Software Engineering and Applications 软件工程与应用, 2018, 7(2), 99-109
Published Online April 2018 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2018.72012

Hans汉斯

A Method Combined with Data Flow Constraint for Generating Mutation Test Cases

Xiaozhi Du*, Qianyao Qiang, Linting Huang

School of Software, Xi'an Jiaotong University, Xi'an Shaanxi
Email: *xzdu@xjtu.edu.cn

Received: Apr. 6th, 2018; accepted: Apr. 17th, 2018; published: Apr. 24th, 2018

Abstract

Mutation testing is a powerful method of software testing. As a means of finding faults and uncovering test suite defects, the method of test case generation is crucially important. If the test case can generate efficiently and achieve a higher mutation score, it has the potential to help mutation testing be widely adopted. At present, most of the mutation test case generation methods are based on the analysis of the control flow graph, using only the control flow constraint between the statements to guide the generation of test cases, without considering the influence of the data flow constraint among the statements. In this paper, a mutation test case generation method combined with data flow constraint is proposed. First, the control flow constraint and the data flow constraint were combined to model the fitness function; second, the model was used in genetic algorithms to guide the selection and evolution of the test cases. Experiments show that the average iteration number is reduced by 60.53% when generating the same scale test set compared to HGA, and the generated test cases get a 27.9% higher mutation score than random method, showing a better error detection ability.

Keywords

Mutation Testing, Test Case Generation, Fitness Function, Mutation Score

一种结合数据流约束的变异测试用例生成方法

杜小智*，强倩瑶，黄琳婷

西安交通大学软件学院，陕西 西安
Email: *xzdu@xjtu.edu.cn

收稿日期：2018年4月6日；录用日期：2018年4月17日；发布日期：2018年4月24日

*通讯作者。

文章引用: 杜小智, 强倩瑶, 黄琳婷. 一种结合数据流约束的变异测试用例生成方法[J]. 软件工程与应用, 2018, 7(2): 99-109. DOI: 10.12677/sea.2018.72012

摘　要

变异测试是一种高效的软件测试方法。作为发现故障和发现测试用例缺陷的一种手段，测试用例生成方法至关重要。如果测试用例能有效地生成并获得较高的变异评分，则有助于变异测试的广泛应用。目前大部分变异测试用例生成方法都是通过分析控制流图，仅使用语句间的控制流约束指导用例的生成，而未考虑语句间数据流约束的影响。本文提出一种结合数据流约束的变异测试用例生成方法，首先，将控制流约束和数据流约束二者结合对适应度函数进行建模；其次，将该模型应用于遗传算法指导测试用例的进化和选择。实验表明，与HGA相比，本方法在生成同样规模的测试集时平均迭代次数减少了60.5%；并且生成的测试用例获得的变异评分比随机方法平均高出了27.9%，表现出了更优的错误检测能力。

关键词

变异测试，测试用例生成，适应度函数，变异评分

1. 引言

软件测试对软件产品质量和生产率的提高有着举足轻重的作用。变异测试是一种行之有效的软件测试方法，可用于测试充分性的评价和测试用例的增强。变异测试基于熟练程序员假设和耦合效应[1]，通过使用变异算子系统地模拟程序中可能存在的各种缺陷产生变异体，然后求解能够杀死这些变异体的测试集[2]。变异测试主要应用于单元测试，也有学者将其应用于集成测试、面向对象测试、面向方面测试等方面[3] [4] [5] [6] [7]。一般情况下，根据变异评分衡量测试集发现程序变异体中错误的能力。一个能将变异体 M 与源程序 P 区分出来的测试用例必须满足三个条件[8]：可达性(必须存在一条从 M 的开始语句到变异语句的执行路径)；状态感染性(通过执行变异语句，M 和 P 的状态彼此不同)；状态传播性(通过执行变异语句，M 和 P 的状态差异一直传播到变异体执行结束)。

如何生成测试用例在变异测试中至关重要，现有的变异测试用例生成方法主要有 CBT (Constraint-Based Test data generation，基于约束的生成方法) [9]、DDR (Dynamic Domain Reduction test data generation，动态域削减方法) [10]、DRD (Domain Reduction approach with Data dependence，考虑数据依赖的域削减方法) [7] 和 GA (Generation Algorithm，遗传算法生成变异测试用例的方法) [11]。CBT 采用控制流分析变异体的可行性条件建立约束关系，是一种高效的测试用例生成方法，但 CBT 难以处理对输入变量有依赖的判断条件和表达式；DDR 根据控制流动态地求解路径约束，改进了对输入变量有依赖的判断条件和表达式的处理，弥补了 CBT 存在的缺点，但未考虑数据依赖；DRD 将数据依赖考虑到约束系统中并使用 DDR 进行求解，但未权衡加入的数据依赖结点的重要性；GA 基于控制流测试充分性准则构建适应度函数模型，考察多种程序结构进行测试，但这种测试不考虑数据流约束，往往是不充分的，且生成测试用例的效率低。

