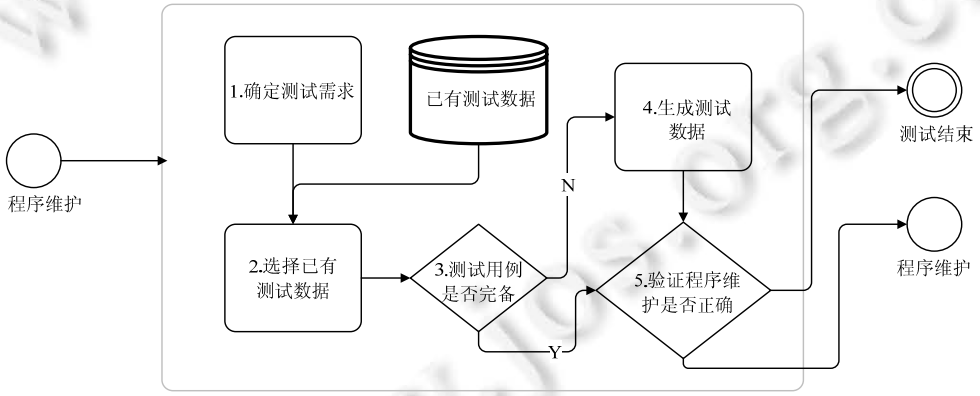


Fig.1 Procedure of traditional selective-form regression testing

图 1 传统选择性回归测试过程

鉴于此,本文研究面向分支覆盖的测试数据选择与生成问题,将其转化为部分路径覆盖问题,确切地说,转化为路径选择问题,通过建立路径选择问题的多目标优化模型并采用遗传算法求解,期望在选择路径的同时,减少测试数据选择以及扩增的代价,以降低测试成本.为此,以若干路径形成的集合作为决策变量,以路径最少、覆盖的分支最多以及包含的未覆盖路径最少为目标,建立路径选择问题的 3 目标优化模型;当采用遗传算法求解

上述模型时,设计了基于目标重要性的个体评价策略.此外,基于已有的测试数据与选择的路径之间的覆盖关系,确定需要生成的测试数据.与测试数据集扩增方法相比,本文将测试数据选择以及测试数据的再生成这两阶段的问题,转换为一个路径选择的问题,并提出了针对性的求解方法,不仅能够利用已有测试数据集,而且能够充分利用已有测试数据的覆盖信息,可以高效地减少需生成的测试数据数量,从而在满足测试需求的前提下,提高回归测试效率.

本文的贡献主要体现在如下 3 个方面:

(1) 建立了路径选择问题的多目标优化模型;

(2) 设计了遗传算法求解上述模型的个体评价策略;

(3) 通过对比实验,验证了所提方法能够提高回归测试效率.

1 相关工作

在软件开发过程中,为了保证程序的修改正确且不引入新的缺陷,采用合适的方法,选择或生成测试数据,以覆盖程序被修改及其影响的元素,是十分必要的.

根据不同的目标,已有测试数据集的优化工作主要分为 4 类:测试数据选择、测试数据集约简、测试数据优先,以及测试数据集扩增[2].在选择已有测试数据方面,测试数据集约简,是指使用尽可能少的测试数据,充分满足给定的测试目标,从而获得更高的回归测试效率[4].测试数据优先,是将不同的测试数据按照其重要程度排序后使用,从而降低测试成本[5].与上述两种技术相比,测试数据选择是选择性回归测试中较为重要的一部分[6].

测试数据选择,是指在程序修改后,从已有的测试数据集中,选择部分测试数据,以验证程序的修改是否正确.在理想情况下,希望选择的测试数据能够覆盖所有的新增测试需求,从而降低回归测试的成本.20 世纪 70 年代以来,测试数据选择问题得到了国内外学者的高度关注,并提出了多种有效的解决方法.Rothermel 和 Harrolad 首次对选择性回归测试问题进行了总结,并提出一种评估不同测试数据选择方法的统一框架[1,7],基于此,Rothermel 等人进一步提出了基于最大语句覆盖率的测试数据选择方法[5].Smith 等人提出了基于测试代价的测试数据选择方法[8].在 El-Hamid 等人提出的方法中,通过语句相关及功能作用图,获取发生变化的句法和语义,用于选择与变化部分相关的测试数据[9].为了选择测试数据,Yoo 和 Harman 以错误检测率、执行时间以及代码覆盖率作为目标函数,建立了该问题的多目标优化模型,并采用遗传算法求解[10].Sampath 等人针对包括选择、约简以及优先等在内的测试数据集选择问题,给出了统一的求解框架和基于多准则评价的测试数据选择方法[11].Panichella 等人研究测试数据选择问题的优化求解方法,通过增加经过正交设计的个体,改进用于求解优化问题的种群多样性,提升进化种群的性能,从而提升测试数据选择效率[12].Nardo 等人针对工业软件,利用真实的回归缺陷,评价多种已有测试数据集的优化方法以及在不同覆盖准则下缺陷检测的性能[13].Suri 等人采用群体优化算法选择测试数据,使得回归测试的执行时间在设定的范围内,以此减少回归测试的成本[14].Gligoric 等人根据程序代码文件之间的依赖关系,选择文件依赖关系变化的被测试的类或函数,减少需要执行的测试数据集,并减少了收集测试信息的代价[15].

以上测试数据选择方法提高了已有测试数据的利用率,降低了回归测试成本.但是,这些方法均假设已有的测试数据集是完备的,即程序修改之后,已有的测数据集仍具有测试充分性,而没有考虑已有测试数据集不能覆盖被修改及其影响元素的情况,并基于此生成新的测试数据,因而难以验证被修改及其影响元素的正确性.

为了解决上述问题,研究者们提出了测试数据集扩增技术,促使新生成的测试数据能够覆盖新版本程序的修改部分和新增部分.Ye 等人通过对比程序修改前后的活动图,寻找影响、没有影响以及删除的路径,并基于与上述路径的关系,将已有的测试数据分为失效和没有失效两类,通过对失效测试数据的修改,生成新的测试数据[16].Kumar 等人选择程序修改之后新增和发生改变的路径作为覆盖目标,并生成覆盖这些目标的测试数据[17].Santelices 等人基于数据与控制依赖关系,选择与修改点相关的路径作为覆盖目标,使得基于该目标覆盖生成的测试数据能够覆盖修改点影响的语句[18].

以上测试数据集扩增方法仅针对受修改影响的元素生成测试数据,减少了测试目标.但是,这些方法均没有

充分利用已有测试数据.当上述方法执行的频度增高时,测试数据生成的成本也会增大.

Xu 等人认为,已有测试数据集包含了大量的有用信息,研究通过使用已有测试数据集,提高测试数据集扩增的效率[19−22].首先,他们基于已有测试数据对维护后程序语句的覆盖变化,寻找发生改变的语句,采用深度优先策略,将这些变化的语句排序后,利用已有测试数据,生成新的测试数据[19].但是,该方法有可能生成冗余的测试数据,也在一定程度上增加了回归测试的成本.然后,Xu 等人在采用遗传算法生成覆盖分支的测试数据时发现,不同算法、不同的测试目标执行顺序以及是否利用新生成的测试数据,对测试数据生成的效率有很大影响[20],然而,他们没有利用影响测试数据生成的因素,提高回归测试数据生成效率.接着,Kim 和 Xu 等人在文献[19,20]研究的基础上,进一步提出利用已有测试数据提高测试数据再生成的效率[21,22].通过合理的设定测试目标顺序,并且给出已有测试数据的复用准则,在遗传算法或具体符号执行方法生成测试数据过程中,使用已生成测试数据,减少了测试数据生成的代价.实验结果也充分表明了他们提出的方法的有效性.

