软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(7):2103–2117 [doi: 10.13328/j.cnki.jos.006264]

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

基于偶然正确性概率的回归测试选择方法*

周小莉, 赵建华

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 赵建华, E-mail: zhaojh@nju.edu.cn

摘 要: 数据驱动的智能系统的核心是处理数据的算法,对算法正确性的要求高,导致其测试开销大,需要有效地缩减测试的规模,其中回归测试选择是控制测试规模的有效手段.数据驱动的智能系统由于其动态信息流强度弱的原因,发生偶然正确性现象的概率较高,并且该现象会导致常用的回归测试选择技术所选择出的测试集包含大量检测不到故障的测试用例.因此,从偶然正确性现象的角度出发,提出一种基于偶然正确性概率的回归测试选择技术,进一步排除可能发生偶然正确性现象的用例.该方法能够兼顾代码覆盖,同时从偶然正确性的角度保证缩减后的测试用例集合对被修改的代码的测试是充分的.根据在用例缩减和故障检测能力之间侧重的不同,提出了基于最小化和安全性技术的两种选择策略,并给出 3 种具体的选择算法.在实验中将所提方法与一种安全的测试选择技术进行比较,结果表明,这3种选择算法都很好地缩减了测试集合的规模,提高了测试选择的精度,并提高了安全性和精度的综合指标.

关键词: 偶然正确性现象;回归测试;回归测试选择

中图法分类号: TP311

中文引用格式: 周小莉,赵建华.基于偶然正确性概率的回归测试选择方法.软件学报,2021,32(7):2103–2117. http://www.jos.org.cn/1000-9825/6264.htm

英文引用格式: Zhou XL, Zhao JH. Regression test selection method based on coincidental correctness probability. Ruan Jian Xue Bao/Journal of Software, 2021,32(7):2103–2117 (in Chinese). http://www.jos.org.cn/1000-9825/6264.htm

Regression Test Selection Method Based on Coincidental Correctness Probability

ZHOU Xiao-Li, ZHAO Jian-Hua

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: For data-driven intelligent systems, the data processing algorithms are very important and need to be tested adequately. Because of the high safety requirement, the cost of testing becomes very high and reducing such cost is needed. Regression test selection is an effective mean to control the scale of testing. For data-driven intelligent systems, the coincidental correctness happens frequently because of the weak dynamic information flows, and leads that the regression test sets contain a lot of redundant tests. Therefore, a regression test selection technique is proposed based on the coincidental correctness probability. This method considers the probability of coincidental correctness in addition to the code coverage. The selected tests not only cover the modified code, but have a higher probability to transfer the intermediate results produced by the modified code to the program output. Such selection can reduce the impact of coincidental correctness. The empirical results show that the proposed selection method can improve the precision of selection and reduce the size of the regression tests.

Key words: coincidental correctness; regression testing; regression test selection

1 引 言

数据驱动的智能化软件系统是一类非确定性软件系统,且日益应用于一些安全相关的领域,对其软件质量

*本文由“面向非确定性的软件质量保障方法与技术”专题特约编辑陈俊洁副教授、汤恩义副教授、何啸副教授以及马晓星教授推荐.

收稿时间: 2020-09-14; 修改时间: 2020-10-26; 采用时间: 2020-12-14; jos 在线出版时间: 2021-01-22

的要求也日益提高.这类软件使用算法来解析数据,从数据中学习到规律,并掌握这种规律,然后对真实世界中的事件做出决策或预测.与传统软件不同,数据驱动的智能化软件通常使用大量的数据来训练,通过各种算法从数据中学习如何完成任务.数据驱动的智能化软件的关键是良好的数据,其核心是算法,如果算法或其实现存在任何问题,那么很可能造成严重的后果,因此对其算法正确性的要求是非常高的.如何对这类系统进行充分的测试引起了越来越多的研究关注.由于对软件质量的更高的要求,其测试所消耗的开销也更大,需要更有效的手段对测试用例的规模进行控制.软件测试的预算大部分都花费在回归测试上,因此,如何对回归测试用例集合进行有效的缩减进而降低这一类非确定性软件整体的测试开销是一个值得研究的主题.

在程序执行过程中,如果在某一程序点观测到的变量 y 的取值可以降低更早之前某一点上变量 x 取值的不确定性,则称从 x 到 y 产生了一个信息流,而信息流强度表示 x 取值的不确定性降低的程度.信息流及其强度可在一定程度上反映程序执行时变量之间的依赖程度.Masri 和 Podgurski 等人[1]的工作表明,实际场景中弱信息流是非常普遍的,甚至其中大多数信息流的强度为 0.数据驱动的智能系统包含大量数据处理过程,并产生许多中间值,而这些中间值之间的关联性很弱,因此在数据驱动的智能系统中信息流弱的现象更加严重.弱信息流现象意味着如果程序执行过程中产生了一个错误的中间值,这个错误的状态很可能无法传递到程序输出或检查点上,从而无法被测试人员发现,即发生偶然正确性现象(coincidental correctness,简称 CC).偶然正确性现象是指程序中包含错误的语句被执行但仍通过了测试的现象,其原因是错误语句被执行时没有产生错误的中间结果或状态,或者这个错误的中间结果或状态没有被传递到程序的输出或检查点中.当程序执行时的信息流强度较弱时,程序输出或检查点上的值与各个中间结果之间的关联性较弱,很可能会出现偶然正确性现象.

许多研究工作已经证明偶然正确性现象在实际场景中是非常普遍的[2,3],并且影响了许多测试技术的有效性[4].其中,回归测试技术也受到了偶然正确性现象的影响.常用的回归测试用例选择技术是基于代码修改分析选择出与代码修改相关的用例.然而,这种方法选择出的回归测试集合包含了大量发生偶然正确性现象的用例,这些测试用例虽然覆盖了被修改代码但仍无法检测其中包含的错误.对于数据驱动的智能系统等偶然正确性现象的发生概率相对较高的系统,回归测试用例选择的结果更易受到偶然正确性现象的不良影响.因此,我们从偶然正确性现象的角度出发,提出一种有效缩减回归测试规模的测试用例选择方法,从而降低测试开销.

在以往工作中[5,6],我们提出了一种估算偶然正确性现象发生概率的方法,该方法通过估算错误在执行过程中的传播概率来估计偶然正确性现象的发生概率.基于这项工作,我们提出一种基于偶然正确性概率的回归选择技术.基于代码修改的回归测试选择技术所选出的测试集合包含大量无法检测出故障的用例(即发生偶然正确现象的用例),本文根据偶然正确性概率对覆盖修改的用例作进一步选择,以提高测试选择的准确性.

本文第 2 节介绍偶然正确性现象和回归测试选择技术,并给出估算偶然正确性现象的发生概率的方法.第 3 节给出根据偶然正确性概率进行回归测试选择的方法.第 4 节通过实验对基于偶然正确性概率的回归测试选择方法进行评估.最后第 5 节对本文工作进行总结.

2 背景知识

2.1 偶然正确性现象

偶然正确性现象的概念最初是由 Budd 和 Angluin[7]提出的,是指程序中包含错误的语句被执行但仍通过了测试的现象[8].Masri 根据 PIE(propagation-infection-execution,传播-感染-执行)模型对偶然正确现象给出了更明确的定义.PIE 模型是由 Voas[9]提出来的,强调了程序缺陷的执行并不是程序失效的充分条件,还需要满足将错误的中间状态传播到程序的输出中.类似地,在 Ammman 和 Offutt 于 2008 年提出的 RIP(reachiability-infection-propagation,到达-感染-传播)模型等其他研究工作[10,11]中也对该问题进行了讨论.Voas 指出,当且仅当满足下面 3 个条件时错误才会被观测到.

