计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20120866
51(4):874-881,2014

MujavaX:一个支持非均匀分布的变异生成系统

孙昌爱[1,2]　王　冠[1]
[1](北京科技大学计算机与通信工程学院　北京　100083)
[2](北京科技大学材料领域知识工程北京市重点实验室　北京　100083)
(casun@ustb.edu.cn)

MujavaX: A Distribution-Aware Mutation Generation System for Java

Sun Chang'ai[1,2] and Wang Guan[1]
[1](*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083)
[2](*Beijing Key Laboratory of Knowledge Engineering for Materials Science, University of Science and Technology Beijing, Beijing* 100083)

Abstract Mutation analysis is widely employed to evaluate the effectiveness of various software testing techniques. Existing mutation analysis techniques commonly insert faults into original programs uniformly, while actual faults tend to be clustered, which has been observed in empirical studies. This mismatch may result in the inappropriate simulation of faults, and thus may not deliver the reliable evaluation results. To overcome this limitation, we proposed a distribution-aware mutation analysis technique in our previous work, and it has been validated that the mutation distribution has impact on the effectiveness result of software testing techniques under evaluation. In this paper, we implement a mutation system called MujavaX to support distribution-aware mutation analysis. Such a system is an extension and improvement on Mujava which has been widely employed to mutation testing for Java programs. A case study is conducted to validate the correctness and feasibility of MujavaX, and experimental results show that MujavaX is able to generate a set of mutants for Java programs with respect to the given distribution model specified by testers.

Key words mutation analysis; mutation system; software testing; fault-based testing; performance evaluation

摘　要 变异分析是一种广泛用来评估软件测试技术性能的方法.已有的变异分析技术通常将变异算子平均地应用于原始程序.由于现实程序中的故障分布往往具有群集的特征,采用平均分布的变异分析方法不能客观地评估软件测试技术的性能.前期研究工作中提出了非均匀分布的变异分析方法,采用实例研究验证了不同的故障分布对测试技术性能评估的影响.为了增强非均匀分布的变异分析方法的实用性,开发了支持非均匀分布的变异生成系统MujavaX,该系统是对广泛实践的Mujava工具的扩展与改进.采用一个实例系统验证了开发的MujavaX的正确性与可行性,实验结果表明该系统能够生成指定分布的非均匀变体集合.

关键词 变异分析;变异系统;软件测试;基于故障的测试;性能评估

中图法分类号 TP311.5;TP311.56

收稿日期:2012-09-28;**修回日期**:2012-12-17

基金项目:国家自然科学基金项目(60903003,61370061);北京市自然科学基金项目(4112037);中央高校基本科研业务费专项资金项目(FRF-SD-12-015A);北京市优秀人才培养资助项目(D类)(2012D009006000002);材料领域知识工程北京市重点实验室2012年度阶梯计划项目(Z121101002812005)

变异分析是一种基于故障的软件测试技术[1].其基本思想是向待测试程序中植入各种类型的故障,产生的错误版本称为"变体",用来模仿某种故障的操作称为"变异算子".如果某个测试用例导致一个变体与待测程序产生不同的结果,那么该变体被"杀死",即与变体相关的故障能够被检测出来.针对给定的测试用例集,能够"杀死"的变体数量与所有变体数量的比例称为该测试用例集合的"变异得分".变异得分可以用来定义某个测试用例集的测试充分性.大量的实践表明,变异分析具有很强的故障检测能力[2],而且与手动植入故障相比,自动生成的变体更接近真实软件中的故障[3].尽管如此,变异分析技术并没有广泛应用于工业界[4],主要原因如下:1)变异分析计算代价昂贵;2)难以确定等价变体(即不存在一个测试用例使得该变体与原始程序产生不同的输出);3)缺少高效的自动化变异测试工具.

近年来,人们致力于改进变异分析技术的实用性[2],包括减少变异分析的开销以及等价变体的确定.

一方面,尽量减少变异分析的开销,包括减少变体的数量,或者减少测试用例生成与执行的开销.如果用多种变异算子模仿程序中可能潜藏的故障类型,那么即使一个普通的程序也会产生大量的变体.为了减少测试的成本,主要方法包括从大量的变体中随机挑选一些变体进行变异分析(随机选择)[5];或者挑选效果"好"的变异算子生成变体(选择性变异)[6].除此之外,人们提出了采用弱变异的方法[7].强变异是指当一个测试用例执行原始程序和变体之后的输出不同时,这个测试用例杀死变体,这包括可达性(变体故障部分必须能达到)、必要性(一旦达到,必须引起错误状态)和充分性(错误状态必须传播到输出).弱变异仅需要前两个条件成立,即认为不变体被杀死.