不言而喻,当已有的测试数据不能覆盖所有的测试需求时,除了需要选择部分测试数据以外,还需要针对未能覆盖的测试需求,生成一些新的测试数据.Xu 等人的工作通过减少测试数据集扩增的成本,使得回归测试的时间耗费降低.但是,针对测试数据集扩增问题的这些研究,没有提出选择并生成新的测试数据的有效策略,也没有考虑在测试数据选择的同时降低需要生成的测试数据数.虽然利用了已有测试数据集,但是没有充分利用已有测试数据的结构覆盖信息,容易产生冗余的测试数据,这将使得回归测试的效率降低,测试成本增加.

如果能在利用已有测试数据集的基础上进一步利用已有测试数据的路径覆盖信息,从待覆盖的路径中选择已有测试数据覆盖的路径以及覆盖这些路径的测试数据,并生成一些新的测试数据,以覆盖剩余的路径,那么,将能够提高已有测试数据覆盖信息的利用率,并减少需要生成的测试数据,从而降低回归测试的成本.这对执行次数较高的选择性回归测试来说,是非常有意义的.

本文建立的路径选择问题的多目标优化模型,是将传统的基于分支覆盖的回归测试数据选择和测试数据的再生成,转化为覆盖部分路径问题,着重解决最优路径子集的选择问题.与测试数据选择以及测试数据扩增方法相比,由于求解问题的不同,本文的决策向量、搜索空间都有所区别,因此,本文给出了具有针对性的进化求解方法,设计了基于目标重要性的个体评价策略.虽然本文方法和测试数据扩增方法都生成了一些新的测试数据,但与测试数据扩增方法不同,扩增方法在获得新的测试数据时,注重利用已有测试数据提高测试数据生成的效率,而本文致力于通过路径选择,在满足测试需求的前提下,减少需要生成的测试数据数目,避免生成的测试数据冗余,从而使得回归测试开销降低,使用的测试数据生成方法仍然是传统的方法.进一步地,本文使用的已有测试信息更充分,除了已有测试数据集以外,还利用了已有测试数据的覆盖信息,并且基于路径选择,解决测试数据集优化问题,不但可以利用已有的测试数据集优化方法和路径分析技术[14,16,17,23],还可以与进化测试方法[24−26]相结合,根据路径的变化,自动生成测试数据.

2 路径选择问题的多目标优化模型

本节首先简要描述需要完成的测试任务;然后,将回归测试的分支覆盖问题,转化为覆盖路径问题;最后,建立路径选择问题的多目标优化模型.

2.1 测试任务描述

记被测程序为 G,G 的输入向量为 X,回归测试之前,已经存在的测试数据为 $X_1,X_2,...,X_m$,这些测试数据覆盖的路径集合为 P_E.此外,记 P_E 包含的路径条数为 $|P_E|$.由于不同测试数据有可能覆盖相同的路径,因此,$|P_E|\leqslant m$.

程序 G 修改之后,该程序的部分(或所有)分支将受到影响,记受到影响的分支构成的集合为 $B=\{b_1,b_2,\ldots,b_\beta\}$,为了验证修改之后的程序 G 是否正确,需要对 B 中的每一分支进行测试,因此,B 的所有分支即为目标分支.鉴于此,本文的测试任务是,从已有的测试数据中选择测试数据,或者生成新的测试数据,使得这些数据能够覆盖目标分支.

2.2 分支覆盖转化

由于回归测试的频度高,测试数据的生成和执行均耗费大量的时间.因此,在回归测试中,应该充分利用已有的测试数据.

第 2.1 节的测试任务包含两个子任务:(1) 从已有的测试数据集中,选择部分测试数据;(2) 如果选择的测试数据不能覆盖全部目标分支,还需要生成一些新的测试数据.这是一个典型的两阶段决策问题.由于第 1 阶段选择的测试数据影响了第 2 阶段测试数据的生成,因此,每一子问题的最优解不一定构成整个问题的最优解.

考虑到已有多种路径覆盖测试数据生成方法,因此,如果将本文研究的分支覆盖问题,转化为覆盖部分路径问题,并采用已有的方法生成覆盖路径的测试数据,那么,将能够有效地生成分支覆盖测试数据.为此,需要首先将分支覆盖问题转化为覆盖路径问题.

对于子任务 1,如果在已有的测试数据集中,存在某一测试数据 X_i,该数据覆盖的分支为 $b(X_i)\subseteq B$, 路径为 $p(X_i)\in P_E$, 那么,路径 $p(X_i)$将覆盖 $b(X_i)$.因此,已有测试数据的选择问题可以转化为:从已覆盖的路径集中,选择覆盖目标分支的路径,并从已有测试数据集中,选择覆盖这些路径的测试数据.

对于子任务 2,由于覆盖某一目标分支的测试数据,也可能覆盖其他目标分支,因此,如果针对每一剩余的目标分支生成测试数据,那么,将生成冗余的测试数据.如果寻找一个包含较少路径的集合,该集合包含的路径覆盖所有剩余的目标分支,那么,基于该集合生成测试数据,将能够避免测试数据的冗余.

可以看出,子任务 1 和子任务 2 均将分支覆盖问题转化为覆盖部分路径问题,确切地说,即转化为路径选择问题.对于子任务 1,是从已覆盖的路径中选择部分路径;对于子任务 2,是从所有的可执行路径中选择若干路径.

2.3 路径选择问题的多目标优化模型

所谓路径选择是指,从被测程序中选择一定数量的路径作为目标路径,使得基于该路径的测试能够满足既定的测试充分性准则[27].

本文从被测程序所有可执行路径形成的路径集合中,选择若干路径,作为目标路径,但是,与传统的路径选择不同,本文在选择路径时,将第 2.2 节中子任务 1 和子任务 2 一起解决,充分利用已覆盖路径的信息,使得选择的路径尽可能少、覆盖的分支尽可能多,以及包含的未覆盖路径尽可能少,以减少需要生成的测试数据,提高回归测试的效率.

可以看出,路径选择问题,能够建模为一个多目标优化问题.下面,给出建立路径选择问题的多目标优化模型.

记程序 G 的所有可执行路径形成的集合为 P,P 的幂集为 2^P.由于本文需要寻找的是一个路径子集,记为 P_T,因此,$P_T\in 2^P$,2^P 即为优化问题的搜索空间.

