基于变异分析和集合进化的测试用例生成方法

张功杰[1),3)] 巩敦卫[2)] 姚香娟[4)]

[1)](中国矿业大学计算机科学与技术学院 江苏 徐州 221116)

[2)](中国矿业大学信息与电气工程学院 江苏 徐州 221116)

[3)](江苏师范大学计算机科学与技术学院 江苏 徐州 221116)

[4)](中国矿业大学理学院 江苏 徐州 221116)

摘 要 变异分析能够辅助生成有效的测试用例集,然而,高昂的测试代价,严重影响了这一技术在实际软件测试中的广泛应用.文中基于弱变异分析,研究求解测试用例生成问题的新方法,以高效地生成具有很高缺陷检测能力的测试用例集.该方法首先利用变异前后的语句,构造变异分支,并将所有变异分支集成到原程序,形成新的被测程序;然后,以测试用例集作为决策变量,根据该测试用例集对变异分支的覆盖信息,构造目标函数,建立新的测试用例生成问题的数学模型;最后,采用集合进化优化方法求解上述模型时,设计具有针对性的适应度函数和进化策略,使得一次求解该模型,生成杀死所有变异体的测试用例.将所提方法应用于 13 个基准和工业程序的测试,并与传统的遗传算法进行了比较.实验结果表明,所提方法能够高效地生成测试用例,且生成的测试用例具有更高的缺陷检测能力.

关键词 软件测试;变异测试;变异分支;测试用例生成;集合进化

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02318

Test Case Generation Based on Mutation Analysis and Set Evolution

ZHANG Gong-Jie[1),3)] GONG Dun-Wei[2)] YAO Xiang-Juan[4)]

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

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

[3)](*School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, Jiangsu* 221116)

[4)](*College of Science, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

Abstract Mutation analysis can assist in generating effective test cases. However, the high cost in mutation testing has affected its widespread application in practical software testing. Based on the weak mutation analysis, we proposed a new method to effectively generate test cases with a high ability in detecting defects. In our method, mutant branches are constructed from statements before and after mutation, and a new program is formed by integrating all mutant branches into the original program. Then, a novel model is established for optimizing test case generation. In the model, the decision variable is a test suite, and the objective is defined based on the coverage information of the test suite to all mutant branches. Finally, when solving the above model by using the set-based evolutionary optimization method, an appropriate fitness function and genetic operators are designed, so as to generate test cases that kill all mutants by running the method once. The proposed method was applied to thirteen benchmark and industrial programs,

收稿日期:2014-07-16;最终修改稿收到日期:2015-03-20.本课题得到国家自然科学基金(61375067,61203304)、江苏省自然科学基金(BK2012566)、中央高校基本科研业务费专项资金(2012QNA41)资助.**张功杰**,男,1979 年生,博士研究生,讲师,主要研究方向为软件测试.E-mail:zhanggongjie@126.com.**巩敦卫**,男,1970 年生,博士,教授,博士生导师,主要研究领域为基于搜索的软件工程、智能优化与控制.**姚香娟**,女,1975 年生,博士,副教授,主要研究方向为基于搜索的软件工程.

and compared with the traditional genetic algorithm. Our experimental results showed that our method can effectively generate test cases, and the generated test cases have a higher ability in detecting defects.

Keywords　software testing; mutation testing; mutant branch; test case generation; set-based evolutionary algorithm

1　引　言

变异测试通过向原程序人为地注入缺陷[1]，以模拟软件中的实际缺陷. 被注入缺陷的程序副本称为变异体. 采用相同的测试输入，分别执行原程序和变异体，如果某测试用例能够从执行结果上区分某变异体和原程序，则称该变异体被杀死；如果任何测试用例都不能杀死某变异体，则称该变异体为等价变异体.

变异测试常被用于评价现有测试用例集的质量；也被用于辅助生成单元测试用例，其准则为所生成的测试用例集能够杀死所有变异体. 对于给定的测试用例集，其杀死的变异体数量占所有非等价变异体的百分比，称为该测试用例集的变异得分. 变异得分是衡量测试用例缺陷检测能力的重要指标[2]. 因此，基于变异分析，能够生成具有更高缺陷检测能力的单元测试用例集.

变异测试不仅可以模拟现实中的各种缺陷，还能够选择缺陷发生的位置[3]. 但是，实际的软件规模大、复杂度高[4]，从而产生的变异体为数众多. 此时，为了生成有效的测试用例集，以最大限度地杀死变异体，必须反复执行原程序和变异体，这显著增加了测试用例生成的成本，从而，严重影响了这一测试用例生成技术的实用性.

变异体约简是减小变异测试代价的有效途径，然而，现有变异体约简方法，其约简力度和有效性仍需进一步提高. 弱变异测试能够降低程序的执行代价，但是，其降低幅度仅为 50%左右. 更重要的是，现有基于变异分析的测试用例生成方法，每次仅以杀死一个变异体作为目标，生成测试用例，使得测试用例生成的效率降低. 这表明，研究新的方法，以提高基于变异分析的测试用例生成效率，是非常必要的. 迄今为止，相关的研究仍非常少.

本文基于弱变异测试准则，研究新的测试用例生成方法，以高效地生成具有很高缺陷检测能力的测试用例. 所提方法根据弱变异测试转化思想[5]，将所有变异体转化为变异分支，并集成到原程序中，形成新的被测程序之后，建立新的测试用例生成问题的数学模型，并采用集合进化优化方法求解上述优化模型，以生成杀死所有变异体的测试用例. 13 个基准和工业程序的测试结果表明，所提方法能够有效提高测试用例的生成效率.

本文第 2 节综述相关的研究工作，并说明本文的研究动机；第 3 节详细阐述所提方法，包括变异测试转化、问题的表示以及测试用例生成问题的数学模型；所建模型的集合进化求解方法，将在第 4 节给出；第 5 节通过对比实验，评价所提方法的性能；最后，第 6 节总结本文所做的工作，并提出后续的研究问题.

2　相关工作

本节首先简述变异测试的思想；然后，综述现有基于变异分析的单元测试用例生成方法；考虑到进化优化在测试用例生成中已经取得丰硕的研究成果，并对基于变异分析的测试用例生成问题具有很强的指导作用，最后，阐述已有的基于进化优化的测试用例生成方法.

2.1　变异测试

变异测试最早由 DeMillo 等人[6]和 Hamlet[7]提出，目前，已广泛应用于 C、Java、SQL 等各种语言以及单元、集成等软件测试的各个层次. 变异测试已经取得丰硕的研究成果，并应用于工业程序的测试[8]. 然而，高昂的测试代价，始终是变异测试难以回避的问题. 研究合适的方法，降低变异测试代价，成为该领域的热点方向之一.

通过抽样和选择变异测试方法，以减少所需杀死的变异体数量，是降低变异测试代价的有效方法. 其中，抽样方法仅选择一定比例的变异体执行测试[9-10]，但是，随着抽样比例的减少，变异测试的充分度将明显下降[11]. 选择方法通过舍弃某些变异算子，减少生成的变异体数量[12-13]，然而，被舍弃变异算子的增多，将导致变异得分显著降低. 此外，高阶

变异体虽然也能够有效减少变异体的数量[14-16]，可是，高阶变异体生成的代价，将随着阶次的增多而急剧攀升.

弱变异测试是另一类降低测试代价的有效方法[17]. 弱变异测试要求测试用例必须能够执行到变异语句，且执行变异语句后的状态发生改变，即满足可达性和必要性条件[18]. 而强变异测试则进一步要求改变的状态必须影响程序的执行结果，即满足充分性条件. 研究表明，满足必要性条件的测试用例，能够在很大程度上满足充分性条件[19-20]. 绝大多数情况下，弱变异测试是强变异测试的有效替代，且弱变异测试能够节省一半的测试成本[21-22].