(1) 执行(execution),错误代码需要被执行到.

(2) 感染(infection),执行错误代码时必须触发一个错误的中间状态.

(3) 传播(propagation),这个错误的中间状态必须能够传播到最后并输出,使得我们能够观测到它与预期的

输出不一致,即失效.

测试不一定执行错误代码,错误代码被执行后也不一定触发错误的中间状态.即使程序执行时产生了错误的中间状态,这个错误的中间状态也不一定能传播到输出而被测试人员观测到.只有“执行”“感染”和“传播”这 3 个条件同时被满足时才能观测到错误.

基于 PIE 模型,Masri 等人提出了强偶然正确性现象和弱偶然正确现象的概念.强偶然正确性现象是指满足“执行”和“感染”条件,不满足“传播”条件的情况.弱偶然正确性现象是指满足“执行”同时不满足“传播”的情况,且不考虑“感染”条件是否被满足.弱偶然正确性现象是广义上的偶然正确性现象,而强偶然正确性现象是更狭义的概念.

许多工作都证明了偶然正确性现象在实际场景中是非常普遍的.本文在西门子程序集进行了初步实验,其结果表明,强、弱偶然正确性现象出现的平均概率分别为 17%和 57%.这个结果验证了偶然正确性现象的普遍性,并且弱偶然正确性现象的发生频率显著高于强偶然正确性现象.本文的回归测试选择方法是针对弱偶然正确性现象而提出来的.

偶然正确性现象使得即使错误语句被多次覆盖,仍可能无法被发现.偶然正确性现象影响了许多测试技术的有效性,尤其是对基于代码覆盖的测试技术产生了负面影响[4].基于代码覆盖的测试技术通常包含一个隐含的假设:错误代码没有被执行一定不会产生错误的结果,而错误代码被执行则很可能产生错误的程序输出.然而,本文在实验中发现,弱偶然正确性现象出现的平均频率为 57%,这表明了错误代码被执行后产生错误程序输出的概率只有 43%.已有的研究工作也证明了代码覆盖和故障的发生之间没有必然的联系.偶然正确性现象的频繁出现,使得错误代码的执行很可能无法产生错误的程序输出,从而打破基于代码覆盖的测试技术的潜在假设,使其有效性受到了影响.Denmat 等人[12]对经典的基于代码覆盖的错误定位技术 Tarantula 的局限性进行了研究,并表明要使其有效,必须满足错误语句的执行在大多数情况下会导致程序出错,即大多数程序的执行中不会发生偶然正确性现象.

由于判断程序执行过程中是否发生偶然正确性现象是非常困难的,消除偶然正确性现象影响的研究工作还不是很多.已有的相关工作都集中在错误定位领域[3,13−15],而在回归测试选择等其他测试领域目前还没有偶然正确性现象的相关工作.

2.2 回归测试用例选择

随着软件开发过程的迭代和演化,因移除缺陷、完善功能等原因,代码不断被修改,需要通过回归测试来保证代码修改的正确性,以避免代码的修改给其他模块引入新的缺陷.此外,还需要针对新增功能等被修改代码设计新的测试用例.因此,在软件持续演化过程中,测试用例集合的规模快速扩大.回归测试的开销占整个软件测试预算的 80%以上,并占软件整体维护预算的 50%以上[16].

回归测试的核心问题是对测试用例的高效维护,重新执行原有的测试集合是最简单的维护策略,但是会导致测试的开销过大、用例失效等一系列问题.针对测试用例的维护,研究人员提出了一系列的技术,包括识别失效用例、测试选择技术、测试优先技术、测试集合缩减技术和测试用例扩充技术等.其中,测试选择技术是缩小测试用例规模的一种有效手段,其目标是在降低测试用例集合大小的同时,将原有测试集中可能检测到故障的用例尽可能多地选择出来,为进一步的错误定位提供足够的测试信息,提高错误定位的有效性.常用的回归测试选择方法分为下面几类.

(1) 全部重新测试.

除了过时的测试用例,将原有的测试集合中的所有用例都在新版本程序上重新测试.

(2) 随机选择策略.

从原始的测试集合中随机地或者根据测试人员的经验选取测试用例.例如,基于风险用例的回归测试,为测试用例设定一个风险级别,首先运行最关键和可疑的测试用例,忽略那些非关键的、优先级别低或者稳定性高的测试用例,因为即使这些测试用例可以发现缺陷,所发现的缺陷的严重性也不高.

(3) 安全的选择策略.

安全的选择策略将所有覆盖程序中受影响部分的测试用例作为回归测试用例,例如基于代码修改的回归测试用例选择策略.

(4) 最小化策略.

在原始程序的测试集合中选择可以保证程序中被修改代码至少被覆盖 1 次的最小子集作为回归测试集合.

全部重新测试用例所消耗的执行成本大,而随机选择策略依赖于测试人员的经验,稳定度低,精确度不高.最小化策略所选择出的测试用例集合最小,执行成本低,但是错误检测能力较差.安全的测试选择策略所选出的测试集检测错误的能力较强,用例数目相对不是很多,因此通常使用安全的选择策略进行回归测试的选择.但随着程序规模的扩大,安全选择策略中进行程序分析的成本也随之增加.

随着软件开发规模的扩大,研究人员对回归测试的关注越来越多,提出了很多有效的回归测试选择策略,其中包括整数规划、数据流分析、符号执行、动态切片、graph-walking、源代码文本差异、SDG 系统依赖图切片、路径分析、变更检测、防火墙、控制流图聚类识别、基于设计的测试等方法.很多研究工作对回归测试用例技术进行了总结和系统性综述[16−20].Rothermel 等人[6]提出一种基于程序依赖图的安全的回归测试选择技术,这是一种非常经典的回归测试选择技术.这种技术通过构造原始程序 P 和修改后的程序 P'的依赖图,深度优先遍历两个依赖图并识别出依赖图中被修改的节点,然后挑选出所有覆盖修改节点的测试用例.对于被修改的程序,一个能够检测出故障的测试用例必然是覆盖修改节点的测试用例,因此,Rothermel 等人提出的回归测试选择技术是安全的,其回归测试集合包含了所有能够检测故障的测试用例.然而,这种测试用例选择技术选择出的测试用例集合包含了许多无法发现故障的测试用例.这种方法虽然能够保证安全性,但是并不能有效地降低测试成本,因此很多测试用例选择方法在测试集合大小与错误发现能力之间进行了权衡,在保障一定安全性的前提下减小了测试集合大小.Graves 等人[21]指出,回归测试的核心问题是选择和执行测试用例所需要的时间和测试集合的故障检测能力之间的权衡问题.

常用的回归测试选择策略基于代码修改分析等方法找出代码修改相关的测试用例,并将其作为回归测试选择用例.这里将覆盖了被修改代码的测试用例称为“覆盖修改的用例”,将可检测出被修改程序中的错误的测试用例称为“故障检测用例”.覆盖修改用例不一定是故障检测用例,覆盖修改的用例集合还包含了虽然覆盖修改但发生了偶然正确性现象的用例,图 1 所示为这 3 类测试用例之间的关系.偶然正确性现象在实际场景中非常频繁地发生,这导致覆盖修改用例中包含大量的偶然正确性用例,因此,目前的回归测试选择策略受到了偶然正确性现象的不良影响.