为了更高效地生成变异测试用例，本文提出了一种结合数据流约束的变异测试用例生成方法。此方法充分考虑数据流约束并且权衡所加入数据依赖结点的重要性，首先，结合控制流约束和数据流约束建立合适的适应度函数模型；然后，将测试用例求解问题转换为适应度函数优化问题，基于遗传算法所具备的全局优化功能进行求解，使用个体适应度指导测试用例的进化和选择；最终得到目标测试用例。

2. 结合数据流约束的变异测试用例生成方法

本文的方法关注变异语句中变量的定义和使用，考虑变异语句所依赖的数据流约束，使用了一种新的适应度函数模型，逐步引导测试用例过程。首先，将控制流约束和数据流约束结合起来，提供了充分的代码覆盖，共同构建适应度函数模型；然后，使用遗传算法，根据遗传算法“优胜劣汰”的进化特点，依据测试用例的适应度自适应调整搜索方向；最终高效地生成变异测试用例。图 1 为结和数据流约束的变异测试用例生成方法。如图 1 所示，本方法由适应度函数建模和测试用例生成两大部分组成。

适应度函数建模部分开始于被测试程序，分别根据程序控制流图 CFG (Control Flow Graph)和程序数据流图 DFG (Data Flow Graph)计算得到控制流约束函数 $cf\_constraint(X)$ 和数据流约束函数 $df\_constraint(X)$，然