另一方面,尽量缓解等价变体判定的难题[8].人们探索对被测程序设置约束或采用程序切片技术等方法判定等价变体[9-10].

变异分析除了用来评估测试用例集充分性,还广泛用于评估各种软件测试技术的有效性[3].目前已有的变异分析技术将变异算子均匀运用于待测程序.例如,Mujava是一个广泛使用的面向Java程序变异生成系统[11],该系统在Java程序中均匀地植入各种故障.然而,大量经验研究发现,软件开发实践中,软件故障通常是"群束"的[12],换言之,部分模块代码中潜藏较多故障,其他模块代码中潜藏较少的故障.因此,采用均匀分布的变异分析方法评估某种软件测试技术的有效性是不可信的.

在前期研究中,我们提出了一种非均匀分布的变异分析技术[13],该技术是针对现有变异分析技术的改进与扩展.经验研究的结果表明,变体的分布对测试方法的有效性评估结果有显著影响.例如,我们采用均匀分布的变异分析方法与非均匀分布的变异分析方法评估了动态随机测试技术有效性[14],评估结果随着变异分布模型不同而不同.此外,非均匀分布的变异分析技术在某种程度上减少了变异测试过程的开销.

为了增强非均匀分布的变异分析技术的实用性,我们开发了非均匀分布的变异分析工具MujavaX.MujavaX是对Mujava的改进与扩展,能够根据给定的变异分布模型生成非均匀变体集合.本文介绍MujavaX的设计与实现关键技术,采用一个实例验证其正确性与可行性.

1 非均匀分布的变异分析技术

1.1 传统的变异分析框架

图1描述了常见的变异分析框架[13],该框架可用于评估某种测试技术的有效性.由原始程序P产生变体集合$\{P'_1,P'_2,\cdots,P'_n\}$的过程称为变异.向原始程序$P$实施某种变异算子得到$P$的一个变体$P'$.例如,逻辑运算符替换是一个常见的变异算子,在原始程序逻辑连接处使用该算子可得到一个变体集.运用不同的变异算子,可以得到不同的变体集合.

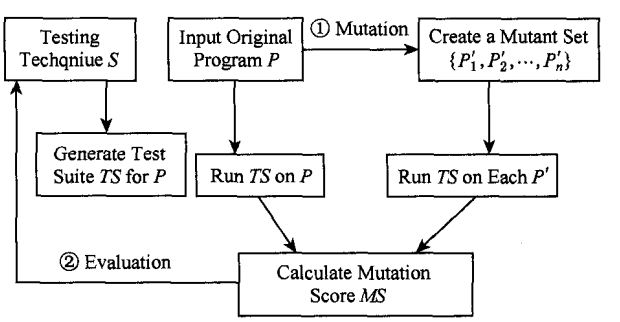


Fig. 1 A generic evaluation framework using traditional mutation analysis.

图1 基于传统变异测分析的一般评估框架

为了评估测试技术S的有效性,通常会选择若干待测程序进行评估.对于每一个待测程序P,使用S生成测试用例集TS,并在P和每个变体P'上

执行 TS,最后计算变异得分 MS,作为 S 的评估指标.变异得分高表明采用该种测试技术 S 产生的测试用例集合可以检测得到较多潜藏的故障,也就是说,测试技术 S 具有较好的有效性;反之,测试技术 S 有效性差.

在已有的变异分析方法中,变异算子均匀地应用在待测程序 P 上,这导致了采用变体模拟的故障均匀潜藏于待测程序中.

1.2 非均匀变异分析的模型与实现过程

大量的经验研究表明现实软件中的故障并不是平均分布的,而是以一种"群束"的方式集中地分布于某段代码[12].这一观测结果导致了软件测试中广泛适用的 Pareto 原则(如果在某个模块中发现了较多的故障,那么该模块中可能潜藏更多的故障).采用变异分析评估某种软件测试技术有效性时,为了得到更可靠的评估结果,必须考虑变体集合模拟故障的分布情况.

