软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(9):3407−3421 [doi: 10.13328/j.cnki.jos.006291]

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

DFSampling: 一种数据流分析指导的变异体精简策略*

孙昌爱[1,2], 卫新洁[1], 刘镇贤[1], 宫云战[3]

[1](北京科技大学 计算机与通信工程学院, 北京 100083)

[2](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[3](北京邮电大学 网络技术研究院, 北京 100876)

通信作者: 孙昌爱, E-mail: casun@ustb.edu.cn

摘　要: 软件测试是一种广泛使用的软件质量保证手段. 变异测试是一种基于故障的软件测试方法, 广泛用于评估测试用例集的充分性与软件测试技术的有效性. 数量庞大的变异体导致变异测试的成本非常高. 提出一种数据流分析指导的变异体精简方法 (DFSampling), 设计了启发式规则, 基于这些规则对随机选择技术与基于路径感知的变异体精简技术 (PAMR) 进行了改进. 采用经验研究的方式评估了 DFSampling 的有效性, 比较了 DFSampling 与随机选择技术、PAMR 技术的有效性, 实验结果表明 DFSampling 是一种有效的变异体精简策略, 提高了变异测试的效率.

关键词: 软件测试; 变异测试; 数据流分析; 变异体精简; 随机选择策略

中图法分类号: TP311

中文引用格式: 孙昌爱, 卫新洁, 刘镇贤, 宫云战. DFSampling: 一种数据流分析指导的变异体精简策略. 软件学报, 2022, 33(9): 3407–3421. http://www.jos.org.cn/1000-9825/6291.htm

英文引用格式: Sun CA, Wei XJ, Liu ZX, Gong YZ. DFSampling: Mutant Reduction Technique Guided by Data Flow Analysis. Ruan Jian Xue Bao/Journal of Software, 2022, 33(9): 3407–3421 (in Chinese). http://www.jos.org.cn/1000-9825/6291.htm

DFSampling: Mutant Reduction Technique Guided by Data Flow Analysis

SUN Chang-Ai[1,2], WEI Xin-Jie[1], LIU Zhen-Xian[1], GONG Yun-Zhan[3]

[1](School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

[2](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[3](Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Software testing is a commonly used software quality assurance technique. Mutation testing is a fault-based software testing technique that is widely applied to evaluate the sufficiency of test suites and the effectiveness of software testing techniques. However, the cost of mutation testing is extremely high due to the large number of mutants. This study proposes a mutant reduction technique, DFSampling, guided by data flow analysis and designs three heuristic rules. The random selection technique and the path-aware mutant reduction technique (PAMR) are improved in line with these rules. An empirical study is conducted to evaluate the effectiveness of DFSampling and compare DFSampling with the random selection technique and the PAMR technique in terms of effectiveness. The experimental results show that DFSampling is an effective mutant reduction strategy, which can increase the efficiency of mutation testing.

Key words: software testing; mutation testing; data-flow analysis; mutant reduction; random selection strategy

变异测试是一种基于故障的软件测试方法[1]. 主要思想是, 根据被测程序的特征设计变异算子, 对被测程序使用变异算子生成变异体, 比较被测程序和变异体的输出结果. 如果输出结果不一致, 则变异体被杀死; 否则, 变异体没有被杀死. 通过计算杀死的变异体数量与全部变异体的数量的比例得到变异得分. 如果变异得分不满足测试人

* 基金项目: 国家自然科学基金 (61872039); 中国科学院软件研究所计算机科学国家重点实验室开放课题 (SYSKF1803); 中央高校基本科研业务费专项 (FRF-GF-19-019B).

收稿时间: 2020-08-28; 修改时间: 2020-10-28, 2020-12-02; 采用时间: 2020-12-18; jos 在线出版时间: 2022-07-15

员的要求, 则添加额外的测试用例, 直到把所有变异体杀死或达到预定的变异得分[2].

变异测试通过不同类型的故障来衡量测试用例集的充分性. 研究表明, 变异测试具有较强的故障检测能力. 然而, 当被测程序规模较大时, 变异测试生成大量的变异体, 需要消耗大量的时间和资源. 近年来, 在如何降低变异测试的计算开销方面已经出现了大量的研究工作[3−8], 人们从变异体精简与加速变异执行两个方面对变异测试进行改进或优化. (1) 变异体数量精简方法: 从减少变异体的数量角度优化变异测试, 主要方法包括变异体随机选择[9−11]、选择性变异算子[12−14]、高阶变异测试[15,16]、变异体聚类方法[17−19]等. 这些方法通过选取变异体子集来减少需要执行的变异体的数量, 从而减少变异测试的时间与资源; (2) 加速变异执行方法: 从缩短变异测试执行时间的角度优化变异测试, 主要方法有变异体检测优化[20−22]、变异体编译优化[23,24]、并行执行变异体[25,26]等. 这些方法通过对变异测试过程进行优化来减少变异测试的时间开销. 上述两类优化方法均有效改进了变异测试的性能.

本文从程序数据流分析的角度出发研究变异体精简问题. 由于程序中的变量必须遵循先定义后使用的规定, 相应地, 同一变量的定义变异体与使用变异体的检测存在一定关联性. 具体说来, 对于某个变量 v 而言, 在 v 定义处引入的变异体称为定义变异体, 在 v 使用处引入的变异体称为使用变异体. 当测试资源受限时, 通常需要对变异体的数量进行精简. 一个有效的做法是, 保留变量 v 的定义变异体和使用变异体中的二者之一即可, 从而实现变异体的数量精简. 通常说来, 当被测程序中存在较多变量的<定义-使用>对时, 则可能存在较多的定义变异体和使用变异体, 则可以较大程度地精简变异体数量. 为此, 本文提出一种数据流分析指导的变异体精简技术 DFSampling. DFSampling 首先分析程序中变量的<定义-使用>对, 识别出相应的定义变异体和使用变异体; 然后依据定义的变异体精简规则, 删除相应的变异体. DFSampling 在不影响变异测试的故障检测能力的前提下, 实现变异体数量精简, 提高变异测试的效率.

本文第 1 节介绍变异体精简技术方面的相关工作; 第 2 节讨论基于数据流分析的变异体精简技术 DFSampling, 包括基本原理、概念、精简规则、算法描述以及方法示例; 第 3 节采用经验研究的方式评估 DFSampling 的有效性与效率; 最后总结全文.

1 相关工作

变异测试被广泛用于衡量测试集的有效性和充分性[27], 帮助测试人员构建高效测试用例[28]. 然而变异体规模较大, 导致变异测试计算开销增加[29]. 近年来, 研究人员围绕如何降低变异测试的计算开销、提高变异测试效率开展了大量的研究工作.

人们从不同的角度研究了变异体精简方法. Acree 和 Budd[30]提出从所有变异体中随机选择一定比例的变异体, 仅用选择出来的变异体子集进行测试, 称为“随机选择法”. Mathur 等人[31]设置了不同的变异体随机选择比例, 研究了不同比例下随机选择变异测试的有效性, 实验结果表明随机选择 10% 变异体时构造的测试用例集的充分性比用所有变异体构造的测试用例集低 16%. Zhang 等人[32]采用经验研究评估了实验对象的程序规模对选择变异测试可扩展性的影响. 实验结果表明, 当程序规模或非等价变异体数量增加时, 选择的变异体数量缓慢增加, 但所选变异体比例减少; 为取得比较充分的变异得分 (例如 90%–99%), 当程序的可执行代码行数为 E 时, 选择的变异体数量为 E_c, 选择的变异体比例为 E_b (b 和 c 小于 1).

Papadakis 等人[33]提出一种将变异前后的语句转化为变异分支的方法, 将原程序的弱变异测试问题转化为新程序的分支覆盖问题. 在此基础上, Gong 等人[34]进一步提出了基于占优关系的变异体精简技术, 通过分析测试用例对变异分支的覆盖情况识别变异分支之间的占优关系, 将不被占优变异分支对应的变异体作为变异体子集进行变异测试. 实验结果表明该技术能有效减少变异体数量, 但在识别变异分支间的占优关系方面引入了较大的时间开销.

