基于路径分析的蜕变测试组生成与优先级排序技术

孙昌爱 邢嘉煜 刘宝莉 付 安
(北京科技大学计算机与通信工程学院 北京 100083)

摘 要 蜕变测试依据待测软件的蜕变属性(通常表示为蜕变关系),由原始测试用例生成衍生测试用例,通过检查它们对应的输出结果是否满足蜕变关系确定测试是否通过,由于无需构造单个测试用例的预期输出结果,因此有效地缓解了测试预期问题。不难看出,蜕变关系和原始测试用例决定了蜕变测试的故障检测有效性。尽管已经存在一些面向蜕变测试的测试用例生成方法,这些方法存在如下不足:忽略了蜕变关系的作用范围,存在易于生成无效的测试用例的问题;仅仅考虑原始测试用例之间的差异,导致生成的蜕变测试组(即原始测试用例与衍生测试用例对)不充分问题;未考虑测试用例的故障检测能力差异,从而影响蜕变测试的故障检测效率。针对上述问题,本文提出了一种基于路径分析的蜕变测试组生成与优先级排序技术(简称 PaMTG)。在待测程序路径分析的基础上,PaMTG 首先获得满足蜕变关系的可行路径对,然后生成覆盖可行路径对的蜕变测试组,最后依据执行路径信息对蜕变测试组进行优先级排序。开发了相应的支持工具,并采用一组程序从测试用例的有效性、故障检测能力、故障检测效率和时间开销四个方面对 PaMTG 进行了实验评估。实验结果表明,PaMTG 能够生成有效的蜕变测试组,且生成的蜕变测试组的故障检测能力与效率优于现有基准技术。

关键词 软件测试;蜕变测试;符号执行;测试用例生成;测试用例优先级排序
中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2025.00675

Metamorphic Testing Group Generation and Prioritization Technique Based on Path Analysis

SUN Chang-Ai XING Jia-Yu LIU Bao-Li FU An
(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083)

Abstract Metamorphic testing (MT) leverages metamorphic properties (usually known as metamorphic relationships, MR) of the software under test (SUT) to generate the follow-up test cases from the source test cases, and verify the test results by examining whether the MR is followed by the corresponding outputs. MT effectively alleviates the oracle problem because it is no longer necessary to obtain the expected outputs of individual test cases. Obviously, the used MRs and source test cases play a key role in the fault detection effectiveness of MT. Although there are already some test case generation methods for MT, they have the following limitations. Firstly, the scope of applicable input domains for an MR is not carefully considered, which may result in invalid test cases. Secondly, only the differences between source test cases are considered during the test case generation, which may result in insufficient metamorphic testing groups (a pair of source test case and follow-up test case, briefly as MTG). Finally, the fault detection capability of test cases is not considered, which may affect the fault detection efficiency of MT. In order to solve the above limitations, we propose a metamorphic testing group generation and prioritization technique based on path analysis (PaMTG). PaMTG first obtains all path pairs of the MR through analyzing the possible paths of the SUT, then generates MTGs to cover as many

收稿日期:2024-01-19;在线发布日期:2024-11-11.本课题得到国家自然科学基金(62272037,61872039)资助。**孙昌爱**(通信作者),博士,教授,博士生导师,中国计算机学会(CCF)杰出会员,主要研究领域为软件测试、程序分析、服务计算。E-mail:casun@ustb.edu.cn。**邢嘉煜**,硕士研究生,主要研究方向为软件测试。**刘宝莉**,硕士研究生,主要研究方向为软件测试。**付 安**,博士研究生,主要研究方向为软件测试。

path pairs as possible, and finally prioritizes the derived MTGs according to their covered path information. A supporting tool was developed and an empirical study was conducted to evaluate PaMTG in terms of valid MTG ratio, fault detection capability, fault detection rate, and time overhead. The experimental results show that PaMTG is able to generate valid MTGs, and the fault detection capability and fault detection rate of the generated MTGs are better than that of the existing baseline techniques.

Keywords software testing; metamorphic testing; symbolic execution; test case generation; test case prioritization

1 引 言

软件测试是一种广泛采用的软件质量保障手段,其基本思想是使用有限的测试用例运行待测软件,通过比较预期输出与实际输出是否一致判断被测软件中是否存在故障。由于软件系统的功能与结构复杂,针对任意测试用例难以或无法构造预期输出,即所谓的测试预期问题[1-3](Oracle 问题)。针对测试预期问题,研究人员从不同角度出发提出了一些解决方案,例如 N 版本编程[4]、断言[5]、统计测试[6-7]、蜕变测试[8]等。其中,蜕变测试依据待测软件存在的蜕变属性对其进行测试,即通过多个测试用例执行待测程序并判断其输出是否满足某种蜕变关系来检测软件中潜藏的故障。蜕变测试无需构造单个测试用例的测试预期,因此有效缓解了测试预期问题。蜕变关系不仅用来验证测试结果,而且还用于生成衍生测试用例,是蜕变测试实施的关键。为此,人们提出了多种蜕变关系获取与选择方法,如基于复合函数的蜕变关系构造方法[9]、组合蜕变关系方法[10]、使用机器学习自动预测蜕变关系的方法[11]、基于搜索的多项式蜕变关系推理方法[12]等。

已有研究表明[13]:除了蜕变关系,原始测试用例的生成与选择也严重影响蜕变测试的有效性。在 2016 年之前,57% 的工作使用随机测试[14]生成原始测试用例,34% 的工作从现有测试池中选择原始测试用例[15]。一些研究提出蜕变测试的测试用例生成技术,例如 Wu[16]提出一种迭代的测试用例生成方法;董等人[17]提出面向路径覆盖准则的蜕变测试用例生成方法;Batra 等人[18]使用遗传算法生成原始测试用例;Barus 等人[19]基于自适应随机测试为蜕变测试选择有效的原始测试用例;Alatawi 等人[20]基于动态符号执行生成原始测试用例;课题组前期工作[21]提出一种基于符号执行的测试用例生成技术。

分析已有的蜕变测试用例生成技术,我们发现存在如下问题:

(1)产生较多无效的测试用例:即产生不属于蜕变关系适用范围的测试用例。随机测试、覆盖测试[22]、基于搜索的测试[23]和符号执行[24]等蜕变测试的测试用例生成技术,仅关注原始测试用例生成问题,未考虑蜕变关系的作用范围,导致可能产生无效的原始与衍生测试用例。

(2)测试用例的充分性问题:现有的测试用例生成策略仅考虑原始测试用例之间的差异,并基于原始测试用例转换生成衍生测试用例。例如,课题组前期工作[21]能够生成覆盖所有可行路径的原始测试用例,然而单个原始测试用例的最小生成区间可能对应多个不同的衍生测试用例区间,因而存在蜕变测试组的充分性问题。

(3)蜕变测试的效率问题:现有的面向路径覆盖准则的蜕变测试用例生成方法[17]仅关注测试用例的路径覆盖情况,没有考虑测试用例的执行顺序,从而影响了蜕变测试的故障检测效率。

针对上述问题,本文从蜕变测试组的角度出发,提出一种基于路径分析的蜕变测试组生成与优先级排序方法,产生有效且高效的蜕变测试用例,提高蜕变测试效率。该方法在路径分析的基础上,获得原始测试用例和衍生测试用例的路径信息和约束条件,生成覆盖所有可行路径对的蜕变测试组;进一步设计了基于路径距离的优化策略,优先选择原始测试用例和衍生测试用例执行路径差异大的蜕变测试组。本文方法同时考虑原始测试用例和衍生测试用例的差异,生成覆盖蜕变关系所有可行路径对的蜕变测试组,并基于路径距离进行优先级排序。实验结果表明,该方法较好地解决了蜕变测试用例生成面临的有效性、充分性和高效性问题。

本文研究工作的主要贡献如下:

(1)提出一种基于路径分析的蜕变测试组生成方法：使用符号执行技术分析路径约束，通过约束求解技术生成充分的蜕变测试组。

(2)使用基于路径距离的蜕变测试组优先级排序方法，提高蜕变测试的故障检测效率。

(3)开发了基于路径分析的蜕变测试用例生成方法支持工具。

(4)使用一组实验程序评估所提方法的有效性、故障检测能力、故障检测效率和时间开销。

本文第 2 节介绍研究的相关基础知识；第 3 节讨论研究动机以及所提方法的技术框架；第 4 节介绍相关支持工具；第 5 节介绍实验评估设置；第 6 节讨论实验结果；第 7 节介绍相关研究工作；第 8 节总结全文。

2　基础知识

本节介绍蜕变测试、符号执行与约束求解等相关技术。

2.1　蜕变测试

蜕变测试作为一种可以有效缓解测试预期问题的软件测试技术[3]，其主要思想是通过两个或多个存在某种关系的测试用例执行程序后的实际输出之间是否符合蜕变关系来检验待测程序是否存在故障。通常，待测程序的两个和多个测试用例之间存在的某种关系和对应输出结果之间也存在的某种关系称为蜕变关系(Metamorphic Relation，MR)，测试用例之间存在的关系称为蜕变关系的输入关系，输出结果之间存在的关系称为蜕变关系的输出关系。蜕变关系可以从软件的规格说明或程序代码内部属性中获得。在蜕变测试中，两个或多个存在某种关系的一组测试用例称为蜕变测试组(Metamorphic Testing Group，MTG)。蜕变测试组包含的测试用例可以分为两类，原始测试用例和衍生测试用例。原始测试用例可以在执行蜕变测试之前由各种测试用例生成技术生成，如人工生成、随机生成、特殊值。衍生测试用例由现有的原始测试用例按照蜕变关系的输入关系进行推导得到。

蜕变测试原理如图 1。首先，根据待测程序的规格说明分析其具有的必要属性并设计相应的蜕变关系；然后，使用原始测试用例生成方法设计出原始测试用例，根据蜕变关系的输入关系 R 推导出衍生测试用例；其次，分别将两类测试用例执行待测程序，并分别获取对应的实际输出；最后，通过判断这两类实际输出结果是否符合蜕变关系的输出关系 R_f 来判断程序是否存在潜在的故障。综上，蜕变测试只需判断测试用例多次执行待测程序的实际输出之间是否满足蜕变关系，而不需要预先构造测试用例的预期输出，因而有效缓解了测试预期问题。

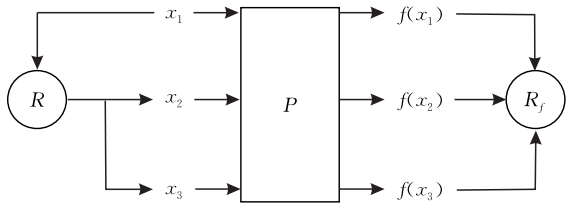


图 1　蜕变测试原理

蜕变测试的形式化定义如下[25]：

假设程序 P 实现函数 f 的功能，$x_1, x_2, \cdots, x_n (n>1)$ 是函数的不同变量，如果一个建立在 $x_1, x_2, \cdots, x_n (n>1)$ 之间的关系 R 使得函数输出 $f(x_1), f(x_2), \cdots, f(x_n)$ 满足关系 R_f，即

$$R(x_1 x_2, \cdots, x_n) \Rightarrow R_f(f(x_1), f(x_2), \cdots, f(x_n)) \quad (1)$$