传统的变异分析技术并没有考虑故障的分布情况.传统的变异分析定义为 5 元组 $E=\langle P,S,D,L,A\rangle$[15],$P$ 是待测程序,S 是规格说明,D 是指定域,$L=(l_1,l_2,\cdots,l_i,\cdots,l_n)$是 P 中表示位置的 n 元组,$A=(A_1,A_2,\cdots,A_i,\cdots,A_n)$,其中 A_i 是位置 l_i 的一个替代集合,$1\leqslant i\leqslant n$.

与传统变异分析不同,我们提出的非均匀变异分析生成的故障 A 在 P 上不是平均分布的.在传统的变异分析模型的基础上,引入概率 Pr 实现上述分布.因此,非均匀变异分析定义为 6 元组 $E=\langle P,S,D,L,A,Pr\rangle$,其中 Pr 是 A_i 中每个替代 $a_{i,j}$ 出现的概率,$0\leqslant j\leqslant|A_i|$,$A_i$ 是替代集合,$|A_i|$是变异算子的数量.

非均匀分布的变异分析是传统平均分布变异分析的一种改进与扩展.对于指定分布,我们可以由平均分布的变体集合随机筛选而得.我们采用以下步骤得到指定分布的变体集合:

1) 设置分布模型 DM 以及变异算子的类型(在 6 元组中称作 A);

2) 使用传统变异分析方式将若干变异算子植入待测程序 P;

3) 随机选择 P 中一段"群束"区间并保留此区间内所有变体.对该区间外的变体进行随机删除,使之满足指定的分布模型 DM.若一个变体被移除(在 6 元组中成为 $a_{i,j}$),那么与之关联的 Pr 设为 0,反之 Pr 设为 1.

2 MujavaX 的设计与实现

为了支持非均匀分布的变异分析技术,我们开发了面向 Java 程序的变异生成工具 MujavaX.该工具基于 Mujava 提供的应用程序接口 API 生成指定分布的变体集合.

2.1 功 能

MujavaX 变异生成工具的主要功能如下:

1) 变体生成.实现多种变异算子,在 Java 程序植入故障,生成均匀分布变体.

2) 变体载入.载入所有均匀分布的变体,统计其变异位置等信息.

3) 变体分布规则的输入.定义变体分布规则.

4) 变体筛选.根据预先设定的分布规则,筛选已载入的均匀分布变体.

5) 变体输出.输出非均匀分布的变体集合.

2.2 工作流程

本文提出的 MujavaX 继承了 Mujava 生成变体的高效率特点,在 Mujava 生成的变体集合的基础上,使用变体筛选模块,生成满足指定分布规则的非均匀分布的变体集合.MujavaX 的主要工作流程如图 2 所示.首先,调用变异分析工具 MuJava 的应用编程接口 API,对某待测 Java 程序随机生成大量均匀分布的变体.然后定义筛选规则(修改筛选程序的部分参数),根据指定的筛选规则进行删减,筛选所需变

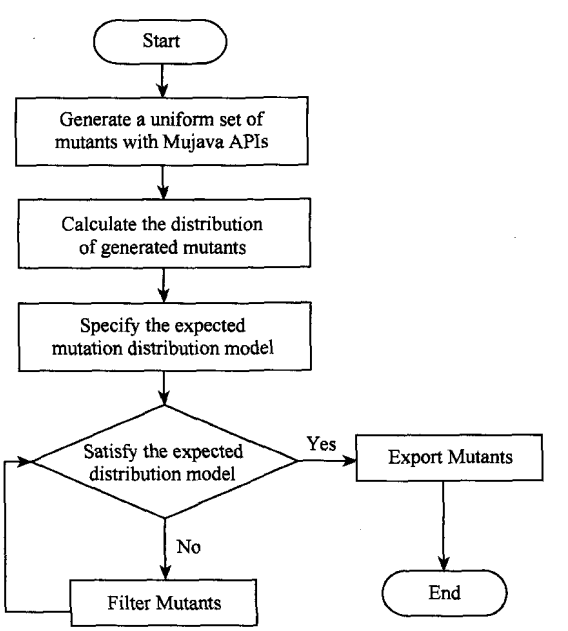


Fig. 2 The flowchart of MujavaX.

图 2 MujavaX 系统流程图

体.这个过程划分为载入变体、定义筛选规则、进行筛选、输出结果几个部分.其中,筛选完成并输出结果之后,根据需求可再次修改筛选程序(修改筛选程序的部分参数).因此,变体筛选过程是可以循环往复的.

使用该工具时,用户可以根据测试需求重复生成变体集合,或决定何时结束变异分析的过程.