Sun 等人[35]提出一种基于路径感知的变异体精简技术, 在程序结构分析的基础上设计了基于深度优先的变异体选择启发式规则, 通过赋予启发式规则不同的优先级, 提出了变异体精简策略. 实验结果表明基于路径感知的变异体精简技术可以在不显著降低变异得分的情况下提高变异测试效率, 而且在选取同样比例的变异体前提下, 深度优先的变异体精简策略比随机选择策略更高效.

已有变异体精简技术主要从随机选择角度或者是控制流约束的角度, 探讨变异体精简策略, 并没有考虑不同变异体之间存在的数据流约束的影响. 本文提出一种基于数据流约束的变异体精简技术, 该方法充分考虑了不同变异体的变量的定义与使用之间的依赖关系, 使用程序静态分析的方法确定变异体对测试充分性评估的作用. 设计了基于数据流分析的变异体精简规则, 在测试执行前对变异体进行精简, 从而在不引入大量时间开销的基础上减少变异体的数量.

2 数据流分析指导的变异体精简技术 DFSampling

为了提高变异测试效率, 本文提出一种数据流分析指导的变异体精简技术 DFSampling. 本节介绍 DFSampling 基本原理、精简规则与算法, 最后用一个例子演示方法的使用.

2.1 研究动机

在介绍 DFSampling 之前, 我们首先示例数据流分析对变异体检测的影响. 图 1 示意一个源程序及其错误版本 (变异体). 其中, 变异体 m_1 与变异体 m_2 分别为同一变量 *temp* 的定义变异体与使用变异体; 变异体 m_3 与变异体 m_4 分别为同一变量 *y* 的定义变异体与使用变异体. 在测试资源有限的情况下, 我们考虑对变量 *temp* 与变量 *y* 处的变异体进行精简.

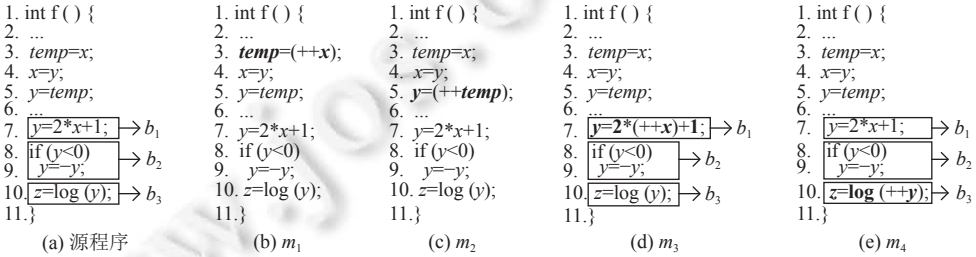

图 1 研究动机示例

对于变量 *temp* 而言, 从图 1(a) 可以看出, 变量 *temp* 在 *temp=x* 处被定义, 在 *y=temp* 处被使用. 变量 *temp* 对应的定义变异体与使用变异体分别如图 1(b)、图 1(c) 所示. 变量 *temp* 在变异体 m_1 处定义性出现 *def* (*temp*, 3), 在变异体 m_2 处使用性出现 *use* (*temp*, 5), 且变异体 m_1 与变异体 m_2 处在同一个基本块 (第 3–5 行) 中. m_1 于 *temp* 的定义变异体集 *M* (*def*, *temp*, 3), 对程序状态的直接影响是变量 *temp* 的值加 1, 并通过 *temp* 的使用将上述影响传播到第 5 行使变量 *y* 的值加 1. m_2 属于 *temp* 的使用变异体集 *M* (*use*, *temp*, 5), 对程序状态的改变是变量 *temp* 和变量 *y* 的值加 1. 由于存在变量 *temp* 的<定义-使用>对 *du* (*temp*, 3, 5), 两个变异体 m_1 与 m_2 对程序第 5 行的状态改变相似, 在进行变异测试时选择其中一个即可.

对于变量 *y* 而言, 从图 1 可以看出, 变量 *y* 在变异体 m_3 中定义性出现 *def* (*y*, 7), 在变异体 m_4 中使用性出现 *use* (*y*, 10). m_3 属于变量 *y* 的定义变异体集 *M* (*def*, y, 7), 对程序状态的影响是变量 *x*、*y* 的值改变, 通过 *y* 的使用 *use* (*y*, 10) 将上述影响传播到第 10 行, 使变量 z 的值发生改变. m_4 属于变量 *y* 的使用变异体集 *M* (*use*, *y*, 10), 对程序状态的影响是变量 *y*、*z* 的值改变. m_3 和 m_4 分别属于基本块 b_1 和 b_3, 且 b_3 是 b_1 的间接顺接块 (b_1 的直接顺接块为选择块 b_2, b_2 的直接顺接块为 b_3). 由于存在变量 *y* 的<定义-使用>对 *du* (*y*, 7, 10), m_3 和 m_4 对程序在第 10 行的状态改变相似, 在执行变异测试时选择其中一个即可. 由此可见, 变量的<定义-使用>对影响其对应的变异体的检测结果, 数据流分析提供了一种变异体精简的手段.

2.2 相关定义

借助程序分析技术分析可以得到不同变量的<定义-使用>对. 相应地, 与这些变量相关的变异体集合包含定义变异体与使用变异体. 定义变异体与使用变异体的定义如下[36].

定义 1. 定义变异体. 对于给定的变量 *v* 在 *m* 处的定义语句 *du* (*v*, *m*) 进行变异, 生成的变异体称为变量 *v* 在语

句 m 处的定义变异体, 记作 M (*def*, v, m).

定义 2. 使用变异体. 对于给定的变量 v 在 n 处的使用语句 *du* (v, n) 进行变异, 生成的变异体称为变量 v 在语句 n 处的使用变异体, 记作 M (*use*, v, n).

2.3 方法框架

本文提出数据流分析指导的变异体精简技术 DFSampling, 如图 2 所示. 首先对待测程序 P 进行数据流分析, 分别得到程序中各个变量 v 的<定义-使用>对, 记为<v, m, n>; 依据数据流分析结果对初始变异体集合 P<v, m, n>进行分析, 分别得到变量 v 的定义变异体 M (*def*, v, m) 与使用变异体 M (*use*, v, n); 依据提出的变异体精简规则, 对定义变异体集合 M (*def*, v, m) 与使用变异体集合 M (*use*, v, n) 进行变异体精简, 得到精简后的变异体集合 P'<v, m, n>. 最后在测试用例集 T 上使用精简后的变异体集合 P'<v, m, n>进行变异测试. 不难看出, DFSampling 通过数据流分析指导变异体的选择过程, 从而减少变异体数量, 提高变异测试效率.

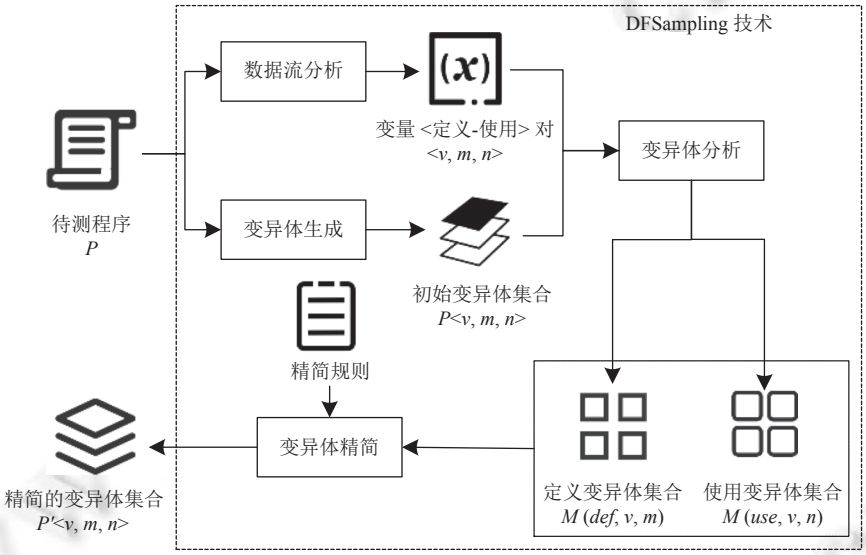

图 2 DFSampling 基本流程图

2.4 变异体精简规则

程序中的变量必须遵循先定义, 后使用的原则, 即存在数据依赖关系. 假设程序中存在变量 v 的一个<定义-使用>对, 记作<v, m, n>. 若 v 定义后被使用, 则使定义变异体 M (*def*, v, m) 的变异语句 m 被执行的测试用例 t 也能够使使用变异体 M (*use*, v, n) 的变异语句 n 被执行, 即 M (*use*, v, n) 满足可达性条件. 若 t 能够使 m 执行后 M (*def*, v, m) 的状态不同于原程序 P 的状态, 则 t 也能够使 n 执行后 M (*use*, v, n) 的状态不同于原程序 P 的状态, 即 M (*use*, v, n) 满足必要性条件. 即一个杀死定义变异体 M (*def*, v, m) 的测试用例非常有可能杀死使用变异体 M (*use*, v, n). 因此在故障检测能力方面, 定义变异体 M (*def*, v, m) 等同于使用变异体 M(*use*, v, n). 当测试资源受限时, 仅选择定义变异体, 而不选择使用变异体. 基于数据流分析指导的变异体精简规则定义如下.