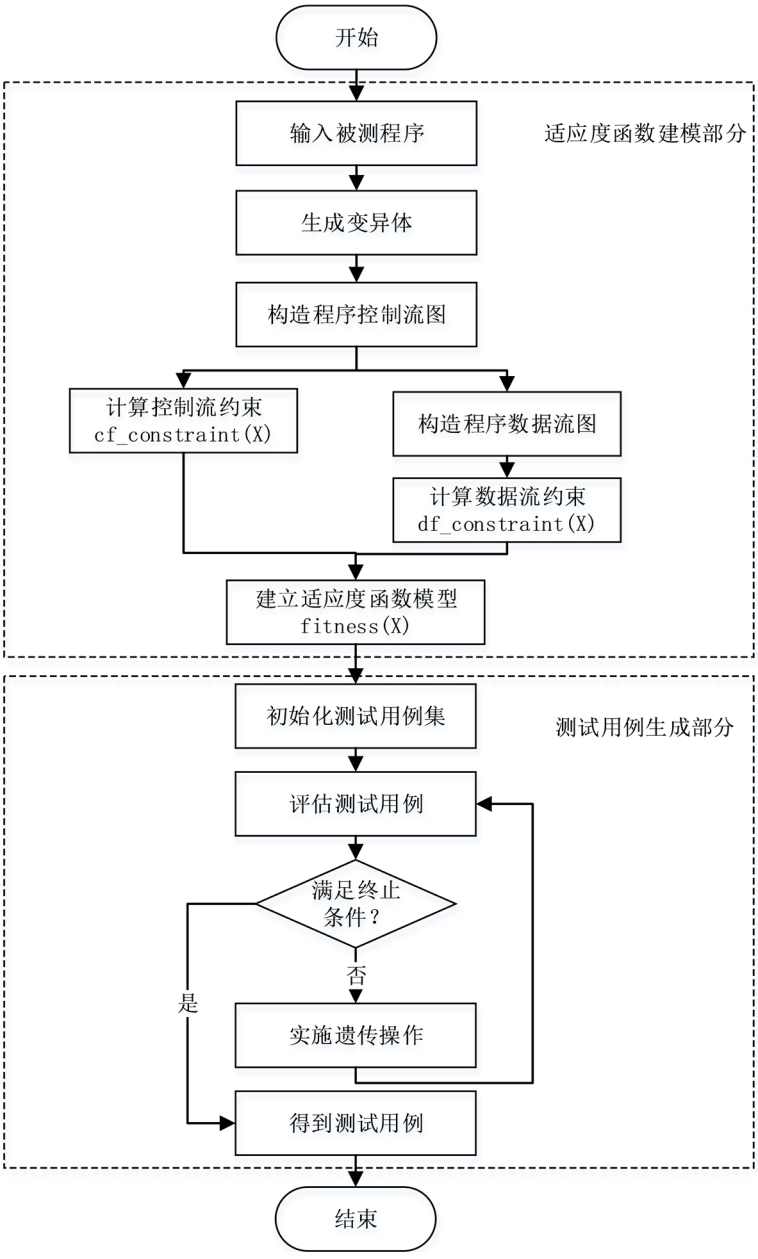

Figure 1. The method flow chart
图 1. 方法流程图

后组合二者构造出适应度函数 $fitness(X)$；测试用例生成部分将用例生成问题转化成了适应度函数 $fitness(X)$ 的优化问题，使用遗传算法优化适应度函数，通过计算每一个测试用例的适应度值，评估用例的优劣，指导对用例的搜索朝着最大化目标分支覆盖的方向，逐渐将测试集收敛至搜索空间的最优点，直至得到最终测试用例。

2.1. 适应度函数建模

适应度函数是基于搜索的方法的核心，它引导测试用例生成过程朝着搜索空间中适应度最好的领域进行，直接决定着搜索结果的好坏。本文的方法改变以往采用控制流约束建立适应度函数模型的思想，将控制流约束和数据流约束结合起来，采用分支函数叠加法建立适应度函数模型，根据个体适应度值指导评价测试用例并指导其选择和进化。适应度函数建模部分如图 2 所示，分 3 个主要步骤完成：1) 构建控制流约束函数 $cf\_constraint(X)$；2) 构建数据流约束函数 $df\_constraint(X)$；3) 建立适应度函数模型 $fitness(X)$。

1) 构建控制流约束函数 $cf\_constraint(X)$

控制流约束用于满足可达性条件，其计算依赖于被测程序的 CFG 和内部结构。首先分析 CFG 上变异语句的逼近水平，然后根据变异语句所在分支的谓词表达式计算分支距离，最后将逼近水平和分支距离线性叠加构造出控制流约束函数。

逼近水平用来度量测试用例如何覆盖目标变异语句，表示测试用例 X 的执行路径与目标变异语句的偏离程度，记作 $appr(X)$，它是根据未被执行的目标变异语句的控制依赖结点的数量计算的。举例说明，程序 1 是示例程序，图 3 是程序 1 的控制流图。假定语句 4 是程序 1 的目标测试语句，给出三组不同的输入数据：$i_1=(1,2,3)$、$i_2=(5,6,6)$、$i_3=(10,3,10)$，相应的穿越路径为 $P(i_1)=\{1,2,5,6,7\}$、$P(i_2)=\{1,2,3,5,6,7\}$ 和 $P(i_3)=\{1,2,3,5,6,7\}$。在这三组不同的测试数据中，i_2 和 i_3 都要执行条件语句 3，而 i_1 则在执行完条件语句 2 之后偏离目标语句 4 所在的分支，所以测试数据 i_2 和 i_3 比测试数据 i_1 的适应度要好。因此，分别定义 i_1、i_2 和 i_3 的逼近水平为：$appr(i_1)=1$、$appr(i_2)=0$ 和 $appr(i_3)=0$。

分支距离用来评估目标分支被选择的远近程度，表示使目标分支条件为真或假的满足程度，记作 $dist(X)$，它是根据目标变异语句所在分支的条件语句计算的，具体计算表达式如表 1 所示[12]。$dist(X)$ 越小，测试用例的执行路径对目标变异语句的覆盖程度越好。

控制流约束函数记为 $cf\_constraint(X)$，定义 $cf\_constraint(X)$ 为 $appr(X)$ 和 $normalize(dist(X))$ 的和，如式(1)所示：