2.3 系统设计

MujavaX 的体系结构如图 3 所示:

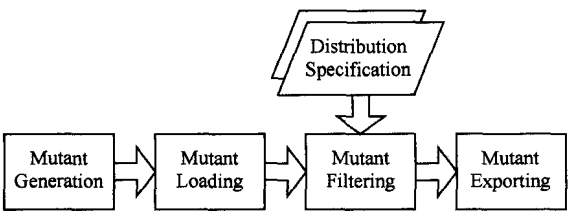


Fig. 3 The architecture of MujavaX.

图 3 MujavaX 系统结构

1) 变体生成.该模块通过调用 Mujava API 对某待测 Java 程序进行变体生成.该模块接受待测原始程序以及变异算子,输出均匀分布的变体集合.

2) 变体载入.该模块主要完成载入变体生成模块生成的均匀分布变体集合,并统计变体信息(变异位置、变异内容等).该模块输出变体统计信息,包括变体的编号 ID、变异发生位置 mLoc 等.

3) 变体筛选.根据用户指定的分布规则进行删减、筛选变体,生成满足用户指定分布的变体集合统计信息,该信息采用文本及图形方式展示.

4) 变体输出.该模块读取经变体筛选后的非均匀变体集合统计信息,输出满足指定分布的变体集合(.java 源文件和.class 类文件).同时,以折线图方式显示变体集合中变体的分布情况.

2.4 非均匀分布变异分析的实现

我们采用如下算法步骤生成指定分布的变体集合.

步骤 1. 调用 Mujava API 得到均匀分布的变体集合.

步骤 2. 标识与分析每个变体的代码行分布,得到如表 1 所示的描述信息,包括变体编号 ID、变异发生的位置 mLoc(发生在第几行代码)、变异内容 mDetail(在 mLoc 行发生了什么类型的变异).

Table 1 A Brief Description of Mutants in MujavaX

表 1 MujavaX 变体信息简要描述

ID	mLoc	mDetail
mutant001	88	money>maxTransferAmount_Once ‖ money<=0=>money>maxTransferAmount_Once && money<=0
mutant002	88	money=>--money
mutant003	88	money>maxTransferAmount_Once=>money==maxTransferAmount_Once
mutant004	89	EXCEPTION_BEYOND_TRANSFERLIMIT=>~EXCEPTION_BEYOND_TRANSFERLIMIT
mutant005	110	same_bank=>! same_bank
mutant006	110	same_bank==false=>same_bank! =false
mutant007	110	same_bank==false=>! (same_bank==false)
mutant008	110	same_bank==false && same_location==false=>same_bank==false^same_location==false

步骤 3. 对于输入的变异分布规则进行正确性检查.如果正确则进入步骤 4;否则,重复执行步骤 3.尽管分布规则有多种表示形式,我们采用 $X\%$ 和 $Y\%$ 分别表示代码行比率与故障比率,而且 $X+Y=100$.换言之,X-Y 分布规则的含义是 $Y\%$ 的故障分布在 $X\%$ 的代码中.

步骤 4. 对于发生变异的代码行区间(假设该区间包含 N 行代码),在 $1\sim N$ 之间生成一个随机数 R.若此随机数 $R/N<X\%$,则重新生成;否则,选择 R-$X\%\times N$ 至 R 区间为连续变异区间 M_c.

步骤 5. 该区间 M_c 内变体数量记为 N_1.M_c 以外的区间 M_{nc} 上变体数量记为 N_2.若 $N_1/(N_1+N_2)>Y\%$,则随机删除 M_c 上的变体,使其满足 $N_1/(N_1+N_2)=Y\%$;若 $N_1/(N_1+N_2)<Y\%$,则随机删减 M_{nc} 上的变体,使其满足 $N_2/(N_1+N_2)=(100-Y)\%$.换言之,删除变体的对应变异 Pr 设为 0;否则,Pr 设为 1.

基于上述算法实现的 MujavaX 支持只含一个连续区间的"群束"变异,且分布规则必须满足特定的约束.至于如何设定变异分布模型不属于本文的讨论范围,因为 MujavaX 仅提供生成指定分布模型的变体集合.

2.5 工具使用

本文基于 JSP 开发了工具原型,如图 4 所示.

首先,用户通过"Browse"按钮输入待变异的 Java 文件;然后在"Mutation Operators"栏中,选择适用的变异算子的类型(在相应的变异算子的复选框中打勾);在"Parameters"栏"Distribution"文本框中设置变异分布模型. 上述参数设置完毕后,点击右侧的"Mutant Generation"栏的"Mutate"按钮生成变体.