精简规则 1 (R_1). 变量 v 的一个<定义-使用>对<v, m, n>, 若 $m \in Block_i \wedge n \in Block_i$, 且 $Block_i$ 为基本块 (i = 1, 2, 3, …), 则选择 m 处生成的定义变异体 M(*def*, v, m).

精简规则 2 (R_2). 变量 v 的一个<定义-使用>对<v, m, n>, 若 $m \in Block_i \wedge n \in Block_j$, 其中, $Block_i$ 为基本块、选择块或循环块之一; $Block_j$ 为基本块 (i = 1, 2, 3, …, j = 1, 2, 3, …且 $i \neq j$), 而且 $Block_j = NextBlock^+(Block_i)$, 则选择 m 处生成的定义变异体 M (*def*, v, m). 这里, $Block_j = NextBlock^+(Block_i)$ 表示 $Block_j$ 是 $Block_i$ 的直接或间接顺接块, 即 $Block_i$ 执行后一定会执行 $Block_j$, 且 $Block_j$ 的执行不会导致 $Block_i$ 的执行.

精简规则 3 (R_3). 变量 v 的一个<定义-使用>对<v, m, n>, 若 $(m \in Block_i) \wedge (n \in Block_j)$, 其中, $Block_i$ 为基本块、选择块或循环块之一; $Block_j$ 为基本块 (i=1, 2, 3, …, j=1, 2, 3, …且 $i \neq j$), 且 $Block_j = NextBlock(UpperBlock(Block_i))$,

则选择 m 处生成的定义变异体 M (def, v, m). 这里, $Block_k$ =$UpperBlock$($Block_i$), k = 1, 2, 3, …且 $k\neq i$, 表示 $Block_k$ 为 $Block_i$ 的上层块, 即 $Block_k$ 执行可能引起 $Block_i$ 执行, 且 $Block_i$ 执行后 $Block_k$ 继续执行.

上述精简规则的直观描述如图 3 所示. 对于精简规则 1 而言, 变量 v 的一个定义使用对<v, m, n>, 若定义语句 s<v, m>与使用语句 s<v, n>属于同一个基本块 $Block_i$, 则选择语句 s<v, m>处的变异体 M (def, v, m); 对于精简规则 2 而言, 变量 v 的一个定义使用对<v, m, n>, 若定义语句 s<v, m>属于块 $Block_i$, 使用语句 s<v, n>属于基本块 $Block_j$, 且 $Block_j$ 为 $Block_i$ 的直接或间接顺接块, 则选择语句 s<v, m>处的变异体 M (def, v, m); 对于精简规则 3 而言, 变量 v 的一个定义使用对<v, m, n>, 若定义语句 s<v, m>属于块 $Block_i$, 使用语句 s<v, n>属于基本块 $Block_j$, 且块 $Block_i$ 包含于 $Block_k$, $Block_j$ 为 $Block_k$ 的顺接块, 则选择语句 s<v, m>处的变异体 M(def, v, m).

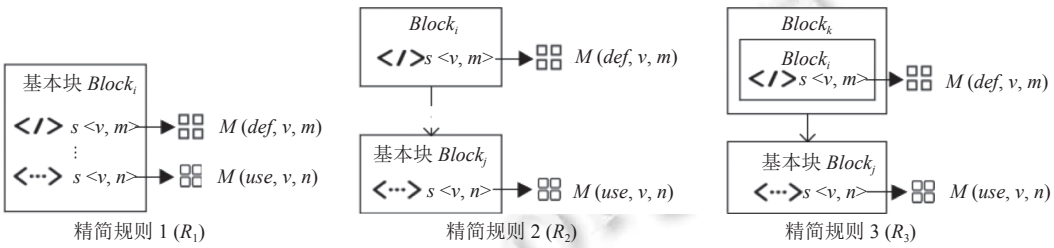

图 3 变异体精简规则示意图

2.5 算法描述

数据流分析指导的变异体精简技术 DFSampling 的执行过程如算法 1 所示. 首先分析源程序中各个变量的定义、使用语句 du (v, s_1, s_2), 获取变量 v 的定义变异体集 M (def, v, s_1) 和使用变异体集 M (use, v, s_2), 并判断其定义性出现所在语句 s_1 中的程序块 b_i 以及使用性出现所在语句 s_2 所属的程序块 b_j; 对于这两个变异体集合 M (def, v, s_1) 和 M (use, v, s_2), 依据规则 R_1~R_3 去除不满足规则的变异体, 得到精简后的变异体集合 $M_{all}-M_{use}$; 具体来说: 代码行 6–7 收集满足 R_1 的变异体集合; 代码行 8–15 收集满足 R_2 的变异体集合; 代码行 16–27 收集符合 R_3 的变异体集合; 最后从剩余的变异体集合 $M_{all}-M_{use}$ 中按照一定比例随机选取若干变异体并返回.

算法 1. DFSampling 算法.

Input: M={m_1, m_2, …}, B={b_1, b_2, …}, DU={du(v_1, s_{11}, s_{21}), du(v_2, s_{12}, s_{22}), …}, $ratio\in$(0, 100);
Output: M_{re}={m_{re1}, m_{re2}, …}.

PROCEDURE

1. **Initialize** M_{use}, M_{re} to an empty set
2. **for** each du (v, s_1, s_2) of v *in* DU **do**
3. M (def, v, s_1)←du (v, s_1, s_2)
4. M (use, v, s_2)←du (v, s_1, s_2)
5. b_i←s_1, b_j←s_2
6. **if** ((b_i == b_j) && (type(b_j) == $BasicBlock$) **then**
7. M_{use}=M_{use}∪M(use, v, s_2)
8. **else if** (type(b_j) == $BasicBlock$) && (b_i <$^+$ b_j)) **then**
9. **do**
10. **if** (b_j == $NextBlock^+$ (b_i)) **then**
11. M_{use}=M_{use}∪M(use, v, s_2)
12. **break**

13. **end if**
14. $b_i = NextBlock^+(b_i)$
15. **while** ($b_i <^+ b_j$)
16. **else if** (type(b_j) == *BasicBlock*) && ($b_i <^+ b_j$)) **then**
17. **do**
18. $b_k = UpperBlock(b_i)$
19. **do**
20. **if** (b_j == $NextBlock(b_k)$) **then**
21. $M_{use}=M_{use} \cup M(use, v, s_2)$
22. **break**
23. **end if**
24. $b_k = NextBlock(b_k)$
25. **while** ($b_k <^+ b_j$)
26. $b_i = UpperBlock(b_i)$
27. **while** ($b_i <^+ b_j$)
28. **end if**
29. **end for**
30. Randomly take *ratio* percent of mutant from $M_{all} - M_{use}$ to M_{re}
31. **return** M_{re}

2.6 方法示例

本节采用一个示例程序 (如图 4 所示) 展示上述变异体精简规则的应用.