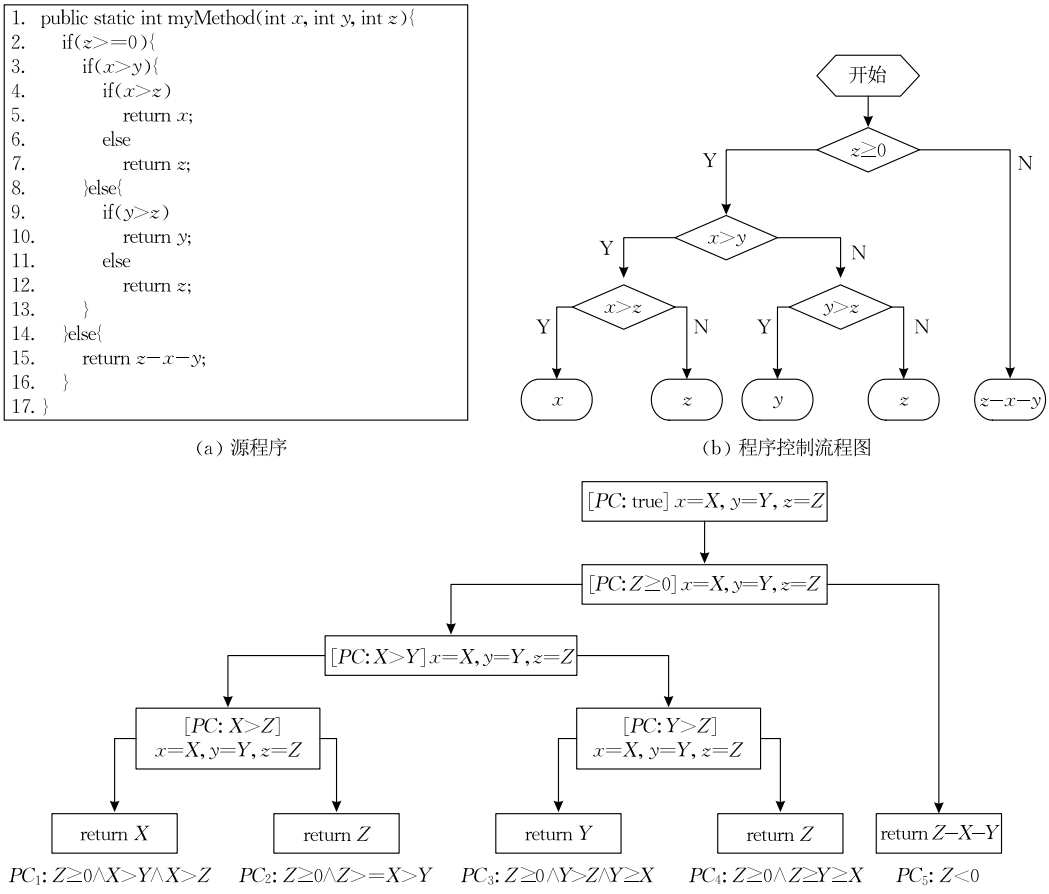
则称 $MR=(R, R_f)$ 为程序 P 的蜕变关系。

蜕变测试具有概念简单、易于自动化、开销低等优点[2,26-27]。蜕变测试的研究工作主要集中在测试用例的生成、蜕变关系的识别与选择、与其他测试验证方法结合、蜕变测试在不同领域的应用等方面。特别地，蜕变测试技术已成功应用于多个领域，如 Web 服务[28]、嵌入式系统[29-30]、机器学习分类器[31-32]、搜索引擎[33-34]、编译器[35]等。另一方面，蜕变测试仍然存在有待进一步解决的问题。例如，如何选择与识别有效的蜕变关系和如何生成有效且高效的测试用例等问题。

2.2　符号执行与约束求解

符号执行是一种程序分析技术，利用抽象的符号值而不是具体的数据作为程序的输入执行程序，并沿着程序的路径系统地收集路径约束，使用符号值表达式表示程序执行过程中的变量和输出。符号执行创建两个全局变量：(1)符号状态(σ)：记录程序的每个输入变量到符号值表达式的映射；(2)路径约束(PC)：是符号值表达式表示的约束集合，用来表示每个程序的执行路径信息和路径约束，初始值为“true”。符号执行搜索完所有的可执行路径，则路径约束代表每条路径的输出约束条件，对该约束条件进行约束求解可以生成执行该路径的输入。

符号执行过程中探索的可执行路径通过符号执行树进行描述。例如，图 2(a)示意的程序代码对应的程序控制流程图与符号执行树分别为图 2(b)与



(a) 源程序　(b) 程序控制流程图

(c) 符号执行树

图 2　符号执行示例

图 2(c)。符号执行树的每个节点代表一个程序状态,边表示程序状态的转换。符号执行初始时的符号状态为空且符号路径约束值初始值为 true。在程序执行期间,用符号值 X、Y 和 Z 分别表示参数列表的变量 x、y 和 z。当执行判断语句(第 2 行)时,出现是否满足"$Z\geq 0$"的两种情况。相应地,依次对两条分支进行探索,并记录路径约束条件"$Z\geq 0$"和"$Z<0$"。遍历完程序的所有可执行路径后,即可生成多条可执行的程序路径信息及其约束条件。

符号执行可以探索尽可能多的程序可执行路径,同时也能检查在程序路径上是否存在错误,如内存损坏、未捕获异常、断言违规等。符号执行的实现存在不同的方式:静态符号执行是使用符号值代替程序的实际输入执行代码;动态符号执行是使用符号值和具体数值混合参数作为程序的输入参数执行代码。符号执行需要遍历程序的所有可行路径,因此计算代价较大。近年来,随着约束求解方面的研究进展和计算资源的增加[36],出现了多种较为实用的符号执行支持工具。定向自动随机测试 DART[37] 是一个代表性的动态符号执行技术,随机生成需要具体数值执行的参数,符号值和具体值作为输入同时执行待测程序。此外,JPF[38] 为 Java 字节码提供符号执行的运行环境。SPF[39-40] 通过已有的决策方法和约束求解器对程序路径约束条件求解,能够生成路径覆盖率高的输入集合,并检查断言和并发等方面程序违规信息。SPF 可以处理布尔值、整数、实数、字符串等复杂数据结构的输入[41],可以系统地搜索符号执行的路径,也可以利用 JPF 中提供的深度优先或广度优先的搜索策略。Z3 是一款代表性的可满足性模理论的约束求解器[42],广泛应用于静态类型检查、测试用例生成和谓词抽象等领域,如 Pex[43]、HAVOC[44]、Vigilante[45]。

本文通过符号执行待测程序获取待测程序的路径信息和约束条件,借助约束求解技术生成测试用例。基于符号执行支持工具 SPF 和约束求解器 Z3,进一步开发蜕变测试组生成支持工具。

3 方　法

首先介绍本文工作的研究动机,然后讨论所提

方法的框架与关键技术，最后采用程序实例演示方法的应用。

3.1　研究动机

实施蜕变测试时，如果不考虑蜕变关系的约束条件，则易于产生不适用的测试用例，即无效的蜕变测试用例。第一，随机测试、自适应的随机测试等广泛采用的测试用例生成方法[14,19]生成的测试用例难以有效覆盖较多的程序路径；第二，基于符号执行的测试用例生成方法[21,24]能够生成有效覆盖程序执行路径的测试用例集，但仅仅考虑了原始测试用例的路径覆盖率情况，却忽略了衍生测试用例的适用范围与路径覆盖情况；最后，面向路径覆盖准则的蜕变测试用例生成方法[17]忽略了测试用例优先级排序问题，影响蜕变测试的故障检测效率。

以图 2(a)示意的程序为例，当参数 $z\geq 0$ 时，$myMethod(\text{int } x,\ \text{int } y,\ \text{int } z)$的返回值是整数参数 x,y,z 的最大值。不难发现，该程序应满足如下蜕变关系：如果测试输入中 y 和 z 的值互换，相应的输出不变。即 $MR:(Z\geq 0,Z'\geq 0,Y'=Z,Z'=Y)\Rightarrow myMethod(X',Y',Z')=myMethod(X,Y,Z)$，其中 X'、Y'、Z'分别表示衍生测试用例中的参数 x、y 和 z 值。显然，若不考虑该约束条件“$Z\geq 0$”和“$Z'\geq 0$”，则易生成无效的测试用例。假设原始测试用例为 $tc:(X=1,Y=-1,Z=2)$，依据上述蜕变关系生成的衍生测试用例为 $tc':(X'=1,Y'=2,Z'=-1)$，由于生成的衍生测试用例 tc'不满足约束条件“$Z'\geq 0$”，因此是无效的。使用基于符号执行的测试用例生成方法时，由于仅考虑原始测试用例执行程序路径情况，在原始测试用例覆盖路径 PC_3 时仅生成一组蜕变测试组。实际上，根据路径 PC_3 的约束条件和蜕变关系可以推导出衍生测试用例的约束条件为“$Z'\geq 0\wedge Y'\geq 0\wedge X'\leq Z'\wedge Z'>Y'$”，该约束条件与路径 PC_2、路径 PC_4 均存在交集，即在原始测试用例覆盖路径 PC_3 时存在覆盖情况为(PC_3，PC_2)和(PC_3，PC_4)的两个蜕变测试组。显然，基于符号执行的测试用例生成方法仅覆盖其中一组，无法执行蜕变关系满足的所有可行路径对。

为了减少无效的测试用例与提升蜕变测试组的覆盖率，本文充分挖掘与利用蜕变关系覆盖的路径信息，在路径分析的基础上引入蜕变关系约束条件，生成充分覆盖的蜕变测试组。另一方面，为了提高蜕变测试组的故障检测效率，进一步考虑了蜕变测试用例的优先级排序。具体说来，课题组前期工作[21]根据原始测试用例的语句覆盖率的贡献程度确定优先级排序，不考虑衍生测试用例的排序问题。本文则从蜕变测试组的执行路径的差异性角度出发，探索高效的蜕变测试组优先级排序方法。

3.2　方法框架

本文提出的基于程序路径分析的蜕变测试组生成与优先级排序方法 PaMTG，如图 3 所示。该方法包括三个主要步骤：(1)基于符号执行的程序路径约束获取：通过符号执行技术获取待测程序路径和约束条件；(2)基于约束求解的蜕变测试组生成：首先根据蜕变关系满足的原始域与每条路径的约束条件，获得每条路径的原始子域；然后依据蜕变关系的输入关系，由每条路径的原始子域推导出对应的衍生域，在衍生域上搜索适用的路径并分析获得其衍生子域；最后通过衍生子域的约束条件求解出衍生测试用例，根据蜕变关系逆推导出原始测试用例；(3)基于路径距离的蜕变测试组优先级排序：以原始测试用例和衍生测试用例执行路径的差异性为指导，对蜕变测试组进行优先级排序。

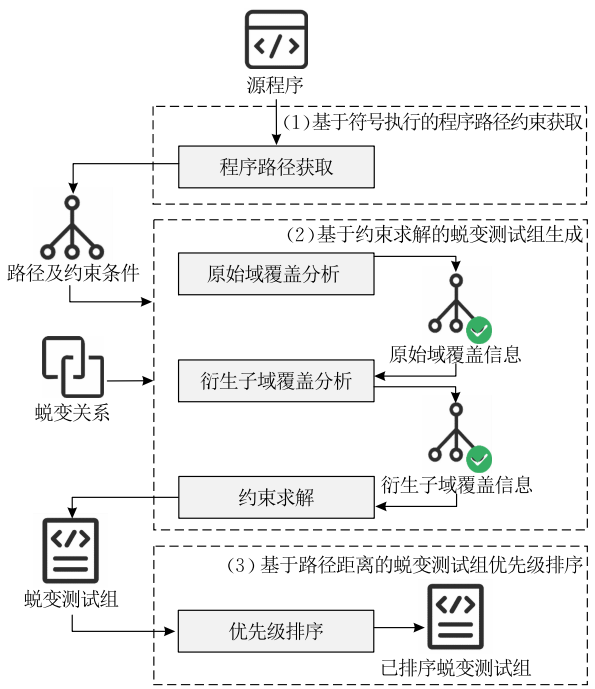


图 3　PaMTG 方法框架

3.2.1　基于符号执行的程序路径约束获取

采用符号执行获取程序的所有可执行路径信息与路径约束条件。为此首先需要创建执行期间的程序状态空间，然后获取待测程序的路径约束表达式。具体说来，(1)配置符号执行文件：符号执行需指定被符号执行的类、函数、参数列表、监听器的配置、路径探索深度等符号执行配置信息；(2)创建符号值变量：符号执行根据配置的符号执行文件信息，寻找

需要用符号值替代的参数，并使用符号值作为输入执行程序；(3)构建符号执行树：使用符号值表达式表示程序执行中的变量，通过约束求解器判定当前探索的路径可行性，删除不可行路径的分支并记录可行路径的约束条件；(4)获取路径约束条件表达式：遍历符号执行树生成程序的所有可执行路径的输出约束条件。

3.2.2 基于约束求解的蜕变测试组生成

为了生成覆盖所有可行路径对的蜕变测试组，需要基于蜕变关系与程序路径约束推理原始测试用例与衍生测试用例的可行路径对。需要指出的是，待测程序的输入域不一定等同于特定蜕变关系的适用范围；换言之，蜕变关系进一步限定了测试用例的范围。为了便于讨论，我们首先定义蜕变测试组的输入域概念。