$$cf\_constraint = appr(X) + normalize(dist(X)) \tag{1}$$

其中：$normalize(dist(X))$ 为标准化分支距离，如式(2)所示：

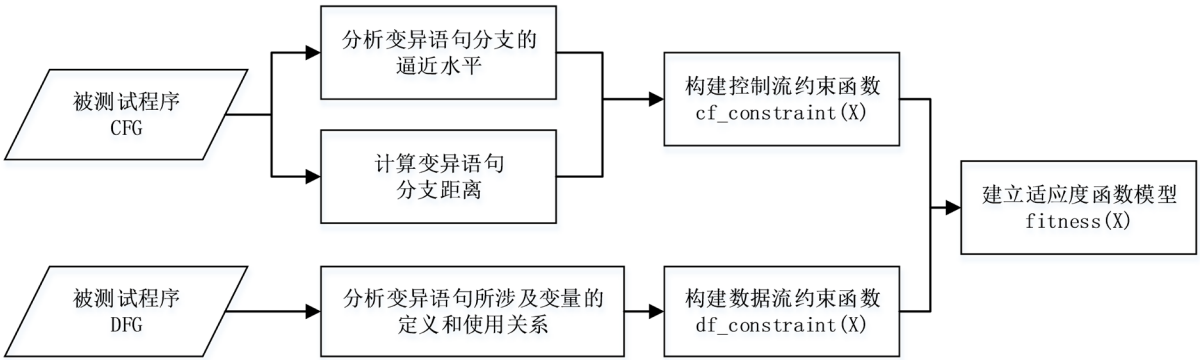

Figure 2. Mapping of fitness function modeling
图 2. 适应度函数建模示意图

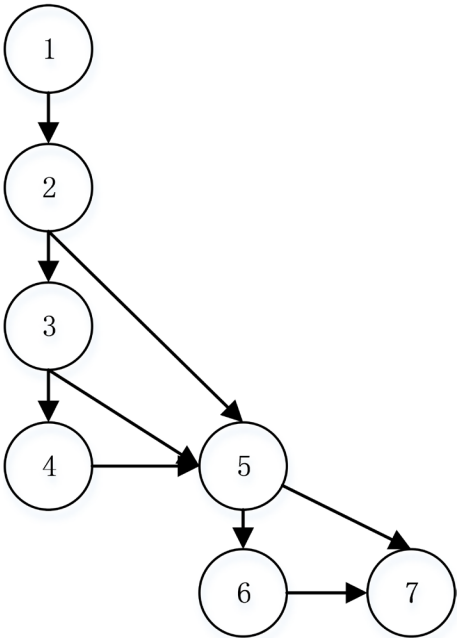

Figure 3. The control flow graph of program 1
图 3. 程序 1 的控制流图

```
1       public int function(int x,int y,int z){
2               if(x>3){
3               if(y==1){
4                       z=z+x;
                }
        }
5               if(z>=2){
6                   z=z-1;
                }
7           return z;
```

Program 1. Sample program
程序 1. 示例程序

Table 1. Branch distance calculation table
表 1. 分支距离计算表

分支条件	真分支	假分支
a == b	0	abs (a − b)
a != b	0	1
a < b	0	abs (a − b) + k
a <= b	0	Abs (a − b)
a > b	0	abs (a − b) + k
a >= b	0	abs (a − b)
a || b	min(dist(a),dist(b))	dist (a) + dist (b)
a && b	dist(a) + dist(b)	min(dist(a),dist(b))

$$normalize\left(dist\left(X\right)\right)=1-1.01^{-dist(X)} \tag{2}$$

控制流约束函数 $cf\_constraint\left(X\right)$ 的值越小，说明测试用例越满足必要性条件；反之，若控制流约束函数 $cf\_constraint\left(X\right)$ 的值越大，则说明测试用例越偏离必要性条件。

2) 构建数据流约束函数 $df\_constraint\left(X\right)$

数据流约束用于表示测试用例的执行路径对目标变量定义——使用路径的覆盖程度，由变异语句中变量的定义——使用路径覆盖定义。注意，构造 DFG 时，CFG 中的结点、边和所有路径都在 DFG 中保留，CFG 中的结点对应于 DFG 中的基本块。计算数据流约束时，首先由 DFG 计算得到变异语句变量的定义表和使用表，然后将定义表和使用表信息合并得到定义——使用表，最后根据定义——使用表计算数据流约束。

数据流约束函数记为 $df\_constraint\left(X\right)$，定义其计算如式(3)所示：

$$df\_constraint\left(X\right)=1-\frac{def-use\left(X\right)}{\sum_{i=1}^{M}def-use} \tag{3}$$

其中：$def-use\left(X\right)$ 为测试用例的执行路径覆盖目标变量的定义——使用对数量；M 为测试用例集的规模。若数据流约束函数 $df\_constraint\left(X\right)$ 的值越小，即 $\frac{def-use\left(X\right)}{\sum_{i=1}^{M}def-use}$ 的值越大，则测试用例的执行路径提供的代码覆盖越充分；反之，若数据流约束函数 $df\_constraint\left(X\right)$ 的值越大，即 $\frac{def-use\left(X\right)}{\sum_{i=1}^{M}def-use}$ 的值越小，测试用例的执行路径提供的代码覆盖越不充分。