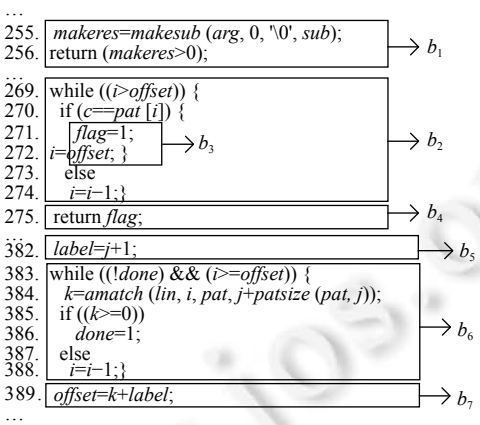

图 4 示例程序

对图 4 示例的变异体应用精简规则的过程如下.

R_1. *makeres* 是程序中的一个变量, <*makeres*, 255, 256>是 *makeres* 的一个<定义-使用>对, 语句 255 和语句 256 属于同一个基本块 b_1, 满足规则 1. 因此选择在语句 255 处发生的变异体.

R_2. *label* 是程序中的一个变量, <*label*, 382, 389>是 *label* 的一个<定义-使用>对, 语句 382 属于基本块 b_5, 语句 389 属于基本块 b_7, 且循环块 b_6 是 b_5 的顺接块, b_7 是 b_6 的顺接块, 则 b_7 为 b_5 的间接顺接块, 满足规则 2. 因此选择在语句 382 处发生的变异体.

R_3.*flag* 是程序中的一个变量, <*flag*, 271, 275>是 *flag* 的一个<定义-使用>对, 语句 271 属于基本块 b_3, 其上层块是 b_2, 语句 275 属于基本块 b_4, 且 b_4 是 b_2 的顺接块, 满足规则 3. 因此选择在语句 271 处发生的变异体.

3 实验评估

本节采用经验研究的方式验证与评估 DFSampling 技术的可行性与有效性.

3.1 研究问题

本文选取 10 个实例程序作为研究对象, 同时将 DFSampling 技术分别与随机选择策略、基于路径感知的变异体精简策略进行对比. 其中, 随机选择策略是一种典型的变异测试优化技术; 基于路径感知的变异体精简策略则是新近提出的变异测试优化策略.

本文实验评估试图回答如下 3 个研究问题.

(1) 数据流分析指导的变异体精简技术 DFSampling 是否可行?

选取基准程序进行实例研究, 验证 DFSampling 技术的可行性.

(2) 与随机选择策略相比, DFSampling 技术是否更有效?

在基准程序上分别使用 DFSampling 技术与随机选择策略, 比较两者的相对变异得分与执行时间.

(3) 与基于路径感知的变异体精简策略 (PAMR) 相比, DFSampling 技术是否更有效?

选取 PAMR 策略进行实例研究与对比, 以验证 DFSampling 技术的有效性.

3.2 实验设计

变异得分用来检测变异体集合的故障检测评估能力, 如公式 (1) 所示:

$$MS = \frac{M_k}{M_{all} - M_{eq}} \times 100\% \tag{1}$$

其中, *MS* 为变异得分, M_{all} 为总的变异体数量, M_{eq} 为等价变异体数量, M_k 为被杀死的变异体数量, 且 $M_k \leqslant M_{all} - M_{eq}$.

为了评估变异体精简对故障检测能力的影响, 本文采用相对变异得分衡量不同变异精简策略的有效性. 实验中, 按照某一精简策略从变异体集合 M 中选择一个变异体子集 M_{selected}, 构造变异得分能达到 100% 的测试用例集 TC_{selected}. 在变异体集合 M 上执行测试用例集 TC_{selected} 并计算变异得分, 这样的变异得分称为相对变异得分. 相对变异得分越高, 表明变异体子集 M_{selected} 越有效, 变异体精简策略的效果越好. 相对变异得分如公式 (2) 所示:

$$MS_r^{MSt} = \frac{M_k}{M} \times 100\% \tag{2}$$

其中, MS_r^{MSt} 为变异测试优化策略 *MSt* 在变异体选取比例为 r 时的相对变异得分, M_k 为测试用例集 TC_{selected} 作用在所有变异体集合 M 上所杀死的变异体数量, M 为总的变异体数量, 且 $M_k \leqslant M$.

变异测试执行时间 *MST* 用来衡量不同变异体精简策略的性能, 包括变异体选择时间 *MSS* 与变异测试运行时间 *MSR*. 实验中, 计算每个实验对象在不同变异体选取比例下不同精简策略的执行时间 *MST*. 变异测试执行时间 *MST* 越短, 表示变异测试的时间开销越少, 变异体精简策略效果越好. 变异测试执行时间如公式 (3) 所示:

$$MST = MSS + MSR \tag{3}$$

其中, 变异体选择时间 *MSS* 指的是每个变异测试优化策略按照某一比例选择合适变异体所需的时间; 变异测试运行时间 *MSR* 指的是从变异测试开始到结束的实际运行过程所需的时间.

DFSampling 通过确定的精简规则实现变异体精简, 即采用“保留定义变异体、舍弃使用变异体”的策略. 对于给定的初始变异体集合, DFSampling 精简后的变异体集合中的变异体数量是相对固定的. 在此基础上, 如果选择不同比例的变异体集合进行对比实验, DFSampling 选取的变异体集合的数量与现有方法 (随机选择、PAMR) 选取的变异体集合的数量不一致, 破坏了实验的公平性. 为此, 本文将 DFSampling 技术应用于现有方法, 比较 DFSapmling 应用前后不同精简比例下的相对变异得分, 验证 DFSampling 在变异体精简方面的有

效性.

不同精简比例下的实验流程如下.

1. DFSampling 技术与随机选择策略对比实验流程

(1) 随机选择策略实验流程

① 从所有变异体集合 M 中随机选取固定比例 r 的变异体集合 M_r^R;

② 使用 M_r^R 在原始测试用例集 T 上执行变异测试, 得到杀死该变异体集合 M_r^R 的测试用例集 T_r^R;

③ 使用 T_r^R 在所有变异体集合 M 上执行变异测试, 得到相对变异得分 MS_r^R.

(2) DFSampling 实验流程

① 从所有变异体集合 M 中随机选取固定比例 r 的变异体集合 M_r^R;

② 使用 DFSampling 技术对 M_r^R 进行变异体精简, 得到变异体集合 M_r^D;

③ 从未被选择的变异体集合中补充满足 DFSampling 技术的变异体 (例如, 当变异体选取比例为 5% 时, 从未被选取的 95% 的变异体集合中随机选择满足 DFSampling 技术的变异体), 直至 M_r^D 的变异体数量与 M_r^R 的变异体数量一致;

④ 使用 M_r^D 在测试用例集 T 上执行变异测试, 得到杀死该变异体集合 M_r^D 的测试用例集 T_r^D;

⑤ 使用 T_r^D 在所有变异体集合 M 上执行变异测试, 得到相对变异得分 MS_r^D.

2. DFSampling 技术与 PAMR 策略对比实验流程

(1) PAMR 实验流程

① 从所有变异体集合 M 中使用 PAMR 策略选取固定比例 r 的变异体集合 M_r^P;

② 使用 M_r^P 在测试用例集 T 上执行变异测试, 得到杀死该变异体集合 M_r^P 的测试用例集 T_r^P;

③ 使用 T_r^P 在所有变异体集合 M 上执行变异测试, 得到相对变异得分 MS_r^P.

(2) DFSampling 实验流程

① 从所有变异体集合 M 中使用 PAMR 策略选取固定比例 r 的变异体集合 M_r^P;

② 使用 DFSampling 技术对 M_r^P 进行变异体精简, 得到变异体集合 M_r^D;

③ 从未被选择的变异体集合中补充满足 DFSampling 技术的变异体 (例如, 当变异体选取比例为 5% 时, 从未被选取的 95% 的变异体集合中选择满足 PAMR 与 DFSampling 技术的变异体), 直至 M_r^D 的变异体数量与 M_r^P 的变异体数量一致;

④ 使用 M_r^D 在测试用例集 T 上执行变异测试, 得到杀死该变异体集合 M_r^D 的测试用例集 T_r^D;

⑤ 使用 T_r^D 在所有变异体集合 M 上执行变异测试, 得到相对变异得分 MS_r^D.