首先,我们希望选择的路径最少.这是因为,路径的条数决定了测试数据的个数,路径越少,覆盖该路径需要的测试数据也越少,从而减少了执行被测程序的次数,提高了回归测试的效率.反映该目标的目标函数可以表示为

$$f_1(P_T)=|P_T| \tag{1}$$

然后,我们希望选择的路径覆盖的目标分支最多.假设 P_T 包含 n 条路径 $p_1,p_2,...,p_n$, 那么,P_T 可以表示为 $P_T=\{p_1,p_2,...,p_n\}$. 考虑路径 p_i,该路径包含的目标分支构成的集合为 B_{p_i}, 那么,P_T 包含的目标分支可以表示为 $\bigcup_{p_i\in P_T} B_{p_i}$. 反映该目标的目标函数可以表示为

$$f_2(P_T)=\left|\bigcup_{p_i\in P_T} B_{p_i}\right| \tag{2}$$

最后,我们希望选择的路径包含的未覆盖路径最少.这是因为,如果选择的部分(或全部)路径已经被覆盖,那么,需要覆盖的路径将减少,从而需要生成的测试数据也减少.这将提高回归测试的效率.由于已覆盖的路径集为 P_E,因此,P_T 包含的未覆盖路径集可以表示为 $P_T\cap(P-P_E)$. 反映该目标的目标函数为

$$f_3(P_T)=|P_T\cap(P-P_E)| \tag{3}$$

基于以上分析,本文建立的路径选择问题的多目标优化模型如下:

$$\begin{cases}\max \quad f(P_T)=(-f_1(P_T),f_2(P_T),-f_3(P_T))\\ \qquad\quad =\left(-|P_T|,\left|\bigcup_{p_i\in P_T}B_{p_i}\right|,-|P_T\cap(P-P_E)|\right)\\ \text{s. t.}\quad P_T\in 2^P\end{cases} \tag{4}$$

3 基于遗传算法的路径选择问题求解

本节主要给出采用遗传算法,求解第 2.3 节所建立的多目标优化模型的方法.

3.1 路径选择过程

对第 2.3 节中的式(4),采用遗传算法求解的伪代码如算法 1 所示,其中,遗传算法终止的条件为相邻若干代种群平均适应值的变化小于某一设定的阈值,记为θ,或达到设定的最大进化代数.

记采用上述方法得到的最优路径子集为 P_T^*. 第 3.2 节~第 3.5 节给出了算法 1 的具体实现细节.

算法 1. 基于遗传算法的路径选择.

```
Input: 路径选择参数;
Output: 路径选择子集.
BEGIN
    SetParameters();   //参数设置
    Initialize();   //初始化种群
    generation→0;   //进化代数
    DO WHILE (generation ≤ max_allow_evolve)
                //如果大于最大迭代次数,终止
        generation ← generation + 1;
        Crossover operation to Pop(generation);
          //个体之间的交叉
        Mutation operation to Pop(generation);
          //个体的变异
        Repair operation to Pop(generation);
         //个体的修补
        FOR j=1 to popSize DO
            Evaluate fitness of P_T^j;
        END FOR;
            //计算个体的适应值
        Select operation to Pop(generation);
           //选择操作
        Elitism operation to Pop(generation);
            //最优个体保留
        IF (ΔavgFitness < θ) THEN
            BREAK;
            //如果种群相邻若干代平均适应值变化小于阈值,终止
        END IF
    END WHILE
END
```

3.2 个体编码

与一般的优化问题不同,本文优化问题的决策变量是若干路径形成的集合.因此,在采用遗传算法求解该问题时,进化个体应为若干路径形成的集合.记 l 为编码长度,表示个体包含路径的最大数,$l\leqslant|P|$,那么,基于整数编

码方式,第 j 个个体的编码可以表示为

$$P_T^j = p_1^j, p_2^j, ..., p_{n_j}^j, \#, ..., \# \tag{5}$$

式中,$n_j \leqslant l$,为个体包含的路径条数; $p_i^j \in \{1,...,|P|\}$, 对应于程序的控制流程图中,依某种方式遍历路径集合 P 之后,得到的第 p_i^j 条路径.采用路径的序号表示路径,有利于降低个体编码的复杂度.

当 $n_j<l$ 时,个体由 n_j 条路径和 $l–n_j$ 个占位符“#”构成,“#”表示该部分编码不表示任何路径.

使用“#”作为占位符的原因是:首先,本文寻找的最优路径子集包含的路径条数事先并不知道;其次,种群进化过程中,实施交叉和变异操作之后,生成的个体可能包含重复的路径,那么,此时需要修复个体,修复个体后,重复的路径采用“#”表示.

3.3 种群初始化

种群初始化方法如算法 2 所示,该方法每次迭代生成 *popSize* 个个体.为了生成个体 P_T^j, 首先,构造候选集 $\Upsilon = \{1,...,|P|\}$; 然后,采用满足均匀分布的随机函数 uniformRand(Υ),生成 Υ 内的某一路径,并将该路径从 Υ 中删除.重复上述过程,直到生成 l 条路径;最后,将这些路径依生成的顺序排列,形成个体 P_T^j. 这样做的好处是, P_T^j 包含的路径不但不重复,而且在路径集合 P 中的分布也是均匀的,从而 P_T^j 中不同路径覆盖的相同分支减少,提高了分支覆盖率.

算法 2. 种群初始化.

Input: l,$|P|$;

Output: 初始化种群.

```
BEGIN
i=0,j=0, Υ = {1,...,|P|};
DO WHILE (j<popsize)
        FOR i<l DO
            P_T[j][i] = uniformRand (Υ);
            Υ ← Υ – P_T[j][i];
            //从Υ中删除已生成的路径
            i←i+1;
        END FOR
        j←j+1;
END WHILE
END
```

3.4 适应度函数

为了评价进化个体的性能,通过加权,将式(4)的 3 个目标转化为 1 个目标.为此,需要确定不同目标的权重.本文的目的是,选择若干合适的路径,使得这些路径尽可能地覆盖所有的目标分支.因此,在这 3 个目标中,目标分支的覆盖是最重要的.为了利用尽可能多的已覆盖路径的信息,我们希望选择的路径包含尽可能少的未覆盖路径.因此,包含的未覆盖路径条数的重要性其次.鉴于此,评价个体性能的适应度函数可以表示为

$$F(P_T) = -w_1 f_1(P_T) + w_2 f_2(P_T) - w_3 f_3(P_T) \tag{6}$$

式中, $w_1, w_2, w_3 \in [0,1], w_2 >> w_3 >> w_1$.

根据式(6),进化种群的个体适应值有以下特性:

(1) 某个体覆盖的目标分支越多,那么,该个体的适应值就越大;

(2) 当多个个体覆盖的目标分支个数相同时,某个体包含未有测试数据覆盖的路径越少,那么,该个体的适应值就越大;

(3) 若多个个体覆盖的目标分支个数相同,且包含未有测试数据覆盖的路径数也相等,那么,某个体包含的可执行路径数越少,该个体的适应值就越大.

鉴于此,本文种群进化的方向是,优先寻找覆盖最多目标分支的路径子集;如果上述子集不止一个,那么,接着寻找包含未有测试数据覆盖路径数最少的路径子集;如果后者还不止一个,那么,再寻找包含最少可执行路径的路径子集.

3.5 进化算子

本小节将设计进化算子,包括:交叉、变异、修补以及选择算子等.