变体集合生成后,"Mutation Distribution"一栏将显示用户指定分布的变体分布折线图. 横坐标为异发生位置(用代码行表示);纵坐标为某行代码位置上的变体数量. 点击"download"按钮,可以下载生成的变体集合(生成 zip 格式的压缩文件).

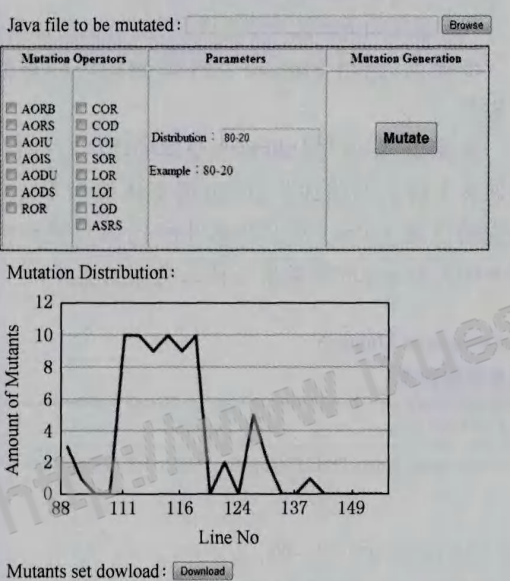


Fig. 4 A snapshot of MujavaX.

图 4 MujavaX 界面截图

目前,MujavaX 支持如下 15 种变异算子:

1) AORB(arithmetic operator replacement). 替换一个二元算术运算符为另一个二元算术运算符. 例如,"$a+b$"替换为"$a-b$".

2) AORS(arithmetic operator replacement). 替换一个 short-cut 运算符为另一个一元运算符. 例如,"$a++$"替换为"$a--$".

3) AOIU(arithmetic operator insertion). 插入基本一元运算符. 例如,"a"替换为"$-a$".

4) AOIS(arithmetic operator insertion). 插入 short-cut 运算符. 例如,"a"替换为"$a++$".

5) AODU(arithmetic operator deletion). 删除基本一元运算符. 例如,"$-a$"替换为"a".

6) AODS(arithmetic operator deletion). 删除 short-cut 运算符. 例如,"$a++$"替换为"a".

7) ROR(relational operator replacement). 替换关系运算符为其他的关系运算符. 例如,"$a<b$"替换为"$a\geqslant b$".

8) COR(conditional operator replacement). 替换二元条件运算符为其他的二元条件运算符. 例如,"$a\&\&b$"替换为"$a \| b$".

9) COI(conditional operator insertion). 插入一元条件运算符. 例如,"a"替换为"$!a$".

10) COD(conditional operator deletion). 删除一元条件运算符. 例如,"$!a$"替换为"a".

11) SOR(shift operator replacement). 替换移位运算符为其他的移位运算符. 例如,"$a\ll b$"替换为"$a\gg b$".

12) LOR(logical operator replacement). 替换一个二元逻辑运算符为其他的二元逻辑运算符. 例如,"$a\&b$"替换为"$a|b$".

13) LOI(logical operator insertion). 插入一元逻辑运算符. 例如"a"替换为"$\sim a$".

14) LOD(logical operator deletion). 删除一元逻辑运算符. 例如,"$\sim a$"替换为"a".

15) ASRS (short-cut assignment operator replacement). 替换 short-cut 赋值操作符为其他的 short-cut 赋值操作符. 例如"$a+=b$"替换为"$a*=b$".

3 实例验证

我们采用一个 Java 语言实现的 Web 服务验证开发的 MujavaX 的正确性与可行性.

3.1 待测程序

我们使用一个部署在 Tomcat 上的电子支付 Web 服务作为待测程序[16]. 该系统提供存款、取款、转账、查询等功能,数据存储在 MySQL 数据库中. 转账业务应用广泛、业务逻辑实现相对复杂,因此我们选择转账功能的 Java 实现,对 MujavaX 工具进行验证. 该功能实现涉及的代码位于第 88~154 行.

3.2 实验过程

首先,使用 MujavaX 工具生成平均分布的变体集合,其分布如图 5 所示. 其中,横坐标表示代码行,纵坐标表示该代码行上发生变异的变体数量.

采用传统变异分析技术得到的变体集合在各区间上的分布情况如表 2 所示.