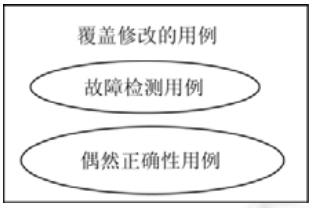

Fig.1 The set of modification-traversing test

图 1 覆盖修改测试用例集合

2.3 偶然正确性概率的估算

为了消除偶然正确性现象对回归测试选择的影响,首先需要判断测试用例执行时是否发生偶然正确性现象.在以往的工作中[2,5],我们提出了估计程序执行中发生偶然正确性现象概率的方法,该方法从代码执行过程中内存空间中值变化的角度出发,通过动态的数据流和控制流两个方面对发生偶然正确性的概率进行估算.根据 PIE 模型,错误被发现需要满足执行、感染和传播 3 个条件,而这 3 个事件的发生有一定的包含关系.假设程序中语句 s 包含错误,在执行测试 t 时该错误被发现的概率 $T(s,t)$可用条件概率进行计算:

$$T(s,t)=PR(E)\times PR(I|E)\times PR(P|E\cap I) \tag{1}$$

其中,$PR(E)$表示 s 被执行的概率,这个概率可以通过错误语句 s 在此次执行中是否被覆盖直接得到.如果错误语

句被执行到,则 $PR(E)=1$,否则,$PR(E)=0$.$PR(I|E)$表示在“执行”被满足的情况下“感染”发生的概率,也就是错误语句被执行时产生错误的中间状态的概率.这个概率可以通过统计实验或者根据经验而设定.$PR(P|E\cap I)$表示在满足“执行”和“感染”两个条件的情况下满足“传播”条件的概率,即当错误语句被执行并产生了错误的中间结果时,这个错误的状态被传播到输出的概率.通过对实际场景的观测,我们发现错误的中间结果通过下面两种途径将这种错误的中间状态传递到其他语句的执行实例,并最终影响程序输出值的正确性.

(1) 参与运算

当错误的中间结果被计算性使用时,例如被用于赋值语句表达式、输出语句、当作函数调用的参数或被用于索引表达式中,此时可能会导致运算结果出错.对于包含运算的语句,其结果的正确性和操作数的正确性有很大关系.在操作数都正确的情况下,运算结果也一定是正确的;如果至少一个操作数是错误的,运算结果也很可能是错误的.对于不同的运算,操作数对运算结果正确性的影响程度也不同.对不同类型的运算,通过为每个操作数都设定不同的影响因子来量化该操作数对运算结果正确性的影响程度,从而可以根据操作数的取值及其正确性概率来估算运算结果的正确性概率.

(2) 影响控制流

当错误的中间结果被判定性使用时,即被用于分支语句的条件表达式时,会导致程序的执行路径出错,被实际执行的路径上和应该被执行的路径上的语句所改变的变量值的正确性都受到了影响.对于一条语句实例 s,为了估算控制流对其结果的正确性概率的影响,我们首先找到影响其执行的所有控制表达式的执行实例,然后根据这些表达式的正确性概率修正 s 的结果的正确性概率.

通过对 $PR(E)$、$PR(I|E)$和 $PR(P|E\cap I)$的估计,可以估算出语句 s 中所包含的错误被发现的概率,同样可以估算偶然正确性现象发生的概率.假设 s 包含错误,而该错误在执行 t 时因偶然正确性现象被掩盖的概率为

$$CCP(s,t)=PR(E)\times(1-PR(I|E)\times PR(P|E\cap I)) \tag{2}$$

这个概率是满足“执行”条件,却没有满足“传播”条件的概率.发生偶然正确性现象,首先需要满足“执行”条件.在执行条件满足的情况下,如果同时满足“感染”和“传播”,那么错误会被发现,否则,错误会被隐藏.此时,偶然正确性概率的估算是针对一个测试用例和程序中一条语句的,可以比较精细地分析出偶然正确性现象的发生情况.

3 基于偶然正确性概率的回归测试选择技术

以往回归测试选择相关工作的重点集中在如何根据代码修改找出与代码修改相关的用例,并将这些用例作为回归测试用例.然而,不是每个覆盖了被修改代码的用例都能够检测出故障.即使覆盖了被修改代码,也可能因偶然正确性现象而无法发现故障.

即使测试用例集合对被修改的代码覆盖过很多次,如果这些用例发生偶然正确性现象的概率高,则被修改代码中的错误仍很可能会被掩盖.因此,需要对覆盖修改的用例作进一步的分析,以提高测试选择的准确性.本文提出的回归测试选择方法考虑了偶然正确性现象的影响,将回归测试用例集合因发生偶然正确性现象而导致错误未被发现的可能性控制在一定程度内.

如果一个测试用例对所有被修改语句所估算的偶然正确性概率都高,这个用例很可能发生偶然正确性现象.如果一个测试用例针对至少一条被修改语句的偶然正确性概率非常低,那么该用例很可能是故障检测用例.因此,本文根据估算得到的偶然正确性概率,来评判覆盖修改代码用例中哪些用例更可能检测到修改代码中的故障,将这些更可能检测故障的用例加入到回归测试集合中.

图 2 展示了基于偶然正确性概率进行回归测试选择的工作流程,其中,P 和 P'分别表示修改前后的程序,T 和 T'分别表示原始测试集合和回归测试集合.在对原始程序 P 进行测试时,获取 P 执行 T 中各个测试用例时发生偶然正确性的概率.当代码进行修改后,通过代码修改分析,根据 P 在 T 上的偶然正确性概率估计被修改后的程序 P'在执行 T 中各测试用例时发生偶然正确性现象的概率.最后,基于被修改后代码的偶然正确性现象的发生概率,选择出测试集合 T'作为回归测试集合.

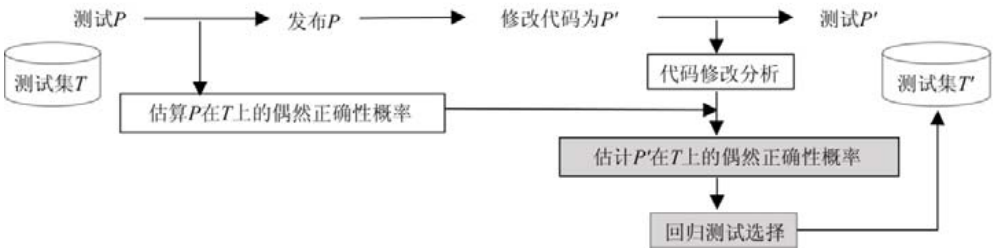

Fig.2 The process of CCP-based regression selection

图 2 基于 CCP 的回归测试选择的工作流程

本文提出的测试选择方法主要包含两部分工作,首先根据原始测试集合在原始版本程序上的偶然正确性现象的发生情况,来估计该测试集合在修改后代码上的偶然正确性现象的发生概率.然后,基于被修改代码执行时发生偶然正确性现象的概率来指导测试选择,将发生偶然正确性现象的概率控制在一定程度内.根据对测试用例缩减和其故障检测能力之间不同程度的折中,我们分别在基于代码覆盖的最小化和安全的选择技术基础上,给出两种利用偶然正确性概率指导测试用例选择的策略.