定义 1. 原始域。针对一个给定的蜕变关系 MR，满足 MR 的测试用例约束条件 C 的输入域称为原始域 $\mathbb{D}^s$，其中 $\mathbb{D}$ 表示待测程序的输入域。

定义 2. 原始子域。原始域 $\mathbb{D}^s$ 与路径约束条件 PC 的非空子集称为原始子域 $\mathbb{D}^s_{sub}$。

定义 3. 衍生域。对于一个给定的蜕变关系 MR，依据输入关系 R 由原始子域 $\mathbb{D}^s_{sub}$ 推导出的满足 MR 测试用例约束条件 C 的非空子集称为衍生域 $\mathbb{D}^f$。

定义 4. 衍生子域。衍生域 $\mathbb{D}^f$ 与路径约束条件 PC 的非空子集称为衍生子域 $\mathbb{D}^f_{sub}$。

通过程序路径约束条件 PCs 和人工获取的蜕变关系 MR 约束条件 C 及输入关系 R 约束求解。在上述定义的基础上，本文进一步提出基于约束求解的蜕变测试组生成算法 GenMTG(算法 1)，蜕变测试组的推导过程如图 4 所示，主要步骤如下：

算法 1. GenMTG

输入：$PCs, C, R, \mathbb{D}$

输出：$MTGs$

1. Initialize $MTGs$ to an empty set
2. $\mathbb{D}^s = C \cap \mathbb{D}$
3. FOR each PCs_i in PCs DO
4. IF PCs_i and $\mathbb{D}^s$ can solve THEN
5. $\mathbb{D}^s_{sub} = PCs_i \cap \mathbb{D}^s$
6. $\mathbb{D}^f = R(\mathbb{D}^s_{sub}) \cap C$
7. FOR each PCs_j in PCs DO
8. IF PCs_j and $\mathbb{D}^f$ can solve THEN
9. $\mathbb{D}^f_{sub} = PCs_j \cap \mathbb{D}^f$
10. $TC_f = \text{Z3}(\mathbb{D}^f_{sub})$
11. $TC_s = R^{-1}(TC_f)$
12. $MTG = \langle TC_s, TC_f, set(TC_s), set(TC_f) \rangle$
13. $MTGs = MTGs \cup \{MTG\}$
14. END IF
15. END FOR
16. END IF
17. END FOR
18. RETURN $MTGs$

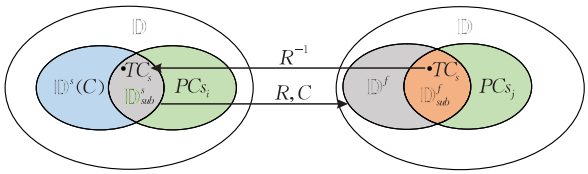


图 4 蜕变测试组推导原理示意图

(1)根据蜕变关系 MR 的测试用例约束条件 C 获取原始域 $\mathbb{D}^s$，根据每条路径约束条件 PCs_i，在原始域 $\mathbb{D}^s$ 推导出每条路径对应的子域，即原始子域 $\mathbb{D}^s_{sub}$(算法 1 第 2～5 行)。

(2)依据输入关系 R 推导出原始子域 $\mathbb{D}^s_{sub}$ 对应的衍生域 $\mathbb{D}^f$(算法 1 第 6 行)。

(3)根据衍生域 $\mathbb{D}^f$ 与每条路径约束条件 PCs_j，在衍生域 $\mathbb{D}^f$ 推导某条路径对应的子域，即衍生子域 $\mathbb{D}^f_{sub}$(算法 1 第 7～9 行)。

(4)通过对衍生子域 $\mathbb{D}^f_{sub}$ 对应的约束条件求解，得出一个能够满足 $\mathbb{D}^f_{sub}$ 的可行解，即衍生测试用例 TC_f(算法 1 第 10 行)。

(5)依据输入关系 R 对衍生测试用例 TC_f 逆推理，得到原始测试用例 TC_s，即生成一个蜕变测试组 $\langle TC_s, TC_f \rangle$。并将原始测试用例与衍生测试用例及其执行的语句集合存储到 MTG 的集合 $MTGs$ 中(算法 1 第 11～13 行)。

(6)重复执行第 4～16 行，直到原始域 $\mathbb{D}^s$ 与所有路径的约束条件都遍历一遍。

算法 1 第 10 行使用约束求解器 Z3 对衍生子域 $\mathbb{D}^f_{sub}$ 进行约束求解。使用 Z3 约束求解器的具体流程如下：

(1)创建约束求解器：Z3 提供创建问题模型的 API，创建约束求解类 Solver 的对象并利用该类的各种方法实现约束求解功能。

(2)增加约束：根据约束条件表达式中变量的类型创建相应的约束变量，并按照约束依次对变量进行约束操作，并将其添加到求解器中。

(3)判定是否有解：判断是否存在至少一个可行解满足约束求解器包含的约束条件。

(4)求解：问题模型进行求解需要让 Solver 对象读取约束模型 Model，根据约束模型 Model 中的

变量类型和约束条件获得一个满足约束的可行解。

3.2.3　基于路径距离的蜕变测试组优先级排序

为了进一步提高蜕变测试组的故障检测效率，本文提出了基于路径距离的蜕变测试组优先级排序方法。首先给出路径距离的相关定义。

定义 5.　测试用例的执行语句集合：给定一个程序 $P=\{s_1,s_2,\cdots,s_n\}$，$s_i(i=1,2,\cdots,n)$表示语句，则测试用例 tc 的执行语句集合用 $set(P,tc)$表示，其计算公式为

$$set(P,tc)=\{s_j \mid s_j\in P\wedge P(tc)\rightarrow s_j\} \quad (2)$$

定义 6.　测试用例的路径距离：给定程序 P 的两个测试用例 tc 和 tc'的执行语句集合分别为 $set(P,tc)$和 $set(P,tc')$，测试用例的路径距离 $dis(P,tc,tc')$表示测试用例 tc 和 tc'覆盖的不同语句数量，其计算公式为

$$dis(P,tc,tc')=|\{s_j \mid s_j\in set(P,tc)\bigcup set(P,tc')\wedge s_j\notin set(P,tc)\bigcap set(P,tc')\}| \quad (3)$$

图 5 示例了测试用例的路径距离的计算过程。测试用例 tc 和 tc'的执行语句集合分别为 $set(tc)=\{s_1,s_2,s_5\}$和 $set(tc')=\{s_1,s_3,s_5\}$，集合的对称差集为$\{s_2,s_3\}$，则测试用例tc 和tc'的路径距离为 $dis(tc,tc')=|\{s_2,s_3\}|=2$。

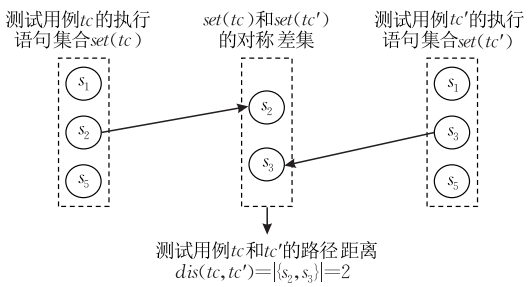


图 5　测试用例的路径距离计算示例

基于路径距离的蜕变测试组优先级排序方法分为两阶段：第一阶段对具有不同组内距离的蜕变测试组赋予一个一阶优先级；第二阶段对具有相同组内距离的蜕变测试组赋予一个二阶优先级。显然，优先执行一阶优先级高的蜕变测试组；如果一阶优先级相同，则优先执行二阶优先级高的蜕变测试组。本文提出了 PriMTG 算法，具体流程如下：

算法 2.　PriMTG

输入：$MTGs$，$Paths$

输出：TCs

1. Initialize MTG_o，MTG_s，$SMTGs$ to an empty set
2. FOR each $MTGs_i=\langle TC_s,TC_f,set(TC_s),set(TC_f)\rangle$ in $MTGs$ DO
3. $d=dis(TC_s,TC_f)$
4. $SMTG=\langle d,MTGs_i\rangle$
5. $MTG_o=MTG_o\cup\{SMTG\}$
6. END FOR
7. WHILE MTG_o is not NULL DO
8. Initialize $SMTG$=NULL
9. IF $getMaxDisNumber$ in MTG_o is 1 THEN
10. $SMTG=getMaxDis(MTG_o)$
11. ELSE
12. Initialize max=0
13. $SMTGs=getMaxDis(MTG_o)$
14. $SMTG_{last}=getLast(MTG_s)=\langle d^{last},TC_s^{last},TC_f^{last},set(TC_s^{last}),set(TC_f^{last})\rangle$
15. FOR each $SMTGs_i=\langle d,TC_s,TC_f,set(TC_s),set(TC_f)\rangle$ in $SMTGs$ DO
16. $d=dis(TC_s,TC_s^{last})$
17. IF $d>max$ THEN
18. $SMTG=SMTGs_i$
19. $max=d$
20. END IF
21. END FOR
22. END IF
23. $MTG_s=MTG_s\cup\{SMTG\}$
24. $MTG_o=MTG_o\backslash\{SMTG\}$
25. END WHILE
26. RETURN MTG_s

(1) 计算每个蜕变测试组 $MTGs$ 的组内距离，并将蜕变测试组的路径距离、测试用例及其执行语句集合存储到集合 MTG_o(算法 2 第 2～6 行)。

(2) 从蜕变测试组 MTG_o 中选取组内距离最大的蜕变测试组 $SMTG$，放入已排序的蜕变测试组 MTG_s 中。若存在多个组内距离最大的蜕变测试组 $SMTGs$ 且已排序的蜕变测试组 MTG_s 为空时，则选取其中任意一个加入已排序的蜕变测试组 MTG_s 中；若存在多个组内距离最大的蜕变测试组 $SMTGs$ 且已排序的蜕变测试组 MTG_s 不为空时，则优先选取与上一对刚加入 MTG_s 的蜕变测试组 $SMTG_{last}$ 的原始测试用例执行路径距离最大的蜕变测试组(算法 2 第 8～22 行)。

(3) 重复执行第 8～22 行，直到无序测试用例集 MTG_o 为空，输出按路径距离优先级排序的蜕变测试组 MTG_s。

3.3　方法示例

以 3.1 节讨论的 $myMethod(x,y,z)$函数和蜕变关系 MR 为例，展示本文方法的应用过程。

(1)符号执行:该例子中,符号执行使用符号值〈X,Y,Z〉作为程序的输入,在 JPF 提供的虚拟机上搜索得到如图 2(c)所示的符号执行树,遍历符号执行树即可获得程序的所有可执行路径($PC_1 \sim PC_5$)的信息和路径约束。

(2)约束求解生成蜕变测试组:不难看出,该 *MR* 的测试用例取值范围应满足约束条件"$Z \geq 0$"。通过上述步骤后,将每条路径的约束条件与原始域进行交集,推导出这些路径的原始子域。依据该 *MR* 的输入关系($Y'=Z, Z'=Y$)以及约束条件"$Z' \geq 0$",由某个路径的原始子域推导出其对应的衍生域(见表 1)。将每条路径的衍生域与所有路径的约束条件匹配,获得可行的衍生子域。

以原始测试用例中路径 PC_3 为例,通过将其对应的衍生域"$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Z' \wedge Z' > Y'$"与路径约束进行交集,发现路径 PC_2 和 PC_4 均适用,并推导出相应的衍生子域(见表 2)。通过遍历原始路径,推导出共 6 个适用的衍生子域,使用约束求解器获得衍生测试用例,并逆推导出原始测试用例,获得 6 个蜕变测试组(见表 3)。