1) 交叉算子

在路径子集优化的过程中,通过交叉算子,能够组合不同的进化个体.对于个体 $P_T^1=p_1^1,p_2^1,...,p_{n_1}^1,\#,...,\#$ 和 $P_T^2=p_1^2,p_2^2,...,p_{n_2}^2,\#,...,\#$, 选择 p_k^1 为交叉点, $k<n_1$ 且 $k<n_2$, 记交叉之后产生的新个体分别为 $P_T^{1'}$ 和 $P_T^{2'}$, 那么, P_T^1 和 P_T^2 的交叉过程如图 2 所示.

$$P_T^1=p_1^1,p_2^1,...,p_k^1,...,p_{n_1}^1,\#,...,\#$$
$$P_T^2=p_1^2,p_2^2,...,p_k^2,...,p_{n_2}^2,\#,...,\#$$
$$\Downarrow$$
$$P_T^{1'}=p_1^1,p_2^1,...,p_k^2,...,p_{n_2}^2,\#,...,\#$$
$$P_T^{2'}=p_1^2,p_2^2,...,p_k^1,...,p_{n_1}^1,\#,...,\#$$

Fig.2 Crossover process between P_T^1 and P_T^2

图 2 P_T^1 和 P_T^2 的交叉过程

需要说明的是:(1) 参与交叉的两个个体可能包含相同的路径,使得交叉操作之后得到的新个体具有重复的路径,导致这些个体成为非法个体,这需要应用后面提出的修补算子进一步加以修复;(2) 参与交叉的两个个体,包含不同个数的路径,使得这两个个体包含“#”的个数也不相等,那么,交叉之后产生的新个体也与其父代个体具有不同个数的“#”.

2) 变异算子

本文方法中,变异是产生新个体的辅助手段.对于个体 P_T^1, 首先,选择路径 p_i^1 实施变异操作,生成的新路径为 $p_i^{1'}\in\{1,...,|P|\}$, 且 $p_i^{1'}\neq p_i^1$; 然后,用 $p_i^{1'}$ 代替 P_T^1 中的 p_i^1, 从而得到一个新个体 $P_T^{1'}$.

与交叉算子类似,某个体变异操作之后得到的新个体也可能出现重复的路径,从而也需要采用修补算子进一步修复.但是,变异算子不会导致“#”的变化.

3) 修补算子

现在,对由于交叉和变异操作导致的非法个体进行修复,使之成为合法的个体.方法如下:

首先,对于非法个体 $P_T^{j'}$, 从 n_j+1 位置开始,如果“#”的位置变成了某一路径,那么,将该路径移动到 n_j+1 位置,并令 n_j=n_j+1.重复上述过程,直到所有的“#”均检查完毕;然后,从第 1 个位置开始,到 n_j 位置结束,如果出现了重复的路径,那么,针对每一重复的路径,采用冒泡的方式,将其移动到 n_j 位置,用“#”替代之后,令 n_j=n_j−1,重复上述过程,直到所有重复的路径均处理完毕;最后,针对 n_j+1 位置之前存在的“#”进行处理,如果当前分支覆盖率达到 100%,那么,采用和重复路径一样的处理方法,依次移动“#”到 n_j 位置,并令 n_j=n_j−1,否则,生成新路径替换当前“#”.

4) 选择算子

在种群进化过程中,通过选择算子,确定进入下一代的个体.方法是:首先,按照式(6)计算个体的适应值;然后,将个体按照适应值大小排序;最后,选择适应值最大的若干个体,进入下一代.

除了上述遗传算子之外,本文还采用最优保留机制.如果当前代种群最优个体的适应值低于上一代,那么,将上一代的最优个体加入到当前代群体中,并将当前代群体中适应值最小的个体删除.

4 测试数据选择与生成

按照第 3.1 节中所提方法,得到最优的路径子集 P_T^* 后,还需要利用 P_T^* 包含的路径选择部分已有的测试数据,并根据需要生成一些新的测试数据.过程如下:

首先,在已有的测试数据集 T 中,寻找穿越 $P_T^* \cap P_E$ 中路径的测试数据.如果有多个测试数据覆盖同一条路径,那么,仅保留一个测试数据.记这些测试数据形成的集合为 T_E.

然后,判断 $P_T^* \cap (P - P_E)$ 是否为空集.如果不为空集,则采用基于路径的测试数据生成方法,生成一些穿越 $P_T^* \cap (P - P_E)$ 中路径的测试数据.记这些测试数据形成的集合为 T_U.如果 $P_T^* \cap (P - P_E)$ 为空集,那么,T_U=∅.

最后,求 $T_E \cup T_U$,并用于回归测试,以验证程序的更新是否正确.

本文的重点在于,基于选择的路径,选择或生成用于回归测试的测试数据.基于路径的测试数据生成,目前已有很多研究成果,这里不再赘述.

5 实 验

为了评价本文提出的路径选择方法的性能,首先,选择 6 个不同规模的程序作为被测程序;然后,基于选择的路径生成测试数据;最后,与其他方法进行比较.实验的硬件环境为 Intel Due-Core 2.0 GHz CPU 和 2G RAM 内存.所有方法均采用 VC++6.0 实现.

实验验证以下 3 个问题:

问题 1:与其他方法相比,本文方法是否能够达到很高的分支覆盖率?

问题 2:与其他方法相比,本文方法在达到很高分支覆盖率的同时,需要生成的测试数据是否很少?

问题 3:与其他方法相比,本文方法达到很高分支覆盖率的同时,是否具有较少的时间消耗?

5.1 被测程序基本信息

实验采用的被测程序分别为 triangle,gsl-find,gsl-minmax,tcas,print_tokens 和 replace,其中,triangle 为三角形分类程序;gsl-sort 和 gsl-minmax 来源于开源编译工具 GCC 的科学计算函数库 gsl;tcas,print_tokens 和 replace 来源于 SIR 测试平台的西门子测试程序集[28].采用不同的方法更新上述程序,更新之后的程序信息见表 1.为了识别修改影响的分支和确认路径与节点的关系,首先,对程序控制流图的所有分支节点广度优先遍历,记录这些分支节点的序号,从修改节点遍历可达的分支即为目标分支;然后,对程序插桩,并记录某路径在不同分支节点处穿越真假分支的情况,穿越真分支记为 1;否则,记为–1,如果没有穿越该分支结点,记为 0;最后,得到路径与分支的关系矩阵.基于此,每个程序采用 20 个不同已有的测试数据集,以评价本文方法的通用性.在这 20 个测试数据集中,分支覆盖率最小为 50%,最大为 100%.

Table 1 Information of tested programs

表 1 被测对象的基本信息

被测程序编号	程序名称	函数个数	代码行数	已有测试数据集大小		目标分支数	维护类型
				Min	Max		
PG1	triangle	1	32	4	6	12	增加
PG2	gsl-find	1	85	18	20	14	删除
PG3	gsl-minmax	3	88	38	45	14	增加
PG4	tcas	8	138	5	12	12	删除
PG5	printtokens	21	563	19	28	74	修改
PG6	replace	21	516	25	32	103	修改

5.2 实验方法和评价指标

为了评价本文方法(记为 GAP)的性能,与使用基于遗传算法选择测试数据的回归测试方法[11]、使用基于贪心算法[3]选择测试数据的回归测试方法以及使用基于贪心算法[3]选择路径的回归测试方法进行比较,这些方法分别记为 GAT、GT 和 GP,参考相关文献,用 VC++6.0 实现用于选择测试数据的方法.