此外, 为了增强实验结果的可靠性, 本文每组实验重复 30 次, 以消除变异体采样随机性造成的偏差.

本文使用一组基准 C 程序来进行实验评估, 实验对象的基本信息如表 1 所示. 其中 7 个程序为常用的西门子程序[35]: tcas 程序为防止航空器空中相撞系统; schedule 和 schedule2 程序为优先级调度器; tot_info 程序针对指定的输入数据生成统计信息; print_tokens 和 print_tokens2 程序是词法分析器; replace 程序完成模式匹配和替换. 另外 3 个程序为应用科学程序: minmax 程序获取一组数据的最大值和最小值; bubble 程序是冒泡排序算法; nextdate 程序用于计算下一天的具体日期. 表格共有 4 列, 其中第 1 列表示实验对象的名称, 第 2 列为实验对象源代码的行数, 第 3 列为实验对象对应变异体的数量, 最后一列为测试用例集的规模.

3.3 实验结果与分析

3.3.1 DFSampling 技术与随机选择策略对比

图 5 为 DFSampling 技术与随机选择策略在 10 组实验对象的相对变异得分情况对比. 从图中可以看出, 在所有选取比例中, DFSampling 的相对变异得分在 60% 及以上实验对象中高于随机选择策略. 具体说来, 当选取比例为 1% 时, DFSampling 的相对变异得分高于随机选择策略; 当变异选取比例为 2%、3%、5% 时, DFSampling 的相对变异得分在 90% 以上实验对象中高于随机选择策略; 当变异选取比例为 4%、10% 时, DFSampling 的相对变异

得分在 80% 的实验对象中高于随机选择策略. 上述结果表明, 在进行变异体精简时, DFSamling 技术比随机选择策略具有更好精简效果, 即选取相同比例的变异体时, DFSampling 技术选取的变异体集合具有更好的故障检测能力.

表 1 实验对象信息

实验对象	代码行	变异体数量	测试用例集规模
print _tokens	343	3 746	4 130
print_tokens2	355	3 601	4 115
replace	513	7 797	5 542
schedule	296	1 548	2 650
schedule2	263	2 089	2 710
tcas	137	2 019	1 052
tot_info	281	3 695	1 608
minmax	33	253	60
bubble	24	311	75
nextdate	83	692	377

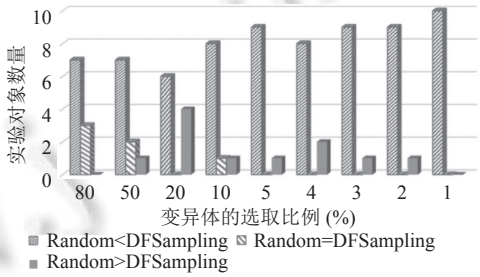

图 5 DFSampling 与随机选择策略相对变异得分对比

表 2 总结了 DFSampling 技术与随机选择策略不同变异体选取比例 (1%、2%、3%、4%、5%、10%、20%、50% 和 80%) 下相对变异得分比较结果. 其中, 加粗数据表示不同变异体选取比例下, 相对变异得分的最高提高率、最低提高率以及提高率的平均值. 从表中可以看出, DFSampling 的相对变异得分比随机选择策略提高了高达 4.61%, 且 DFSampling 的平均相对变异得分高于随机选择策略. 上述结果表明, DFSampling 技术比随机选择策略具有更好的变异体精简效果.

图 6 为 DFSampling 技术与随机选择策略不同变异体选取比例下执行时间对比结果. 其中, 横坐标为 10 组实验对象, 纵坐标为执行时间. 从图中可以看出, DFSampling 的变异测试执行时间与随机选择策略近似, 且 DFSampling 的平均执行时间比随机选择策略减少 0.77%. 上述结果表明, 相较于随机选择策略, DFSampling 技术并没有引入时间开销.

3.3.2 DFSampling 技术与 PAMR 策略对比

图 7 为 PAMR 策略与 DFSampling 技术在 10 组实验对象上相对变异得分对比情况. 从图中可以看出, DFSampling 的相对变异得分在 60% 及以上的实验对象中高于 PAMR. 具体来说, 当选取比例为 2%、3%、5%、10% 时, DFSampling 在 90% 实验对象上的相对变异得分高于 PAMR; 当选取比例为 4% 时, DFSamling 在 80% 实验对象上的相对变异得分高于 PAMR; 当选取比例为 20%、50%、80% 时, DFSamling 在 70% 实验对象上的相对变异得分高于 PAMR. 上述结果表明, DFSampling 技术比 PAMR 策略具有更好精简效果, 即选取相同比例的变异体时, DFSampling 技术选取的变异体集合具有更好的故障检测能力.

表 3 总结了 DFSampling 技术与 PAMR 策略不同变异体选取比例下相对变异得分比较结果. 其中, 加粗数据表示不同变异体选取比例下, 相对变异得分的最高提高率、最低提高率以及提高率的平均值. 从表中可以看出,

与 PAMR 策略相比, DFSampling 相对变异得分提高了高达 4.8%, 且 DFSampling 的平均相对变异得分高于 PAMR. 上述结果表明, DFSampling 比 PAMR 具有更好的变异体精简效果.

表 2 DFSampling 与随机选择策略不同选取比例下相对变异得分对比 (%)

变异体选取比例	80%			50%			20%		
实验对象	Random	DFSampling	提高率	Random	DFSampling	提高率	Random	DFSampling	提高率
schedule	99.93	99.96	0.03	99.65	99.70	0.05	98.67	98.85	0.18
schedule2	99.99	99.99	**0.00**	99.81	99.86	0.05	99.49	99.58	0.09
tcas	99.83	99.85	0.02	99.27	99.34	0.07	98.28	98.20	**−0.08**
tot_info	100.00	100.00	**0.00**	99.92	99.94	0.02	99.64	99.69	0.05
print_tokens	99.98	99.98	**0.00**	99.92	99.92	−0.01	99.69	99.68	0.00
print_tokens2	99.96	99.97	0.01	99.91	99.91	0.00	99.79	99.77	−0.02
replace	99.96	99.97	0.01	99.77	99.79	0.01	99.37	99.40	0.03
bubble	99.68	99.73	0.05	99.66	99.69	0.03	99.04	99.21	0.18
minmax	99.86	99.92	**0.06**	99.23	99.31	**0.08**	98.32	98.74	**0.42**
nextdate	99.65	99.70	0.04	99.15	99.13	**−0.02**	96.85	96.77	**−0.08**
平均值	99.88	99.91	**0.02**	99.63	99.66	**0.03**	98.91	98.99	**0.08**
变异体选取比例	10%			5%			4%		
实验对象	Random	DFSampling	提高率	Random	DFSampling	提高率	Random	DFSampling	提高率
schedule	97.94	97.90	**−0.05**	95.90	96.05	0.16	94.86	95.06	0.22
schedule2	97.95	98.14	0.19	95.58	95.78	0.21	95.95	95.57	**−0.39**
tcas	96.01	96.84	0.86	94.46	94.48	0.03	93.08	93.66	0.62
tot_info	99.42	99.44	0.03	98.99	98.89	**−0.10**	98.46	98.81	0.36
print_tokens	99.19	99.19	0.00	98.83	98.94	0.11	97.85	97.95	0.10
print_tokens2	99.54	99.56	0.02	98.63	98.70	0.08	98.81	99.01	0.20
replace	98.58	98.62	0.04	97.72	97.89	0.17	97.04	97.41	0.38
bubble	98.30	98.39	0.10	97.33	97.75	0.43	98.41	98.23	−0.18
minmax	95.38	96.56	**1.24**	90.71	93.32	2.88	91.19	91.94	0.82
nextdate	94.49	94.94	0.48	87.05	90.43	**3.88**	81.37	84.88	**4.32**
平均值	97.68	97.96	**0.28**	95.52	96.22	**0.74**	94.70	95.25	**0.58**
变异体选取比例	3%			2%			1%		
实验对象	Random	DFSampling	提高率	Random	DFSampling	提高率	Random	DFSampling	提高率
schedule	94.27	94.60	0.36	93.10	93.46	0.39	90.87	92.06	1.32
schedule2	95.59	95.67	0.08	94.35	94.08	**−0.29**	91.84	91.96	0.13
tcas	89.84	90.26	0.47	87.64	87.70	0.06	80.86	80.95	0.11
tot_info	98.29	98.42	0.13	97.55	97.72	0.18	95.44	96.30	0.90
print_tokens	97.14	97.30	0.16	96.29	96.50	0.22	92.64	93.60	1.04
print_tokens2	98.35	98.49	0.14	96.93	97.46	0.55	93.46	94.24	0.83
replace	96.55	96.62	0.08	94.65	95.04	0.41	91.94	92.16	0.23
bubble	97.36	97.54	0.18	97.12	97.49	0.38	96.51	96.78	0.28
minmax	92.83	92.47	**−0.38**	88.10	88.68	0.65	85.40	86.01	0.72
nextdate	75.98	79.49	**4.61**	67.67	69.36	**2.51**	50.92	53.09	**4.24**
平均值	93.62	94.09	**0.50**	91.34	91.75	**0.45**	86.99	87.71	**0.83**