基于弱变异测试准则，Durelli 等人[23]利用现有托管执行环境和底层虚拟指令集，构建虚拟机集成的弱变异测试环境，能够显著提高变异测试效率. Kim 等人[24]针对非解释性系统执行速度快的特点，建立新的弱变异测试方法，该方法降低了非解释性变异测试系统的执行代价. 把变异体转化为变异分支，Papadakis 等人[5]将弱变异测试问题，转化为分支覆盖问题，从而进一步提高弱变异测试的效率. Papadakis 等人[25]还分析变异体所在的路径，以选择变异体执行弱变异测试，从而增强了所生成的测试用例的有效性.

2.2 基于变异分析的测试用例生成

测试用例生成是指根据特定的测试准则，例如分支覆盖，产生满足需求的测试用例. 作为直接面向缺陷的测试技术，变异测试能够辅助生成具有很高缺陷检测能力的单元测试用例. 但是，基于变异测试，如何高效地生成测试用例，相关的研究成果却很少.

基于约束的测试用例生成(Constraint-Based Test data generation，CBT)方法是较早基于变异分析生成测试用例的研究之一，该方法将杀死变异体的条件转化为约束，并通过约束求解的方法，生成测试用例[26]. 实验表明，CBT 生成的测试用例，能够杀死约 90%的变异体[19]. 动态域约简(Dynamic Domain Reduction，DDR)方法对 CBT 进行改进，并通过回溯搜索，生成测试用例[27-28]. 但是，DDR 和 CBT 由于采用符号执行方法，导致生成测试用例的代价高、适应范围窄. 动态符号执行(Dynamic Symbolic Execution，DSE)方法虽然改进了传统符号执行方法[29-30]，但是，由于过分依赖约束求解器，使得 DSE 生成测试用例的效率尚需进一步提高.

上述方法都基于程序的控制流图(Control Flow Graph)，仅考虑程序语句之间的控制依赖关系，而忽略了它们之间的数据依赖关系. 同时考虑这两种关系，对 DDR 方法进行改进，能够提高测试用例的生成效率[31]. 此外，根据同一位置变异语句之间的相似性，将这些变异体组合成少数复合变异体，进一步降低了测试用例生成的代价[32]. 但是，上述研究在问题的求解方法上，仍局限于传统的约束求解方法，导致测试用例生成的效率难以显著提高.

2.3 进化优化的测试用例生成

近年来，进化优化方法广泛应用于软件测试，尤其用于解决传统结构化测试用例的生成[33-41]问题. 在传统的结构化测试中，Harman 等人[38]利用多目标优化方法搜索测试用例，但是，每次仅针对一个测试目标，生成相应的测试用例. Fraser 等人[39]利用遗传算法(Genetic Algorithm，GA)，生成高质量的测试用例集，然而，该方法目前仅应用于传统的结构化测试.

Jia 等人[14-15]利用进化优化方法，生成更难杀死的高阶变异体，以反映实际软件中的复杂缺陷. Fraser 等人[3]利用进化优化方法，提高测试用例生成的自动化程度. Souza 等人[42]统计了基于变异分析生成测试用例的相关 19 篇主要论文，发现采用 GA 的就有 6 篇. 可见，在基于变异分析的测试用例生成问题上，进化优化已经成为主要方法. 尽管如此，与传统结构化测试用例生成相比，对基于变异分析的测试用例生成问题的研究仍然很少；而且，现有的这类方法，其测试用例生成效率仍有待于更进一步提高.

由于弱变异测试效率明显高于强变异测试，Papadakis 等人[5]基于弱变异测试准则，构建变异分支，将弱变异测试问题转化为分支覆盖测试问题，从而，进一步提高弱变异测试效率. 然而，在生成测试用例时，仍采用符号执行等复杂且低效的方法；尽管也使用了进化优化方法，但是，每次仅以一个分支作为目标，生成一个测试用例.

Papadakis 等人的弱变异测试转化方法非常实用，然而，为数众多的变异体导致过多的测试目标. 采用进化优化方法生成测试用例时，一个个体仅表示一个测试用例；当一个个体包含多个测试用例时，相应的进化优化方法称为集合进化[43-44]. 集合进化非常适合于生成杀死多个变异体的测试用例. 鉴于此，本文为了提高测试用例的生成效率，基于弱变异测试转化方法，建立新的测试用例生成问题的数学模型，并通过集合进化，生成杀死所有变异

体的测试用例.

3　问题建模

本节基于弱变异测试转化方法，将变异体杀死问题转化为分支覆盖问题；基于转化后的分支覆盖问题，建立测试用例生成问题的数学模型.

3.1　变异测试转化

记被测程序为 P，s 为 P 的一条语句(取其主要表达式)，对 s 实施某变异算子，得到相应的变异语句 s'，用 s' 替换 P 中的 s，生成变异体 m. 根据弱变异测试准则，对于变异体 m，如果存在测试用例能够执行到变异语句 s'，且执行该语句后的状态发生改变，即 $s!=s'$，那么，变异体 m 被杀死. 如果将 $s!=s'$ 作为条件，构建分支语句 b，那么，覆盖 b 真分支的测试用例，必然满足杀死 m 的必要条件，即以弱变异测试准则杀死 m.

仅需通过“!=”组合变异前后的语句 s 和 s'，就能够构建新的分支 b，其中，s 为变异前语句，s' 则为 s 的一条变异语句. 因此，新分支的构建方法非常容易. 可见，每个分支 b 对应一个变异体 m，每个 m 可以产生一个 b，分支 b 和变异体 m 一一对应，因此，b 称为变异分支. 对于 P，将所有变异前后的语句进行上述组合，得到一个变异分支集 B，对应的变异体集为 M.

为了更清晰说明变异分支的构建方法，采用 MuClipse[45] 的全部 15 类方法级(Method-Level)变异算子，针对各种语句，列出所构造的变异分支，如表 1 所列. 例如，表 1 中对原语句 s，$a+b$，实施 AORB 变异算子，得到一条变异后语句 s'，$a*b$，组合得到变异分支 b，if((a+b)!=(a*b)). 当然，对 $a+b$ 实施 AORB 变异算子，还可以产生其他 3 条变异语句：$a-b$、a/b 和 $a\%b$，相应地，还可以构建 3 个变异分支，表 1 中仅列出 1 条.

表 1　变异分支构建

变异算子	全称	说明	原语句(s)	变异语句(s')	变异分支(b)
AORB	Basic Arithmetic Operator Replacement	基本算术运算符	$a+b$	$a*b$	if((a+b)!=(a*b))
AORS	Short-cut Arithmetic Operator Replacement	简洁算术运算符替换	$a+++b$	$a--+b$	if((a+1+b)!=(a-1+b))
AOIU	Insert basic Unary Arithmetic Operators	插入基本一元算术运算符	$a+b$	$-a+b$	if((a+b)!=(-a+b))
AOIS	Short-cut Arithmetic Operator Insertion	插入简洁算术运算符	$a+b$	$++a+b$	if((a+b)!=(a+1+b))
AODU	Delete Basic Unary Arithmetic Operators	删除一元算术操作符	$-a+b$	$a+b$	if((-a+b)!=(a+b))
AODS	Short-cut Arithmetic Operator Delete	删除简洁算术运算符	$a+++b$	$a+b$	if((a+1+b)!=(a+b))
ROR	Relational Operator Replacement	关系运算符替换	$a>b$	$a<=b$	if((a>b)!=(a<=b))
COR	Conditional Operator Replacement	条件运算符替换	$a>b \| c<d$	$a>b\&\&c<d$	if((a>b | c<d)!=(a>b&&c<d))
COD	Conditional Operator Deletion	条件运算符删除	$!(a>b)$	$a>b$	if(!(a>b)!=(a>b))
COI	Conditional Operator Insertion	条件运算符插入	$a>b$	$!(a>b)$	if((a>b)!=!(a>b))
SOR	Shift Operator Replacement	移位运算符	$a>>b$	$a>>>b$	if((a>>b)!=(a>>>b))
LOR	Logical Operator Replacement	逻辑运算符替换	$a\|b$	$a\&b$	if((a|b)!=(a&b))
LOI	Logical Operator Insertion	逻辑运算符插入	$a+b$	$\sim a+b$	if((a+b)!=(~a+b))
LOD	Logical Operator Deletion	一元逻辑运算符删除	$\sim a+b$	$a+b$	if((~a+b)!=(a+b))
ASRS	Assignment Operator Replacement	赋值运算符替换	$a+=b$	$a*=b$	if((a+b)!=(a*b))