GAT 和 GT 的流程如图 1 所示.与 GP 相同的是,本文方法也是基于选择的目标路径生成测试数据;与 GAT 选择测试数据相同的是,本文方法选择路径,也基于遗传算法;GP 和 GT 也有共同点,即 GP 选择路径和 GT 选择测试数据,均采用贪心算法.此外,上述方法均采用基于 VC++6.0 实现的简单遗传算法,生成测试数据[26],其中,GAT 和 GT 根据测试数据选择结果,以未覆盖分支为目标,依次生成测试数据;在生成覆盖某一分支的测试数据过程中,也检查该测试数据是否覆盖其他分支.如果该测试数据覆盖某一分支,那么,将该分支从未覆盖的目标集合中删除.对于本文方法与 GP,根据选择的路径,以没有覆盖的路径作为目标,生成测试数据.

本文求解路径选择方法的遗传操作如第 3 节所述,w_1=0.01,w_2=1,w_3=0.1;对比方法的遗传操作分别为轮盘赌选择、单点交叉和单点变异.无论是本文方法还是对比方法,交叉和变异概率均分别为 0.8 和 0.15.此外,种群规模均为 50,用于路径或测试数据选择的最大进化代数为 3 000,用于生成覆盖每个目标路径测试数据的最大进化代数为 5 000.

为了比较不同方法的性能,采用如下指标:

目标分支覆盖率(r_{con}),为测试数据覆盖的目标分支数与所有目标分支数的比值.

生成的测试数据数($|T'|$),即生成的测试数据集中包含的测试数据个数.

已有测试数据比率,是指用于回归测试的测试数据集中包含 T 中测试数据的个数,与用于回归测试的数据集中测试数据的个数比值.

测试数据生成时间($t_{generate}$),是指生成测试数据消耗的时间.

执行时间(t_{total}),是测试数据或路径选择与测试数据生成所消耗的时间总和.

等待时间,是指测试数据或路径选择后,生成测试数据所需的时间与执行已有测试数据所需时间的差.

为了减少遗传算法的随机性对不同方法性能的影响,每种方法和每一测试数据集均独立运行 10 次,针对每一指标,求取某一方法 200 次实验的平均值.

5.3 实验结果

表 2 和表 3 列出了相应的实验结果,表中,加粗显示的数据为最优结果.由表 2 可以看出,路径或测试数据选择的时间均较短,而用于测试数据生成的时间则相对较长.

Table 2 Target branches coverage and the number of generated test data of different methods in each tested program

表 2 不同方法对 PG1~PG6 测试的目标分支覆盖率以及生成测试数据数

被测程序	GT		GAT		GP		GAP	
	r_{con} (%)	$\|T'\|$	r_{con} (%)	$\|T'\|$	r_{con} (%)	$\|T'\|$	r_{con} (%)	$\|T'\|$
PG1	96.25	1.05	**96.67**	0.92	92.5	1.15	94.59	**0.65**
PG2	100	0.15	100	0.15	100	0.25	**100**	**0.15**
PG3	94.29	2.15	95.36	1.57	85.71	1.95	**97.5**	**1.43**
PG4	100	1.1	100	1.28	99.16	1.35	**100**	**0.84**
PG5	97.90	2.45	97.64	2.6	93.71	3.05	**98.58**	**2.1**
PG6	95.48	3.9	95	4.26	95.44	4.45	**96.9**	**3.18**

Table 3 Perform time and test data generation time of different methods in each tested program(s)

表 3 不同方法对 PG1~PG6 测试的方法执行时间以及测试数据生成时间(s)

被测程序	GT		GAT		GP		GAP	
	t_{total}	$t_{generate}$	t_{total}	$t_{generate}$	t_{total}	$t_{generate}$	t_{total}	$t_{generate}$
PG1	0.63	0.63	0.72	0.55	0.72	0.72	**0.51**	**0.29**
PG2	**0.37**	0.37	0.67	0.38	0.53	0.53	0.98	**0.42**
PG3	2.24	2.24	**1.75**	1.42	2.13	2.13	2	**1.32**
PG4	1.22	1.22	1.57	1.33	1.78	1.78	**1.16**	**0.85**
PG5	39.84.	39.84.	42.27	41.82	56.65	56.65	**23.99**	**22.61**
PG6	61.67	61.67	67.41	66.92	73.61	73.61	**38.18**	**35.83**

(1) 对问题 1 的回答

由表 2 可以看出,相比其他方法,本文方法用于回归测试的测试数据集达到了很高的目标分支覆盖率.

① 由路径选择的结果我们发现,对于每一程序,采用本文方法选择的路径子集,均能够覆盖所有的目标分支.这说明,本文提出的路径选择模型和求解方法是可行的.

② 与其他方法相比,本文将分支覆盖问题转化为路径覆盖问题,使得需要生成的测试数据数减少.因此,在测试数据生成方面,本文方法具有显著的优势.此外,本文方法形成的回归测试数据集,分支覆盖率优于其他 3 种方法,在程序 PG2~PG6 中达到了最大的目标分支覆盖率.这意味着,本文建立的路径选择问题的模型适合回归测试.但是,对于程序 PG1,本文方法选择的未有测试数据覆盖的部分路径难以覆盖,且没有优化测试数据的生成过程,因此,目标分支覆盖率比 GAT 略低.这表明,对于某些程序的回归测试,如果优先选择容易覆盖的路径,那么,能够进一步提升所提方法的性能.

(2) 对问题 2 的回答

由表 2 和图 3 可以看出,相比其他方法,本文方法在达到很高分支覆盖率的同时,需要生成的测试数据最少.

① 对规模较小的程序 PG1~PG3,需要生成的测试数据最多仅为 1.43 个;对规模较大的程序 PG4~PG6,最多仅需生成 3.18 个测试数据.这充分说明,本文设定的路径选择问题的目标函数是合适的.

② 由图 3 可以看出,与其他方法相比,对于每一程序,本文方法的已有测试数据比率是最高的.这说明,本文方法对已有测试数据集的利用充分,有助于减少需要生成的测试数据.

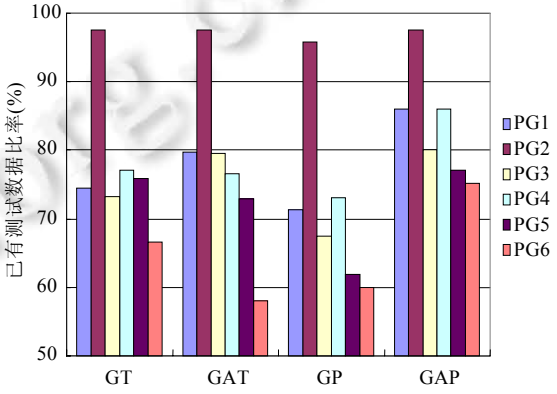


Fig.3 Comparison of existing test data ratio among different methods

图 3 不同方法已有测试数据比率的比较

(3) 对问题 3 的回答

结合表 2 和表 3 可以看出,相比其他方法,本文方法在达到很高分支覆盖率的同时,具有较少的时间消耗.

① 对于规模较大的程序 PG4~PG6,本文方法的执行时间最短;对于规模较小的程序 PG1,虽然本文方法的分支覆盖率为 94.58%,略低于 GAT,但是,本文方法的执行时间少于 GAT;对于程序 PG2 和 PG3,本文方法的执行时间略高于最优方法,但覆盖率是最高的.

② 对于每一程序,不同方法选择路径或测试数据的时间都很短,使得执行时间的大小主要取决于测试数据的生成.在这些方法中,本文方法的测试数据生成时间最短,随着程序规模的扩大,本文方法的优势更加明显.这说明,本文方法的实用性更强.

