

融合因果效应的高效软件产品线缺陷定位方法*

王海宁¹, 向毅^{1,2}, 黄翰^{1,2}, 吴春国², 冯夫健³, 杨晓伟¹

¹(华南理工大学 软件学院, 广东 广州 510006)

²(符号计算与知识工程教育部重点实验室 (吉林大学), 吉林 长春 130012)

³(贵州民族大学 数据科学与信息工程学院, 贵州 贵阳 550025)

通信作者: 向毅, E-mail: xiangyi@scut.edu.cn; 黄翰, E-mail: hhan@scut.edu.cn



摘要: 缺陷定位是软件程序调试中最昂贵、最繁琐和最耗时的活动之一, 同时也是软件维护中不可或缺的关键步骤. 由于缺陷的可变性, 缺陷定位在软件产品线中更具挑战性. 尽管单系统软件的缺陷定位研究取得了显著进展, 但针对软件产品线可变性缺陷定位的研究相对不足. 同时, 现有方法由于忽略了特征交互的重复生成和检查问题, 以及缺陷在程序语句间的传播问题, 从而面临着效率低和根因定位能力差的挑战. 为此, 针对软件产品线提出一种高效且准确的缺陷定位方法, 该方法分为特征级和语句级两个层级的定位: 在特征级定位中, 利用可疑特征选择集合之间存在包含关系和相同子集这两点特性, 实现可疑特征交互的高效识别; 在语句级定位中, 利用一种引入中介变量的约简因果模型, 并融合因果效应和频谱效应, 实现更精确的缺陷定位. 选择 4 种先进的软件产品线缺陷定位方法, 并在 6 个真实的软件产品线系统上进行实验比较. 结果表明, 所提方法在定位效率和准确性方面均显著优于其他主流方法.

关键词: 软件产品线; 缺陷定位; 因果推理

中图法分类号: TP311

中文引用格式: 王海宁, 向毅, 黄翰, 吴春国, 冯夫健, 杨晓伟. 融合因果效应的高效软件产品线缺陷定位方法. 软件学报. <http://www.jos.org.cn/1000-9825/7552.htm>

英文引用格式: Wang HN, Xiang Y, Huang H, Wu CG, Feng FJ, Yang XW. Efficient Fault Localization Method for Software Product Lines Integrating Causal Effects. Ruan Jian Xue Bao/Journal of Software (in Chinese). <http://www.jos.org.cn/1000-9825/7552.htm>

Efficient Fault Localization Method for Software Product Lines Integrating Causal Effects

WANG Hai-Ning¹, XIANG Yi^{1,2}, HUANG Han^{1,2}, WU Chun-Guo², FENG Fu-Jian³, YANG Xiao-Wei¹

¹(School of Software Engineering, South China University of Technology, Guangzhou 510006, China)

²(Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education (Jilin University), Changchun 130012, China)

³(College of Data Science and Information Engineering, Guizhou Minzu University, Guiyang 550025, China)

Abstract: Fault localization is one of the most expensive, tedious, and time-consuming activities in software debugging, and it is also an indispensable step in software maintenance. Due to the variability of faults, fault localization is even more challenging in software product lines. Although significant progress has been made in fault localization for single-system software, research on fault localization for variability in software product lines is still insufficient. Meanwhile, existing methods face challenges such as low efficiency and poor root cause localization due to the issues of repeated generation and checking of feature interactions, as well as the propagation of faults between program statements. To address this, this study proposes an efficient and accurate fault localization method for software product lines, which performs localization at both the feature level and the statement level. At the feature level, based on observations of inclusion

* 基金项目: 国家自然科学基金 (62276103, 62566012); 中央高校基本科研业务费专项资金 (93K172024K03, 93K172024K24); 广东省基础与应用基础研究基金 (2024A1515030022); 广东省普通高校创新团队项目 (2023KCXTD002); 广东省哲学社会科学规划 2023 年度审计理论研究专项重点项 (GD23SJZ09); 贵州省教育厅自然科学研究项目 (黔教技 [2023]061 号); 惠州市科技计划 (2024BQ010011)

收稿时间: 2024-09-22; 修改时间: 2025-05-12, 2025-07-23; 采用时间: 2025-09-02; jos 在线出版时间: 2026-02-11

relationships and identical subsets between suspicious feature selection sets, the method identifies suspicious feature interactions more efficiently. At the statement level, a reduced causal model with mediator variables is used, combining causal effects and spectrum-based effects to achieve more precise fault localization. Four advanced fault localization methods for software product lines are selected, and experiments are conducted on six real-world software product line systems for comparison. The results demonstrate that the proposed method significantly outperforms other mainstream methods in terms of localization efficiency and accuracy.