3.1 被修改后程序的偶然正确性概率的估算

为了能够为 P'选择适当的测试用例,需要得到 P'在执行这些测试用例时发生偶然正确性现象的概率,这个概率是通过原始程序 P 执行各测试用例时的偶然正确性概率来进行估算得到的近似值.

假设程序 P 中的语句 s 被修改为 s',这里用 $CCP(P,s,t)$表示在 s 包含错误的情况下 P 在执行测试用例 t 时发生偶然正确性的概率,$CCP(P',s',t)$表示在 s'包含错误的情况下 P'在执行测试 t 时发生偶然正确性现象的概率.在行回归测试用例选择时,$CCP(P',s',t)$只能通过之前测试 P 时获得的偶然正确性信息来估算得到一些近似值.如果 s 和 s'都是运算语句,且其中间结果被存放到同一个内存地址中,那么 P'执行时语句 s'产生的中间结果和 P 执行时 s 产生的中间结果被传播到输出的过程很可能是类似的,因此,本文将 $CCP(P,s,t)$作为 $CCP(P',s',t)$的一个近似值使用.另外,还有两种特殊情况需要进行处理.

(1) 语句 s 和 s'都是运算语句,但是它们写入的内存有所不同.

例如,P 中的语句"$x=a+b$"在 P'中被修改为"$y=a-b$".比较 P 和 P'运行测试用例 t 的过程,P 执行语句 s 时会把某个值存放到 x 中,而 P'相应地执行 s'时没有对 x 进行赋值、但对 y 进行了赋值.也就是说,与 P 执行时相比,P'执行时变量 x 和 y 中存放的值都发生了改变.本文的估算方法是对源程序进行静态分析,如果 s 所在的基本块内可以找到 s 之前的另一语句 $s1$,其写入的内存位置与 s'相同,那么,P'运行时 y 中值被传播到输出的过程和 P 运行时把 s 运算得到的中间结果传播到输出的过程是类似的,同时,P 和 P'运行时在 y 中存放的值是不同的,也就是说,对 s 的修改也影响了 y 中存放的值.如果这个影响被传播到了输出,测试人员可在某种程度上判断对 s 的修改是否正确.因此,本文将 $1-(1-CCP(P,s,t))\times(1-CCP(P,s1,t))$作为 $CCP(P',s',t)$的估算值.如果同一基本块中存在多条语句与 s'写入的内存位置相同,则选择距 s 最近的一条语句进行估算.如果找不到满足条件的语句 $s1$,此时就将 $CCP(P,s,t)$作为 $CCP(P',s',t)$的估算值.

(2) 语句 s 为控制语句.

此时将 $CCP(P,s,t)$直接作为 $CCP(P',s',t)$的估计值.虽然这种估算不是特别精细,但 $CCP(P,s,t)$和 $CCP(P',s',t)$仍然存在一定的相关性.

程序被修改后,在执行同一个测试用例时,其执行路径和运行时的动态数据流关系都会发生变化.因此,上述方法只是较粗略地估算出测试用例在新版本程序上运行时发生偶然正确性现象的概率,但是这些估算结果仍然能够在一定程度上反映各测试用例对修改后语句的测试有效性,本文的实验结果也表明这种估算是有效的.

3.2 回归测试选择策略

基于偶然正确性概率进行测试选择的过程分为下面 3 个步骤.

(1) 获取测试选择的备选测试用例集合 T'.本文将 T 中所有覆盖被修改代码的测试用例加入 T'.

(2) 估算 T'在 P'上执行时发生偶然正确性现象的概率.根据第 3.1 节给出的估算方法,根据 T'在 P 上的执行信息估算 T 在 P'上执行时发生偶然正确性现象的概率.

(3) 从备选测试集 T'中,根据代码覆盖和偶然正确性概率进行测试选择.

在第 3 步的测试选择过程中,本文给出两种具体的选择策略,分别采用了前文第 2.2 节中描述的最小化策略和安全策略的思想.

3.2.1 基于最小化的策略

本文的第 1 种选择策略是基于 Beena 等人[22]提出的基于代码覆盖的最小化方法,该方法以最小化测试集为目标,选择出满足基本的代码覆盖的最小测试集合.

在最小化方法的基础上,本文加入了基于偶然正确性概率的选择过程.首先基于代码覆盖的最小化技术,将满足代码覆盖的最小集合作为回归测试用例集.仅仅满足代码覆盖的充分度准则无法保障测试集合的错误发现能力,因此,本文从中再选择出满足基于偶然正确性概率的充分度准则的最小测试集,以提高故障检测用例的数目.在进行两次选择之后,回归测试集合满足代码覆盖和基于偶然正确性概率的充分度准则,并保障了测试集合发现错误的能力,同时,测试用例集合的规模也得到了控制.

为了提高回归测试集合中故障检测用例的数目,本文将偶然正确性概率低于一定标准 K_1 的测试用例加入被选集.这种策略更着重于用例的缩减,在测试资源有限的情况下,首先保证代码覆盖率和基于偶然正确性的充分度,从而保障测试集的错误发现能力.然后在资源允许的情况下,选择更可能检测出故障的测试用例,增加被选集合中故障检测用例的数目.测试人员可以通过调整 K_1 的值在测试集合大小与故障用例数目之间作不同程度上的折中.

算法 3.1 中展示了基于最小化策略的选择算法.算法的输入是覆盖修改的测试集合 T、被修改的语句集合 S、被修改程序上的偶然正确性概率以及需要满足的偶然正确性概率的上限 K_1.该算法的目标是输出一个回归测试用例集合 T'',尽可能地使得各个被修改语句的综合偶然正确性概率小于等于 K_1.该算法选择出的回归测试用例集满足代码覆盖和基于偶然正确性概率充分度标准,并将偶然正确性概率低于标准 K_1 的用例加入被选集.

算法 3.1 中使用了我们以往工作[5]所提出的基于偶然正确性概率的充分度准则.一个测试集合 T 在语句 s 上满足基于偶然正确性的充分度为 $AC(s,T)$,当 $AC(s,T)\geqslant\Phi$时,称 s 满足基于偶然正确性概率的充分度标准Φ.第 1 种选择策略包括了 3 个选择过程.

(1) 选择出满足代码覆盖的最小的测试集.

不断地将能够覆盖最多被修改代码的测试用例加入到被选集中,得到一个覆盖所有被修改代码的最小测试集.算法 3.1 中第 2 行~第 6 行描述了基于代码覆盖信息选择测试用例的过程.这里使用贪心算法从测试用例集合中优先选择那些可以覆盖更多未被覆盖的被修改语句的测试用例,当被选择的测试集合可以覆盖所有的被修改语句时停止选择.

(2) 选择满足基于偶然正确性概率的充分度的最小集.

此次选择的目的是得到一个较小的集合,使得被选集合对于被修改的每一条语句都满足基于偶然正确性概率的充分度标准.算法 3.1 中第 7 行~第 19 行描述了具体的选择方法.

(3) 选择偶然正确性概率低的用例加入被选集合.

这里设定了一个可调节的参数 K_1,将至少对于一条被修改语句的偶然正确性概率低于 K_1 的用例加入被选集合.因为只要测试用例可以发现 S 中任意一条语句中的错误,这个测试用例即为故障检测用例,应被选为回归测试用例.K_1 的设定值越高,对偶然正确性概率的要求越低,这种策略选出的回归测试集合也越大.算法 3.1 中第 20 行~第 25 行描述了这一步骤的具体操作.