需要指出的是, 如表 2 与表 3 所示, nextdate 程序在选取比例为 10% 以下时相对变异得分比其他程序低很多. 其原因解释如下: 通过分析 nextdate 程序的结构特征发现, 该程序中存在多个选择分支 (switch). 对于包含多分支语句的程序来说, 一个测试用例通常仅能覆盖其中一条分支. 给定此类程序的一个变异体集合, 其故障语句可能处在不同的程序分支, 能覆盖某个分支的测试用例通常无法覆盖其他的分支. 在进行变异体精简时, 如果当某个分支上的变异体都被移除后, 则能杀死精简后的变异体集合的测试用例集合很难杀死该分支上所有变异体, 导致相对变异得分降低. 当变异体精简到较低比例时, 相对变异得分则急剧下降.

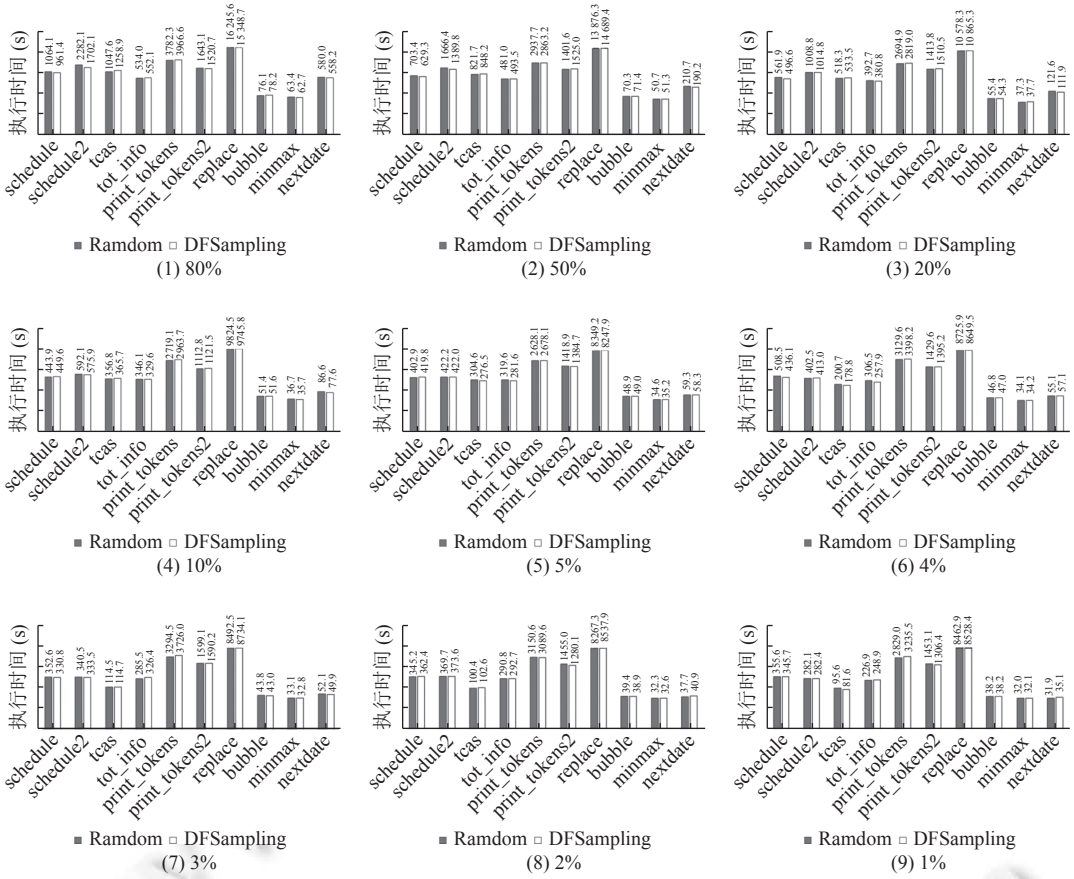

图 6 DFSampling 与随机选择策略执行时间对比

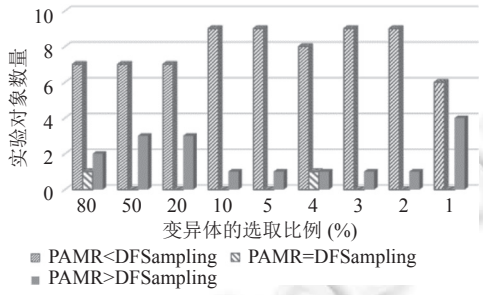

图 7 DFSampling 与 PAMR 策略相对变异得分对比

图 8 为 DFSampling 技术与 PAMR 策略不同变异体选取比例下执行时间对比情况. 其中, 横坐标为 10 组实验对象, 纵坐标为执行时间. 从图中可以看出, DFSampling 技术在 10 组实验对象的变异测试执行时间与 PAMR 策略近似, 且相较于 PAMR 策略, DFSampling 平均执行时间仅增加 0.39%. 上述结果表明, DFSampling 技术并未引入时间开销.

3.3.3 结 论

基于上述分析结果, 可得到如下结论:

(1) DFSampling 技术的平均相对变异得分均高于随机选择策略与 PAMR 策略. 不仅验证了 DFSampling 技术的可行性与有效性, 而且验证了 DFSampling 技术精简后的变异体集合比随机选择策略与 PAMR 具有更好的相对

变异得分;

(2) DFSampling 技术的时间开销与随机选取策略、PAMR 策略下的时间开销基本相当. 具体说来, 与随机选取策略相比, DFSampling 技术平均时间开销减少了 0.77%; 与 PAMR 策略相比, DFSampling 技术平均时间开销增加了 0.39%. 因此, DFSampling 技术在提高故障检测能力的前提下并没有引入过多的时间成本;

(3) DFSampling 技术可以在不增加测试开销且不显著降低相对变异得分的情况下, 实现高效的变异体精简. 因此在测试资源受限的情形下, 优先推荐使用 DFSampling 技术.