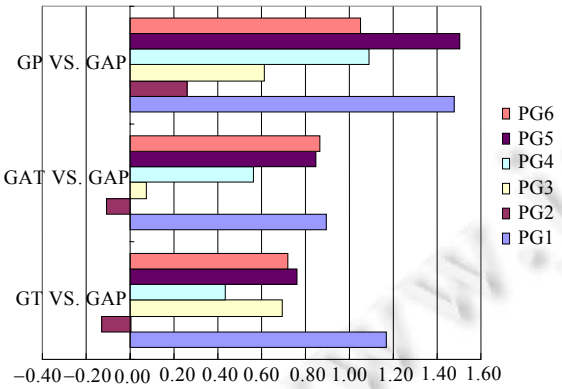


Fig.4 Extra times of waiting time of other methods than our method

图 4 相对本文方法,其他方法的额外等待时间倍数

③ 鉴于测试数据的生成过程可以和已有测试数据的验证过程并行执行,而前者较为耗时,因此,验证已有测试数据之后,需要等待新的测试数据生成.通过实验我们发现,与其他方法相比,对于大部分被测程序,本文方法由于应用较多已有的测试数据验证被测程序,使得等待生成测试数据的时间较短.

将每一对比方法的等待时间减去本文方法的等待时间,再除以本文方法的等待时间,能够得到该方法相对本文方法的额外等待时间倍数,结果如图 4 所示.由图 4 可以看出,对于程序 PG2,虽然本文方法的测试数据生成等待时间多于 GT 和

GAT,但是,等待时间相差并不大.对于其他程序,每一对比方法的等待时间明显多于本文方法,额外等待时间倍数最高为 1.51.

上述实验结果和问题的回答体现了本文方法的优势,但在衡量本文方法与其他方法在回归测试效率方面是否有显著性差异时,还需要通过多次实验采集到的样本之间的差异来推断不同方法总体之间的区别.本文采用 Z 检验的方法,通过 200 次实验结果,对不同方法的分支覆盖率(r_{con})和回归测试执行时间(t_{total})进行预测.不同程序、不同方法运行得到的分支覆盖率和回归测试执行时间是随机变量,这是因为,它们的值都受许多随机因素的影响,在大量实验中,它们近似服从正态分布.比较不同方法随机变量的平均值,如果分支覆盖率越高,回归测试所用的时间越短,那么,该方法回归测试效率越高.

以 PG5 为例,令本文方法和 GAT 的 t_{total} 随机变量的均值为 μ_1 和 μ_2,给出利用 Z 检验方法进行比较 μ_1,μ_2 的具体过程.鉴于样本方差是总体方差的无偏估计,采用样本方差的值当作对总体方差的估计值,也就是用样本标准差的值作为对总体标准差的估计值,得到 σ_1=10.25 和 σ_2=17.39.样本容量 n_1=n_2=200, $\overline{t_{total\,1}}$=23.99, $\overline{t_{total\,2}}$=42.27,显著性水平 α 的值为 0.01.

第 1 步.建立原假设 $H_0:\mu_1 \geqslant \mu_2$ 以及相反假设 $H_1:\mu_1 < \mu_2$.

第 2 步.设置统计量 $Z_1=\dfrac{\overline{t_{total\,1}}-\overline{t_{total\,2}}}{\sqrt{\dfrac{\sigma_1^2}{n_1}+\dfrac{\sigma_2^2}{n_2}}}$.

第 3 步.给定拒绝域 $Z_1=\dfrac{\overline{t_{total\,1}}-\overline{t_{total\,2}}}{\sqrt{\dfrac{\sigma_1^2}{n_1}+\dfrac{\sigma_2^2}{n_2}}} \leqslant -Z_\alpha$;

第 4 步.计算统计量的值 $Z_1=\dfrac{\overline{t_{total\,1}}-\overline{t_{total\,2}}}{\sqrt{\dfrac{\sigma_1^2}{n_1}+\dfrac{\sigma_2^2}{n_2}}}=\dfrac{23.99-42.27}{\sqrt{\dfrac{10.25^2}{200}+\dfrac{17.39^2}{200}}} \approx -12.8$;

第 5 步.得出结论,因为 $Z_1 \leqslant -Z_\alpha$, 落在拒绝域内,所以,拒绝 H_0,接受 H_1.即认为本文方法执行时间的期望值显著性小于 GAT.这说明,与 GAT 相比,本文方法回归测试的总时间明显小于 GAT,回归测试成本消耗优于 GAT.鉴于此,在表 4 中,GAP 与 GAT 在 PG5 关于 t_{total} 的显著性分析结果为 Y.

对于 6 个程序的显著性分析结果的比较情况见表 4.其中,本文方法显著性优于其他方法标记为 Y,否则,标记为 N,由表 4 可以看出,在简单程序中,由于各种方法分支覆盖率都较高,本文方法未体现出明显优势,但是在较大规模的 PG4~PG5 中,本文方法分支覆盖率显著高于其他方法.而在分支覆盖率接近时,在绝大多数情况下,本文方法需要的执行时间明显少于其他方法.

Table 4 Result of significance analysis

表 4 显著性分析结果

被测程序	GAP vs. GT		GAP vs. GAT		GAP vs. GP	
	r_{con} (%)	t_{total}	r_{con} (%)	t_{total}	r_{con} (%)	t_{total}
PG1	N	Y	N	Y	Y	Y
PG2	N	N	N	N	N	N
PG3	Y	Y	Y	N	Y	Y
PG4	N	Y	N	Y	N	Y
PG5	Y	Y	Y	Y	Y	Y
PG6	Y	Y	Y	Y	Y	Y

综合以上实验结果与分析,能够得出如下结论:本文方法能够达到很高的目标分支覆盖率,回归测试需要的耗时短,需要生成的测试数据少,且更适合于较大规模程序的测试.

5.4 多次回归测试中的可重用性分析

根据帕累托准则,软件缺陷具有类聚性,那么,因缺陷做出的软件维护,使得程序控制结构的变化也具有重复性.鉴于此,对某个程序维护版本的选择性回归测试,其中的路径选择或者测试数据选择结果,可以重复利用

到该程序其他版本的回归测试中.显然,这对执行频度较高的回归测试是较有价值的,这将进一步降低回归测试的成本.

为了验证本文方法在多次回归测试中的可重用性,并与GAT方法比较,首先,采用变异测试方法模拟维护行为,使用5个变异算子(ABS,AOR,LCR,ROR和UOI)对表1程序的不同位置实施变异,每个变异体相当于程序的一个维护版本;然后,每个被测程序随机挑选出100个变异体,仍然使用第5.1节中的20个不同的已有测试数据集,分别采用本文方法以及GAT,进行20次连续选择性回归测试,每次的测试需求为:覆盖变异影响到的分支;最后,对每个程序的若干需生成测试数据的变异体,分别统计本文方法以及GAT使用20个不同已有数据集的重用率,记为 *Rate*,*Rate* 可以表示为

$$Rate = \frac{ReuseCount}{AffectCount} \tag{7}$$

具体计算方法如下:

首先,对每一程序的100个变异体,统计目标分支未被已有数据全部覆盖的变异体数,记为 *AffectCount*,这些变异体需要生成一些新的测试数据.