算法 3.1. 采用最小化策略的基于偶然正确性概率的测试选择算法.

Input:

覆盖修改代码的测试集合T';

程序中被修改的语句集合S;

针对S中每一条语句,T中每个测试用例的偶然正确性概率;

偶然正确性概率的上限K_1;

Output:

回归测试用例集合T''.

```
初始情况下,T''=∅;
//第1步:为满足基于代码覆盖标准进行测试选择
while T''没有达到和T'同样的覆盖度并且T'不为空do
  从T'中选择可以得到最大覆盖度增量的测试用例t;
  将这个测试用例t加入到T''中并将其从T'中删除;
end while
//第2步:为满足基于CCP的充分度标准进行测试选择
定义一个语句集合S';//不满足CCP充分度的被修改语句的集合;
for each s∈S do
  if AC(s,T'')<Φ then
    将s加入到S'中;
  end if
end for
while S'不为空并且T'不为空do
  定义AdeqIncrement(t)=将t加入T''后使得S'中的满足基于CCP的充分度Φ的语句数目;
  从T'中选择AdeqIncrement(t)最大的测试用例:
    如果得到最大AdeqIncrement值的测试用例不止一个或所有用例的AdeqIncrement都为0(即所有用
    例都无法增加满足CCP充分度的语句),则选择偶然正确性概率乘积Π_{s∈S}CCP(s,t)最大的测试用例t;
  将t加入T'',并将t从T'中删除;
  在S'中删除在测试集合T''上满足充分度标准Φ的语句;
end while
//第3步:根据偶然正确性概率标准K1选择较低概率发生偶然正确性概率的用例
for each t∈T' do
  if S中至少一条语句s满足CCP(s,t)≤K1 then
    将t加入T'',并将t从T'中删除;
  end if
end for
```

3.2.2 基于安全技术的策略

基于安全技术的策略是在安全选择技术的基础上,根据偶然正确性概率进一步进行选择.安全的选择技术将所有覆盖修改的用例作为回归测试用例.然而,这种技术所选择出的测试集合中包含因偶然正确性现象的发生而无法检测故障的测试用例.因此,安全的选择策略无法进一步有效地缩减测试用例集合的大小.当全部运行这些测试用例所需要的资源超过了资源限制时,就需要按照一定的准则有效地缩减测试用例集合.本文采用了两种根据偶然正确性概率进行删减的方法,其具体的算法分别展示在算法 3.2 和算法 3.3 中.

第 1 种算法采取了一个较为简单的测试用例删减标准,它在不降低代码覆盖并满足基于偶然正确性概率

的充分度的前提下,根据偶然正确性的发生概率,从被选择集合中删除对于大部分被修改语句都具有较高偶然正确性概率的测试用例,达到进一步缩减回归测试集合大小的目的.对一个测试用例 t,如果存在至少一条被修改的语句s,使得$CCP(s,t)$的值较低,我们就认为这个用例很可能发现故障.如果测试用例对于所有被修改语句的偶然正确性概率都非常高,则认为这个用例很可能无法检测故障,在测试资源受限时将其从测试集合中删除.

算法 3.2. 采用安全策略的基于偶然正确性概率的测试选择算法 1.

Input:

覆盖修改代码的测试集合T';

程序中被修改的语句的集合S;

针对S中每一条语句,T中每个测试用例的偶然正确性概率;

偶然正确性概率选择下限K_2;

Output:

回归测试用例集合T''.

初始情况下,$T''=T'$;
for each $t\in T''$ **do**
  if 如果对于S中的90%以上的所有语句s,t都使得$CCP(s,t)\geqslant K_2$ **then**
    if 将t从T''中删除后,对于S中的任意一条语句s,都满足s被T''覆盖并且满足基于CCP的充分度Φ即$AC(s,T'')\geqslant\Phi$ **then**
      将t从T''中删除;
    end if
  end if
end for

算法 3.3. 采用安全策略的基于偶然正确性概率的测试选择算法 2.

Input:

覆盖修改代码的测试集合T';

程序中被修改的语句集合S;

针对S中每一条语句,T中每个测试用例的偶然正确性概率;

基于CCP的充分度的变化幅度的限制值K_2;

Output:

回归测试用例集合T''.

初始情况下,$T''=T'$;
//计算测试用例对充分度的影响程度
for each $t\in T''$ **do**
  for each $s\in S$ **do**
    计算t被删除后对测试集针对s的CCP充分度的影响$AC_{dec}(s,t,T'')=\dfrac{AC(s,T'')-AC(s,T''-t)}{AC(s,T'')}$;
    //$ACdec(s,t,T'')$表示t被删除后测试集对于s的充分度的影响程度
  end for
  //将被删除后测试集仍满足充分度标准的用例加入备选集
  计算其对所有语句的充分度降低程度的平均值$AC_{avg}(t)$;
  if $AC_{avg}(t)\leqslant K_2$并且对于S中的任意一条语句s,都满足s被$T''-t$覆盖并且满足基于CCP的充分度Φ即$AC(s,T''-t)\geqslant\Phi$ **then**

11: 将t加入备选集T''';
12: **end if**
13: **end for**
14: //为备选集中的每一个用例计算其评估变量AC_{meas}
15: **for** each $t \in T'''$ **do**
16: $\rho(S,t,T'')=\sum_{s\in S} AC_{dec}(s,t,T'')$; //表示$AC_{dec}(s,t,T'')$的和;
17: $\sigma(S,t,T'')=\sqrt{\frac{\sum_{s\in S} AC_{dec}(s,t,T'')-AC_{avg}}{|S|}}$; //表示$AC_{dec}(s,t,T'')$的标准差;
18: 定义评估变量$AC_{meas}(s,t,T'')=\rho(s,t,T'')\times\sigma(s,t,T'')$;
19: **end for**
20: //从T''中优先删除备选集中AC_{meas}最小的用例
21: **for** each $t \in T'''$ **do**
22: 选择$AC_{meas}(s,t,T'')$最小的测试用例t,并将其从T''中删除;
23: //更新备选集T''';
24: 从T''中选出被删除后仍使得T''满足充分度标准的用例加入T''';
25: **end for**

算法 3.2 展示了第 1 种删除方法的算法描述,其输入是覆盖修改的测试集合 T、被修改的语句集合 S、被修改程序上的偶然正确性概率以及需要满足的偶然正确性概率的下限 K_2.该算法选择出的回归测试集满足代码覆盖和基于偶然正确性概率的充分度标准,并且在此前提下尽可能地不包含发生偶然正确性概率较高的用例.这里设定了一个可调节参数 K_2,其取值在 0~1 之间,在满足覆盖率要求及基于偶然正确性概率的充分度的前提下,将那些对大多数被修改语句的偶然正确性概率都高于 K_2 的用例从测试集中删除.如果测试用例对于所有被修改语句的偶然正确性都非常高,表明它发现错误的概率很低,这个用例不应该被选为回归测试用例.K_2 设定得越高,对偶然正确性概率的要求就越宽松,其选出的测试用例数目也越多.