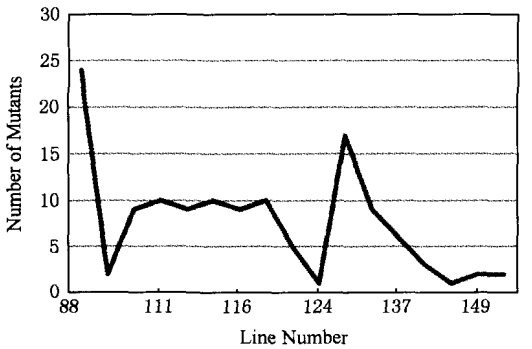


Fig. 5 An illustration of the distribution of mutants generated using traditional mutation analysis.

图 5 传统变异分析得到变体集合分布情况

Table 2 A Summary of Distribution of Mutant Numbers using Traditional Mutation Analysis

表 2 传统变异分析得到的变体数量分布情况

Line Number	Number of Mutants	Percentage /%
110～122	57	44.19
122～135	32	24.81
80～109	26	20.16
136～151	12	9.30
151～165	2	1.55

现假设要生成 80/20 分布的变体集合,MujavaX 随机选择一段 20%的"群束"变异区间.得到该"群束"区间代码行为第 110～122 行,该"群束"区间变体数量为 57,"群束"区间外变体数量为 72.由于 57/(57+72)<80%,故在第 110～122 行区间外随机删除变体,使删减后的数量 $N_{2'}$ 满足 $57/(57+N_{2'})=80\%$.最终,得到了满足 80/20 分布的变体集合,如图 6 所示:

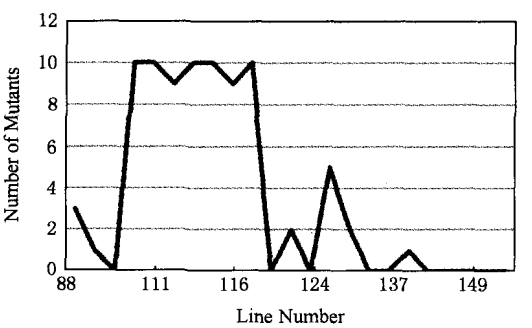


Fig. 6 An illustration of distribution of mutants using 80/20 distribution mutation analysis.

图 6 满足 80/20 分布的变体集合分布情况

采用 MujavaX 删减后的各区间上变体数量如表 3 所示.

Table 3 A Summary of Distribution of Mutant Numbers using 80/20 Distribution Mutation Analysis

表 3 80/20 分布的变体数量分布情况

Line Number	Number of Mutants	Percentage/%
110～122	57	79.17
122～135	9	12.50
80～110	4	5.56
136～151	1	1.39
151～165	1	1.39

相似地,MujavaX 可以生成其他指定分布的变体集合.该实例研究的结果验证了 MujavaX 的正确性与可行性.

4 相关工作

近年来人们尝试从多个方面改进与扩展变异分析技术[2].已有研究工作侧重于降低变异分析的计算开销.这些方法可分为两类:降低生成变体数量和降低变体执行开销.

常用的一种方法是通过有选择性的植入某些变异算子[17],其思想是如果难以被发现的错误被植入原始程序,那么容易被发现的错误就没有植入的必要.Kuhn[18]使用布尔规格的故障分层来指导变异算子的选择.Mresa 和 Bottaci 在实证研究的基础上对变异算子进行了分类,观察到如下变异算子更为有效:算术运算符号替换(arithmetic operator replacement, AOR)、语句分析(statement analysis, SAN)、语句删除(statement deletion, SDL)、关系运算符替换(relational operator replacement, ROR)、一元运算符插入(unary operation insertion, UOI)[6].相似地,Offutt 等人也指出,采用算术运算符号替换 AOR、关系运算符替换 ROR、一元运算符插入 UOI、绝对值插入(absolute value insertion, ABS)和逻辑连接符替换(logical connector replacement, LCR)产生的测试用例集的变异得分可以达到采用全部变异算子变异得分的 99%,并且减少了 77%的变体数量[17].非均匀变异分析技术与上述方法均不相同.该方法通过删减非群束区间的变体达到减少变体总数的目的,我们的方法可在上述减少变体数量方法的基础上选出具有群束特征的变体集合.

另外一种减少变体总数的方式是通过随机地选取变体集合的一个子集[19].人们发现选择 10%被杀死的变体可以用来进行变异分析,但是该方法仍然

存在生成、编译和链接所有变体的大量开销[20]. 有趣的是基于变异算子的变体筛选并没有优于变体随机选择[21]. 这种方式与我们的方法共同点是均采用随机选取的方式;不同的是我们的方法在随机筛选、删减变体时的目的是使变体集合满足指定的分布.