表 1 示例程序的原始子域与衍生域推导过程

原始路径	路径约束条件	原始域/蜕变关系约束条件	是否有解	原始子域约束条件	衍生域约束条件
PC_1	$Z \geq 0 \wedge X > Y \wedge X > Z$	$Z \geq 0$	true	$Z \geq 0 \wedge X > Y \wedge X > Z$	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' > Z' \wedge X' > Y'$
PC_2	$Z \geq 0 \wedge X > Y \wedge X \leq Z$	$Z \geq 0$	true	$Z \geq 0 \wedge X > Y \wedge X \leq Z$	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' > Z' \wedge X' \leq Y'$
PC_3	$Z \geq 0 \wedge X \leq Y \wedge Y > Z$	$Z \geq 0$	true	$Z \geq 0 \wedge X \leq Y \wedge Y > Z$	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Z' \wedge Z' > Y'$
PC_4	$Z \geq 0 \wedge X \leq Y \wedge Y \leq Z$	$Z \geq 0$	true	$Z \geq 0 \wedge X \leq Y \wedge Y \leq Z$	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Z' \wedge Z' \leq Y'$
PC_5	$Z < 0$	$Z \geq 0$	false	N/A	N/A

表 2 示例程序原始路径 PC_3 的衍生子域推导过程

衍生路径	衍生域约束条件	路径约束条件	是否有解	衍生子域约束条件
PC_1		$Z' \geq 0 \wedge X' > Y' \wedge X' > Z'$	false	N/A
PC_2	$Z' \geq 0 \wedge Y' \geq 0$	$Z' \geq 0 \wedge X' > Y' \wedge X' \leq Z'$	true	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' > Y' \wedge X' \leq Z' \wedge X' \leq Z' \wedge Z' > Y'$
PC_3	$\wedge$	$Z' \geq 0 \wedge X' \leq Y' \wedge Y' > Z'$	false	N/A
PC_4	$X' \leq Z' \wedge Z' > Y'$	$Z' \geq 0 \wedge X' \leq Y' \wedge Y' \leq Z'$	true	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Y' \wedge Y' \leq Z' \wedge X' \leq Z' \wedge Z' > Y'$
PC_5		$Z' < 0$	false	N/A

表 3 示例程序的蜕变测试组及优先级排序

序号	衍生子域约束条件	衍生测试用例	原始测试用例	衍生路径	原始路径	路径距离	优先级
1	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' > Y' \wedge X' > Z' \wedge X' > Z' \wedge X' > Y'$	$X'=1, Y'=0, Z'=0$	$X=1, Y=0, Z=0$	PC_1	PC_1	0	5
2	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Y' \wedge Y' > Z' \wedge X' > Z' \wedge X' \leq Y'$	$X'=1, Y'=1, Z'=0$	$X=1, Y=0, Z=1$	PC_3	PC_2	4	1
3	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' > Y' \wedge X' \leq Z' \wedge X' \leq Z' \wedge Z' > Y'$	$X'=1, Y'=0, Z'=1$	$X=1, Y=1, Z=0$	PC_2	PC_3	4	2
4	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Y' \wedge Y' \leq Z' \wedge X' \leq Z' \wedge Z' > Y'$	$X'=0, Y'=0, Z'=1$	$X=0, Y=1, Z=0$	PC_4	PC_3	2	4
5	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Y' \wedge Y' > Z' \wedge X' \leq Z' \wedge Z' \leq Y'$	$X'=0, Y'=1, Z'=0$	$X=0, Y=0, Z=1$	PC_3	PC_4	2	3
6	$Z' \geq 0 \wedge Y' \geq 0 \wedge X' \leq Y' \wedge Y' \leq Z' \wedge X' \leq Z' \wedge Z' \leq Y'$	$X'=1, Y'=1, Z'=1$	$X=1, Y=1, Z=1$	PC_4	PC_4	0	6

(3)蜕变测试组优先级排序:计算每个蜕变测试组中的原始测试用例和衍生测试用例的路径距离并进行优先级排序(见表 3)。首先选取组内路径距离最大的蜕变测试组,如果存在路径距离相同的多个蜕变测试组,则选择与上一个已排序的蜕变测试组之间原始测试用例路径距离最大的 1 组。本例中,第 2、3 组的路径距离均为 4,假设首先随机选取了第 2 组,将其优先级设为 1,则第 3 组的优先级为 2;其次,路径距离为 2 的蜕变测试组有两组(第 4、5 组),计算与上一组(第 3 组)的原始测试用例路径距离,第 4 组与第 3 组的原始路径均为 PC_3,因此选取了第 5 组,相应地其优先级设为 3,而第 4 组的优先级则设为 4;最后,路径距离为 0 的蜕变测试组也有两组(第 1、6 组),计算与上一组(第 4 组)的原始测试用例路径距离,第 1、4 组之间路径距离 $dis(tc_1, tc_4) = dis(PC_1, PC_3) = 4$,第 4、6 组之间路径距离 $dis(tc_6, tc_4) = dis(PC_4, PC_3) = 2$,因此第 1 组优先级设为 5,第 6 组优先级设为 6。依据上述过程将优先级从小到大进行排序,从而生成优先级排序的蜕变测试组。

4 支持工具

使用 Java 语言开发了支持工具 MTG-GEN。主要的功能特色包括:(1)分析用户上传的待测程序,生成符号执行配置文件,支持符号执行;(2)根据用户指明的蜕变关系的输入关系和输入域等信息,生成有序蜕变测试组;(3)展示蜕变测试组覆盖的路径信息。

图 6 示意了 MTG-GEN 的系统架构。(1) 路径生成：使用符号路径查找器 SPF 遍历待测程序中所有可行路径，分析可行路径的代码覆盖情况，获得待测程序的路径信息；(2) 约束求解：调用蜕变关系和路径信息提取路径约束，分析所有可行路径对的蜕变测试组；使用 Z3 约束求解器对约束条件进行约束求解，依据蜕变关系的输入关系进行推导操作，生成蜕变测试组；(3) 优先级排序：计算蜕变测试组的原始测试用例和衍生测试用例在执行路径上的距离，进行两阶段的优先级排序。

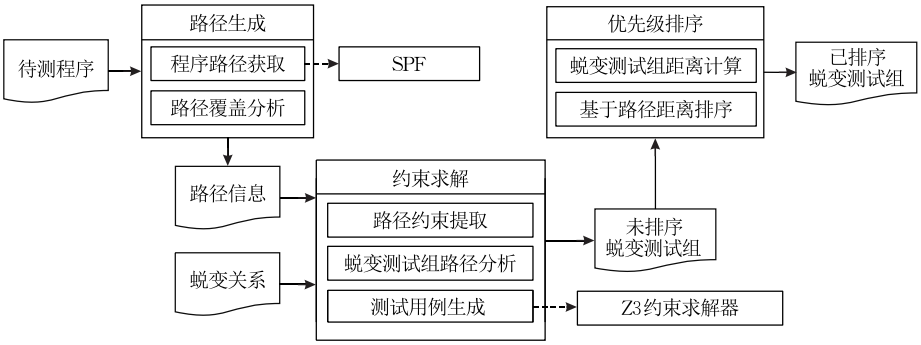


图 6　MTG-GEN 结构图

5　实验设置

本节介绍实验设置，包括基准技术、研究问题、实验对象、度量指标、实验步骤等。

5.1　基准技术

为了验证 PaMTG 的有效性，本文选取如下测试用例生成方法：

(1) 随机方法(RT)[14]：根据待测程序的输入规格随机生成原始测试用例。

(2) 适应性随机方法(ART)[19]：将待测程序的输入域划分为多个互斥区域，每次随机生成多个测试用例，计算所有测试用例与上一个原始测试用例的距离，选取距离最大的作为下一个原始测试用例。实验中，ART的候选原始测试用例集的大小设为 10。

(3) 基于符号执行的原始测试用例生成方法(PaDMT)[21]：针对待测程序进行符号执行，获得所有可行路径的约束条件，通过约束求解技术生成原始测试用例。

(4) 面向路径对覆盖准则的蜕变测试用例生成方法(APPCEMST)[17]：首先使用符号执行技术获取每条可执行路径的约束条件，然后针对蜕变关系所有适用的路径对生成相应的测试用例。由于 APPCEMST 方法的具体实现没有讨论，本文采用符号执行工具 SPF 获得可执行路径的约束条件，采用随机生成方法与约束求解器 Z3 实现测试用例生成，对应的实现分别记为 APPCEMST-R 和 APPCEMST-Z3。其中，APPCEMST-Z3 等同于不包含优先级排序的 PaMTG 方法(即 GenMTG 算法的单独实现，且生成的蜕变测试组按照随机的顺序执行)。

5.2　研究问题

我们采用经验研究的方式，围绕以下四个问题展开实验评估：

(1) RQ1：本文所提方法能否避免无效测试用例的生成?本文所提方法生成的蜕变测试组的路径对覆盖情况如何?

使用 PaMTG 与基准方法的测试用例有效率进行比较，评估所生成的测试用例的有效性；比较 PaMTG 和 PaDMT、APPCEMST-R、APPCEMST-Z3 生成的蜕变测试组的路径对覆盖数量。

(2) RQ2：本文所提方法生成蜕变测试组的故障检测能力如何?

比较 PaMTG 与基准方法生成的测试用例集的故障检测能力。

(3) RQ3：本文所提方法生成的蜕变测试组的故障检测效率如何?

从测试用例序列中选取前 10%至 100%的测试用例进行蜕变测试，比较 PaMTG 与 APPCEMST-Z3、APPCEMST-R、ART、RT 基准技术生成的有序测试用例集的故障检测效率。由于 PaDMT 无法生成与 PaMTG 数量相同的测试用例集，不宜进行故障检测效率的比较。

(4) RQ4：本文所提方法的时间开销大小?

计算 PaMTG 的时间开销(包括测试用例生成时间和测试执行时间)，并与基准方法的时间开销进行比较。

5.3 实验对象

为回答上述研究问题，我们选取 7 个来自不同应用领域的程序作为实验对象。表 4 总结了实验对象的代码行数、蜕变关系数以及生成的变异体数和变异体类型。具体说来，实验对象简要说明如下：

(1) LUGGAGE[46]：航空行李托运计费程序，参照航空公司行李计费标准，根据用户的座舱等级、飞机票价、携带行李总重等信息计算用户需要交纳的行李托运费用；

(2) PHONE[46]：话费计费程序，根据用户的话费套餐价格、通话时间、使用流量等数据计算用户的通信费用；

(3) PARKING[47]：停车计费程序，以停车场的计费方式为原型，从车的种类、停车时间等信息计算停车费用；

(4) CHARGE[21]：转账手续费程序，模拟账号资金向银行卡转账的操作，根据用户的个人账号信息，如用户转账免费额度、转账金额、转账方式等信息计算转账方需要提供的手续费用；

(5) MATH[48]：数学函数程序，计算两个输入参数的最大公约数；

(6) TAX[21]：税收程序，根据顾客购买产品的信息，如产品价格、数量、当地税率、是否属于进口产品等信息计算缴纳的税款。

(7) NUMBER[48]：数据类型转换程序，以 Apache Common Lang 库中的数据类型转换程序为原型，根据前缀类型、后缀类型、数字、是否有小数点等信息计算数据的类型和数值。

表 4 实验对象信息描述

实验对象	代码行数	MR	变异体	变异体类型
LUGGAGE	101	36	56	MuJava 中的变异算子 AOIS, AORB, COI, LOI, ROR, AOIU。
PHONE	113	36	112	MuJava 中的变异算子 AOIS, AOIU, AORB, COI, LOI, ROR。
PARKING	266	4	754	MuJava 中的变异算子 AODU, AOIS, AORB, COD, COI, COR, LOI, ROR。
CHARGE	1008	20	847	Mu2ava 中的变异算子 AODU, AOIS, AORB, AOIU, CDL, COD, COI, COR, LOI, ODL, ROR, SDL, VDL。
MATH	2002	2	435	MuJava 中的变异算子 AOIS, AORB, AOIU, CDL, COI, LOI, ODL, ROR, SDL, VDL。
TAX	2150	24	1565	MuJava 中的变异算子 AODU, AOIS, AORB, AOIU, CDL, COR, COI, LOI, ODL, ROR。
NUMBER	1843	24	1336	MuJava 中的变异算子 AODU, AOIS, AOIU, AORB, AORS, ASRS, CDL, COI, COR, LOI, ODL, ROR, SDL, VDL。