算法 3.3 中给出了另一种删除用例的方法,这种方法优先删除对各语句的基于偶然正确性概率的测试充分度的综合影响程度较小,并且使得各语句的测试充分度较均匀地下降的用例,目标是使得测试资源更平均地分配到各个被修改语句.对于每个测试用例,首先计算出该用例被删除后各语句的基于偶然正确性的充分度,然后根据其总体的降低程度和对各语句的具体降低情况定义一个评估标准,在保障测试集充分度的前提下优先删除那些综合影响程度较低并且对各语句的影响较为平均的用例.算法 3.3 的输入和输出与算法 3.2 一致,其中,K_2 是控制充分度变化幅度上限的一个可调节参数.首先,对每个测试用例计算出该用例被删除后基于偶然正确性概率的测试充分度的降低程度.然后,将对测试充分度的影响程度的平均值低于 K_2,并且被删除后仍能够使得测试集满足代码覆盖和基于偶然正确性概率的充分度标准的用例作为备选集 T'''.最后,在 T'''中选出使得充分度的综合下降程度较低并且使得各语句的充分度比较平均地下降的测试用例,并将其从 T''中删除.在算法中,用 $\rho(S,t,T'')$之和来表示 t 使得充分度下降的总体程度,用$\sigma(S,t,T'')$来衡量各语句的充分度下降的具体情况,然后基于$\rho(S,t,T'')$和$\sigma(S,t,T'')$定义了对用例进行评估的变量 AC_{meas}.该算法在保证测试集充分度的前提下,优先删除 AC_{meas} 较低的用例.

4 实验设计和评估

4.1 回归测试技术的评估方法

Rothermel 和 Harrold[23,24]提出了评估回归测试选择技术的统一框架,采用了包括安全性、精度、执行效率和可扩展性等评估标准.本文从中选择几个比较重要的指标对测试选择的结果进行评估,其中包括测试集合规模的缩减、安全性、精度变化以及一种衡量安全性和精度的综合指标 PR.

(1) 测试集合规模的缩减

测试集合规模的缩减(test size reduction)是衡量测试选择技术的一个重要指标,回归测试选择的目标之一是降低测试用例的规模.测试集合规模的缩减是依靠测试集合大小的变化来体现的.如果被选择的测试用例集合包含的用例越少,测试所花费的成本越小,则认为该回归测试选择技术在测试集合规模缩减上的优势越大.

(2) 安全性

第 2 个评估标准是安全性(safety)的变化.假设原始的测试集合为 T,用例选择技术选择出的测试用例集合为 T',其中可检测出故障的测试用例构成集合 T_f',T 中所有可检测出故障的测试用例的集合为 T_f.召回率(recall)的计算公式为

$$Recall=T_f'/T_f \tag{3}$$

测试选择技术的安全性是通过召回率来衡量的,是对测试用例选择完整性的度量,它表示被选择的故障检测用例占所有故障检测用例的比例.回归测试用例技术是安全的,当且仅当其召回率为 100%.一个安全的回归测试用例选择技术应将所有覆盖修改的测试用例都选择出来.

(3) 精度

假设原始的测试集合为 T,用例选择技术选择出的测试用例集合为 T',其中可检测出故障的测试用例构成集合 T_f',T 中所有可检测出故障的测试用例的集合为 T_f.精度(precision)的计算公式为

$$Precision=T_f'/T' \tag{4}$$

测试选择的精度是对测试选择技术的准确性的度量,测试用例选择结果的精度用于表示被选择出的测试集合中包含故障检测用例的比例.

(4) 综合指标 PR

理想情况下,我们希望一个测试选择技术可以同时具有高安全性和高精度.然而,安全性和精度有时是相互矛盾的,一种测试用例选择方法往往难以同时兼顾高的精度和高的安全性.因此,这里定义一种综合的度量 PR 作为评估精度和安全性的综合标准,具体的计算方法参照统计量 F-measure 的计算公式.F-measure 是数据挖掘中广泛使用的综合度量方法,常用来评估召回率和精确率的综合结果,其值为精度和召回率的加权调和平均,F-measure 的计算方法如下:

$$F\text{-measure}=(a^2+1)\times 精确率\times 召回率/(a^2\times 精确率+召回率) \tag{5}$$

在上式中,召回率表示安全性,精确度对应精度,本文将 PR 指标设置为参数 a=1 时的 F-measure 值,即

$$PR=2\times 精度\times 安全性/(精度+安全性) \tag{6}$$

4.2 实验设计和结果

本文在 software-artifact infrastructure repository(SIR)中提供的西门子测试套件上进行了实验,这些程序也是在偶然正确性相关研究工作[3,14]中常使用的实验程序.本文选取了 20 个版本的小于 1 000 行的 C 程序,SIR 对于每个程序提供了多种版本的错误变种程序,这里将正确版本的程序作为回归测试的原始程序,各个包含错误的程序版本被认为是被修改后的程序.

由于目前没有关于偶然正确性概率在测试选择领域的相关工作,因此在实验中将基于代码覆盖的安全的测试选择技术作为对比技术,与本文提出的 3 种测试选择方法在测试集合的缩减、安全性、精度和 PR 指标这 4 个方面进行比较.用于对比的安全测试选择技术将所有覆盖修改的测试用例作为回归测试用例.在实验中,基于 CCP 的测试选择技术 1 使用了本文提出的第 1 种选择策略,基于 CCP 的测试选择技术 2 和技术 3 则分别使用了第 2 种策略的两种选择算法.

图 3 展示了各目标程序发生偶然正确现象的频率,图 4~图 7 分别展示了基于 CCP 的 3 种测试选择技术与基于代码覆盖的技术在测试集合缩减、精确度、安全性以及 PR 指标的具体实验数据.表 1 是上述实验数据的综合统计结果,展示了相对于基于代码覆盖的测试选择技术、3 种基于 CCP 的技术在用例缩减、精确度、安全性以及 PR 指标上的变化幅度.

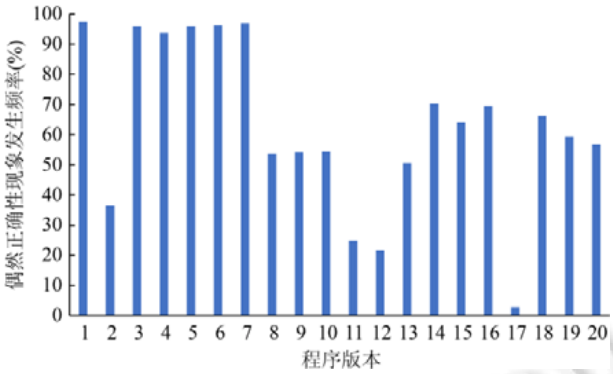

Fig.3 Prevalence of CC in subject programs

图 3 目标程序发生偶然正确性现象的频率

Table 1 Changes of CCP-based techniques on three indicators relative to the coverage-based technique

表 1 基于 CCP 的选择技术相对于基于代码覆盖技术在评估指标上变化幅度的均值

指标变化的幅度	用例缩减(%)	精度(%)	安全性(%)	PR(%)
基于 CCP 的技术 1	929	28.36	–47.49	19.72
基于 CCP 的技术 2	344	23.14	–12.11	20.19
基于 CCP 的技术 3	396	25.09	–16.89	21.14