3) 建立适应度函数模型 $fitness\left(X\right)$

建立合适的适应度函数模型是本文方法的关键，适应度作为测试用例优劣的衡量标准，指导测试集的进化和测试用例的选择。适应度函数记为 $fitness\left(X\right)$，将其定义为控制流约束函数和数据流约束函数的线性叠加，如式(4)所示：

$$fitness\left(X\right)=a*cf\_constraint\left(X\right)+b*df\_constraint\left(X\right) \tag{4}$$

其中：a、b 为控制流约束函数和数据流约束函数的权重系数，其定义域均为 $\left(0,1\right]$ 且 $a+b=1$。

X 包含全部测试用例是个体适应度函数 $fitness\left(X\right)=0$ 的充要条件。由于控制流约束函数 $cf\_constraint\left(X\right)$ 越小，测试用例越满足可达性条件，数据流约束函数 $df\_constraint\left(X\right)$ 越小，测试用例的执行路径提供的代码覆盖越充分。所以，个体适应度函数 $fitness\left(X\right)$ 越小，则测试用例的执行路径与目标变异语句间的距离越小，且对变异语句中变量的定义——使用路径覆盖越充分。因此，可将测试用例生成问题可转换为适应度函数 $fitness\left(X\right)$ 的最小化问题。

2.2. 测试用例生成

为了提高生成测试用例的效率，本文方法将测试用例生成问题转换成了适应度函数 $fitness\left(X\right)$ 的最小化问题，并使用遗传算法解决。根据适应度值动态调整测试用例的生成过程，提高其质量并且减少测试冗余。

测试用例生成过程开始于随机法生成的初始用例集，然后按照适应度函数 $fitness\left(X\right)$ 计算初始用例集中各用例的适应度，评估用例的优劣；种群进化时，选择出优良个体来参与交叉、变异操作得到新一代用例集，最后判断终止条件是否满足，若满足，则选择出目标用例，若不满足，则依据个体适应度逐渐将用例集收敛至搜索空间的最优点，不断进化，直至生成最优测试用例。具体流程如图 4 所示。