将 B 的变异分支，依次插入到 P 的相应位置，使得被插入的分支，与变异前语句具有相同的可达性. 这样，所有变异体转化为等量的变异分支，并全部融入到 P 中，即可得到一个新的被测程序 P'. 鉴于变异分支在 P' 中不执行任何实质性的操作，因此，对原程序的执行不会构成任何影响. 对于转化后的程序 P'，覆盖 B 的测试用例集，必然能够以弱变异测试准则杀死 M. 至此，杀死 P 的所有变异体 M 的变异测试问题，转化为以 P' 的变异分支集 B 为目标分支的覆盖测试问题.

图 1 以三角形分类程序 Triangle[14] 为例，说明弱变异测试的转化过程. 其中，(a)为 Triangle 的部分代码；利用变异测试工具 MuClipse，对语句 5 实施 ROR 变异算子，得到的变异分支如(b)所示；对语句 6 实施 AORB 变异算子，得到的变异分支如(c)所示. 将(b)和(c)的分支植入到原程序中，其中，(b)所示的分支被植入到(a)的语句 5 之前，(c)所示的分支被植入到语句 6 之前. 将 Triangle 的所有可变异语句，都进行上述操作，即可得到转化后的新程序 P'.

```
   ...
1. if(a<=0||b<=0||c<=0){
2.     return 4;
3. }
4. int trian=0;
5. if(a==b){
6.     trian=trian+1;
7. }
8. if(a==c){
9.     trian=trian+2;
10.}
11.if(b==c){
12.    trian=trian+3;
13.}
   ...
```

(a) Triangle 的部分代码

```
if((a==b)!=(a<b))
if((a==b)!=(a!=b))
if((a==b)!=(a>b))
if((a==b)!=(a<=b))
if((a==b)!=(a>=b))
```

(b) 对(a)中语句 5 实施 ROR 变异算子后，得到的变异分支

```
if((trian+1)!=(trian-1))
if((trian+1)!=(trian/1))
if((trian+1)!=(trian*1))
if((trian+1)!=(trian%1))
```

(c) 对(a)中语句 6 实施 AORB 变异算子后，得到的变异分支

图 1 弱变异测试转化过程

这种转化方法能够进一步减少弱变异测试代价. 覆盖 P' 中所有变异分支的测试用例集，必然能够以弱变异测试准则杀死所有变异体. 但是，P' 中融入大量的变异分支，使得被测程序异常复杂. 事实上，选择 MuClipse 的所有方法级变异算子，对 Triangle 实施变异，共产生 325 个变异体，即 P' 新增了 325 个变异分支. 可见，该转化方法显著增加了转化后问题求解的难度.

尽管进化优化方法能够生成期望的测试用例集，但是，一次运行仅生成一个测试用例，使得测试用例生成的效率大大降低. 鉴于此，建立合适的测试用例生成问题的数学模型，以提高测试用例生成的效率，是非常必要的.

3.2 测试用例生成问题的数学模型

通过变异测试转化，不需要同时执行变异体和原程序，仅需分析 P' 中每个变异分支是否被覆盖，根据弱变异测试准则，即可判定某变异体是否被杀死. 因此，覆盖 B 的测试用例集，就是所要生成的测试用例集. 然而，对于转化后的新程序 P'，由于融入太多的变异分支，使得测试用例生成的效率显著降低.

对于转化后的被测程序 P'，需要覆盖的分支集为 B，包含 $|B|$ 个变异分支. 设测试输入 $\vec{x}\in D$，D 为输入域，对于变异分支 $b_i\in B$，设 $\vec{x}$ 对 b_i 真分支的分支距离为 $f_i(\vec{x})$[34]. 以 $\vec{x}$ 为决策变量，分支距离 $f_i(\vec{x})$ 为目标函数，记为 y_i，那么，y_i 取得最小值的充要条件是 $\vec{x}$ 覆盖 b_i 的真分支.

对于简单谓词的分支距离，其计算方法如表 2 所列；而复合谓词的计算方法则如表 3 所列. 表 3 中 $dist_A(\vec{x})$($dist_B(\vec{x})$)表示，以 $\vec{x}$ 作为测试输入，复合谓词 A(B)的分支距离. 对于本文如果采用“!=”连接的是两个复合谓词，例如 $A!=B$，可以转化为 $(!A\&\&B)\|(A\&\&!B)$的形式.

表 2 简单谓词分支距离计算方法

分支条件	分支距离	
	T	F
$a>=b$	0	$\|a-b\|$
$a>b$	0	$\|a-b\|+0.0001$
$a<=b$	0	$\|a-b\|$
$a<b$	0	$\|a-b\|+0.0001$
$a==b$	0	$\|a-b\|$
$a!=b$	0	1

表 3 复合谓词分支距离计算方法

复合形式	分支距离
$A\&\&B$	$dist_A(\vec{x})+dist_B(\vec{x})$
$A\|\|B$	$\min(dist_A(\vec{x}),dist_B(\vec{x}))$

这样，覆盖分支 b_i 的测试用例生成问题，就转化为函数 $f_i(\vec{x})$ 的最小化问题. 由于共有 $|B|$ 个变异分支，因此，共有 $|B|$ 个最小化问题，其数学表示为

$$\begin{aligned}&\min\ (f_i(\vec{x}))\\&\text{s.t. } \vec{x}\in D;\\&i=1,2,\cdots,|B|\end{aligned}\tag{1}$$

例如，变异分支 b_i 为“if((a==b)!=(a<b))”(如图 1(b)所示). 该分支条件为复合谓词，等价于“(a==b)||(a<b)”. 因此，以 $\vec{x}$ 为输入，其分支距离 $f_i(\vec{x})$，为两个简单谓词“(a==b)”和“(a<b)”分支距离的复合形式：

$$\begin{aligned}&f_i(\vec{x})=\min(dist_i^1(\vec{x}),dist_i^2(\vec{x}))\\&dist_i^1(\vec{x})=\begin{cases}0, & a=b\\|a-b|, & \text{其他}\end{cases}\\&dist_i^2(\vec{x})=\begin{cases}0, & a<b\\|a-b|+0.0001, & \text{其他}\end{cases}\end{aligned}\tag{2}$$

式(2)中，$dist_i^1(\vec{x})$ 为简单谓词“(a==b)”的分支距离，$dist_i^2(\vec{x})$ 为“(a<b)”的分支距离. 当 $f_i(\vec{x})=0$ 时，$\vec{x}$ 覆盖“if((a==b)!=(a<b))”的真分支. 覆盖 b_i 的测试输入，可能同时覆盖 B 的其他

分支.因此,选取未被覆盖的变异分支 $b_i \in B$,作为下一个目标分支,生成覆盖 b_i 真分支的测试用例,同时记录 B 中其他被覆盖的分支,直至 B 中所有分支都被覆盖.

对于式(1)中的每一个最小化问题,仅生成一个测试用例.为了生成覆盖 B 的测试用例集,必须反复执行优化过程.下一小节将建立一个新的测试用例生成问题的数学模型,通过求解该模型,能够生成覆盖 B 的测试用例集.