5.4 度量指标

使用以下度量指标评估所提方法的有效性、故障检测能力、故障检测效率和时间开销。

(1) 测试用例有效率(Valid Test Case Ratio, *VTCR*)：属于蜕变关系 *MR* 适用范围的测试用例数量占所有生成测试用例总数量的百分比。其计算公式如下：

$$VTCR(MR,TC)=\frac{TC_e}{TC_a}\times 100\% \tag{4}$$

其中，TC_e为有效测试用例数量，TC_a为生成测试用例总数。

(2) 变异得分(*MS*)[17,21]：一个测试用例集 *TC* 能够检测到的程序 *P* 的变异体数占所有非等价变异体数的百分比。变异得分反映蜕变测试组(或测试用例集)的故障检测能力。变异得分越高，则表明测试用例集的故障检测能力越强。

$$MS(P,TC)=\frac{N_k}{N_m-N_e}\times 100\% \tag{5}$$

其中，N_k为被杀死的变异体数，N_m为变异体总数，N_e为等价变异体数。

(3) 故障检测率(*FDR*)[21]：该指标度量按一定顺序执行测试用例过程中检测到的平均累计故障百分比，量化测试用例序列的故障检测效率。以 10% 为步进，从测试用例序列中选取前 10% 至 100% 的测试用例计算 *FDR* 值。*FDR* 越大，则表明测试用例集的检测错误速度越快。*FDR* 计算公式如下：

$$FDR(TS,P)=\frac{D}{N_m-N_e}\times 100\% \tag{6}$$

其中，*TS* 为具有顺序的测试用例集，*P* 为待测程序，*D* 为当前测试用例集能够检测出的故障数量，N_m为变异体总数，N_e为等价变异体数。

(4) 测试用例生成时间开销(*T*)[21]：即测试用例生成所消耗的时间 *T*。

$$T=T_s-T_e \tag{7}$$

其中，T_s和T_e分别表示测试用例生成开始和结束时刻。

5.5 实验步骤

本节详细介绍对技术评估的实验步骤，具体过程如下：

(1) 获取蜕变关系：分析待测程序的蜕变属性，

采用基于数据变异的蜕变关系获取方法[46]和人工方式获取实验对象的蜕变关系，人工提取原始测试用例约束条件。

(2) 获取实验对象的故障版本：采用 MuJava[49]生成变异体，选择方法级别变异算子；使用人工方式识别并剔除等价变异体。

(3) 生成测试用例集：PaMTG 首先通过符号执行获取待测程序的路径信息和路径约束，然后使用 GenMTG 算法生成蜕变测试组，最后使用 PriMTG 算法对蜕变测试组进行优先级排序。PaDMT 与 APPCEMST 均使用符号执行工具 SPF 生成测试用例。ART 和 RT 均生成与 PaMTG 相同规模的测试用例集。

(4) 执行测试：将步骤(3)生成的蜕变测试组分别执行步骤(2)生成的非等价变异体。执行测试过程使用自动化测试，判断两类测试用例的实际输出结果是否符合蜕变关系。

(5) 结果分析：设计对照实验，分析与回答本文所提的四个研究问题。为了克服随机方式存在的偶然性，本文所有实验均重复 30 次。为了进一步衡量平均值与多次实验结果之间的差异情况，我们对多次实验结果进行了 Wilcoxon 符号秩检验[50]，计算统计假设检验 P 值。如果 $P>0.05$，则表示差异很小；否则，表示差异较大。

6　结果分析

6.1　蜕变测试组的有效性(RQ1)

图 7 和图 8 分别总结了 PaMTG 与基准方法生成的测试用例的有效率和可行路径对覆盖结果。表 5 进一步给出了统计评估结果，其中 AV、SD 和 P 分别表示测试用例有效率的平均值、标准差与符号秩检验 P 值。从图中可以看出，(1) 7 个实验程序上，PaMTG、APPCEMST-Z3 和 APPCEMST-R 的测试用例有效率始终为 100%，覆盖可行路径对数量高于 PaDMT 方法。主要原因是：① PaMTG、APPCEMST-Z3 和 APPCEMST-R 在生成测试用例时考虑了蜕变关系的作用域，因此测试用例均有效；② PaMTG、APPCEMST-Z3 和 APPCEMST-R 考虑了原始测试用例和衍生测试用例的执行路径差异，可行路径对覆盖更全面；(2) LUGGAGE、PHONE、PARKING、NUMBER 等 4 个实验程序上，PaMTG、APPCEMST-Z3 和 APPCEMST-R 生成测试用例有效率明显高于其他方法的有效率；(3) TAX 程序上，PaMTG 和其他方法生成的测试用例有效率相当，均为 100%。进一步分析发现，TAX 程序的输入域与蜕变关系的原始域相同，因此不会出现无效的测试用例。

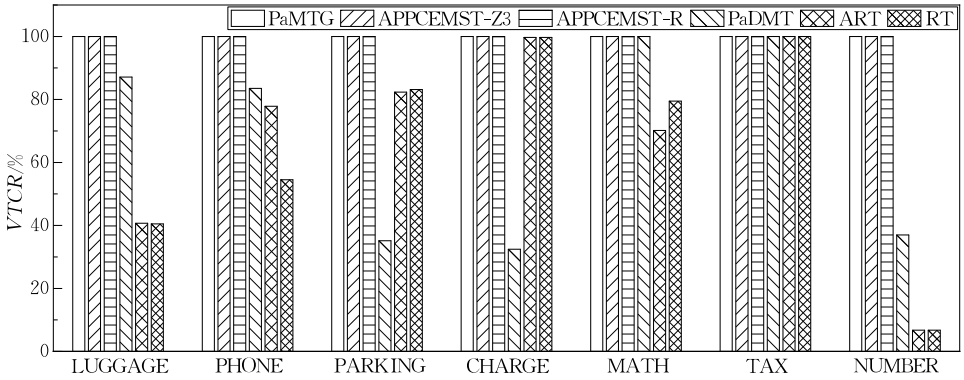


图 7　测试用例有效率评估结果

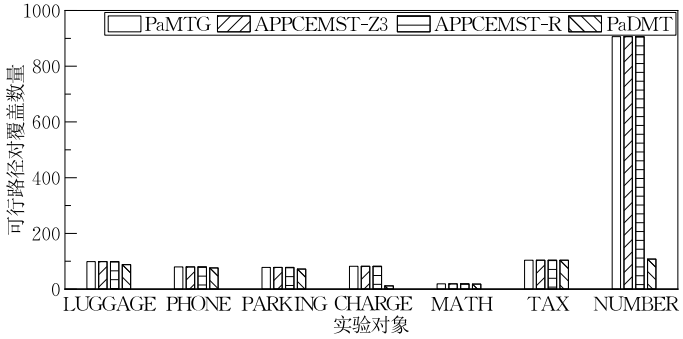


图 8　可行路径对覆盖数量

表 5 测试用例有效率评估结果 （单位：%）

测试方法	LUGGAGE			PHONE			PARKING			CHARGE			MATH			TAX			NUMBER		
	AV	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100	0	1.00	100.00	0.00	1.00
APPCEMST-Z3	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100	0	1.00	100.00	0.00	1.00
APPCEMST-R	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00	100	0	1.00	100.00	0.00	1.00
PaDMT	87.13	0.00	1.00	83.52	0.00	1.00	35.12	0.00	1.00	32.43	0.00	1.00	100.00	0.00	1.00	100	0	1.00	36.99	0.00	1.00
ART	40.71	7.77	0.79	77.83	5.29	0.71	82.35	5.59	0.97	99.72	0.52	0.35	70.18	20.47	0.61	100	0	1.00	6.73	1.39	1.00
RT	40.51	9.50	0.89	54.50	8.31	0.68	83.16	4.05	0.68	99.72	0.61	0.14	79.47	7.49	0.73	100	0	1.00	6.72	1.46	0.67

6.2 故障检测能力（RQ2）

图 9 总结了 PaMTG 与基准方法生成蜕变测试组的变异得分情况，表 6 进一步给出了统计评估结果，其中 *AV*、*SD* 和 *P* 分别表示变异得分的平均值、标准差与符号秩检验 *P* 值。不难看出：(1) PaMTG 生成的测试用例集的变异得分高于已有基准方法。进一步分析发现，PaMTG 生成原始测试用例和衍生测试用例能够覆盖蜕变关系的所有路径对，因而具有更高的变异得分；PaMTG 和 APPCEMST-Z3 均使用约束求解器 Z3，因此生成的测试用例集相同，变异得分相当；(2) PaMTG 生成的测试用例集的变异得分比 APPCEMST-R 更稳定。在 LUGGAGE、PHONE、NUMBER 实验中，PaMTG 与 APPCEMST-R 在重复实验中的变异得分均没有变化；在 PARKING、CHARGE、MATH、TAX 实验中，PaMTG 在重复实验中的变异得分始终一致。主要原因是：约束求解器 Z3 对同一约束条件的执行结果相同，因此 PaMTG 每次生成的测试用例集相同，变异得分不变；(3) 对于 TAX 程序而言，PaMTG 和 PaDMT 生成的测试用例变异得分相当。主要原因是：适用于 TAX 程序的每个蜕变关系的路径对唯一，即每个原始域对应的路径的衍生子域覆盖路径唯一，PaMTG 与 PaDMT 生成的蜕变测试用例没有区别，因而获得了相同的变异得分。

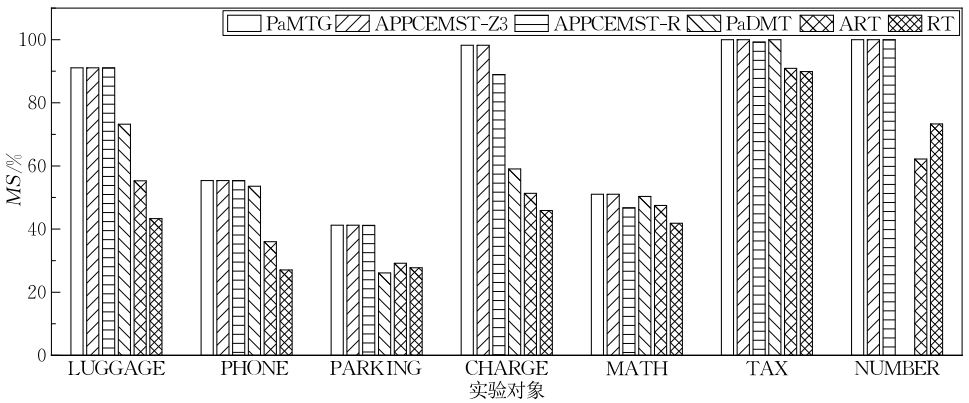


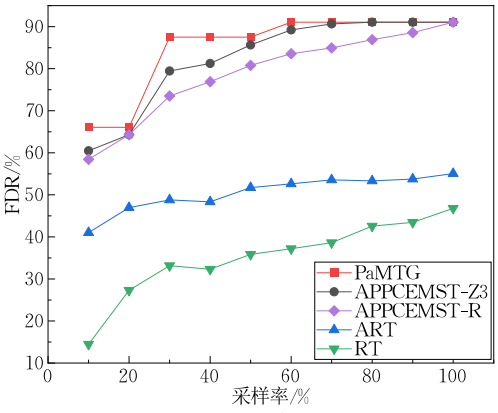
图 9 PaMTG 及基准方法生成的蜕变测试组的变异得分

表 6 变异得分评估结果 （单位：%）

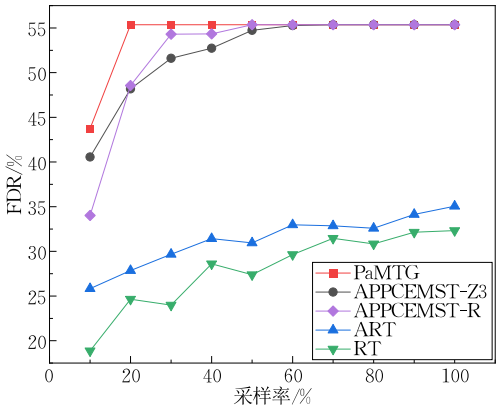
测试方法	LUGGAGE			PHONE			PARKING			CHARGE			MATH			TAX			NUMBER		
	AV	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	91.07	0.00	1.00	55.36	0.00	1.00	41.25	0.00	1.00	98.23	0.00	1.00	51.03	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00
APPCEMST-Z3	91.07	0.00	1.00	55.36	0.00	1.00	41.25	0.00	1.00	98.23	0.00	1.00	51.03	0.00	1.00	100.00	0.00	1.00	100.00	0.00	1.00
APPCEMST-R	91.07	0.00	1.00	55.36	0.00	1.00	41.18	0.43	0.59	88.96	3.79	1.00	46.77	4.64	0.30	99.27	0.09	0.21	100.00	0.00	1.00
PaDMT	73.21	0.00	1.00	53.57	0.00	1.00	26.13	0.00	1.00	59.03	0.00	1.00	50.34	0.00	1.00	100.00	0.00	1.00	0.00	0.00	1.00
ART	55.30	8.43	0.73	36.01	4.63	0.69	29.17	1.25	0.90	51.33	6.43	0.52	47.48	9.81	0.26	90.90	0.05	0.14	62.20	35.16	0.89
RT	43.33	12.40	0.82	27.05	3.95	0.07	27.76	1.58	0.98	45.89	5.64	0.87	41.85	5.72	0.81	89.89	0.87	0.43	73.31	32.54	0.65