Key words: software product line; fault localization; causal inference

在现代软件工程中, 软件的开发效率与质量至关重要, 而软件产品线 (software product line, SPL) 是实现这一目标的关键开发技术^[1,2]. Linux 操作系统、AUTOSAR 汽车电子系统软件、Firefox 浏览器和 ZipMe 压缩系统等常见的基础和应用软件均是采用 SPL 技术开发的. 特征是 SPL 中不同功能的抽象^[3], 特征模型 (feature model, FM)^[4]是用来描述和管理所有特征及其相互之间约束关系的有效工具. 软件工程师或终端用户根据特征模型所定义的约束关系选择相应的特征自动构建满足需求的软件产品 (系统), 有效地提高了软件开发的效率和质量, 但也增加了软件维护, 特别是缺陷定位的挑战性^[5,6]. 其挑战性的根源在于 SPL 软件系统的缺陷 (bug) 具有可变性^[7], 即仅能暴露在 SPL 中某些特定软件产品的缺陷, 而在其他产品中则通过了所有测试.

与传统的单系统软件相比, SPL 系统的缺陷定位有显著区别^[8,9]. 在单系统软件中, 开发人员只需关注一个特定系统中的缺陷即可, 所有的代码、配置和依赖关系都是固定的, 便于追踪和修复^[10]. 与此不同, SPL 系统的缺陷可能位于共享代码块或变异点, 导致多个产品存在功能缺陷. 因此, 由于需要考虑多个产品之间的共享代码和特征组合 (称为特征交互), SPL 系统中缺陷来源的追踪更加复杂^[11]. SPL 的缺陷定位包含两个相互关联的子任务, 即特征级缺陷定位和语句级缺陷定位^[5]. 前者旨在识别带缺陷的特征交互, 有助于从抽象层面理解和定位缺陷; 后者则聚焦于定位到缺陷的语句, 是更细粒度的定位. 特征级缺陷定位是语句级缺陷定位的基础, 二者相辅相成, 共同实现 SPL 系统的缺陷定位^[11].

近年来, 研究人员提出了一些面向 SPL 系统的缺陷定位方法^[5,6,11]. 但是, 与单系统软件的缺陷定位^[10,12,13]相比, 相关研究仍然非常稀缺. Arrieta 等人^[5]利用基于频谱的缺陷定位 (spectrum-based fault localization, SBFL) 技术计算所有二阶特征交互 (即所涉及特征的个数为 2) 的可疑性, 实现了特征级缺陷定位. 针对语句级缺陷定位方法, Nguyen 等人^[11]提出了 VarCop 方法, 该方法利用可变性缺陷的特性识别出可疑的特征交互, 接着基于程序切片技术隔离出可疑语句. 最后, VarCop 从产品和单元测试两种粒度基于 SBFL 技术计算语句的频谱效应以实现语句级缺陷定位, 一定程度上增强了频谱效应的准确性. 据我们所知, VarCop 是目前唯一针对 SPL 系统可变缺陷的语句级定位方法.

然而, 现有方法在特征级缺陷定位的效率和语句级缺陷定位的精确性上均面临不足. 例如, Arrieta 等人^[5]提出的方法忽略了可变缺陷的特性, 即可变缺陷往往由不超过 6 个特征之间的交互作用导致^[14,15], 仅关注两个特征之间的交互无法准确地实现特征级定位. 其次, 计算所有特征交互的可疑性会面临“组合爆炸”带来的挑战, 导致定位过程极为低效. 因此, 该方法很难在实践中实现有效的缺陷定位. 对于语句级缺陷定位技术, SBFL 技术固有的局限性为 VarCop 计算语句的可疑值带来了挑战. 例如, 对于相同覆盖路径的语句, 基于频谱效应的方法会计算出相同的可疑值, 导致无法准确地识别出根因缺陷语句, 这在许多单系统的缺陷定位研究中已被证实^[12,16,17]. 进一步地, 当缺陷的影响通过程序中的控制依赖或数据依赖向其他语句传播时, SBFL 方法往往无法区分传播效应与缺陷本身的直接影响, 从而导致根因缺陷语句难以被正确识别. 需要说明的是, 根因缺陷是指系统、产品或流程中直接导致问题或缺陷产生的最本质、最核心的因素. 它通常位于问题因果链的源头, 是根因定位过程中最终锁定的核心原因^[18]. 在软件系统中, 根因缺陷语句特指那些直接引发故障的源代码语句, 位于问题因果链的起点, 代表导致故障发生的根本原因^[19].