3.3　测试用例集生成问题的数学模型

鉴于式(1)中的决策变量 $\vec{x}$ 仅表示一个测试输入,因此,每次求解仅能生成一个测试用例.如果期望生成覆盖 B 中所有变异分支的测试用例集,需要对式(1)进行多次求解.为了提高测试用例生成效率,本节将建立一个新的数学模型,使得一次求解能够得到覆盖 B 的所有变异分支的测试用例集.

令 $X=(\vec{x}_1,\vec{x}_2,\cdots,\vec{x}_m),\vec{x}_k\in D,k=1,2,\cdots,m,1\leq m\leq|B|$,那么,$X$ 为一个测试输入集.对于变异分支 $b_i\in B$,我们期望存在能够覆盖 b_i 的 $\vec{x}_k$,即 $f_i(\vec{x}_k)=0$.令 $F_i(X)=\min(f_i(\vec{x}_1),f_i(\vec{x}_2),\cdots,f_i(\vec{x}_m))$,那么,$\exists\vec{x}_k,\ni f_i(\vec{x}_k)=0,k\in\{1,2,\cdots,m\}$ 的充要条件为 $F_i(X)=0$.

对于变异分支集 B,令 $F(X)$ 为反映测试输入集 X 覆盖 B 的函数.以 X 为决策变量,$F(X)$ 为目标函数,那么,$F(X)$ 取得最小值的充要条件是 X 覆盖 B 中的所有变异分支,即 $F_1(X),F_2(X),\cdots,F_{|B|}(X)$ 同时取得最小值.因此,基于变异测试的测试用例生成问题的数学模型可以表示为

$$\begin{aligned}&\min F(X)=(F_1(X),F_2(X),\cdots,F_{|B|}(X))\\&\text{s.t. } X=(\vec{x}_1,\vec{x}_2,\cdots,\vec{x}_m);\\&\quad\vec{x}_k\in D,\ k=1,2,\cdots,m\end{aligned}\tag{3}$$

比较式(1)和(3)可知,① 式(3)的决策变量包含若干测试输入,而式(1)的决策变量仅表示一个测试输入;② 式(3)以 X 覆盖 B 的所有变异分支的情况,作为优化目标,而式(1)仅以 $\vec{x}$ 覆盖变异分支 b_i 的情况,作为优化目标;③ 对式(3)的一次求解,将生成一个覆盖 B 的测试用例集,而对式(1)的一次求解,仅生成覆盖 b_i 的一个测试用例,这样一来,为了生成覆盖 B 的测试用例集,需要多次求解式(1).由此可知,通过求解式(3),能够生成覆盖所有变异分支的测试用例集,它们即可杀死所有的变异体.

下一节将给出通过集合进化求解式(3),生成测试用例的方法.

4　基于集合进化的测试用例生成

本节阐述用于求解式(3)的集合进化优化方法,包括适应度函数、进化算子以及算法步骤.

4.1　适应度函数

根据程序输入的类型,采用合适方法,得到与某一决策变量取值对应的个体编码.不失一般性,仍记决策变量 $X=(\vec{x}_1,\vec{x}_2,\cdots,\vec{x}_m)$ 的编码为 X,那么,X 覆盖 B 的所有变异分支的情况,可以通过 $\sum_{i=1}^{|B|}F_i(X)$ 反映.考虑到 $\sum_{i=1}^{|B|}F_i(X)$ 的取值范围难以确定,为了评价 X 的性能,对 $\sum_{i=1}^{|B|}F_i(X)$ 进行归一化处理是非常必要的.记归一化 $\sum_{i=1}^{|B|}F_i(X)$ 之后的函数为 $Fit(X)$,以此作为 X 的适应度函数,那么,$Fit(X)$ 可以表示为

$$Fit(X)=1-\mathrm{e}^{-\sum_{i=1}^{|B|}F_i(X)}\tag{4}$$

由式(4)可知,X 中某一分量越接近于覆盖 B 的某一变异分支,那么,$Fit(X)$ 越小;当 X 中的某一或多个分量覆盖 B 的所有变异分支时,$Fit(X)=0$.

需要说明的是,在变异测试中,对于某一被测程序,往往存在一或多个等价变异体.利用第 3.1 节的转化方法,得到的转化后程序中,也存在一定数量的变异分支,使得任何测试用例都不可能覆盖它们,这类变异分支称为不可覆盖变异分支.对于不可覆盖变异分支,不存在 X,使得 $Fit(X)=0$.为了避免不可覆盖变异分支对测试用例生成的影响,在生成测试用例之前,我们通过静态分析方法,排除这些变异分支.

4.2　进化算子

本小节将设计集合进化算子,包括交叉、变异以及选择算子.

(1) 交叉算子

交叉是产生新个体的主要来源.鉴于式(3)的决策变量包含若干测试用例,因此,不仅考虑个体之间的交叉,还考虑个体内部的交叉,以提高测试用例的性能.

对于个体 $X^1=(\vec{x}_1^1,\vec{x}_2^1,\cdots,\vec{x}_m^1)$ 和 $X^2=(\vec{x}_1^2,\vec{x}_2^2,\cdots,\vec{x}_m^2)$,其中,$\vec{x}_i^1=(x_i^{11},x_i^{12},\cdots,x_i^{1l'}),i=1,2,\cdots,m$.选择 $\vec{x}_k^1$ 为交叉点,那么,个体 X^1 和 X^2 之间的交叉如图 2 所示,图中,通过交叉产生的新个体分别记为 $X^{1'}$ 和 $X^{2'}$.

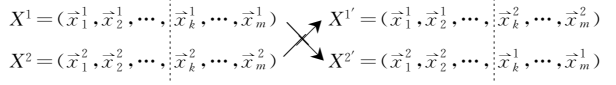

图 2 个体之间的交叉

对于个体 $X^1=(\vec{x}_1^1,\vec{x}_2^1,\cdots,\vec{x}_k^1,\cdots,\vec{x}_m^1)$，$\vec{x}_i^1=(x_i^{11},x_i^{12},\cdots,x_i^{1l})$ 和 $\vec{x}_j^1=(x_j^{11},x_j^{12},\cdots,x_j^{1l})$ 分别为 X^1 的两个测试输入. 选择 x_i^{1k} 为交叉点，那么，个体 X^1 内部的交叉如图 3 所示. 记通过交叉产生的两个新的测试输入分别为 $\vec{x}_i^{1'}$ 和 $\vec{x}_j^{1'}$，用 $\vec{x}_i^{1'}$ 和 $\vec{x}_j^{1'}$ 代替 X^1 中的 $\vec{x}_i^1$ 和 $\vec{x}_j^1$，得到的新个体分别为 $X^{1'}$ 和 $X^{1''}$.

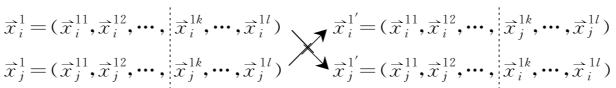

图 3 个体内部的交叉

(2) 变异算子

在集合进化中，变异是产生新个体的辅助手段. 对于个体 X^1，首先，选择测试输入 $\vec{x}_i^1$ 实施变异操作，即生成新的测试输入 $\vec{x}_i^{1'}\in D$，且 $\vec{x}_i^{1'}\neq\vec{x}_i^1$；然后，用 $\vec{x}_i^{1'}$ 代替 X^1 中的 $\vec{x}_i^1$，从而得到一个新个体 $X^{1'}$.

(3) 选择算子

在集合进化中，通过选择算子，确定遗传到下一代的个体. 交叉和变异操作不断产生新的个体，并加入到当前种群中. 选择这些个体进入下一代的方法是：首先，计算这些个体的适应值；然后，对种群中个体按照适应值大小排序；最后，选择适应值最小的若干个体，形成下一代种群.

4.3 算法步骤

本文提出的测试用例生成过程，如算法 1 所示.