图 4 展示了 3 种基于 CCP 的测试选择技术和基于代码覆盖的安全选择技术在测试用例缩减方面的对比数据,其中,横坐标表示不同版本的程序,纵坐标表示经过测试选择后的回归测试集相对于原始测试集合在用例数目上缩减的比例.假设原始测试集合的用例数目为 x,经过测试选择后集合的用例数目为 y,则用例数目缩减比例为 $(x–y)/x\times100\%$.对图 4 所示数据进行统计可得,基于代码覆盖的安全选择策略和 3 种基于 CCP 的技术在用例缩减基于代码覆盖的安全技术用例缩减比例的均值为 33%,基于 CCP 的技术 1 的用例缩减比例的均值为 74%,基于 CCP 的技术 2 的用例缩减的平均比例为 51%,而基于 CCP 的技术 3 的用例缩减的平均比例为 54%.表 1 展示了基于 CCP 的 3 种技术相对于基于代码覆盖技术在用例缩减部分的提升幅度的平均值.假设基于 CCP 的技术所缩减用例的比例为 x,基于代码覆盖的技术缩减的比例为 y,则基于 CCP 的技术在用例缩减方面所提升的幅度为 $(x–y)/y\times100\%$.结果表明,基于 CCP 的选择技术在用例缩减上的提升幅度分别高达 929%、344%和 396%.实验结果表明,3 种基于 CCP 的测试选择技术都较大程度地缩减了被选择的测试用例的数目,并且其缩减的程度显著大于基于代码覆盖的测试选择技术所缩减的程度.

对照图 3 中发生偶然正确性现象的频率及图 4 中的测试用例缩减情况可得,对于偶然正确性现象发生频率高的程序,基于 CCP 的选择技术对其测试集合缩减的提升程度也相对较高.反之,对于偶然正确性现象发生频率低的程序,在测试集合缩减上的提升也相对低一些.例如,对于编号为 1、3、4、5、6 和 7 的程序,偶然正确性现象发生的概率非常高,从图 3 可以发现,基于安全的选择技术在这几个程序上的用例缩减比例是非常低的(都低于 10%),而我们的技术在这些程序上达到了较好的提升.其中,基于 CCP 的技术 1 使得这几个程序对应的测试集合缩减比例达到 60%以上,相对于安全选择技,技术 1 在用例缩减比例上提升的差值高达 56%~61%(用图 4 中基于 CCP 的技术 1 的测试集合缩减比例与安全技术的缩减比例的差值来计算).而对于偶然正确现象的频率相对较低的程序,例如编号 11、12 和 17 的程序,基于安全技术已经得到了较高程度的用例缩减,而基于 CCP 的技术对用例缩减的提升有限.偶然正确性现象的出现频率和用例缩减的对照结果表明,基于 CCP 的选择技术可以有效地消除偶然正确性现象对测试用例集合缩减的影响.

测试用例集合较大幅度的缩减可能会影响测试选择的安全性和精度,为了进一步证明基于 CCP 的技术的有效性,下文给出精度、安全性和综合指标 PR 上的实验结果.

有关精度的实验数据如图 5 所示,图上的数据表明了两种基于 CCP 的测试选择技术在一定程度上提升了测试选择结果的精度.对于大多数版本的程序,基于 CCP 的 3 种技术都提高了测试选择的精度.如表 1 所示,相对于基于代码覆盖的技术,3 种基于偶然正确性现象的选择技术在精度上提升幅度的均值为 28.36%、23.14%和 25.09%.

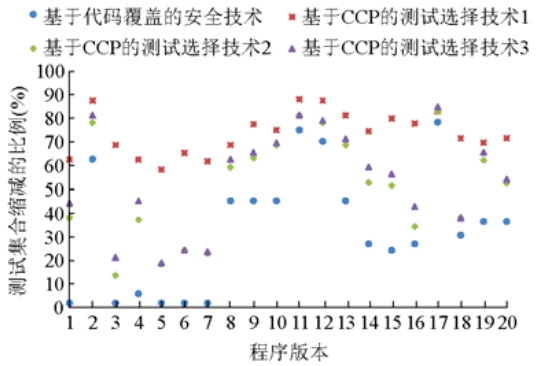

Fig.4 Reduction of the test set

图 4 测试集合的缩减

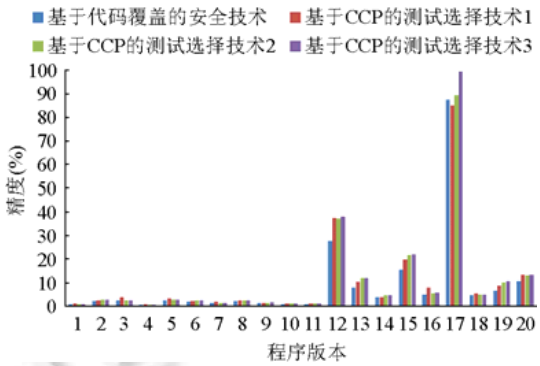

Fig.5 Changes of precision

图 5 精度变化

有关安全性的实验数据如图 6 所示,其横坐标为不同版本的程序,纵坐标为针对选择技术的安全性指标,图上展示了 4 种技术在不同版本程序上进行测试选择的安全性.由图可知,基于代码覆盖的安全技术的安全性始终为 100%,而基于 CCP 的选择技术在不同程度上降低了安全性.

为了综合衡量精度和安全性的变化,图 7 展示了 4 种技术在 PR 指标上的变化情况.对图 7 中的数据进行统计可得,基于 CCP 的技术 1 提升了 85%版本的程序的 PR 指标,而基于 CCP 的技术 2 和技术 3 提升了 95%版本的程序的 PR 指标.这 3 种技术降低了小部分版本程序的 PR 值,但其降低的幅度非常小.如表 1 所示,基于 CCP 的 3 种技术在 PR 指标上的总体提升幅度分别为 19.72%、20.19%和 21.14%.PR 指标的提升表明,基于 CCP 的测试选择技术虽然降低了安全性,但精度方面的提高所产生的收益大于安全性降低的损失.

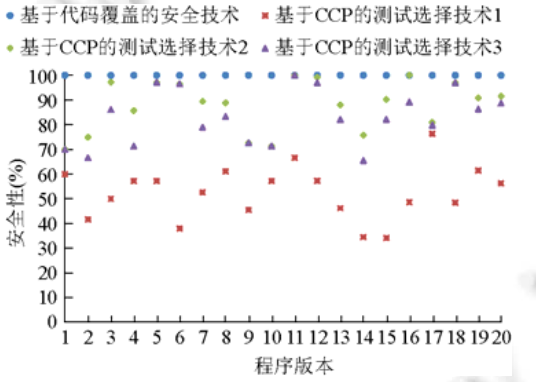

Fig.6 Changes of safety

图 6 安全性变化

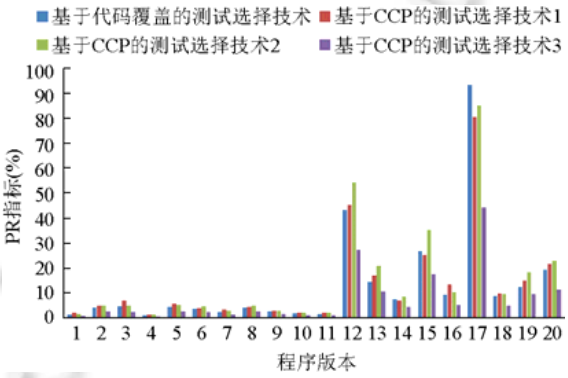

Fig.7 Changes of PR

图 7 PR 指标的变化

综合上述实验结果,本文提出的 3 种基于 CCP 进行测试选择的技术都大幅度地缩减了测试集合的大小,并提升了测试选择的精度,虽然降低了测试选择的安全性,但是 3 种技术都使得精度和安全性的综合指标 PR 值得到了较大的提升,这表明,基于 CCP 的测试选择技术比基于代码覆盖的技术能够更有效地对测试用例进行选择.