本文通过一个真实 SPL 系统 (BankAccountTP) 的可变缺陷进行说明. 图 1 展示了 BankAccountTP 系统的特征 Transaction 中 transfer 函数实现, #ifdef 宏指令用来控制特征的选择. 该函数涉及的特征有 Transaction、Lock、BanAccount 和 DailyLimit. 第 11 条语句为缺陷语句 (应为 amount 而非~amount). 这是一个可变缺陷, 因为它不会导致所有的产品无法通过测试. 为简化问题, 假设已确定缺陷语句为 transfer 中的语句. 图 1 给出了每条语句在实

际测试环境中未通过测试样例中执行的次数 f_s 、通过测试样例中覆盖语句的次数 p_s 以及所有通过的测试样例总数 Σ 。若使用先进的 SBFL 指标 (OP2) 计算每条语句的可疑程度 (如列 OP2_{SBFL} 所示), 可以得到第 12 和 13 条语句的可疑程度最高, 其次为第 11 条语句, 这意味着开发人员需要第 3 次检查才能找到错误的语句。VarCop (如列 OP2_{VarCop} 所示) 除了从单元测试结果这一粒度计算语句的频谱效应, 还考虑了产品测试结果这一粒度, 从而提升频谱效应的准确性。然而, 即使采用 VarCop 计算语句的可疑值, 第 11 条语句的可疑值仍排至第 3。产生这一现象的原因是 SBFL 和 VarCop 主要依赖统计数据, 不考虑语句之间的依赖关系和执行顺序。这意味着, 对于一条语句, 它在未通过测试样例/产品中被覆盖次数越多, 而在通过测试样例/产品中被覆盖次数越少, 其可疑程度越高。因此, 由于语句 11、12 和 13 在未通过测试样例中执行次数一致, 但语句 11 在通过测试样例中执行次数更多, 从而导致其频谱效应更低。事实上, 该案例缺陷语句的执行直接影响到后续语句的执行 (语句 12 和语句 13), 并且缺陷传播到后续语句, 导致程序失败。以上结果表明, 频谱效应在一些情形下尽管能反应部分代码语句块的可疑程度, 但无法更为精确地用于定位缺陷语句, 尤其在缺陷通过依赖关系传播的情况下可能会误判可疑语句。此外, 在大规模的 SPL 系统中, VarCop 识别缺陷特征交互时忽略了特征交互的重复生成和检查问题 (详细解释见第 3.3 节), 导致其特征级定位效率仍然低下。

#ifdef Transaction	f_s	p_s	Σ	OP2 _{SBFL}	OP2 _{VarCop}	C
public boolean transfer(Account source, Account destination, int amount) {						
#ifdef Lock						
1 if (!lock(source, destination)) {	11	106	663	10.840	0.873	0.364
2 return false;	0	30	663	-0.045	0.000	0.000
3 }						
4 try {	11	61	663	10.908	0.000	0.000
5 if (amount <= 0) {	11	61	663	10.908	0.935	0.402
6 return false;	0	30	663	-0.042	0.000	0.000
7 }						
#ifdef BankAccount/Dailylimit						
8 if (!source.update(amount*-1)) {	11	31	663	10.953	0.977	1.260
9 return false;	2	15	663	1.977	0.118	0.500
10 }						
#ifdef BankAccount/Dailylimit						
11 if (!destination.update(~amount)) {	11	16	663	10.976	0.998	5.000
12 source.undoUpdate(amount*-1);	11	12	663	10.982	1.000	2.500
13 return false;	11	12	663	10.982	1.000	2.500
14 }						
15 return true;	0	4	663	-0.006	0.000	0.000
16 } finally {						
#ifdef Lock						
17 source.unlock();	11	61	663	10.908	0.935	1.700
18 destination.unlock();	11	61	663	10.908	0.935	1.700
19 }						
20 }						