算法 1. 基于集合进化的测试用例生成.

```
输入：集合进化参数
输出：测试用例集
BEGIN
  setParameters(); //参数设置
  initialize(pop); //初始化种群
  generation←0;  //进化代数
  DO WHILE (generation<max_allow_evolve)
    //如果大于最大迭代次数，终止
    generation←generation+1;
    inner_crossover(pop); //个体内部的交叉
    inter_crossover(pop); //个体之间的交叉
    mutate(pop); //变异操作
    evaluate(pop);//计算个体的适应值
    select(pop);  //选择操作
    IF (pop.getChrosome(0).getFitness()=0) THEN
      BREAK; //如果个体最小适应值为 0，终止
    END IF
  END WHILE
END
```

5 实 验

本节通过实验评价所提方法的性能，首先，列出需要解决的问题；然后，介绍采用的基准与工业程序；接着，说明实验过程；最后，分析实验结果.

5.1 需要解决的问题

本文提出一种基于变异分析和集合进化的测试用例生成方法，为了评价该方法的性能，需要回答如下 4 个问题：

(1) 所提方法能否降低测试用例生成的成本？这里，通过生成测试用例需要的迭代次数和时间，反映测试用例生成的成本.

(2) 生成的测试用例能否基于弱变异测试准则有效杀死变异体？这里，通过基于弱变异测试准则，被杀死的变异体个数与非等价变异体个数的比，反映测试用例杀死变异体的有效性.

(3) 生成的测试用例能否基于强变异测试准则有效杀死变异体？这里，通过基于强变异测试准则，被杀死的变异体个数与非等价变异体个数的比，反映测试用例杀死变异体的有效性.

(4) 集合进化参数的取值是否影响所提测试用例生成方法的性能？鉴于集合规模是集合进化特有的参数，因此，这里通过集合规模的取值对生成测试用例的性能，反映集合进化参数对所提方法的影响.

5.2 被测程序

本节选择 13 个 Java 源程序，作为被测程序，如表 4 所列，表中 J1、J2 和 J3 选自文献[5]的变异测试实验程序；J4 不但是[5]的实验程序，也是文献[14-15]的案例分析程序；J5 是文献[46]的变异测试实验程序；J6～J12 是 Apache 开源项目的程序类①，其中，J6、J8、J9、J12 和 J13 曾作为文献[3]的部分实验程序. 在所有 13 个实验程序中，J1～J5 是变异测试常用的基准程序；J6～J13 常作为变异测试或其他软件测试的部分程序.

利用 Metrics 1.3.6 统计程序的基本信息②，可以得到，这 13 个程序共包含 2412 行代码，其中，J1～J5 包含 166 行，而后 8 个程序类占 90% 以上的代码行数. 此外，这些实验程序共包含 151 个方法. 鉴于这些方法中的多数为形如“$getXXX()$”、“$setXXX()$”的简单方法，因此，仅测试 46 个较复杂的方法. 采用 MuClipse 1.3 的所有方法级(Method-Level)变异算子，共生成 4320 个变异体.

① http://commons.apache.org
② http://sourceforge.net/projects/metrics

表 4　被测程序

ID	程序	行数	方法		变异体个数	说明
			总数	被测试个数		
J1	TranshAndOut	30	2	2	111	未知
J2	Mid	26	1	1	115	3 个整数的最小值
J3	FourBalls	28	1	1	213	4 个球体的相对重量
J4	Triangle	36	1	1	325	三角形类型判定
J5	Cal	46	2	2	314	两个日期之间天数
J6	MD5Crypt	107	7	2	158	org. apache. commons. codec. digest
J7	DurationFormatUtils	365	9	1	377	org. apache. commons. lang3. time
J8	HelpFormat	416	39	2	301	org. apache. commons. cli
J9	UnixCrypt	311	12	5	805	org. apache. commons. codec. digest
J10	WordUtils	173	12	2	243	org. apache. commons. lang3. text
J11	NumberUtils	636	47	21	912	org. apache. commons. lang3. math
J12	PatternOptionBuilder	96	3	3	204	org. apache. commons. cli
J13	FieldUtils	142	15	3	242	org. joda. time. field
	Sum.	2412	151	46	4320	

5.3　实验过程

实验的运行环境为 Intel(R) Core(TM)2 Duo CPU E7400@208 GHz 2.79 GHz，2.00 GB 内存，Microsoft Windows XP SP3 操作系统，以及 Eclipse SDK 4.2.2 集成开发环境. 变异体的执行和强变异测试结果的统计，都采用 MuClipse 工具. 基于 JUnit4① 框架，每个测试类封装若干测试方法，每个测试方法包含多个形如“*assertXXX*()”的测试用例.

基于弱变异测试转化方法，首先，对每个被测程序运行 MuClipse，以生成所有的变异体；然后，对变异体生成过程中的日志文件“mutation_log”自动解析，并构造相应的变异分支，得到变异分支集 B；最后，将 B 中所有分支依次插入到原程序 P 中，得到转化后的新程序 P'.

基于第 4 节的方法，开发实验的原型系统，以生成期望的测试用例. 该原型系统包含 10 个基本类和 36 个方法，分别实现基于 GA 和本文方法的测试用例生成. 鉴于 GA 是一种常用的优化方法，而且，目前 GA 已经作为基于变异分析的测试用例生成的主要方法[3,47-51]，因此，利用本文方法和 GA 分别生成测试用例，并进行相应的实验分析和比较. 为此，将式(1)中的目标函数归一化为 $fit_i(\vec{x})=1-e^{-f_i(\vec{x})}$，以此作为 GA 中进化个体 $\vec{x}$ 的适应度函数.

图 4 以分支语句“if($a==b$){$trian=trian+1$;}”(图 1 中(a)的 5～7 行)为例，列出实验的部分实例代码. 其中，(a)为利用 GA 生成测试用例的部分实例代码；(b)为采用本方法生成测试用例的部分实例代码. (a)中数组 $fx[i]$ 与 3.2 节中的 $f_i(\vec{x})$ 相对应，用于记录对应变异分支的分支距离；(b)中 $FX[i]$ 与 3.3 节中的 $F_i(X)$ 相对应，用于记录对应变异分支的分支距离.

```
…                                          …
if((a==b)!=(a<b)){                         if((a==b)!=(a<b)){
  fx[11]=0;                                  FX[11]=0;
}else{                                     }else{
  fx[11]=Math.min(Math.abs(a-b),             tmp=Math.min(Math.abs(a-b), Math.abs(a-b)+0.0001);
          Math.abs(a-b)+0.0001);             if(tmp<FX[11]) FX[11]=tmp;
}                                          }
…                                          …
if (a==b) {                                if(a==b) {
  …                                          …
  if((trian+1)!=(trian*1)){                  if((trian+1)!=(trian*1)){
    fx[22]=0;                                  FX[22]=0;
  }else{                                     }else{
    fx[22]=1;                                  if(FX[22]!=0) FX[22]=1;
  }                                          }
  …                                          …
  trian=trian+1;                             trian=trian+1;
}                                          }
…                                          …
```

(a) GA 方法生成测试用例　　(b) 本文方法生成测试用例

图 4　实例代码

① http://www.junit.org

采用 GA 生成测试用例时，设当前目标分支为 $fx[11]$所对应的变异真分支，当测试输入为 $\vec{x}=(3,2,4)$时，$fx[11]=1$，适应度函数 $fit_{11}(\vec{x})=1-e^{-1}$；通过相应的遗传操作后，当 $\vec{x}=(3,3,4)$时，$fx[11]=0$，适应度函数 $fit_{11}(\vec{x})=1-e^{-0}=0$；此时，$fit_{11}(\vec{x})$取得最小值，即 $fx[11]$所对应的变异真分支被覆盖.