6.3 故障检测效率（RQ3）

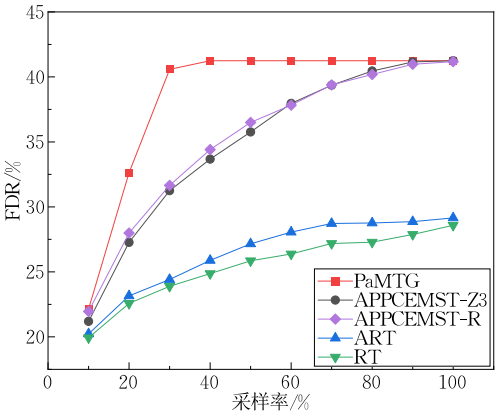
图 10 比较了 PaMTG 与基准方法生成的测试用例集的故障检测效率。可以看出：(1) 总体说来，PaMTG 生成的测试用例集故障检测效率明显优于 APPCEMST-Z3、APPCEMST-R、ART 和 RT；且随着选取比例的增加，PaMTG 的故障检测率提高更快；(2) PriMTG 能够有效提高蜕变测试的故障检测效率。与 APPCEMST-Z3 相比，包含优先级排序方法的 PaMTG 具有更好的故障检测效率。特别地，在 PARKING 实验中，每条蜕变关系覆盖的路径对数目



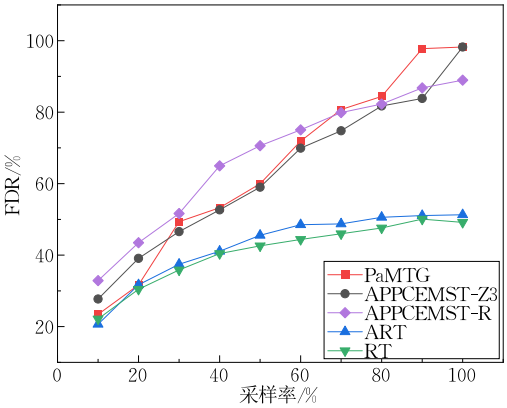
(a) LUGGAGE



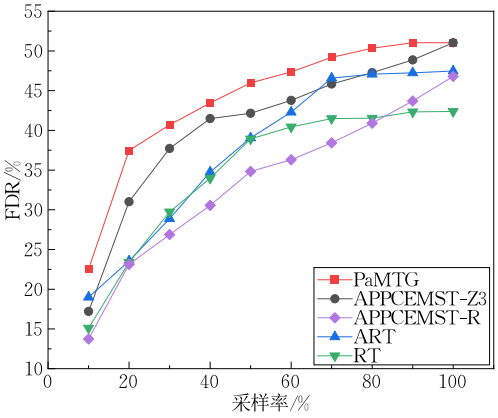
(b) PHONE



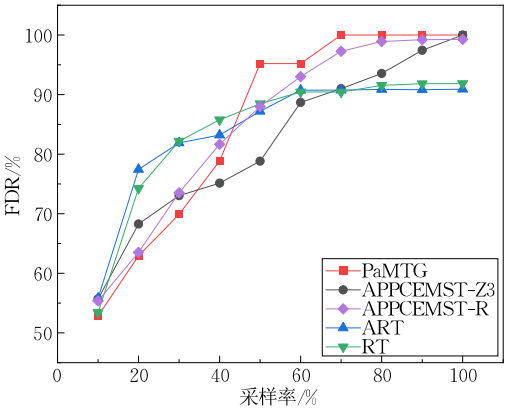
(c) PARKING



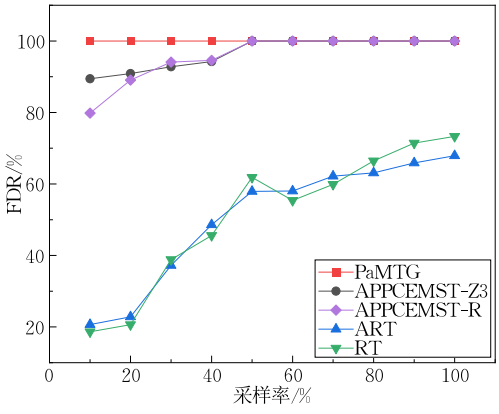
(d) CHARGE



(e) MATH



(f) TAX



(g) NUMBER

图 10　故障检测效率评估结果

较多且蜕变测试组的组内距离差异大，PaMTG 优势明显；(3)在不同实验对象、不同选取比例的实验中，PaMTG 生成的测试用例集故障检测效率的优势有所区别。在 LUGGAGE、PHONE、PARKING、MATH、NUMBER 实验中，PaMTG 的故障检测效率在所有的选取比例下均优于所有的基准技术；在 CHARGE 实验中，PaMTG 的故障检测效率与 APPCEMST-Z3、APPCEMST-R 基本相当，但优于 ART 和 RT；在 TAX 实验中，选取比例低于 40% 时，PaMTG 的故障检测率低于其他基准方法。进一步分析发现：在 TAX 实验对象的每条程序路径中的原始子域与衍生子域覆盖的路径唯一且相同，因而蜕变测试组的原始域和衍生域之间的路径差异均为零，无法展示基于蜕变测试组组内距离的优先级排序的效果。

6.4 时间开销(RQ4)

表 7 总结了 PaMTG 和基准方法生成测试用例与测试执行的时间开销，其中 *AV*、*SD* 和 *P* 分别表示平均值、标准差与符号秩检验 *P* 值。在 LUGGAGE、PARKING、MATH、TAX 实验中，PaMTG 生成测试用例的时间开销高于基准方法；在 PHONE、CHARGE、NUMBER 实验中，PaMTG 生成测试用例的时间开销低于 APPCEMST-R，但高于其他基准方法。由于 PaMTG 方法使用了符号执行和约束求解操作，引入大量时间开销。然而，这样的时间开销是值得的，主要理由如下：(1)RT 与 ART 方法生成测试用例的时间开销较少，但故障检测能力弱于 PaMTG；(2)RT、ART 和 PaDMT 方法需要判断生成的测试用例是否属于某个蜕变关系的适用范围，其中产生的无效测试用例将造成测试资源的浪费；(3)PaDMT 生成的测试用例数量少于 PaMTG，然而故障检测能力弱于 PaMTG；(4) APPCEMST-R 方法通过生成大量的随机测试用例，并判断生成的测试用例是否满足约束条件，以获得特定路径对的测试用例，存在测试资源浪费的问题；(5) PaMTG 与 APPCEMST-Z3 方法的时间开销基本相当，PaMTG 具有更高的故障检测效率；(6) 测试用例执行时间与测试用例数量以及程序的复杂程度相关。当测试用例数量较多或程序较为复杂时，测试执行将占据

表 7 时间开销 (单位：ms)

测试方法	LUGGAGE						PHONE					
	测试用例生成			测试执行			测试用例生成			测试执行		
	AV	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	8.12×10^{4}	1.71×10^{3}	0.85	1.44×10^{5}	1.79×10^{4}	0.55	4.30×10^{4}	1.16×10^{3}	0.76	1.73×10^{5}	4.90×10^{3}	0.98
APPCEMST-Z3	3.65×10^{4}	8.79×10^{2}	0.77	1.46×10^{5}	1.60×10^{4}	0.55	2.29×10^{4}	7.44×10^{2}	0.73	1.72×10^{5}	1.32×10^{4}	0.24
APPCEMST-R	3.98×10^{4}	1.07×10^{3}	0.68	1.39×10^{5}	2.99×10^{4}	0.61	9.39×10^{4}	1.29×10^{4}	0.98	1.68×10^{5}	3.39×10^{4}	0.68
PaDMT	6.02×10^{2}	6.21	0.07	1.27×10^{5}	4.31×10^{3}	0.70	7.08×10^{2}	6.72×10^{-1}	0.47	1.36×10^{5}	2.68×10^{4}	0.59
ART	2.15	7.73×10^{-1}	0.18	4.46×10^{4}	1.38×10^{4}	0.20	1.70	1.15	0.06	1.30×10^{5}	3.23×10^{4}	0.41
RT	1.41	9.81×10^{-1}	0.08	4.19×10^{4}	7.33×10^{3}	0.85	5.39×10^{-1}	2.79×10^{-1}	0.06	7.04×10^{4}	2.40×10^{4}	0.13

表 7 时间开销(续表 1) (单位：ms)

测试方法	PARKING						CHARGE					
	测试用例生成			测试执行			测试用例生成			测试执行		
	AV	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	1.33×10^{5}	3.15×10^{4}	0.52	2.84×10^{5}	7.40×10^{4}	0.09	2.53×10^{4}	1.17×10^{3}	0.25	4.04×10^{4}	6.41×10^{3}	0.62
APPCEMST-Z3	1.09×10^{5}	2.36×10^{4}	0.76	2.84×10^{5}	7.65×10^{4}	0.11	1.51×10^{4}	9.15×10^{2}	0.17	4.16×10^{4}	6.81×10^{3}	0.62
APPCEMST-R	1.10×10^{5}	2.36×10^{4}	0.73	3.02×10^{5}	7.95×10^{4}	0.25	1.14×10^{5}	3.87×10^{4}	0.79	4.43×10^{4}	6.95×10^{3}	0.06
PaDMT	2.04×10^{3}	2.49	0.18	1.79×10^{5}	1.90×10^{4}	0.14	6.63×10^{2}	2.22	0.18	2.15×10^{4}	3.37×10^{3}	0.08
ART	2.42	1.17	0.07	1.35×10^{5}	1.85×10^{4}	0.54	2.87	1.76	0.14	4.30×10^{4}	1.95×10^{3}	0.50
RT	2.33×10^{-1}	1.63×10^{-1}	0.11	1.29×10^{5}	1.30×10^{4}	0.07	5.15×10^{-1}	2.77×10^{-1}	0.14	4.34×10^{4}	1.96×10^{3}	0.55

表 7 时间开销(续表 2) (单位：ms)

测试方法	MATH						TAX					
	测试用例生成			测试执行			测试用例生成			测试执行		
	AV	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	1.29×10^{4}	3.17×10	0.31	2.25×10^{6}	1.04×10^{4}	0.17	5.00×10^{5}	1.10×10^{4}	0.46	1.05×10^{6}	9.16×10^{4}	0.37
APPCEMST-Z3	1.26×10^{4}	2.65×10	0.31	2.23×10^{6}	6.94×10^{3}	0.10	1.01×10^{5}	8.61×10^{3}	0.68	1.03×10^{6}	8.29×10^{4}	0.16
APPCEMST-R	1.27×10^{4}	4.88×10	0.29	2.24×10^{6}	5.62×10^{3}	1.00	1.01×10^{5}	8.72×10^{3}	0.68	1.09×10^{6}	1.12×10^{5}	0.20
PaDMT	1.14×10^{4}	3.11×10^{-1}	0.06	2.25×10^{6}	7.45×10^{2}	0.54	1.18×10^{3}	3.95×10^{-1}	0.50	1.23×10^{6}	9.77×10^{4}	0.31
ART	1.05×10^{-1}	5.30×10^{-2}	0.54	1.33×10^{6}	2.20×10^{5}	0.16	1.49	1.31	0.07	1.21×10^{6}	1.08×10^{4}	0.64
RT	5.04×10^{-2}	1.37×10^{-2}	0.84	1.55×10^{6}	1.24×10^{5}	0.92	1.10	6.39×10^{-1}	0.67	1.62×10^{6}	3.54×10^{5}	0.70