对比 3 种基于 CCP 的选择技术,其精确度和 PR 的提升幅度的差距不大,在测试用例缩减程度和安全性上

有较大的差异.其中,基于CCP的技术1更着重于测试用例的缩减,更大幅度地降低测试集合大小,但是也伴随着安全性的降低.基于CCP的技术2和技术3更着重于选择更多的故障检测用例,在一定程度上缩减了测试集合,并且安全性指标的降低幅度较小.基于CCP的技术2和技术3是策略2的两种算法的具体实现,较为复杂的技术3在用例缩减、精度和PR指标上略优于技术2,而技术2在安全性上的表现更好.在使用基于偶然正确性概率的测试选择方法时,可以根据测试集合缩减和故障检测间侧重的不同来选择具体的测试选择技术.

5 总 结

回归测试选择技术是降低测试用例集合规模的一种有效手段,而偶然正确性现象的普遍存在使得回归测试用例集合包含了大量无法检测故障的用例.因此,本文提出一种根据偶然正确性现象的发生概率进行回归测试选择的方法,进一步缩减了测试用例集合的规模,提升了测试选择的精度.本文首先给出一种根据测试用例在原始程序上的执行信息来估算被修改程序执行时发生偶然正确性现象的概率的方法,然后基于得到的偶然正确性概率的估值来指导测试选择.根据在测试集合缩减和故障检测能力之间不同程度的侧重,本文提出了基于最小化和安全性的两种选择策略,并给出3种具体的选择技术.相对于基于代码覆盖的安全性选择技术,本文提出的3种测试选择技术在缩减测试集合规模上有明显的优势,并提升了测试选择的精度以及精度和安全性的综合指标.对于数据驱动的智能系统等偶然正确性现象发生概率高的软件,本文的方法可以更大程度上缩减测试用例的数目,从而节省这类系统的测试开销.

References:

[1] Masri W, Podgurski A. Measuring the strength of information flows in programs. ACM Trans. on Software Engineering and Methodology, 2009,19(2):5:1–5:33. [doi: 10.1145/1571629.1571631]

[2] Zhou XL, Wang HF, Zhao JH. A fault-localization approach based on the coincidental correctness probability. In: Proc. of the Int'l Conf. on Software Quality, Reliability and Security. 2015. 292–297. [doi: 10.1109/QRS.2015.48]

[3] Masri W, Abou-Assi R. Cleansing test suites from coincidental correctness to enhance fault-localization. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation. 2010. 165–174. [doi: 10.1109/ICST.2010.22]

[4] Masri W, Abou-Assi R. Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Trans. on Software Engineering and Methodology, 2014,23(1):8:1–8:28. [doi: 10.1145/2559932]

[5] Zhou XL, Wang LZ, Li XD, Zhao JH. An empirical study on the test adequacy criterion based on coincidental correctness probability. In: Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering, 2014. 632–635. [doi: 10.1145/2020723.2020743]

[6] Rothermel G, Harrold MJ. A safe, efficient regression test selection technique. ACM Trans. on Software Engineering and Methodology, 1997,6(2):173–210. [doi: 10.1145/248233.248262]

[7] Budd TA, Angluin D. Two notions of correctness and their relation to testing. Acta Informatica, 1982,18(1):31–45. [doi: 10.1007/BF00625279]

[8] Hierons RM. Avoiding coincidental correctness in boundary value analysis. ACM Trans. on Software Engineering & Methodology, 2006,15(3):227–241. [doi: 10.1145/1151695.1151696]

[9] Daran M. Software error analysis: A real case study involving real faults and mutations. In: Proc. of the ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 1996. 158–171. [doi: 10.1145/226295.226313]

[10] Voas JM. PIE: A dynamic failure-based technique. IEEE Trans. on Software Engineering, 1992,18(8):717–727. [doi: 10.1109/32.153381]

[11] Ammann P, Offutt P. Introduction to Software Testing. Cambridge University Press, 2008.

[12] Denmat T, Ducasse M, Ridoux O. Data mining and crosschecking of execution traces: A re-interpretation of Jones, Harrold and Stasko test information visualization. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. 2005. 369–399. [doi: 10.1145/1101908.1101979]

[13] Miao Y, Chen ZY, Li SH, Zhao ZH, Zhou YM. A clustering-based strategy to identify coincidental correctness in fault localization. Int'l Journal of Software Engineering and Knowledge Engineering, 2013,23(5):721–741. [doi: 10.1142/S0218 194013500186]

[14] Li Y, Liu C. Identifying coincidental correctness in fault localization via cluster analysis. Journal of Software Engineering, 2014, 8(4):328–344. [doi: 10.3923/jse.2014.328.344]

[15] Wang XM, Cheung SH, Chan WK, Zhang ZY. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 45–55.

[16] Chen X, Gu WJ, Xu H, Gu Q, Chen DX. Regression testing techniques: A state-of-the-art review. Computer Science, 2013,40(10): 1–9 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2013.10.001]

[17] Leung HKN, White L. Insights into regression testing (software testing). In: Proc. of the Conf. on Software Maintenance. 1989. 60–69. [doi: 10.1109/icsm.1989.65194]

[18] Beydeda S, Gruhn V. Integrating white- and black-box techniques for class-level regression testing. In: Proc. of the Int'l Computer Software and Applications Conf. 2001. 357–362. [doi: 10.1109/CMPSAC.2001.960639]

[19] Soetens QD, Demeyer S, Zaidman A. Change-based test selection in the presence of developer tests. In: Proc. of the European Conf. on Software Maintenance and Reengineering. 2013. 101–110. [doi: 10.1109/CSMR.2013.20]

[20] Mirarab S, Akhlaghi S, Tahvildari L. Size-constrained regression test case selection using multicriteria optimization. IEEE Trans. on Software Engineering, 2012,38(4):936–956. [doi: 10.1109/TSE.2011.56]

[21] Graves TL, Harrold MJ, Kim JM, Porter A, Rothermel G. An empirical study of regression test selection techniques. ACM Trans. on Software Engineering and Methodology, 2001,10(2):184–208. [doi: 10.1145/367008.367020]

[22] Beena R, Sarala S. Code coverage-based test case selection and prioritization. Int'l Jornal of Software Engineering and Applications, 2013, 39–49. [doi: 10.5121/ijsea.2013.4604]

[23] Rothermel G, Harrold MJ. A framework for evaluating regression test selection techniques. In: Proc. of the Int'l Conf. on Software Engineering. 1994. [doi: 10.1109/ICSE.1994.296779]

[24] Rothermel G, Harrold MJ. Analyzing regression test selection techniques. IEEE Trans. on Software Engineering, 1996,22(8): 59–551.[doi: 10.1109/32.536955]

附中文参考文献:

[16] 陈翔,顾卫江,徐慧,顾庆,陈道蓄.回归测试用例选择技术研究综述.计算机科学,2013,40(10):1–9. [doi: 10.3969/j.issn.1002-137X.2013.10.001]

周小莉(1988－),女,博士生,CCF 学生会员,主要研究领域为程序分析,软件测试.

赵建华(1971－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,形式化方法.