图 1 针对 BankAccoutTP 系统, 不同定位方法所计算的语句可疑值

为弥补以上不足, 本文针对 SPL 提出了一种融合因果和频谱效应的高效缺陷定位方法 FCS-FL, 旨在提升特征级缺陷定位的效率以及语句级缺陷定位的精准性。FCS-FL 首先识别缺陷的特征交互, 进而检测出缺陷语句。在特征级定位上, FCS-FL 利用通过产品与未通过产品间的特征选择差异 (具体定义见第 3.3 节) 来识别出可疑特征交互。与现有方法不同的是, FCS-FL 采用了一种可疑特征交互去重策略, 该策略利用缓存技术以大幅提升识别效率。接着, 对可疑特征交互进行程序切片以隔离出可疑语句集合。在语句级定位中, 基于程序依赖关系构建一种引入中介变量的约简因果图模型, 从而识别和估计可疑语句对测试结果的因果效应, 并将其与频谱效应融合以实现更准确的语句级缺陷定位。如图 1 所示, 列 C 为使用本文提出的约简因果图模型所计算出可疑语句导致缺陷的因果效应, 结果显示第 11 条语句的可疑程度最高。本文提出的模型既考虑了影响语句 11 的语句, 也考虑了语句 11 所影响的语句。因此, 在这一案例中可以更精确地定位到缺陷语句, 即根因缺陷定位。在实验上, 选择了 4 个先进基

准算法,在6个真实SPL系统上完成多组对比实验,以验证本文提出方法在SPL系统上语句级缺陷定位问题上的精确性.本文的主要贡献包括以下3方面.

1) 在特征级定位过程中,基于特征选择差异设计一种可疑特征交互去重策略.该策略通过可疑特征选择集合之间存在包含关系和相同子集两点特性,利用缓存技术避免重复生成和检查大量的特征交互,有效地提升特征级定位的效率.

2) 提出一种引入中介变量的约简因果图模型评估语句对测试结果的因果效应.该模型既约简语句间的因果关系以提升评估效率,又通过考虑中介变量使因果效应评估更加准确.值得说明的是,由于代码结构的可变性,因果技术尚未被应用于SPL的缺陷定位中.

3) 融合语句的频谱效应和因果效应以实现更稳定的缺陷定位:当因果模型因样本稀疏或模型简化出现偏差时,频谱度量可提供冗余支持;而当统计相关误导频谱度量时,因果分析可加以修正.实验结果表明,这两种效应的融合可以提升方法定位根因缺陷语句的能力.

本文第1节介绍缺陷定位的相关方法和研究现状.第2节介绍本文所需的基础知识,包括SPL和潜在结果模型.第3节详细描述本文提出的方法.第4节通过多组对比实验说明提出方法的有效性.第5节总结全文.

1 缺陷定位相关工作

本节分别从单系统软件和SPL系统两方面介绍和总结缺陷定位问题的相关工作.