采用本文方法生成测试用例时，由于 X 包含若干测试输入，依次运行这些测试输入，如果，X 能够使所有 $FX[i]$都取“0”，那么，$\sum FX[i]=0$，适应度函数 $Fit(X)=1-e^{-0}=0$，取得最小值，即 X 覆盖了所有变异真分支.

GA 和本文方法相关参数的取值如表 5 所列，表中，$p^{cr\_inter\_set}$ 表示个体之间的交叉概率，$p^{cr\_inner\_set}$ 表示个体内部的交叉概率；p^{cross} 表示 GA 的交叉概率；$p^{mutation}$ 表示 GA 和集合的变异概率. 此外，这两种方法的种群规模均为 100. 为了得到性能优越的算法参数，经过多次实验运行确定一组优化的参数取值，如表 5 所列. 当然这些算法的取值并非最优，事实上，对该方法参数取值的优化，已经超出本文的研究范围.

表 5 参数设置

参数	取值	说明
p^{cross}	0.8	GA 的个体交叉概率
$p^{mutation}$	0.2	GA 的个体变异概率
	0.1	集合进化的个体变异概率
$p^{cr\_inter\_set}$	0.7	集合进化的个体之间的交叉概率
$p^{sel\_cr\_inner\_set}$	0.1	集合进化选中执行个体内部交叉操作的概率
$p^{cr\_inner\_set}$	0.2	个体内部的交叉概率
$size^{population}$	100	GA 和集合进化的种群规模

5.4 结果与分析

对转化后的实验程序，利用 GA 和本文方法，分别生成测试用例，并回答第 5.1 小节提出的 4 个问题.

(1) 测试用例生成的成本

为了评价不同方法生成测试用例的成本，分别采用 GA 与本文方法，生成测试用例，统计生成这些测试用例需要的迭代次数和时间(单位：ms). 实验中，根据 GA 生成的测试用例数量，设置本文方法中集合的规模. 10 次实验结果的平均值，如表 6 所列，表中，迭代次数(时间)比是 GA 生成测试用例需要的迭代次数(时间)与本文方法的比值.

表 6 生成测试用例需要的迭代次数和时间

ID	GA			本文方法			比较	
	迭代次数	时间/ms	测试用例个数	迭代次数	时间/ms	测试用例个数	迭代次数比	时间比
J1	74.1	54.63	5.0	5.2	27.17	5	14.3	2.01
J2	410.5	97.60	12.2	24.6	43.39	10	16.7	2.25
J3	590.3	304.19	22.8	30.4	116.37	20	19.4	2.61
J4	2071.5	4624.36	30.9	81.2	3504.41	30	25.5	1.32
J5	14326.0	8640.16	22.1	668.3	5128.57	20	21.4	1.68
J6	550.0	804.64	6.7	123.2	499.89	6	4.5	1.61
J7	71.8	57764.81	6.9	13.4	8634.91	6	5.4	6.69
J8	5048.9	22240.35	20.9	170.3	8887.48	20	29.6	2.50
J9	67.5	391.56	5.3	6.1	325.89	5	11.1	1.20
J10	1033.8	359.61	9.4	20.6	233.61	9	50.2	1.54
J11	11086.7	37126.63	76.8	1825.6	22699.55	75	6.1	1.64
J12	9.5	69.06	8.3	2.2	52.64	7	4.3	1.31
J13	1042.6	193.20	20.5	286.9	150.56	20	3.6	1.28
Sum.	36383.2	132670.80	247.8	3258.0	50304.44	233	Avg.=16.3	Avg.=2.13

由表 6 可知，① 对于所有被测程序，采用本文方法，共生成 233 个测试用例，迭代次数为 3258.0，耗时为 50304.44 ms；采用 GA 共生成 247.8 个测试用例，迭代次数为 36383.2，耗时 132670.8 ms，迭代次数和耗时分别为本文方法的 16.3 和 2.13 倍；② 迭代次数比最大的程序是 J10，为 50.2，说明采用本文方法，能够大幅度降低生成测试用例需要的迭代次数；③ 时间比最大的程序是 J7，将近 7，说明采用本文方法，能够显著减少生成测试用例需要的时间.

上述实验结果表明，采用本文方法生成测试用例，需要的成本明显低于 GA.

(2) 测试用例对弱变异测试的有效性

根据弱变异测试转化方法，测试用例对变异分支的覆盖率，能够反映对弱变异测试的有效性. 表 7 列出了不同程序的变异分支，以及不同方法生成的测试用例对变异分支的覆盖情况.

表 7　测试用例对变异分支的覆盖情况

ID	变异分支		GA		本文方法	
	总数	不可覆盖变异分支个数	覆盖的变异分支个数	覆盖率/%	覆盖的变异分支个数	覆盖率/%
J1	111	0	111.0	100.00	111.0	100.00
J2	115	0	115.0	100.00	115.0	100.00
J3	213	0	213.0	100.00	213.0	100.00
J4	325	8	317.0	100.00	317.0	100.00
J5	314	15	299.0	100.00	299.0	100.00
J6	158	10	148.0	100.00	148.0	100.00
J7	377	29	348.0	100.00	348.0	100.00
J8	301	26	275.0	100.00	275.0	100.00
J9	805	127	656.8	96.87	657.0	96.90
J10	243	26	212.0	97.70	212.0	97.70
J11	912	132	768.8	98.56	769.9	98.71
J12	204	12	192.0	100.00	192.0	100.00
J13	242	5	237.0	100.00	237.0	100.00
Sum.	4320	390	3892.6	Avg.=99.47	3893.9	Avg.=99.49

由表 7 可知，①13 个被测程序共有 4320 个变异分支，其中，不可覆盖变异分支有 390 个；②采用 GA 生成的测试用例，覆盖了 4320−390=3930 个可覆盖变异分支中的 3892.6 个，平均覆盖率为 99.47%；③采用本文方法生成的测试用例，覆盖了 3893.9 个变异分支，平均覆盖率为 99.49%.

这说明，基于弱变异测试准则，本文方法得到变异测试有效性不弱于 GA.

(3)测试用例对强变异测试的有效性

基于强变异测试准则，分别采用 GA 和本文方法生成的测试用例，执行变异测试，根据被杀死的变异体个数与非等价变异体个数的比，计算变异得分，如表 8 所列. 鉴于强变异测试准则杀死变异体的条件，强于弱变异测试准则，因此，表 8 中等价变异体的个数为 642，多于表 7 中不可覆盖变异分支的个数.

表 8　测试用例对强变异测试的有效性

ID	变异体		GA		本文方法	
	总数	等价变异体个数	杀死的变异体个数	变异得分/%	杀死的变异体个数	变异得分/%
J1	111	29	82.0	100.00	82.0	100.00
J2	115	18	95.9	98.87	96.4	99.38
J3	213	34	175.3	98.17	179.0	100.00
J4	325	40	276.5	97.02	277.7	97.44
J5	314	43	264.4	97.57	265.1	97.83
J6	158	12	146.0	100.00	146.0	100.00
J7	377	39	325.6	96.33	326.7	96.66
J8	301	36	256.7	96.87	255.4	96.38
J9	805	156	618.8	95.35	618.2	95.25
J10	243	38	197.8	96.49	198.1	96.63
J11	912	154	735.3	97.01	736.5	97.16
J12	204	15	186.3	98.57	186.7	98.78
J13	242	28	207.7	97.06	208.6	97.48
Sum.	4320	642	3568.3	Avg.=97.64	3576.4	Avg.=97.92

由表 8 可知，①采用两种方法生成的测试用例，均能杀死超过 95%的变异体，其中，GA 生成的测试用例，能够杀死 3568.3 个变异体，变异得分为 97.64%，而本文方法生成的测试用例，能够杀死 3576.4 个变异体，变异得分为 97.92%；②采用本文方法生成的测试用例，对于程序 J1、J3 和 J6，得到 100%的变异得分，对程序 J9 的变异得分最低，为 95.25%；③利用 GA 生成的测试用例，对于程序 J1 和 J6，得到 100%的变异得分，对程序 J3 则取得 98.17%的变异得分，对程序 J9 的变异得分最低，为 95.35%.