Jia 和 Harman 提出了一种高阶变异 HoM[22]. HoM 可以代替一些难以被发现的错误,其思想是使用一个 HoM 代替若干一阶变体,从而减少变体总数. 相似的方法还有 Polo 等人提出的一种二阶变异[23],一个二阶变体可代替两个一阶变体,从而使变体总数降低一半.

降低变异分析开销的另外一种方法是减少测试用例数量或减少测试用例执行开销. 根据杀死同一位置的多个变体条件相近的特点,文献[24]提出一种能够同时杀死该位置多个变异体的测试数据生成方法,减少了变异分析的测试用例生成开销. 文献[25]提出了一种在变异测试过程中引进测试用例选择以降低测试代价的方法,该方法为每个变体选择一定数量的测试用例,并且限制变体执行的最大次数. 文献[26]提出了一种考虑数据依赖的变异测试数据域削减方法,该方法在变异测试数据生成的成功率和执行效率方面有较大的提高.

变体群束(mutant clustering)[2]技术是一种将所有变体分类到不同簇(cluster)中. 某个簇中的所有变体可被一些相似的测试用例杀死. 该方法中"簇"的概念与我们提出的具有群束特征的变异是完全不同的,其动机以及筛选标准与非均匀变异分析也完全不同.

我们提出的非均匀变体生成技术采用减少变体总数的方法降低变异测试的开销. 与此同时,非均匀变体生成技术增强了传统变异分析技术对测试技术的性能评估的可信度,特别是那些与故障分布有关的测试技术. 例如,采用传统的变异分析与非均匀分布的变异分析方法评估了动态随机测试技术 DRT[14]的性能评估时,变异分布模型显著影响评估结果(详细的经验研究请参考文献[13]). 这意味着本文提出的非均匀分布的变异分析技术及支持工具 MujavaX 对测试技术的性能评估有重要的意义.

5 结 论

在分析传统变异分析技术不足的基础上,我们提出了非均匀变异分析的概念. 本文通过扩展 Mujava 开发了一个支持非均匀变异分析工具 MujavaX. 对于给定的变异分布模型,MujavaX 能够产生指定的非均匀分布的变体集合. 实例研究的结果表明,MujavaX 可以生成指定分布的变体集合. 借助 MujavaX 可以采用非均匀变异分析对软件测试技术的有效性进行有效评估,改进评估的效率(无需手工进行变异分析)与可信性(支持不同的变异分布模型).

致谢 作者感谢北京科技大学朱亚俊同学参与了 MujavaX 系统的部分开发工作!

参 考 文 献

[1] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: Help for the practicing programmer [J]. IEEE Computer, 1978, 1(4): 31-41

[2] Jia Yue, Harman M. An analysis and survey of the development of mutation testing [J]. IEEE Trans on Software Engineering, 2011, 37(5): 649-678

[3] Andrews J H, Briand L C, Labiche Y. Is mutation an appropriate tool for testing experiments? [C] //Proc of the 27th Int Conf on Software Engineering (ICSE 2005). New York: ACM, 2005: 402-411

[4] Offutt J, Untch R H. Mutation 2000: Uniting the orthogonal [C] //Proc of 2000 Mutation Testing for the New Century. Norwell, MA: Kluwer Academic Publishers, 2001: 34-44

[5] DeMillo R A, Guindi D S, McCracken W M, et al. An extended overview of the Mothra software testing environment [C] //Proc of the 2nd Workshop on Software Testing, Verification, and Analysis. Piscataway, NJ: IEEE, 1988: 142-151

[6] Mresa E S, Bottaci L. Efficiency of mutation operators and selective mutation strategies: An empirical study [J]. Software Testing, Verification and Reliability, 1999, 9(4): 205-232

[7] Howden W E. Weak mutation testing and completeness of test sets [J]. IEEE Trans on Software Engineering, 1982, 8(4): 371-379

[8] Frankl P G, Weiss S N, Hu C. All-Uses versus mutation testing: An experimental comparison of effectiveness [J]. Journal of Systems and Software, 2007, 38(3): 235-253

[9] Offutt A J, Pan J. Automatically detecting equivalent mutants and infeasible paths [J]. Software Testing, Verification and Reliability, 1997, 7(3): 165-192