表 7　时间开销(续表 3)　　(单位：ms)

测试方法	NUMBER					
	测试用例生成			测试执行		
	AV	*SD*	*P*	*AV*	*SD*	*P*
PaMTG	5.39×10^4	1.95×10^4	0.92	4.96×10^5	6.44×10^4	0.43
APPCEMST-Z3	4.16×10^4	1.95×10^4	0.92	4.98×10^5	1.06×10^5	0.10
APPCEMST-R	2.92×10^5	4.83×10^4	0.79	5.48×10^5	1.42×10^5	0.16
PaDMT	1.57×10^3	1.43	0.17	8.80×10^4	5.19×10^3	0.84
ART	1.60×10	6.53	0.14	1.40×10^5	2.84×10^4	0.77
RT	5.14	1.58	0.29	7.74×10^4	1.20×10^4	0.40

大部分时间开销。

6.5　有效性威胁分析

(1)实验对象的选取：实验对象的数量与类型可能影响实验结果的可靠性。理论上讲，采用更多、更大规模的真实实验对象进行实验评估，结果更可靠。但由于真实的工业界软件的源代码包含敏感的商业秘密，通常不会分享给研究人员；受限于实验资源与时间，本文尽量复用已有研究的实验对象，并且选择了不同领域、不同规模的实验程序，以减少对实验结果有效性的影响；与本领域的相关工作相比，本文的实验对象数量较多且规模较大。此外，虽然本文实验不涉及数百万行数的大型程序，但是本文方法可以推广到更大规模的程序。具体说来，首先将大规模程序划分为多个可以独立测试的子系统；对于每个子系统，然后分析相应的蜕变关系；最后采用本文方法生成每个子系统的蜕变测试组。

(2)实验对象的故障版本设置：本文使用变异分析方法评估所提技术的故障检测能力，变异体的数量与类型可能影响实验结果。一方面，本文采用变异分析工具 MuJava 生成变异体，由于该工具在多个相关研究中得到广泛应用[21,51-52]，其生成的变异体的正确性是受控的；另一方面，我们使用了工具支持的所有适用的变异算子，以生成多种类型的变异体。已有研究表明，这些变异算子生成的变异体能够模拟很多真实情况下常见的软件故障[53-54]。

(3)实验结果随机性：RT、ART、APPCEMST-R 等基准技术均涉及随机方法，实验结果具有随机性。本文将相关实验重复 30 次并计算平均值，以保证实验结果的可靠性。

7　相关工作

如何生成有效且充分的测试用例集是蜕变测试领域需要解决的关键问题之一。介绍已有的蜕变测试用例生成方法[55]，并与本文方法进行比较。

Chen 等人[56]比较了蜕变测试与特殊值测试方法，发现蜕变测试能检测出特殊值测试方法检测不到的错误。若待测程序中存在测试预期问题，则无法使用特殊值测试方法。吴等人[57]比较了蜕变测试与特殊值测试方法，发现随机测试方法能提供更大范围的测试数据，从而能够检测出更多的程序故障。Segura 等人[51]研究了随机测试方法对蜕变测试的故障检测效率的影响，发现随机测试方法生成的原始测试用例的故障检测效率高于手工设计方法。Alzahrani 等人[58]通过规范化地定义约束条件、测试用例分布方式等信息，随机生成原始测试用例。Sobania 等人[59]使用随机方法生成原始测试用例，在基于遗传编程的程序合成方法中检查候选程序是否满足目标程序的必要属性。

惠等人[60]提出了一种基于测试用例距离的自适应随机测试的蜕变测试用例生成方法。该方法生成的原始测试用例的故障检测效率高于随机测试方法，但引入较大计算开销。对于大规模、多故障的程序，该方法的有效性有待进一步验证。Barus 等人[19]通过比较随机测试方法和自适应随机测试方法生成的原始测试用例，进一步分析原始测试用例对蜕变测试有效性的影响。Hui 等人[61]提出了一种基于自适应随机测试的原始测试用例生成方法。该方法考虑原始测试用例和衍生测试用例与已执行测试用例的距离，能够提高故障检测效率，但时间开销较大，其时间复杂度为 $O(n^4)$。

Wu[16]提出了迭代蜕变测试技术，将蜕变关系的衍生测试用例用作下一轮某个蜕变关系的原始测试用例，以链式的方式应用蜕变关系序列。使用稀疏矩阵乘法程序评估了该技术的有效性，实验结果表明：与传统蜕变测试和特殊值测试相比，迭代蜕变测试能检测出更多故障，但存在较大的时间开销。Dong 等人[62]通过路径分析技术，将满足特定路径条件的衍生用例作为下一次迭代的原始用例。Lv 等人[63]融合了粒子群优化算法和蜕变关系的多路径覆盖的测试用例生成方法，使用粒子群优化算法生成第一个测试用例，然后使用蜕变关系迭代生

成其他测试用例,该方法的效率受蜕变关系和目标路径的影响。Sun 等人[13]提出了一个面向 Web 服务的迭代蜕变测试技术,通过构造的测试用例集迭代使用多条蜕变关系产生衍生测试用例,实现测试用例集的扩充,从时间和规模上提高了蜕变测试的测试用例生成效率。

Batra 和 Sengupta[18]使用遗传算法生成原始测试用例,通过最大化遍历被测程序的路径产生测试用例,实现生成一个小型但高效的原始测试用例集的目标。Chen 等人[64]从黑盒的角度提出选择覆盖等价类的测试用例选择算法,将被测程序的输入域划分为等价类。在等价类中,使用遗传算法对待测程序进行搜索生成原始测试用例。选择覆盖等价类的测试用例选择算法可以生成较小的测试用例集,检测出更多的程序故障。Saha 等人[65]采用进化搜索方法生成满足语句覆盖、分支覆盖和弱变异覆盖标准的原始测试用例集。实验结果表明:与语句覆盖和分支覆盖相比,满足弱变异覆盖的原始测试用例集具备更好的故障检测能力。

Dong 等人[66]提出一种符号执行的原始测试用例生成方法。符号执行分析程序的源代码以确定每个路径执行的符号约束,通过用实际数值替换符号输入生成原始测试用例,采用较小规模的 C 程序评估了所提方法的故障检测效率。Alatawi 等人[20]提出了一种基于动态符号执行的原始测试用例生成方法,使用动态符号执行工具 Pex 对待测程序进行动态符号执行产生原始测试用例,生成的测试用例明显提高了蜕变测试的程序覆盖率。Sun 等人[21]提出了一种基于符号执行的原始测试用例生成与优先级排序方法。首先使用符号执行工具 SPF 获取程序的所有可行路径及其约束条件,对获取的约束条件求解出原始测试用例集;然后基于路径距离对原始测试用例进行优先级排序。

上述工作从随机测试或者程序路径覆盖的角度探讨了原始测试用例的生成方法,基于原始测试用例转换生成衍生测试用例,没有考虑蜕变关系隐含的约束条件以及可行路径对的覆盖充分性问题。

董等人[17]从覆盖测试角度提出了三种蜕变测试路径覆盖准则,包括可行路径覆盖、每个蜕变关系的可行路径覆盖以及每个蜕变关系的可行路径对覆盖(简称 APPCEMST),并设计了相应的测试用例生成方法。其中,APPCEMST 首先采用符号执行获取所有的可执行路径信息;然后推导满足不同覆盖准则的测试用例集合;采用随机测试的方法生成测试用例。虽然 APPCEMST 与本文方法都采用路径分析技术生成蜕变测试用例集合,但二者存在如下不同之处:(1)动机方面:APPCEMST 主要考虑路径覆盖,而本文方法是主要为了避免生成无效的测试用例;(2)测试用例生成方面:APPCEMST 针对给定的路径采用随机的方法生成测试用例,本文方法则采用约束求解技术生成可行路径对的测试用例;(3)蜕变测试组执行方式方面:APPCEMST 随机执行,本文方法则考虑了蜕变测试组优先级问题,使用一定的策略对其进行排序。最后,在实验评估方面,APPCEMST 仅通过一个小程序进行了方法示例,而本文不仅开发了支持工具,采用 7 个不同领域的应用程序评估了本文方法的有效性,并与现有相关方法进行了比较。

8 总 结

蜕变测试是一种简单有效的测试方法,能够缓解测试预期问题。高效测试用例的生成是影响着蜕变测试的效率的主要因素之一。针对已有蜕变测试用例生成方法存在的易于产生无效蜕变测试用例、由于未考虑原始测试用例和衍生测试用例的差异难以生成充分且高效的蜕变测试组的问题,本文提出了一种基于路径分析的蜕变测试组生成与优先级排序技术 PaMTG。PaMTG 通过符号执行获得待测程序的路径约束,使用约束求解技术生成能够覆盖可行路径对的有效蜕变测试组,从而提高测试效率;设计了基于路径距离的蜕变测试组优先级排序算法,进一步提升蜕变测试的故障检测效率。在上述方法框架的基础上,开发了支持工具 MTG-GEN。采用一组 Java 程序实例评估了 PaMTG 生成蜕变测试组的故障检测能力。实验结果表明:本文提出的 PaMTG 不仅能够生成有效的蜕变测试组,而且有效提高了蜕变测试的故障检测效率。

进一步的研究工作包括:(1)支持更加复杂类型的蜕变关系的测试用例生成方法;(2)将 MTG-GEN 集成到课题组前期研制的蜕变关系识别支持工具 MR-GEN+[52],支持蜕变关系识别与蜕变测试组生成;(3)采用更大规模的实验程序进一步评估 PaMTG 的有效性。

参 考 文 献

[1] Weyuker E J. On testing non-testable programs. The Computer Journal, 1982, 25(4): 465-470

[2] Chen T Y. Metamorphic testing: A simple approach to alleviate the oracle problem//Proceedings of the 5th IEEE

International Symposium on Service Oriented System Engineering. Nanjing, China, 2010: 1-2

[3] Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: A new approach for generating next test cases. Hong Kong, China, 1998

[4] Brilliant S S, Knight J C, Ammann P. On the performance of software testing using multiple versions//Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS). Newcastle Upon Tyne, UK, 1990: 408-415

[5] Sim K Y, Low C S, Kuo F C. Eliminating human visual judgment from testing of financial charting software. Journal of Software, 2014, 9(2): 298-312

[6] Jameel T, Lin M X, Liu C. Test oracles based on metamorphic relations for image processing applications//Proceedings of the 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). Takamatsu, Japan, 2015: 1-6

[7] Mayer J, Guderlei R. On random testing of image processing applications//Proceedings of the 6th International Conference on Quality Software. Beijing, China, 2006: 85-92

[8] Dong Guo-Wei, Xu Bao-Wen, Chen Lin, et al. Survey of metamorphic testing. Journal of Frontiers of Computer Science and Technology, 2009, 3(2): 130-143(in Chinese)
(董国伟，徐宝文，陈林等. 蜕变测试技术综述. 计算机科学与探索，2009，3(2)：130-143)

[9] Lin Ren-Chao, Liu Xiao-Ming, Huang Song, Hui Zhan-Wei. Metamorphic relation construction method based on compositional function. Command Information System and Technology, 2013, 4(1): 66-69, 74(in Chinese)
(林仁超，刘晓明，黄松，惠站伟. 基于复合函数的蜕变关系构造方法. 指挥信息系统与技术，2013，4(1)：66-69，74)

[10] Liu H, Liu X, Chen T Y. A new method for constructing metamorphic relations//Proceedings of the 12th International Conference on Quality Software (QSIC). Xi'an, China, 2012: 59-68

[11] Kanewala U, Bieman J M. Using machine learning techniques to detect metamorphic relations for programs without test oracles//Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE). Pasadena, USA, 2013: 1-10

[12] Zhang J, Chen J J, Hao D, et al. Search-based inference of polynomial metamorphic relations//Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE). New York, USA, 2014: 701-712

[13] Sun C A, Fu A, Liu Y Q, et al. An iterative metamorphic testing technique for web services and case studies. International Journal of Web and Grid Services, 2020, 16(4): 364-392

[14] Hamlet R. Random testing. Encyclopedia of Software Engineering. America: American Cancer Society, 2002. doi: 10.1002/0471028959.sof268