这说明，基于强变异测试准则，采用本文方法生成的测试用例，总体上其有效性不低于 GA.

(4)集合规模对本文方法性能的影响

测试用例的数量和质量决定测试的成本和有效性[52]. 第 1 部分的实验结果表明，采用本文方法，能够生成更少的测试用例，从而降低了变异测试的成本. 鉴于本文方法生成测试用例的个数，与集合规模

密切相关,因此,本部分考察集合规模对本文方法性能的影响,并基于此,设定集合规模的合理取值.

通过设置集合规模的不同取值,生成不同个数的测试用例,统计生成这些测试用例需要的迭代次数和时间,此外,计算这些测试用例的分支覆盖率和变异得分,如表 9 所列.

表 9 最小测试用例生成成本及其有效性

ID	最小测试用例个数	迭代次数	时间/ms	变异分支覆盖率/%	变异得分/%
J1	5	5.5	28.56	100.00	100.00
J2	9	39.4	51.43	100.00	98.64
J3	10	223.1	287.16	100.00	100.00
J4	23	155.1	5927.55	100.00	97.02
J5	13	835.3	6027.92	100.00	96.38
J6	3	325.1	747.50	100.00	100.00
J7	5	60.3	43244.78	100.00	94.59
J8	16	258.9	11330.40	100.00	95.06
J9	3	8.9	410.54	96.90	94.64
J10	7	33.0	275.97	97.70	95.41
J11	68	2031.4	31850.14	98.62	96.75
J12	5	6.7	63.01	100.00	97.28
J13	15	384.5	179.67	100.00	96.23
Sum.	182	4367.2	100 424.63	Avg.=99.48	Avg.=97.08

由表 9 可知,① 对于所有 13 个被测程序,采用本文方法,生成的测试用例最少为 182 个,比表 6 中的本文方法减少了 51 个;② 本文方法生成测试用例需要的迭代次数略高于表 6 中的本文方法;③ 对于所有被测程序,本文方法生成最小测试用例,需要的时间略多于表 6 中的本文方法;④ 采用本文方法生成的最小测试用例,虽然能够保持与表 7 中本文方法相同的变异分支覆盖率,但是,强变异得分略低于表 8 中的本文方法.

这说明,无论是迭代次数还是时间,集合规模确实影响本文方法生成测试用例的性能;小的集合规模虽然生成的测试用例个数少,但是,生成这些测试用例需要的迭代次数和时间均增加,使得测试用例生成的成本也增加.

通过上述实验结果与分析,可以得到如下结论:采用本文方法生成测试用例,能够有效降低测试用例生成的成本,且生成的测试用例对于弱变异和强变异测试准则都有效.此外,集合规模也影响变异测试的执行成本和测试用例生成成本.

6 总 结

尽管弱变异测试能够降低变异测试的成本,但是,如何借助弱变异测试准则,以高效生成具有很高缺陷检测能力的测试用例,至今缺乏有效的方法.

本文基于弱变异分析,研究测试用例生成问题,建立了一种新的测试用例生成问题的数学模型,并提出了用于求解上述模型的集合进化方法,以生成期望的测试用例.主要贡献包括:(1)以若干测试用例形成的集合,作为决策变量,以这些测试用例对变异分支的覆盖情况,作为目标函数,建立了本文测试用例生成问题的数学模型,使得通过对该模型的求解,能够生成杀死所有变异体的测试用例;(2)提出了采用集合进化方法求解上述模型时,适应度函数和进化算子的设计方法,其中,适应度函数与归一化后的目标函数密切相关,进化算子不但包括个体之间和内部的交叉,还包括个体变异;(3)通过将所提方法应用于 13 个基准和工业程序的测试中,并与 GA 进行比较,从测试用例生成成本、测试用例对弱变异和强变异测试的有效性多方面,评价所提方法的性能.此外,还考察了集合规模对本文方法生成测试用例性能的影响.

需要说明的是,本文方法的集合规模在整个进化过程中是不变的,而集合规模却影响本文方法生成测试用例的性能.如果采用合适的策略,在种群进化过程中,变化集合规模,将能够进一步降低测试用例生成的成本,从而提高本文方法的性能,这是需要进一步研究的问题.此外,本文方法的性能,仅通过较小规模的被测程序评价,在规模更大、更复杂程序中的应用,也是需要进一步研究的问题.

致 谢 各位审稿专家对本文提出了宝贵评审意见,这些评审意见对提高论文水平具有很大的帮助.编辑付出了辛勤工作.在此一并致谢!

参 考 文 献

[1] Chen Jin-Fu, Lu Yan-Sheng, Xie Xiao-Dong. Research on software fault injection testing. Journal of Software, 2009, 20(6): 1425-1443(in Chinese)
(陈锦富，卢炎生，谢晓东. 软件错误注入测试技术研究. 软件学报，2009，20(6)：1425-1443)

[2] Papadakis M, Traon Y L. Mutation testing strategies using mutant classification//Proceedings of the 28th Annual ACM Symposium on Applied Computing. New York, USA, 2013: 1223-1229

[3] Fraser G, Zeller A. Mutation-driven generation of unit tests and oracles. IEEE Transactions on Software Engineering, 2012, 38(2): 278-292

[4] Lammermann F, Baresel A, Wegener J. Evaluating evolutionary testability for structure-oriented testing with software measurements. Applied Software Computing, 2008, 8(2): 1018-1028

[5] Papadakis M, Malevris N. Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing. Software Quality Journal, 2011, 19(4): 691-723

[6] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 1978, 11(4): 34-41

[7] Hamlet R G. Testing programs with the aid of a compiler. IEEE Transactions on Software Engineering, 1977, 3(4): 279-290

[8] Jia Y, Harman M. Analysis and survey of the development mutation testing. IEEE Transactions on Software and Engineering, 2011, 37(5): 649-678

[9] Acree A T. On Mutation [Ph. D. dissertation]. Georgia Institute of Technology, Atlanta, USA, 1980

[10] Budd T A. Mutation Analysis of Program Test Data [Ph. D. dissertation]. Yale University, New Haven, USA, 1980

[11] Mathur A P, Wong W E. An empirical comparison of mutation and data flow based test adequacy criteria. Software Testing, Verification and Reliability, 1994, 4(1): 9-31

[12] Mathur A P. Performance, effectiveness, and reliability issues in software testing//Proceedings of the 15th Computer Software and Applications Conference. Tokyo, Japan, 1991: 604-605

[13] Offutt A J, Rothermel G, Zapf C. An experimental evaluation of selective mutation//Proceedings of the 15th International Conference on Software Engineering. Baltimore, USA, 1993: 100-107

[14] Jia Y, Harman M. Higher order mutation testing. Information and Software Technology, 2009, 51(10): 1379-1393

[15] Jia Y, Harman M. Constructing subtle faults using higher order mutation testing//Proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation. Beijing, China, 2008: 249-258

[16] Polo M, Piattini M, Garcia-Rodriguez I. Decreasing the cost of mutation testing with second-order mutants. Software testing, Verification and Reliability, 2009, 19(2): 111-131

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

[18] DeMillo R A, Offutt A J. Constraint-based automatic test data generation. IEEE Transactions on Software Engineering, 1991, 17(9): 900-910

[19] DeMillo R A, Offutt A J. Experimental results from an automatic test case generator. ACM Transactions on Software Engineering and Methodology, 1993, 2(2): 109-127

[20] Horgan J R, Mathur A P. Weak mutation is probably strong mutation. Purdue University, Lafayette, USA: Technical Report SERC-TR-92-P, 1990