表 3 DFSampling 与 PAMR 策略不同选取比例下相对变异得分对比 (%)

变异体选取比例	80%			50%			20%		
实验对象	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率
schedule	99.90	99.91	0.01	99.50	99.57	0.07	96.18	95.87	−0.32
schedule2	97.92	97.92	0.00	97.83	98.05	0.23	95.31	95.25	−0.07
tcas	99.96	99.97	0.01	88.99	89.18	0.21	79.05	79.39	0.43
tot_info	99.70	99.68	−0.02	99.04	99.22	0.18	96.97	98.29	**1.36**
print_tokens	99.86	99.81	**−0.05**	98.24	98.65	**0.42**	95.84	96.02	0.18
print_tokens2	99.73	99.78	0.05	99.39	99.57	0.18	99.53	99.55	0.03
replace	99.48	99.54	0.06	98.83	98.99	0.16	98.57	98.74	0.17
bubble	99.61	99.65	0.03	99.65	99.71	0.06	97.94	98.20	0.26
minmax	100.00	100.00	0.00	99.76	99.68	**−0.08**	97.87	97.15	**−0.73**
nextdate	99.55	99.71	**0.16**	98.64	98.61	−0.03	82.17	83.25	1.32
平均值	99.57	99.60	**0.03**	97.99	98.12	**0.14**	93.94	94.17	**0.24**
变异体选取比例	10%			5%			4%		
实验对象	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率
schedule	95.50	95.61	0.12	92.62	92.66	0.05	91.49	92.48	1.09
schedule2	93.07	93.37	0.32	92.36	92.41	0.06	89.24	90.02	0.88
tcas	73.64	74.41	1.04	64.95	65.27	0.50	63.01	63.42	0.64
tot_info	95.86	96.41	0.57	92.99	93.96	1.04	92.89	92.93	0.05
print_tokens	91.08	91.99	1.00	81.88	85.81	**4.80**	81.23	82.15	1.13
print_tokens2	98.52	98.16	**−0.36**	97.53	96.69	**−0.86**	96.68	96.85	0.18
replace	97.65	97.98	0.34	96.21	96.97	0.79	96.01	96.56	0.57
bubble	96.69	97.97	1.33	97.40	97.81	0.43	97.27	96.66	**−0.63**
minmax	93.75	95.81	**2.19**	94.23	94.90	0.71	93.68	93.68	0.00
nextdate	78.71	79.61	1.14	41.17	41.46	0.70	40.53	41.73	**2.96**
平均值	78.71	79.61	**1.14**	85.13	85.79	**0.78**	84.20	84.65	**0.53**
变异体选取比例	3%			2%			1%		
实验对象	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率	PAMR	DFSampling	提高率
schedule	89.99	90.63	0.71	84.03	81.25	−3.31	81.85	83.62	2.15
schedule2	86.13	86.25	0.14	86.60	85.86	−0.85	85.73	86.53	0.93
tcas	60.60	61.06	0.77	60.33	58.31	**−3.35**	45.79	45.78	−0.02
tot_info	92.29	90.49	**−1.95**	91.93	93.35	1.55	90.51	91.06	0.61
print_tokens	80.35	81.80	0.57	78.50	80.63	**2.70**	78.38	78.09	**−0.37**
print_tokens2	95.36	95.43	0.07	90.92	93.16	2.46	90.67	90.78	0.12
replace	95.35	95.74	0.41	94.88	94.97	0.09	92.14	92.26	0.13
bubble	96.56	97.68	1.17	96.40	97.40	1.03	96.05	97.01	1.00
minmax	93.24	93.99	0.81	88.54	89.21	0.76	89.64	91.15	1.68
nextdate	31.79	32.30	**1.59**	31.82	32.14	1.00	32.28	33.45	**3.63**
平均值	78.71	79.61	**1.14**	80.39	80.63	**0.29**	78.30	78.97	**0.85**

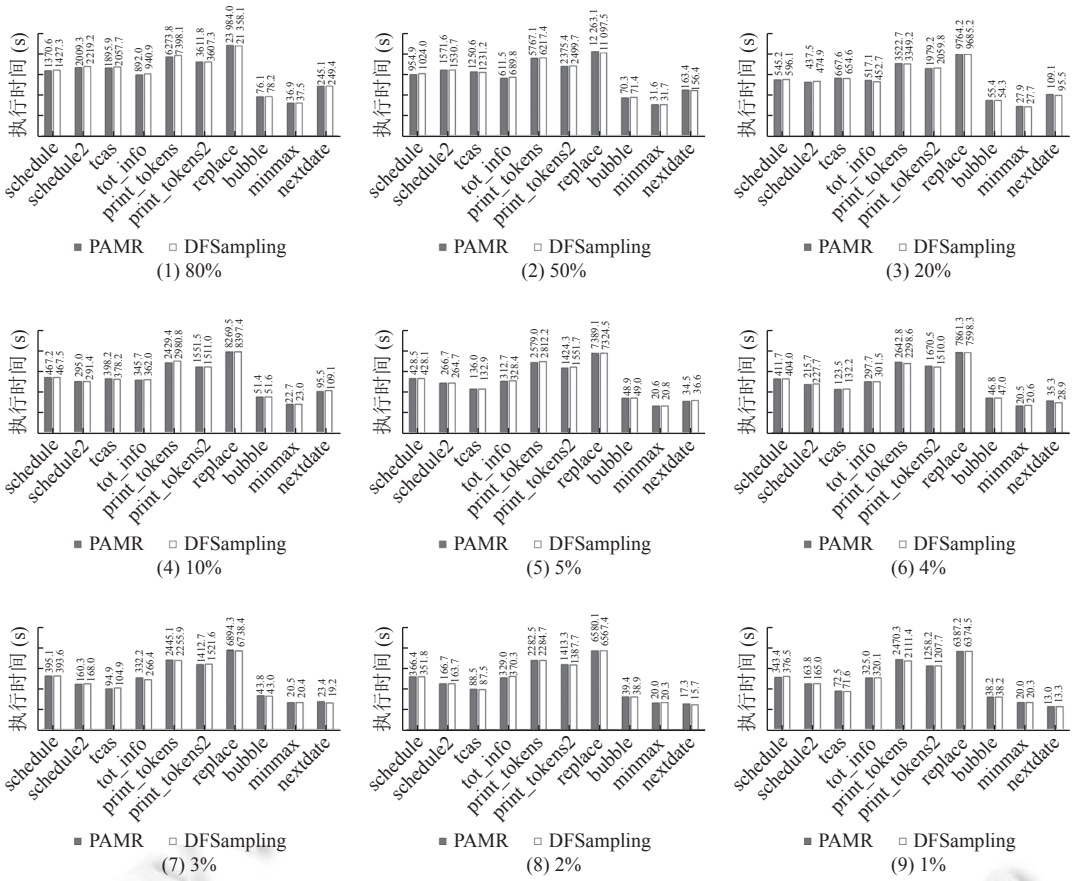

图 8 DFSampling 与 PAMR 策略执行时间对比

4 总结与展望

本文从数据流分析的角度出发, 提出一种数据流分析指导的变异体精简策略 DFSampling. 从程序中变量的<定义-使用>对出发, 充分考虑不同变异体之间存在的数据流约束的影响, 将变异体集合分为定义变异体与使用变异体, 定义了基于数据流分析的启发式规则, 给出了基于数据流分析的变异体精简算法. 采用经验研究评估了 DFSampling 技术的可行性与有效性, 并分别与随机选择策略、基于路径感知的变异体精简策略进行对比. 结果表明, DFSampling 技术可以在不增加测试开销且不影响变异测试故障检测能力的情况下, 实现高效的变异体精简, 从而提高变异测试效率.

我们将探究程序中数据流约束的变化对变异体产生的影响, 研究 DFSampling 技术在其他类型程序中的应用、以及研究针对程序结构特性的新型变异体精简方法.

References:

[1] Al-Hajjaji M, Krüger J, Benduhn F, Leich T, Saake G. Efficient mutation testing in configurable systems. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Variability and Complexity in Software Design. Buenos Aires: IEEE, 2017. 2–8. [doi: 10.1109/VACE.2017.3]

[2] Papadakis M, Kintis M, Zhang J, Jia Y, Traon YL, Harman M. Mutation testing advances: An analysis and survey. Advances in Computers, 2019, 112: 275–378. [doi: 10.1016/bs.adcom.2018.03.015]

[3] Sun CA, Wang Z, Pan L. Optimized mutation testing techniques for WS-BPEL programs. Journal of Computer Research and Development, 2019, 56(4): 895–905 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20180037]

[4] Bianchi FA, Margara A, Pezzè M. A survey of recent trends in testing concurrent software systems. IEEE Trans. on Software Engineering, 2018, 44(8): 747–783. [doi: 10.1109/TSE.2017.2707089]

[5] Zhang J, Zhang LM, Harman M, Hao D, Jia Y, Zhang L. Predictive mutation testing. IEEE Trans. on Software Engineering, 2019, 45(9): 898–918. [doi: 10.1109/TSE.2018.2809496]