1.1 单系统软件的缺陷定位技术

单系统软件中的缺陷定位技术较为成熟,常见的技术包括基于程序频谱的方法、基于突变的缺陷定位方法和基于因果推断的方法等^[20-27].基于频谱的方法一直是研究者关注的重点之一.为处理不同场景中的局限性,研究者提出了许多频谱技术来实现缺陷定位.例如, Jones等人^[21]提出了Tarantula指标来区分和平衡通过与失败测试的影响,进而更直观和有效地定位软件缺陷. Abreu等人^[22]为了处理通过测试对可疑值计算的负面影响,提出了OP2指标.该指标可以平滑怀疑度计算结果,避免极端值对定位结果的干扰,从而提高缺陷定位的准确性和可靠性.然而,研究表明,基于频谱的方法对相同覆盖路径的程序语句无法实现精确的缺陷定位^[12,16,17].为了克服频谱方法在某些情况下的不足,研究人员提出了基于突变的缺陷定位方法.这类方法主要通过分析突变程序与原始程序在执行测试用例时的差异来定位缺陷.例如, Papadakis等人^[23]提出了Metallaxis方法,通过产生多个程序突变体并执行失败测试用例来识别表现出类似失败行为的变体,从而精确推断出缺陷所在位置. Li等人^[24]进一步结合频谱分析与变异分析,通过计算变异体与失败测试之间的相关性,有效缩小了可疑代码范围,提升了定位精确性.然而,基于突变的方法通常需要生成大量的突变体并逐一执行测试,导致方法在实际应用中计算开销较大,耗时较长,限制了其在大规模软件系统中的应用.基于因果推断的方法旨在挖掘程序语句间的因果关系来定位软件缺陷. Cleve等人^[13]最早提出了使用因果推理技术进行细粒度缺陷定位的方法,该方法通过比较和分析程序的成功与失败执行数据,并系统地操纵程序状态,从而识别出特定语句级别的错误.为进一步提升因果效应评估的准确性, Baah等人^[16]利用程序依赖关系构建因果图,并通过回归技术计算因果效应.此外,他们进一步基于程序依赖约简方法缓和混杂因子产生的影响,使算法能更准确地得到程序语句对测试结果的因果效应.在最近的研究中, Zeng等人^[12]利用因子图计算语句的因果效应,并通过程序插桩技术增强观察数据的准确性.

综上所述,基于频谱的缺陷定位技术和因果技术在单系统软件缺陷定位中已被广泛应用,并均在不同情形下表现出了显著的性能优势.不同于现有研究的是,本研究同时考虑了这两种技术,以增强方法在不同情形下的定位能力.此外,因果技术尚未在SPL系统缺陷定位中得到应用,本文的研究对此进行了补充和拓展.

1.2 SPL系统的缺陷定位技术

相比之下,SPL系统的缺陷定位研究非常稀缺.尽管可以直接用单系统软件的缺陷定位技术对单个产品逐一进行缺陷定位,但这极为低效.因此,在使用单系统软件的缺陷定位技术前,常利用程序切片技术^[28,29]来缩小搜索空间,从而提升定位的效率.例如, Chaleshtari等人^[30]考虑到基于突变的缺陷定位极为耗时,通过程序切片技术获

得可疑语句,并只检查被隔离的语句.然而,这些方法无法获得更高的性能,因为它们无法处理特征交互对程序的影响.与之不同,Arrieta等人^[5]首先根据特征选择引发缺陷的概率对特征进行排序;接着提出了一种缺陷隔离方法,通过自动分析特征模型,生成包含最可疑特征的最小规模的产品,以方便缺陷原因的隔离.受他们工作的启发,Nguyen等人^[11]观察到可变缺陷的特性,并基于这些特性实现特征级的定位.接着,他们基于切片技术缩小搜索空间,并利用SBFL技术实现SPL系统的语句级缺陷定位.

尽管现有研究在SPL系统缺陷定位方面取得了一定进展,但在特征级和语句级缺陷定位上仍有进一步研究的空间.本文研究旨在提高特征级缺陷定位的效率和语句级缺陷定位的精准性.

2 基础知识

本节简要介绍相关背景知识,包括SPL、特征交互缺陷和潜在结果模型.

2.1 SPL

SPL是一种软件工程方法论,利用软件复用的理念,通过系统化地共享关键代码库来创建一系列相关软件产品^[31,32].形式上,SPL可由一个三元组 $\Gamma = \langle \mathbb{S}, \mathbb{F}, \varphi \rangle$ 表示,其中 \mathbb{S} 是用于实现系统 Γ 的程序语句集合, \mathbb{F} 表示系统的特征集合, φ 为特征实现函数.对于一个特征 f , $\varphi(f) \subset \mathbb{S}$ 表示特征 f 在系统 Γ 中的语句实现,并且 $\varphi(f)$ 包含在选择特征 f 的所有产品中.对于一个特征集合 $\mathbb{F} = \{f_1, f_2, \dots, f_{|\mathbb{F}|}\}$,每个特征 $f_i \in \mathbb{F}$ ($i = 1, 2, \dots, |\mathbb{F}|$)均有两种状态,选择(打开)或不选择(关闭).SPL系统中特征的存在可视为if-then语句,例如预编译指令`#ifdef`根据特征的选择控制相应代码的编译和运行.配置(或产品)是在所有特征上的不同选择,形式上可表示为 $\{\pm f_1, \pm f_2, \dots, \pm f_{|\mathbb{F}|}\}$,其中每个特征根据选择状态只能表示为 $+f$ 或 $-f$.特征交互在逻辑上即为部分配置.例如,一个特征数为5的SPL系统的特征集合为 $\{f_1, f_2, f_3, f_4, f_5\}$,其一个产品的配置可表示为 $c = \{+f_1, -f_2, -f_3, +f_4, +f_5\}$,其中 $(-f_3, +f_4)$ 为 c 的一个特征交互.满足特征模型的配置为有效配置,本文只针对有效配置执行缺陷定位任务.