[21] Offutt A J, Lee S D. How strong is weak mutation//Proceedings of the 4th Symposium on Software Testing, Analysis, and Verification. New York, USA, 1991: 200-213

[22] Offutt A J, Lee S D. An empirical evaluation of weak mutation. IEEE Transactions on Software Engineering, 1994, 20(5): 377-344

[23] Durelli V H S, Offutt J, Delamaro M E. Toward harnessing high-level language virtual machines for further speeding up weak mutation testing//Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation. Montreal, Canada, 2012: 681-690

[24] Kim S, Ma Y, Kwon Y. Combining weak and strong mutation for a noninterpretive Java mutation system. Software Testing, Verification and Reliability, 2013, 23(8): 647-668

[25] Papadakis M, Malevris N. Mutation based test case generation via a path selection strategy. Information and Software Technology, 2012, 54(9): 915-312

[26] Offutt A J. Automatic Test Data Generation [Ph. D. dissertation]. Georgia Institute of Technology, Atlanta, USA, 1988

[27] Offutt A J, Jin Z, Pan J. The dynamic domain reduction approach for test data generation: Design and algorithms. George Mason University, Fairfax, USA: Technical Report ISSE-TR-94-110, 1994

[28] Offutt A J, Jin Z, Pan J. The dynamic domain reduction procedure for test data generation. Software: Practice and Experience, 1999, 29(2): 167-193

[29] Sen K, Marinov D, Agha G. CUTE: A concolic unit testing engine for C//Proceedings of the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lisbon, Portugal, 2005: 263-272

[30] Tillmann N, Halleux J. Pex-white box test generation for .NET //Proceedings of the 2nd International Conference on Tests and Proofs. Prato, Italy, 2008: 134-153

[31] Liu Xin-Zhong, Xu Gao-Chao, Hu Liang, et al. An approach for constraint-based test data generation in mutation testing. 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. 计算机研究与发展，2011，48(4)：617-626)

[32] Shan Jin-Hui，Gao You-Feng，Liu Ming-Hao，et al. A new approach to automated test data generation in mutation testing. 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生产方法. 计算机学报，2008，31(6)：1025-1034)

[33] McMinn P. Search-based software test data generation：A survey. Software Testing，Verification and Reliability，2004，14(2)：105-156

[34] Korel B. Automated software test data generation. IEEE Transactions on Software Engineering，1990，16(8)：870-879

[35] Wegener J，Baresel A，Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology，2011，43(14)：841-854

[36] Gong D，Yao X. Testability transformation based on equivalence of target statements. Neural Computing and Applications，2012，21(8)：1871-1882

[37] Gong D，Tian T，Yao X. Grouping target paths for evolutionary generation of test data in parallel. The Journal of System and Software，2012，85(11)：2531-2540

[38] Harman M，S. Kim G，Lakhotia K，et al. Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem//Proceedings of the International Workshop on Search-Based Software Testing. Paris，France，2010：182-191

[39] Fraser G，Arcuri A. Whole test suite generation. IEEE Transactions on Software Engineering，2013，39(2)：276-291

[40] Zhang Yan，Gong Dun-Wei. Evolutionary generation of test data for paths coverage based on scarce data capturing. Chinese Journal of Computers，2013，36(12)：2429-2440(in Chinese)
(张岩，巩敦卫. 基于稀有数据扑捉的路径覆盖测试数据进化生成方法. 计算机学报，2013，36(12)：2429-2440)

[41] Gong Dun-Wei，Ren Li-Na. Evolutionary generation of regression test data. Chinese Journal of Computers，2014，37(3)：489-499(in Chinese)
(巩敦卫，任丽娜. 回归测试数据进化生成. 计算机学报，2014，37(3)：489-499)

[42] Souza F C，Papadakis M，Durelli V，et al. Techniques for test data generation for mutation testing：A systematic mapping//Proceedings of the 17th Ibero-American Conference on Software Engineering. Pucon，Chile，2014：419-432

[43] Zitzler E，Thiele L，Bader J. On set-based multi-objective optimization. IEEE Transactions on Evolutionary Computation，2010，14(1)：58-79

[44] Gong D，Wang G，Sun X. Set-based genetic algorithms for solving many-objective optimization problems//Proceedings of the Computational Intelligence. Guildford，UK，2013：96-103

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

[46] Ammann P，Offutt J. Introduction to Software Testing. Cambridge，UK：Cambridge University Press，2008

[47] Masud M，Nayak A，Zaman M，et al. Strategy for mutation testing using genetic algorithms//Proceedings of the Conference on Electrical and Computer Engineering. Saskatoon，Canada，2005：1049-1052

[48] Mishra K K，Tiwari S，Kumar A，et al. An approach for mutation testing using elitist genetic algorithm//Proceedings of the International Conference on Computer Science and Information Technology. Chengdu，China，2010：426-429

[49] Rad M，Akbari F，Bakht A. Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing//Proceedings of the Computational Intelligence and Software Engineering. Wuhan，China，2010：1-6

[50] Louzada J，Camilo-Junior C G，Vincenzi A et al. An elitist evolutionary algorithm for automatically generating test data//Proceedings of the IEEE Congress on Evolutionary Computation. Brisbane，Australia，2012：1-8

[51] Haga H，Suehiro A. Automatic test case generation based on genetic algorithm and mutation analysis//Proceedings of the International Conference on Control System，Computing and Engineering. Penang，Malaysia，2012：119-123

[52] Nie Chang-Hai，Xu Bao-Wen. A minimal test suite generation method. Chinese Journal of Computers，2003，26(12)：1690-1695(in Chinese)
(聂长海，徐宝文. 一种最小测试用例集生成方法. 计算机学报，2003，26(12)：1690-1695)

ZHANG Gong-Jie，born in 1979，Ph. D. candidate，lecturer. His main research interest is software testing.

GONG Dun-Wei，born in 1970，Ph. D.，professor，Ph. D. supervisor. His main research interests include search-based software engineering，intelligent optimization and control.

YAO Xiang-Juan，born in 1975，Ph. D.，associate professor. Her research interest is search-based software engineering.

Background

Mutation analysis is commonly used to evaluate the quality of test cases, and test cases that are mutation adequate reveal more faults than the ones that satisfy other structural coverage criteria. As a fault-based testing technique, therefore, mutation analysis is quite effective at improving the quality of test cases, and attracts widespread attention from both research and industry. However, the high cost in mutation testing makes it a distance from practical applications.

It is no doubt that mutant reduction is an effective way to reduce mutation cost. However, the reduction rate and effectiveness need to be further improved. From the view of optimizing the execution of the traditional mutation testing (strong mutation testing), weak mutation is an effective alternative to its strong counterpart, however, the reduction in time consumption from weak mutation is only around 50%. More importantly, existing methods for generating test cases based on mutation analysis target to kill only one mutant and generate only one test case at a time. This suggests that it is necessary to study a new approach to improving the efficiency of test case generation based on mutation analysis.

In this paper, we focus on improving the efficiency of test case generation based on weak mutation analysis. In our method, a new program is first formed by fusing all mutant branches constructed from statements before and after mutation into the original program, which transforms the weak mutation testing into branch coverage testing. Then, a novel problem solving model for the transformed program is established. Finally, the appropriate fitness function and genetic operators are designed when solving the established model by using set-based evolutionary optimization. Different from previous work, our approach can generate a test case set for a program under test at a time. We evaluated the proposed approach by performing experimental studies on thirteen benchmark and industrial programs and comparing with a commonly used genetic algorithm, and make a conclusive evaluation that the proposed approach can effectively generate test cases with high defect detection ability.

This research is jointly sponsored by the National Natural Science Foundation of China (61375067, 61203304), the Natural Science Foundation of Jiangsu Province (BK2012566), and the Fundamental Research Funds for the Central Universities (2012QNA41).