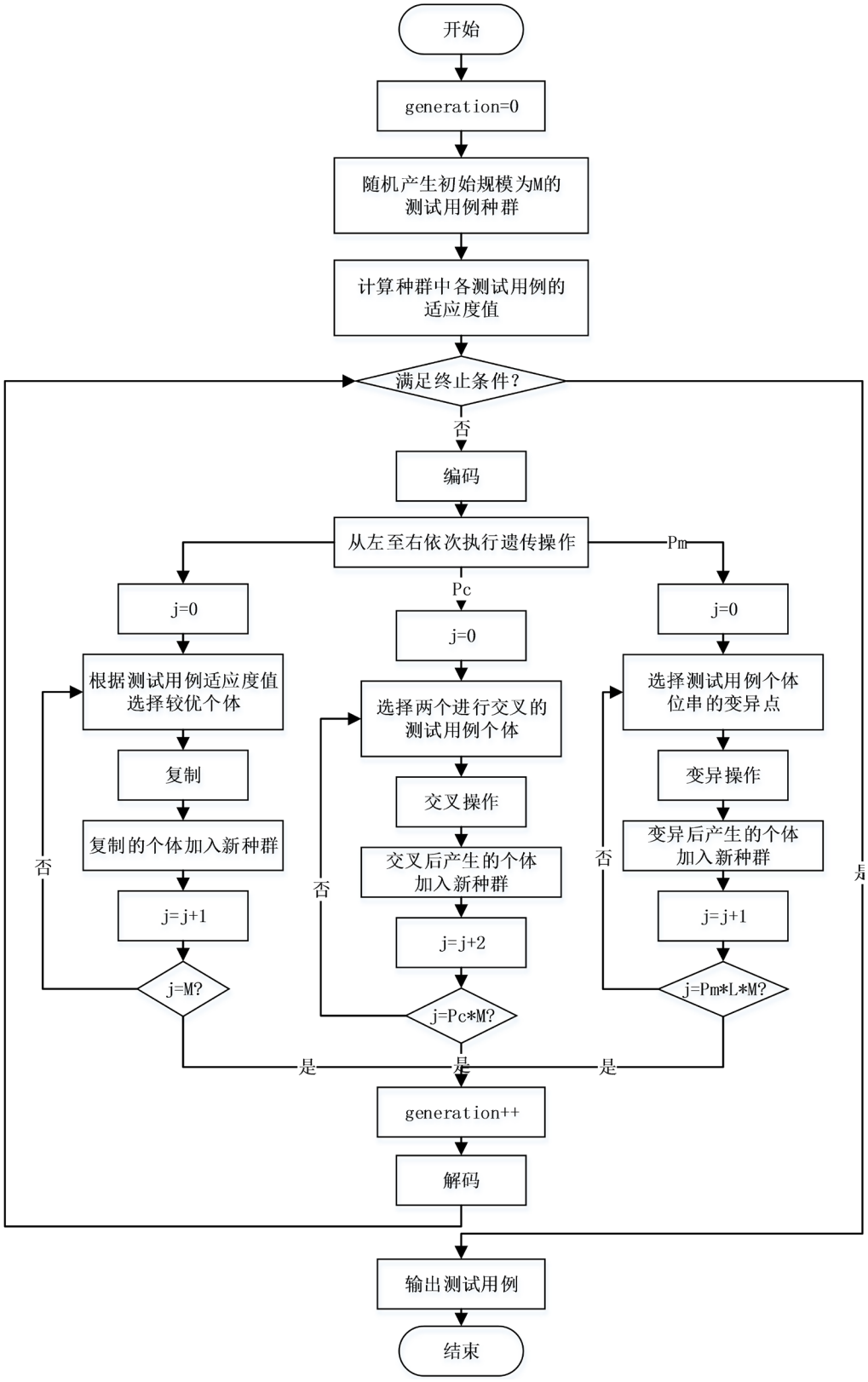

Figure 4. The test case generation flow chart
图 4. 测试用例生成流程图

本方法生成测试用例的过程，其实是适应度函数的优化过程。下面简单介绍使用遗传算法实现该方法的几个重要操作的具体操作：

1) 初始化测试用例集：为保证测试用例集的多样性，采用随机方法生成规模为 M 的初始测试用例集；

2) 编码形式：由于二进制编码易于表达、操作简单的优点，本方法采用二进制编码形式对原始个体进行编码；

3) 计算测试用例适应度：根据 $fitness(X)$ 计算测试用例的适应度以指导测试集的进化；

4) 实施选择操作：有多种不同的选择策略，本方法采用轮盘选择法，将一个圆分为种群规模大小个扇形区间，区间大小为测试用例的相对适应度值。轮盘选择法保证了各用例被选中的概率与适应度值成正比，与本文适应度函数优化的思想相一致，根据测试用例的相对适应度，将父代的优良测试用例复制到子代；

5) 实施交叉操作：此操作的主要目的就是在解空间中搜索不同的可行解。本方法以固定的交叉概率 P_c 为参照，在测试用例集中随机选择两个用例，对其实施动态概率的随机点交叉操作；

6) 实施变异操作：变异操作用于补偿基因丢失情况，保持个体多样性。本方法以固定的变异概率 P_m 为参照，对用例实施动态概率的变异操作；

7) 终止条件：当迭代次数达到预定终止代数或用例适应度达到预定值时算法终止。

3. 实验分析

为了实际证明本文方法的可行性，分别从测试用例的生成效率和检错能力两个方面进行了实验。

1) 测试用例生成效率实验结果

以三角形形状判定程序为例，对变异生成的 4 个一阶变异体进行了实验，该程序以输入的三个整数为三角形的三条边，输出三角形的形状。根据经验值和实验比较，遗传算法交叉概率 P_c 选用 0.7，变异概率 P_m 选用 0.05，最大迭代次数选用 100，种群规模 M 选择 20，对每个变异体进行 10 次实验。实验结果如表 2 所示。

从表 2 可以看出，对于使用 ROR 变异算子生成的 2 个变异体，当生成数量优于 HGA 方法的用例时，本文的方法在平均迭代次数上减少了 56.1%；对于使用 AORB 变异算子生成的 2 个变异体，当生成与 HGA 方法数量相等的测试用例时，本文的方法使用的平均迭代次数是 HGA 方法的 35%，减少了 65%。图 5 为表 2 中两种方法生成测试用例的平均迭代次数对比图，明显显示处了本方法在迭代次数上的优势，所以本的方法表现出了较高的测试用例生成效率。

2) 测试用例检错能力实验结果

选取参数个数和语句复杂度不同的程序，分别使用随机方法、依据路径覆盖的混合遗传算法 HGA 方法和本文方法生成一定数量的测试用例进行实验。程序 Mid 是一个求 3 个整数中间值的程序，该程序结构简单，主要用来测试本文方法在简单结构程序中的应用；程序 TriTyp 是三角形形状判定程序，其特点是条件语句复杂，主要用来评估本文方法在复杂语句程序测试中的应用；程序 Quad 求解一元二次方程的