[6] Sun CA, Jia JT, Liu H, Zhang XY. A lightweight program dependence based approach to concurrent mutation analysis. In: Proc. of the 42nd IEEE Annual Computer Software and Applications Conf. (COMPSAC). Tokyo: IEEE, 2018. 116–125. [doi: 10.1109/COMPSAC.2018.00023]

[7] Abuljadayel A, Wedyan F. An approach for the generation of higher order mutants using genetic algorithms. Int'l Journal of Intelligent Systems and Applications, 2018, 10(1): 34–45. [doi: 10.5815/ijisa.2018.01.05]

[8] Sun CA, Fu A, Guo XL, Chen TY. ReMuSSE: A redundant mutant identification technique based on selective symbolic execution. IEEE Trans. on Reliability, 2022, 71(1): 415–428. [doi: 10.1109/TR.2020.3011423]

[9] Kurtz B, Ammann P, Offutt J, Delamaro ME, Kurtz M, Gökçe N. Analyzing the validity of selective mutation with dominator mutants. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 571–582. [doi: 10.1145/2950290.2950322]

[10] Gopinath R, Ahmed I, Alipour MA, Jensen C, Groce A. Mutation reduction strategies considered harmful. IEEE Trans. on Reliability, 2017, 66(3): 854–874. [doi: 10.1109/TR.2017.2705662]

[11] Derezińska A, Rudnik M. Evaluation of mutant sampling criteria in object-oriented mutation testing. In: Proc. of the 2017 Federated Conf. on Computer Science and Information Systems (FedCSIS). Prague: IEEE, 2017. 1315–1324. [doi: 10.15439/2017F375]

[12] Zhang L, Hou SS, Hu JJ, Xie T, Mei H. Is operator-based mutant selection superior to random mutant selection? In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. Cape Town: IEEE, 2010. 435–444. [doi: 10.1145/1806799.1806863]

[13] Bluemke I, Kulesza K. Reductions of operators in java mutation testing. In: Proc. of the 9th Int'l Conf. on Dependability and Complex Systems. Brunów: Springer, 2014. 93–102. [doi: 10.1007/978-3-319-07013-1_9]

[14] Delgado-Pérez P, Medina-Bulo I, Merayo MG. Using evolutionary computation to improve mutation testing. In: Proc. of the 14th Int'l Work-Conf. on Artificial Neural Networks on Advances in Computational Intelligence. Cadiz: Springer, 2017. 381–391. [doi: 10.1007/978-3-319-59147-6_33]

[15] Omar E, Ghosh S. An exploratory study of higher order mutation testing in aspect-oriented programming. In: Proc. of the 23rd IEEE Int'l Symp. on Software Reliability Engineering. Dallas: IEEE, 2012. 1–10. [doi: 10.1109/ISSRE.2012.6]

[16] Kintis M, Papadakis M, Malevris N. Employing second-order mutation for isolating first-order equivalent mutants. Software Testing, Verification and Reliability, 2015, 25(5–7): 508–535. [doi: 10.1002/stvr.1529]

[17] Derezińska A. A quality estimation of mutation clustering in C# programs. In: Proc. of the 8th Int'l Conf. on Dependability and Complex Systems on New Results in Dependability and Computer Systems. Brunów: Springer, 2013. 119–129. [doi: 10.1007/978-3-319-00945-2_11]

[18] Ji CB, Chen ZY, Xu BW, Zhao ZH. A novel method of mutation clustering based on domain analysis. In: Proc. of the 21st Int'l Conf. on Software Engineering and Knowledge Engineering. Boston, 2009. 1–6.

[19] Ma YS, Kim SW. Mutation testing cost reduction by clustering overlapped mutants. Journal of Systems and Software, 2016, 115: 18–30. [doi: 10.1016/j.jss.2016.01.007]

[20] Kintis M, Malevris N. Using data flow patterns for equivalent mutant detection. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Cleveland: IEEE, 2014. 196–205. [doi: 10.1109/ICSTW.2014.21]

[21] Papadakis M, Jia Y, Harman M, Traon YL. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 936–946. [doi: 10.1109/ICSE.2015.103]

[22] McMinn P, Wright CJ, McCurdy CJ, Kapfhammer GM. Automatic detection and removal of ineffective mutants for the mutation analysis of relational database schemas. IEEE Trans. on Software Engineering, 2019, 45(5): 427–463. [doi: 10.1109/TSE.2017.2786286]

[23] Kintis M, Papadakis M, Jia Y, Malevris N, Traon YL, Harman M. Detecting trivial mutant equivalences via compiler optimisations. IEEE Trans. on Software Engineering, 2018, 44(4): 308–333. [doi: 10.1109/TSE.2017.2684805]

[24] Bogacki B, Walter B. Aspect-oriented response injection: An alternative to classical mutation testing. In: Proc. of the IFIP Working Conf. on Software Engineering Techniques. Warsaw: Springer, 2006. 273–282. [doi: 10.1007/978-0-387-39388-9_26]

[25] Gopinath R, Jensen C, Groce A. Topsy-turvy: A smarter and faster parallelization of mutation analysis. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering Companion. Austin: IEEE, 2016. 740–743.

[26] Mateo PR, Usaola MP. Parallel mutation testing. Software Testing, Verification and Reliability, 2013, 23(4): 315–350. [doi: 10.1002/stvr.

1471]
[27] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]
[28] Pizzoleto AV, Ferrari FC, Offutt J, Fernandes L, Ribeiro M. A systematic literature review of techniques and metrics to reduce the cost of mutation testing. Journal of Systems and Software, 2019, 157: 110388. [doi: 10.1016/j.jss.2019.07.100]
[29] Delgado-Pérez P, Habli I, Gregory S, Alexander R, Clark J, Medina-Bulo I. Evaluation of mutation testing in a nuclear industry case study. IEEE Trans. on Reliability, 2018, 67(4): 1406–1419. [doi: 10.1109/TR.2018.2864678]
[30] Acree AT, Budd TA, DeMillo RA, Lipton RJ, Sayward FG. Mutation analysis. Technical Report, GIT-ICS-79/08, 1979.
[31] Mathur AP, Wong WE. An empirical comparison of data flow and mutation-based test adequacy criteria. Software Testing, Verification and Reliability, 1994, 4(1): 9–31. [doi: 10.1002/stvr.4370040104]
[32] Zhang J, Zhu MY, Hao D, Zhang L. An empirical study on the scalability of selective mutation testing. In: Proc. of the 25th IEEE Int'l Symp. on Software Reliability Engineering. Naples: IEEE, 2014. 277–287. [doi: 10.1109/ISSRE.2014.27]
[33] 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. [doi: 10.1007/s11219-011-9142-y]
[34] Gong DW, Zhang GJ, Yao XJ, Meng FL. Mutant reduction based on dominance relation for weak mutation testing. Information and Software Technology, 2017, 81: 82–96. [doi: 10.1016/j.infsof.2016.05.001]
[35] Sun CA, Xue FF, Liu H, Zhang XY. A path-aware approach to mutant reduction in mutation testing. Information and Software Technology, 2017, 81: 65–81. [doi: 10.1016/j.infsof.2016.02.006]
[36] Sun CA, Guo XL, Zhang XY, Chen TY. A data flow analysis based redundant mutants identification technique. Chinese Journal of Computers, 2019, 42(1): 44–60 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2019.00044]

附中文参考文献:

[3] 孙昌爱, 王真, 潘琳. 面向WS-BPEL程序的变异测试优化技术. 计算机研究与发展, 2019, 56(4): 895–905. [doi: 10.7544/issn1000-1239.2019.20180037]
[36] 孙昌爱, 郭新玲, 张翔宇, 陈宗岳. 一种基于数据流分析的冗余变异体识别方法. 计算机学报, 2019, 42(1): 44–60. [doi: 10.11897/SP.J.1016.2019.00044]

孙昌爱(1974－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件测试, 故障定位, 服务计算.

刘镇贤(1994－), 男, 硕士, 主要研究领域为服务计算, 变异测试.

卫新洁(1995－), 女, 硕士, 主要研究领域为软件测试, 故障定位.

宫云战(1962－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件测试, 程序分析.