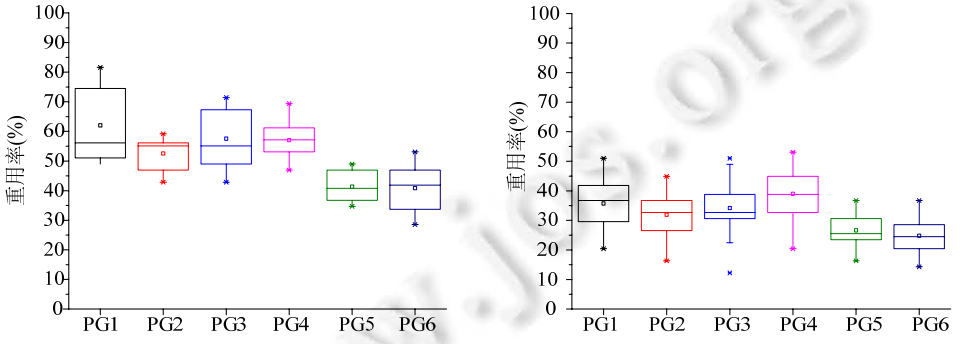
接着,对需生成测试数据的变异体依次生成测试数据,如果某个变异体的已有数据穿越的路径覆盖的分支与其他先生成测试数据的变异体相同,那么,在使用本文方法进行回归测试时,利用相应已测试变异体的路径选择结果,如果已有测试数据覆盖的分支与其他已测试变异体相同,则在使用GAT进行回归测试时,利用对应的测试数据选择结果.

然后,统计可以利用先测试变异体对象选择结果的个数,记为 *ReuseCount*.

最后,按照式(7),计算本文方法以及GAT的重用率,结果如图5所示.

由图5可以看出,通过本文方法选择的路径,在不同程序的多次回归性测试中,其重用率显著高于GAT方法选择的测试数据.在PG1程序中,重用率最高达到81.6%,在PG6程序中,重用率最低也达到28.5%.平均而言,在6个程序中,能够达到 51.8%,这表明,采用本文方法可以减少在多次回归测试中的路径选择的时间,方法的可重用性高.

对于本文方法,分析在不同程序中的重用率我们发现,对于规模较小的程序,如图5所示,本文方法的重用率较大,对于规模较大的程序,本文方法的重用率较低.这进一步说明,当软件缺陷分布相对集中,或软件维护内容在较小的范围内时,采用本文提出的路径选择方法所选择的路径可重复利用的价值高,这在执行频度较高的回归测试中,具有很大的应用价值.



(a) 本文方法在6个程序中20次实验的重用率分布 (b) GAT方法在6个程序中20次实验的重用率分布

Fig.5 Distribution of reuse rate of our method and GAT in each tested program under 20 times continuous regression tests

图5 在6个程序中,本文方法与GAT方法,20次连续回归测试的重用率分布

5.5 有效性分析

本文主要针对分支覆盖问题,研究在取得较高分支覆盖率的同时,回归测试选择测试数据以及生成测试的时间较少,对于其他结构元素覆盖问题,还需要重新设定目标函数并给出相应的求解问题方法.

在外部有效性方面,对于分支覆盖问题,影响本文方法适用性的一个主要问题是:本文方法选择的被测对象以及相应的已有测试数据集的代表性.虽然本文选择的被测对象包含规模大小不同的 6 个 C 语言程序,但是,与工业程序相比,代码行数以及函数个数仍有差距.因此,选择维护后的工业程序作为测试对象可能会影响本文方法的性能.此外,本文不同被测程序的 20 个已有测试数据集,是从已满足程序覆盖准则测试数据集中随机选择出来的,其他测试数据集的产生方法,也有可能带来不同的实验结果.在后续的研究中,我们将继续选择较大规模的工业程序进行回归测试,并根据实际的维护情况,构造已有测试数据集,以进一步验证本文方法的有效性.

在内部有效性方面,本文用于求解路径选择问题的遗传算法参数设置并没有达到最优.针对路径选择问题,本文用遗传算法求解,并设计了个体评价函数、交叉、变异以及个体修复算子.在对不同的程序进行测试时,采用了相同的遗传参数,但针对具体程序,这些遗传参数有可能不是最优的,这将影响本文方法的性能.此外,本文在通过多目标优化选择需要覆盖的路径时,没有考虑这些路径的分布及其覆盖难度,有可能使得覆盖选择路径所需生成的测试数据难度增大,导致回归测试成本增加,选择一些较易生成测试数据的路径,将继续提高本文方法的性能.在实验分析方面,本文主要考察了分支覆盖率以及回归测试的执行时间,当需要生成新的数据时,选用不同的测试数据进化生成方法对本文的实验结果有较大影响,当测试数据生成的效率能够进一步提升时,这将降低本文方法的优势.

对本文所提方法的安全性进行分析,虽然本文方法以较低的测试开销达到了较高的分支覆盖率,期望可以检测出修改后程序内的缺陷,但是,本文方法仍然属于面向结构覆盖准则的测试数据选择以及生成方法,使用本文方法选择以及生成的测试数据对修改程序测试后,有可能修改程序仍然存在隐藏的缺陷,这就需要使用强度更高的结构覆盖准则或者使用面向缺陷检测的测试数据生成方法,在将来的研究中,我们将考虑把缺陷检测能力作为路径选择问题优化模型的目标之一,以增强本文方法的缺陷检测能力.

6 总结及下一步工作

本文将回归测试中的分支覆盖问题转化为路径覆盖问题,并基于选择的路径,生成用于回归测试的数据集,以覆盖程序维护影响的分支.为了选择需要覆盖的路径,将路径选择问题建模含有 3 个目标的优化问题;采用遗传算法生成路径时,设计具有针对性的个体编码方法、遗传算子以及修补算子.

将所提方法应用于 6 个不同规模的被测程序中,并与已有方法比较.实验结果表明,本文方法能够达到很高的目标分支覆盖率,回归测试需要的耗时短,需要生成的测试数据少,且更适合于较大规模程序的测试.

需要说明的是,本文仅通过 6 个基准程序验证所提方法的有效性.这些基准程序的规模还比较小,难以充分说明本文方法的可扩展性.因此,在今后的研究中,需要采用规模更大的实际工业程序来测试所提方法的性能.此外,本文在通过多目标优化选择需要覆盖的路径时,没有考虑这些路径的分布及其覆盖难度,使得覆盖选择路径所需生成的测试数据难度增大,从而限制了测试数据生成的效率.鉴于此,基于选择路径的分布和覆盖难度,建立路径选择问题新的(约束)多目标优化模型,并寻求有针对性的求解方法,也是需要进一步研究的问题.

致谢 感谢各位审稿专家以及本刊编辑的辛勤工作.