Table 2. Experimental results of test case generation efficiency
表 2. 测试用例生成效率实验结果

变异算子	实验次数	平均迭代次数		生成的测试用例数		本文方法平均迭代次数占 HGA 的比率
		HGA [14]	本文方法	HGA	本文方法	
1. ROR [13] (“==”变异为“>=”)	10	65	38	8	8	58.5%
2. ROR (“<=”变异为“<”)	10	17	5	9	10	29.4%
3. AORB [13] (“+”变异为“*”)	10	5	1	10	10	20%
4. AORB (“+”变异为“−”)	10	2	1	10	10	50%

根，程序 Sample 判断两数组内数据与目标数据的相同程度，NextDate 求解输入日期的下一天，用来测试本文方法在复杂路径程序测试中的应用。实验结果如表 3 所示。

如表 3，计算和比较变异评分可知，对程序 Mid 使用 HGA 和本方法生成变异测试用例得到的变异评分相同，相比于随机方法高出了 9.5%；对程序 TriTyp 使用本方法得到的变异评分比 HGA 方法高 3.5%，比随机方法高 33.3%；对于程序 Quad、Sample 和 NextDate，使用本文方法得到的变异评分比 HGA 方法分别高出了 5.1%、4.3%和 8.5%，相比于随机方法有较为明显的提高，分别高出了 41%、26.1%和 29.6%。

三种方法所生成用例分别杀死的变异体数量对比如图 6 所示。从图中可以看出，使用本方法生成的

Table 3. Experimental results of test case error detection ability
表 3. 测试用例检错能力实验结果

被测程序	变异算子	变异体数量	被杀死变异体数量			本文方法与其他方法杀死变异体个数差值比	
			随机方法	HGA	本文方法	随机方法	HGA
Mid	ROR	21	16	18	18	9.5%	0%
TriTyp	AORB、ROR	57	32	49	51	33.3%	3.5%
Quad	ROR	39	20	34	36	41%	5.1%
Sample	AORS, ROR	23	14	19	20	26.1%	4.3%
NextDate	AORBCOR, COD, ROR	71	50	65	71	29.6%	8.5%