[10] Hierons R, Harman M. Using program slicing to assist in the detection of equivalent mutants [J]. Software Testing, Verification and Reliability, 1999, 9(4): 233-262

[11] Offutt J, Ma Y S, Kwon Y R. An experimental mutation system for Java [J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(5): 1-4

[12] Myers G J. The Art of Software Testing [M]. 2nd ed. New York: John Wiley and Sons, 2004: 10-20

[13] Sun Changai, Wang Guan, Cai Kaiyuan, et al. Distribution-aware mutation analysis [C] //Proc of the 9th IEEE Int Workshop on Software Cybernetics (IWSC 2012), Collocated with the 36th Annual IEEE Int Computer Software and Application Conf (COMPSAC 2012). Los Alamitos, CA: IEEE Computer Society, 2012: 170-175

[14] Cai Kaiyuan, Hu Hai, Jiang Changhai, el al. Random testing with dynamically updated test profile [EB/OL] //Proc of the 22nd Annual Int Symp on Software Reliability Engineering (ISSRE 2009). [2012-07-28]. http://www.issre2009.org/papers/issre2009_198.pdf

[15] Morell L J. A theory of fault-based testing [J]. IEEE Trans on Software Engineering, 1990, 16(8): 844-857

[16] Sun Changai, Wang Guan, Mu Baohong, el al. Metamorphic testing for Web services: Framework and a case study [C] // *Proc of the 9th IEEE Int Conf on Web Services* (ICWS 2011). Los Alamitos, CA: IEEE Computer Society, 2011: 283-290

[17] Offutt A J, Lee A, Rothermel G, el al. An experimental determination of sufficient mutation operators [J]. ACM Trans on Software Engineering Methodology, 1996, 5(2): 99-118

[18] Kuhn D R. Fault classes and error detection capability of specification-based testing [J]. ACM Trans on Software *Engineering and Methodology*, 1999, 8 (4): 411-424

[19] Wong W E, Mathur A P. Reducing the cost of mutation testing: An empirical study [J]. Journal of Systems and Software, 1995, 31(3): 185-196

[20] Usaola M P, Mateo P R. Mutation testing cost reduction techniques: A Survey [J]. IEEE Software, 2010, 27(3): 80-86

[21] Zhang Lu, Hou Shanshan, Hu Junjue, et al. Is operator-based mutant selection superior to random mutant selection? [C] //*Proc of the 32nd Int Conf on Software Engineering* (ICSE 2010). New York: ACM, 2010: 435-444

[22] Jia Yue, Harman M. Constructing subtle faults using higher order mutation testing [C] //Proc of the 8th Int Working Conf on Source Code Analysis and Manipulation (SCAM'08). Piscataway, NJ: IEEE, 2008: 249-258

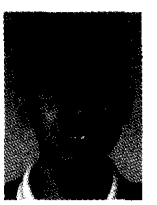
[23] Polo M, Piattini M, Rodríguez G. Decreasing the cost of mutation testing with second-order mutants [J]. Software Testing, Verification and Reliability, 2009, 19(2): 111-131

[24] Shan Jinhui, Gao Youfeng, Liu Minghao. et al. A new approach to automated test data generation in mutation testing [J]. Chinese Journal of Computers, 2008, 31(6): 1025-1034 (in Chinese)
(单锦辉,高友峰,刘明浩,等. 一种新的变异测试数据自动生成方法[J]. 计算机学报, 2008, 31(6): 1025-1034)

[25] Jiang Yuting, Li Bixin. Novel technique for cost reduction in mutation testing [J]. Journal of Southeast University: English Edition, 2011, 27(1): 17-21

[26] Liu Xinzhong, Xu Gaochao, Hu Liang. et al. An approach for constraint-based test data generation in mutation testing [J]. Journal of Computer Research and Development, 2011, 48(4): 617-626 (in Chinese)
(刘新忠,徐高潮,胡亮,等. 一种基于约束的变异测试数据生成方法[J]. 计算机研究与发展, 2011, 48(4): 617-626)

Sun Chang'ai, born in 1974. Currently associate professor at the School of Computer and Communication Engineering, University of Science and Technology Beijing. Senior member of China Computer Federation. His main research interests include software testing, software architecture, and service-oriented computing.



Wang Guan, born in 1988. Master candidate at the School of Computer and Communication Engineering, University of Science and Technology Beijing. Student member of China Computer Federation. His main research interests include software testing and service-oriented computing (jayjaywg@qq.com).