References:

[1] Rothermel G, Harrold MJ. Analyzing regression test selection techniques. IEEE Trans. on Software Engineering, 1996,22(8): 529−551. [doi: 10.1109/32.536955]

[2] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Journal of Software Testing, Verification and Reliability, 2012,22(2):67−120. [doi: 10.1002/stv.430]

[3] Gu Q, Tang B, Chen DX. A test suite reduction technique for partial coverage of test requirements. Chinese Journal of Computers, 2011,34(5):879−888 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.00879]

[4] Shi A, Gyori A, Gligoric M, Zaytsev A, Marinov D. Balancing trade-offs in test-suite reduction. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (SIGSOFT FSE 2014). New York: ACM Press, 2014. 246−256. [doi: 10.1145/2635868.2635921]

[5] Rothermel G, Untch RJ, Chu CY, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. on Software Engineering, 2001,10(27):929−948. [doi: 10.1109/32.962562]

[6] Zhang ZY, Chen ZY, Xu BW, Yang R. Research progress on test case evolution. Ruan Jian Xue Bao/Journal of Software, 2013,24(4):663−674 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4379.htm [doi: 10.3724/SP.J.1001.2013.04379]

[7] Rothermel G, Harrold MJ. A framework for evaluating regression test selection techniques. In: Proc. of the 16th Int'l Conf. on Software Engineering (ICSE 1994). Sorrento: IEEE Computer Society Press, 1994. 201−210. [doi: 10.1109/ICSE.1994.296779]

[8] Smith AM, Kapfhammer GM. An empirical study of incorporating cost into test suite reduction and prioritization. In: Proc. of the 24th ACM Symp. on Applied Computing (SAC 2009). New York: ACM Press, 2009. 461−467. [doi: 10.1145/1529282.1529382]

[9] El-Hamid WSA, El-Etriby SS, Hadhoud MM. A general regression test selection technique. World Academy of Science, Engineering and Technology, 2010,4(2):893−897.

[10] Yoo S, Harman M. Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation. Journal of Systems and Software, 2010,83(4):689−701. [doi: 10.1016/j.jss.2009.11.706]

[11] Sampath S, Bryce R, Memon A. A uniform representation of hybrid criteria for regression testing. IEEE Trans. on Software Engineering, 2013,39(10):1326−1344. [doi: 10.1109/TSE.2013.16]

[12] Panichella A, Oliveto R, Penta MD, Lucia AD. Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Trans. on Software Engineering, 2015,41(4):358−383. [doi: 10.1109/TSE.2014.2364175]

[13] Nardo DD, Alshahwan N, Briand L, Labiche Y. Coverage-Based regression test case selection, minimization and prioritization: A case study on an industrial system. Software Testing, Verification and Reliability, 2015,25(4):371−396. [doi: 10.1002/stvr.1572]

[14] Suri B, Singhal S. Understanding the effect of time-constraint bounded novel technique for regression test selection and prioritization. Int'l Journal of System Assurance Engineering and Management, 2015,6(1):71−77. [doi: 10.1007/s13198-014-0244-3]

[15] Gligoric M, Eloussi L, Marinov D. Practical regression test selection with dynamic file dependencies. In: Proc. of the Int'l Symp. on Software Testing and Analysis (ISSTA 2015). New York: ACM Press, 2015. 211−222. [doi: 10.1145/2771783.2771784]

[16] Ye N, Chen X, Peng J. Automatic regression test selection based on activity diagrams. In: Proc. of the 5th Int'l Conf. on Secure Software Integration & Reliability Improvement Companion (SSIRI-C 2011). Jeju Island: IEEE Computer Society Press, 2011. 166−171. [doi: 10.1109/SSIRI-C.2011.31]

[17] Kumar A, Tiwari S, Mishra KK. Generation of efficient test data using path selection strategy with elitist GA in regression testing. In: Proc. of the 3rd IEEE Int'l Conf. on Computer Science and Information Technology (ICCSIT 2010). Chengdu: IEEE Computer Society Press, 2010. 389−393. [doi: 10.1109/ICCSIT.2010.5564915]

[18] Santelices RA, Chittimalli PK, Apiwattanapong T, Orso A, Harrold MJ. Test-Suite augmentation for evolving software. In: Proc. of the 23th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2008). L'Aquila: IEEE Computer Society Press, 2008. 218−227. [doi: 10.1109/ASE.2008.32]

[19] Xu ZH, Kim Y, Kim M, Rothermel G, Cohen MB. Directed test suite augmentation: Techniques and tradeoffs. In: Proc. of the ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (SIGSOFT FSE 2010). Santa Fe: ACM Press, 2010. 257−266. [doi: 10.1145/1882291.1882330]

[20] Xu ZH, Cohen MB, Rothermel G. Factors affecting the use of genetic algorithms in test suite augmentation. In: Proc. of the 12th Annual Conf. on Genetic and Evolutionary Computation (GECCO 2010). Portland: ACM Press, 2010. 1365−1372. [doi: 10.1145/1830483.1830734]

[21] Kim Y, Xu ZH, Kim M, Cohen MB, Rothermel G. Hybrid directed test suite augmentation: An interleaving framework. In: Proc. of the 7th Int'l Conf. on Software Testing, Verification and Validation (ICST 2014). Cleveland, OH: IEEE Computer Society Press, 2014. 263−272. [doi: 10.1109/ICST.2014.39]

[22] Xu ZH, Kim Y, Kim M, Cohen MB, Rothermel G. Directed test suite augmentation: An empirical investigation. Software Testing, Verification and Reliability, 2015,25(2):77−114. [doi: 10.1002/stvr.1562]

[23] Benedusi P, Cmitile A, De Carlini U. Post-Maintenance testing based on path change analysis. In: Proc. of the Int'l Conf. on Software Maintenance (ICSM'88). Santa Fe: IEEE Computer Society Press, 1988. 352−361. [doi: 10.1109/ICSM.1988.10187]

[24] Gong DW, Zhang WQ, Yao XJ. Evolutionary generation of test data for many paths coverag based on grouping. Journal of Systems and Software, 2011,84(12):2222−2233. [doi: 10.1016/j.jss.2011.06.028]

[25] Xie XY, Xu BW, Shi L, Nie CH. Genetic test case generation for path-oriented testing. Ruan Jian Xue Bao/Journal of Software, 2009,20(12):3117−3136 (in English with Chinese abstract). http://www.jos.org.cn/1000-9825/580.htm [doi: 10.3724/SP.J.1001.2009.00580]

[26] Gong DW, Yao XJ. Testability transformation based on equivalence of target statements. Neural Computing & Applications, 2012,21(8):1871−1882. [doi: 10.1007/s00521-011-0568-8]

[27] Mao CY, Lu YS. A simplified method for generating test path cases in branch testing. Journal of Computer Research and Development, 2006,43(2):321−328 (in Chinese with English abstract). [doi: 10.1360/crad20060220]

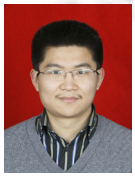
[28] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering, 2005,10(4):405−435. [doi: 10.1007/s10664-005-3861-2]

附中文参考文献:

[3] 顾庆,唐宝,陈道蓄.一种面向测试需求部分覆盖的测试数据集约简技术.计算机学报,2011,34(5):879−888. [doi: 10.3724/SP.J.1016.2011.00879]

[6] 张智轶,陈振宇,徐宝文,杨瑞.测试用例演化研究进展.软件学报,2013,24(4):663−674. http://www.jos.org.cn/1000-9825/4379.htm [doi: 10.3724/SP.J.1001.2013.04379]

[27] 毛澄映,卢炎生.分支测试中测试路径用例的简化生成方法.计算机研究与发展,2006,43(2):321−328. [doi: 10.1360/crad20060220]



吴川(1980－),男,江苏盐城人,博士生,讲师,CCF 会员,主要研究领域为软件测试.

姚香娟(1975－),女,博士,教授,CCF 会员,主要研究领域为基于搜索的软件工程.

巩敦卫(1970－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为智能优化与控制,基于搜索的软件工程.