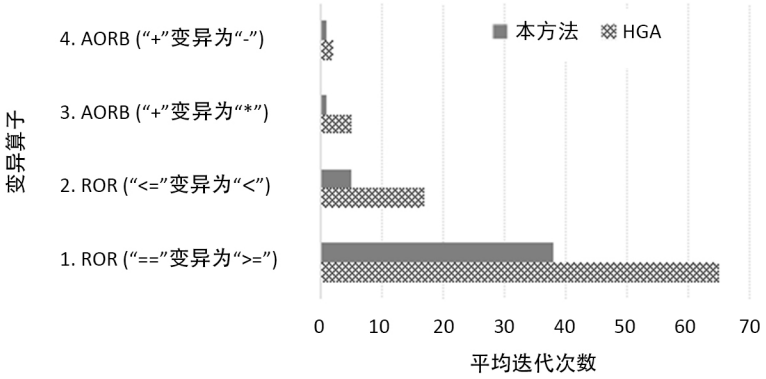

Figure 5. Average iteration number contrast
图 5. 平均迭代次数对比

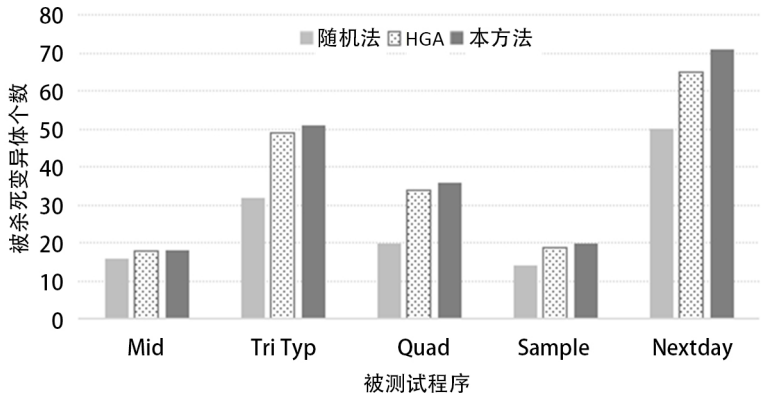

Figure 6. Comparison of the number of killed mutants
图 6. 被杀死变异体数量对比

测试用例杀死的变异体数量最多，比随机方法和 HGA 的变异体杀死数量都有一定程度的提高，表现出了更强的错误检测能力。

4. 结束语

本文提出了一种结合数据流约束的变异测试用例生成方法，充分考虑变异语句中变量数据依赖的影响。首先对被测试程序的预处理分析程序结构和语句关系，通过变异语句分支路径执行条件的组合和非冲突等价类划分量化控制流约束；然后，使用不同测试用例执行路径中变异语句所涉及变量的定义——使用路径覆盖构造数据流约束关系，结合控制流约束和数据流约束共同建立合适的适应度函数模型；最后，基于遗传算法种群进化特征和优越的全局搜索能力，依据测试用例个体适应度自适应调整测试种群的搜索方向，生成测试用例。本文通过实验证明了本方法生成测试用例的效性，以及与其他方法之间的对比实验更进一步验证了本方法的变异体检测能力。

本文的方法主要应用于软件单元测试中一阶变异测试用例的生成，将本文方法应用于自动生成杀死高阶变异体的测试用例还有待进一步的研究。满足多变异语句的分支覆盖条件，且在对变异语句中变量的定义——使用路径覆盖尽可能多的条件下，充分考虑变异语句之间的依赖关系将可能生成同时杀死多个变异体的测试用例。

基金项目

国家自然科学基金资助项目(11575138)；陕西省工业公关项目资助项目(2013K06-20)；中央高校基本科研业务费专项基金资助项目(XJJ2015122)；轨道交通工程信息化国家重点实验室开放课题项目(SKLK16-08)。

参考文献

[1] Demillo, R.A., Lipton, R.J. and Sayward, F.G. (1978) Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, **11**, 34-41. https://doi.org/10.1109/C-M.1978.218136

[2] 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.

[3] Fraser, G. and Zeller, A. (2012) Mutation-Driven Generation of Unit Tests and Oracles. *IEEE Transactions on Software Engineering*, **38**, 278-292. https://doi.org/10.1109/TSE.2011.93

[4] Fraser, G. and Arcuri, A. (2014) Achieving Scalable Mutation-Based Generation of Whole Test Suites. *Empirical Software Engineering*, **20**, 783-812. https://doi.org/10.1007/s10664-013-9299-z

[5] Delamaro, M.E., Maldonado, J.C. and Mathur, A.P. (2001)\ Interface Mutation: An Approach for Integration Testing. *IEEE Transactions on Software Engineering*, **27**, 228-247. https://doi.org/10.1109/32.910859

[6] Delgado-Pérez, P., Segura, S. and Medina-Bulo, I. (2017) Assessment of C++ Object-Oriented Mutation Operators: A Selective Mutation Approach. *Software Testing Verification & Reliability*, e1630. https://doi.org/10.1002/stvr.1630

[7] Bhatia, V. and Singhal, A. (2016) Class/Aspect Level Mutation Operators in Aspect-Oriented Software Systems. 2016 1*st India International Conference on Information Processing* (*IICIP*), Delhi, 12-14 August 2016, 1-7. https://doi.org/10.1109/IICIP.2016.7975397

[8] 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.

[9] DeMillo, R.A. and Offutt, A.J. (1991) Constraint-Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, **17**, 900-910. https://doi.org/10.1109/32.92910

[10] Offutt, A.J., Jin, Z. and Pan, J. (1999) The Dynamic Domain Reduction Procedure for Test Data Generation. *Software: Practice and Experience*, **29**, 167-193. https://doi.org/10.1002/(SICI)1097-024X(199902)29:2<167::AID-SPE225>3.0.CO;2-V

[11] 赵性颂, 顾斌. 遗传算法生成变异测试用例[J]. 计算机应用, 2009, 29(6): 262-264.

[12] Souza, F.C.M., Papadakis, M. and Traon, Y.L. (2016) Strong Mutation-Based Test Data Generation Using Hill Climbing. *International Workshop on Search-Based Software Testing*. *ACM*, Austin, 14-22 May 2016, 45-54. https://doi.org/10.1145/2897010.2897012

[13] Offutt, A.J. and Pan, J. (2015) Automatically Detecting Equivalent Mutants and Infeasible Paths. *Software Testing Verification & Reliability*, **7**, 165-192. https://doi.org/10.1002/(SICI)1099-1689(199709)7:3<165::AID-STVR143>3.0.CO;2-U

[14] Khan, R., Amjad, M. and Srivastava, A.K. (2017) Generation of Automatic Test Cases with Mutation Analysis and Hybrid Genetic Algorithm. 2017 3*rd International Conference on Computational Intelligence & Communication Technology* (*CICT*), Ghaziabad, 9-10 February 2017, 1-4. https://doi.org/10.1109/CIACT.2017.7977265