[15] Sun C A, Dai H P, Liu H, Chen T Y. Feedback-directed metamorphic testing. ACM Transactions on Software Engineering and Methodology, 2023, 32(1): 20:1-20:34

[16] Wu P. Iterative metamorphic testing//Proceedings of the 29th International Computer Software and Applications Conference. Edinburgh, UK, 2005: 19-24

[17] Dong Guo-Wei, Nie Chang-Hai, Xu Bao-Wen. Effectively metamorphic testing based on program path analysis. Chinese Journal of Computers, 2009, 32(5): 1002-1013 (in Chinese)
(董国伟，聂长海，徐宝文. 基于程序路径分析的有效蜕变测试. 计算机学报，2009，32(5)：1002-1013)

[18] Batra G, Sengupta J. An efficient metamorphic testing technique using genetic algorithm//Proceedings of the 5th International Conference on Information Intelligence, Systems, Technology and Management. Gurgaon, India, 2011: 180-188

[19] Barus A C, Chen T Y, Kuo F C, et al. The impact of source test case selection on the effectiveness of metamorphic testing//Proceedings of the 1st International Workshop on Metamorphic Testing. Austin, USA, 2016: 5-11

[20] Alatawi E, Miller T, Søndergaard H. Generating source inputs for metamorphic testing using dynamic symbolic execution//Proceedings of the 1st International Workshop on Metamorphic Testing. Austin, USA, 2016: 19-25

[21] Sun C A, Liu B L, Fu A, et al. Path-directed source test case generation and prioritization in metamorphic testing. Journal of Systems and Software, 2022, 183: 111091: 1-111091:14

[22] Zhu H, Hall P A V, May J H R. Software unit test coverage and adequacy. ACM Computing Surveys, 1997, 29(4): 366-427

[23] Harman M, Yue J, Zhang Y. Achievements, open problems and challenges for search based software testing//Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation. Graz, Austria, 2015: 1-12

[24] Cadar C, Sen K. Symbolic execution for software testing: Three decades later. Communications of the ACM, 2013, 56(2): 82-90

[25] Wang R, Ben K. Classification of metamorphic relations and its application. American Journal of Engineering and Technology Research, 2011, 11(12): 1664-1668

[26] Chen T Y, Kuo F C, Liu H, et al. Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys, 2019, 51(1): 4:1-4:27

[27] Gotlieb A, Botella B. Automated metamorphic testing//Proceedings of the 27th IEEE Annual International Computer Software and Applications Conference. Dallas, USA, 2003: 34-40

[28] Sun C A, Wang G, Mu B H, et al. A metamorphic relation-based approach to testing web services without oracles. International Journal of Web Services Research, 2012, 9(1): 51-73

[29] Chan W K, Chen T Y, Lu H, et al. Integration testing of context-sensitive middleware-based applications: A metamorphic approach. International Journal of Software Engineering and Knowledge Engineering, 2006, 16(5): 677-703

[30] Jiang M Y, Chen T Y, Kuo F C, Ding Z H. Testing central processing unit scheduling algorithms using metamorphic testing//Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science. Beijing, China, 2013: 530-536

[31] Murphy C, Kaiser G, Hu L F, Wu L. Properties of machine learning applications for use in metamorphic testing// Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering. San Francisco, USA, 2008: 867-872

[32] Nakajima S, Bui H N. Dataset coverage for testing machine learning computer programs//Proceedings of the 23rd Asia-Pacific Software Engineering Conference. Hamilton, New Zealand, 2016: 297-304

[33] Zhou Z Q, Xiang S W, Chen T Y. Metamorphic testing for software quality assessment: A study of search engines. IEEE Transactions on Software Engineering, 2016, 42(3): 264-284

[34] Zhou Z Q, Zhang S J, Hagenbuchner M, et al. Automated functional testing of online search services. Software Testing, Verification and Reliability, 2012, 22(4): 221-243

[35] Tao Q M, Wu W, Zhao C, Shen W W. An automatic testing approach for compiler based on metamorphic testing technique //Proceedings of the 17th Asia Pacific Software Engineering Conference. Sydney, Australia, 2010: 270-279

[36] Cadar C, Godefroid P, Khurshid S, et al. Symbolic execution for software testing in practice: Preliminary assessment// Proceedings of the 33rd International Conference on Software Engineering. Honolulu, USA, 2011: 1066-1071

[37] Sen K. DART: Directed automated random testing//Proceedings of the 5th International Haifa Verification Conference. Haifa, Israel, 2009: 4

[38] Visser W, Pasareanu C S, Khurshid S. Test input generation with Java PathFinder. ACM SIGSOFT Software Engineering Notes, 2004, 29(4): 97-107

[39] Pasareanu C S, Rungta N. Symbolic PathFinder: Symbolic execution of Java bytecode//Proceedings of the 25th IEEE/ ACM International Conference on Automated Software Engineering. Antwerp, Belgium, 2010: 179-180

[40] Pasareanu C S, Visser W, Bushnell D H, et al. Symbolic PathFinder: Integrating symbolic execution with model checking for Java bytecode analysis. Automated Software Engineering, 2013, 20(3): 391-425

[41] Khurshid S, Pasareanu C S, Visser W. Generalized symbolic execution for model checking and testing//Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Warsaw, Poland, 2003: 553-568

[42] Moura L M D, Bjørner N S. Z3: An efficient SMT solver// Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Budapest, Hungary, 2008: 337-340

[43] Tillmann N, Schulte W. Unit tests reloaded: Parameterized unit testing with symbolic execution. IEEE Software, 2006, 23(4): 38-47

[44] Lahiri S K, Qadeer S. Back to the future: Revisiting precise program verification using SMT solvers//Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. San Francisco, USA, 2008: 171-182

[45] Costa M, Crowcroft J, Castro M, et al. Vigilante: End-to-end containment of internet worm epidemics. ACM Transactions on Computer Systems, 2008, 26(4): 9:1-9:68

[46] Sun C A, Jin H, Wu S Y, et al. Identifying metamorphic relations: A data mutation directed approach. Software: Practice and Experience, 2024, 54(3): 394-418

[47] Chen T Y, Poon P L, Xie X Y. METRIC: Metamorphic relation identification based on the category-choice framework. The Journal of Systems and Software, 2016, 116: 177-190

[48] Just R, Jalali D, Ernst M D. Defects4J: A database of existing faults to enable controlled testing studies for java programs// Proceedings of the 2014 International Symposium on Software Testing and Analysis. New York, USA, 2014: 437-440

[49] Ma Y S, Offutt J, Kwon Y R. MuJava: A mutation system for Java//Proceedings of the 28th International Conference on Software Engineering. Shanghai, China, 2006: 827-830

[50] Xu Yong-Yong. Medical Statistics. 3rd Edition. Beijing: Higher Education Press, 2014(in Chinese)
(徐勇勇. 医学统计学. 第 3 版. 北京: 高等教育出版社, 2014)

[51] Segura S, Hierons R M, Benavides D, Cortes A R. Automated metamorphic testing on the analyses of feature models. Information and Software Technology, 2011, 53(3): 245-258

[52] Sun C A, Fu A, Poon P, et al. METRIC+: A metamorphic relation identification technique based on input plus output domains. IEEE Transactions on Software Engineering, 2021, 47(9): 1764-1785

[53] Offutt A J, Lee A, Rothermel G, et al. An experimental determination of sufficient mutant operators. ACM Transactions on Software Engineering and Methodology, 1996, 5(2): 99-118

[54] Namin A S, Andrews J H, Murdoch D J. Sufficient mutation operators for measuring test effectiveness//Proceedings of the 30th International Conference on Software Engineering (ICSE 2008). Leipzig, Germany, 2008: 351-360

[55] Segura S, Fraser G, Sanchez A B, Ruiz-Cortes A. A survey on metamorphic testing. IEEE Transactions on Software Engineering, 2016, 42(9): 805-824

[56] Chen T Y, Kuo F C, Liu Y, Tang A. Metamorphic testing and testing with special values//Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Beijing, China, 2004: 128-134

[57] Wu Peng, Shi Xiao-Chun, Tang Jiang-Jun, et al. Metamorphic testing and special case testing: A case study. Journal of Software, 2005, 16(7): 1210-1220(in Chinese)
(吴鹏, 施小纯, 唐江峻等. 关于蜕变测试和特殊用例测试的实例研究. 软件学报, 2005, 16(7): 1210-1220)

[58] Alzahrani N, Spichkova M, Harland J. Application of property-based testing tools for metamorphic testing//Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). 2022: 553-560

[59] Sobania D, Briesch M, Röchner P, Rothlauf F. MTGP: Combining metamorphic testing and genetic programming//Proceedings of the 26th European Conference on Genetic Programming (EuroGP). 2023: 324-338

[60] Hui Zhan-Wei, Huang Song, Li Hui, Rao Li-Ping. A novel metamorphic test case generation method based on ART//Proceedings of the 8th National Academic Conference on Testing. Wuhan, China, 2016: 277-292(in Chinese)
(惠战伟，黄松，李辉，饶莉萍. 基于自适应随机测试策略的蜕变测试用例生成技术//第八届全国测试学术会议. 武汉，中国，2016: 277-292)

[61] Hui Z W, Wang X J, Huang S, Yang S. MT-ART: A test case generation method based on adaptive random testing and metamorphic relation. IEEE Transactions on Reliability, 2021, 70(4): 1397-1421

[62] Dong G W, Nie C H, Xu B W, Wang L L. An effective iterative metamorphic testing algorithm based on program path analysis//Proceedings of the 7th International Conference on Quality Software. Portland, USA, 2007: 292-297

[63] Lv X W, Huang S, Hui Z W, Ji H J. Test cases generation for multiple paths based on PSO algorithm with metamorphic relations. IET Software, 2018, 12(4): 306-317

[64] Chen L L, Cai L Z, Liu J, et al. An optimized method for generating cases of metamorphic testing//Proceedings of the 6th International Conference on New Trends in Information Science, Service Science and Data Mining. Taipei, China, 2012: 439-443

[65] Saha P, Kanewala U. Fault detection effectiveness of source test case generation strategies for metamorphic testing//Proceedings of the 3rd International Workshop on Metamorphic Testing. New York, USA, 2018: 2-9

[66] Dong G W, Guo T, Zhang P H. Security assurance with program path analysis and metamorphic testing//Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science. Beijing, China, 2013: 193-197

SUN Chang-Ai, Ph. D., professor, Ph. D. supervisor. His research interests include software testing, program analysis, software architecture, and service-oriented computing.

XING Jia-Yu, M. S. candidate. Her research interest is software testing.

LIU Bao-Li, M. S. candidate. Her research interest is software testing.

FU An, Ph. D. candidate. His research interest is software testing.

Background

Metamorphic testing (MT) leverages metamorphic properties (usually known as metamorphic relationships, MR) of the software under test (SUT) to generate the follow-up test cases from the source test cases, and verify the test results. MT effectively alleviates the oracle problem because it is no longer necessary to construct the expected outputs of individual test cases. Obviously, the used MRs and source test cases play a key role in the fault detection effectiveness of MT. Although there are already some test case generation methods for MT, they have the following limitations. Firstly, the scope of applicable input domains for an MR is not carefully considered, which may result in invalid test cases. Secondly, only the differences between source test cases are considered during the test case generation, which may result in insufficient metamorphic testing groups (a pair of source test case and follow-up test case, briefly as MTG). Finally, the fault detection capability of test cases is not considered, which may affect the fault detection efficiency of MT. In order to solve the above limitations, we propose a metamorphic testing group generation and prioritization technique based on path analysis (PaMTG). PaMTG first obtains all path pairs of MRs through analyzing the possible paths of the SUT, then generates MTGs to cover as many path pairs as possible, and finally prioritizes the derived MTGs according to their covered path information. A supporting tool was developed and an empirical study was conducted to evaluate PaMTG in terms of valid MTG ratio, fault detection capability, fault detection rate, and time overhead. The experimental results show that PaMTG is able to generate valid MTGs, and the fault detection capability and fault detection rate of the generated MTGs are better than that of the baseline techniques, such as random testing, adaptive random testing, and symbolic execution-based test case generation technique.

This work was supported by the National Natural Science Foundation of China under Grant Nos. 62272037 and 61872039.