2.2 特征交互缺陷

特征之间的交互作用可能导致系统发生无法预料的缺陷^[33].特征交互缺陷可根据系统是否发生功能缺陷,分为非功能型特征交互缺陷和功能型特征交互缺陷^[34].非功能型特征交互缺陷是指由于特征之间的交互作用,影响到软件系统的非功能性属性,如性能、可用性、可靠性、安全性等.这种缺陷不一定会导致功能失效,但会使系统在某些方面表现不佳^[35].相反,功能型特征交互缺陷会直接影响到软件系统的功能性表现,导致系统功能失效或不符合预期的行为.需要注意的是,在无特别说明的情况下,本文提到的特征交互缺陷为功能型特征交互缺陷.

对于一个SPL系统 Γ ,缺陷的特征交互 B 应属于所有未通过产品的配置.检查未通过产品配置的所有特征交互是低效且非必须的.例如,对于采样产品的配置集合 $C = C_p \cup C_f = \{c_1, c_2, \dots, c_n\}$,其中 C_p 和 C_f 分别表示测试通过和未通过的配置集合.任意一个配置 $\exists c \in C_p$ 的特征交互都不会是导致可变缺陷的根本原因.换言之, B 应同时满足缺陷相关性和最小性^[11,14].

- 1) 缺陷相关性是指任何包含 B 的配置都对应于一个未通过的产品.
- 2) 最小性是指 B 中任一子集都无法满足缺陷相关性.

直观地,缺陷特征交互 B 必然属于 C_f 而不属于 C_p .事实上,可能存在多个特征交互同时满足缺陷相关性和最小性,本文将这些特征交互作为可疑特征交互.

2.3 潜在结果模型与因果效应估计

潜在结果模型是一种用于估计因果效应的理论框架^[36].该模型通过比较在不同处理条件下的潜在结果来确定处理对个体或群体的影响.具体而言,每个受试单位(如个体、群体或实验对象)在接受或未接受处理时都有一个潜在结果^[37].通过比较这些潜在结果,研究人员可以估计处理的因果效应.假设处理变量 T 是一个二元变量,取值为1表示接受处理,取值为0表示未接受处理.那么,所有受试单位在 $T = 1$ 时的结果 Y^1 称为“处理后的潜在结果”,在 $T = 0$ 时的结果 Y^0 称为“未处理的潜在结果”.

对于每个受试单位, 处理效应定义为处理与未处理的潜在结果之差^[38]. 由于在实际研究中无法同时观察到同一个体的两种潜在结果, 通常利用平均处理效应 (average treatment effect, ATE) 来衡量总体的因果效应, 即:

$$\tau = E[Y^1] - E[Y^0] \quad (1)$$

其中, $E[\cdot]$ 为期望算子.

在缺陷定位任务中, 数据通常是通过运行一组测试样例并记录它们的执行信息获得的^[17]. 在这种情况下, 实际的实验过程是观察性研究而非随机性实验. 因此, 覆盖一条语句并不是随机分配到处理组或对照组. 为了在观察性研究中估计平均的处理效应, 需要通过控制潜在混杂因子的观察值, 来减少混杂变量对因果推断的影响. 此时, 处理变量 T 对结果变量的平均处理效应可通过调节一组协变量计算, 使得 T 条件独立于结果变量 Y^1 和 Y^0 , 即:

$$(Y^0, Y^1) \perp T | X \quad (2)$$

为此, 公式 (1) 可转换为:

$$\tau = E[E[Y|T=1, X] - E[Y|T=0, X]] \quad (3)$$

此时, 因果效应估计问题即可转化为在控制这些变量条件下, 对处理变量与结果变量之间关系的建模与量化. 回归模型是一种常用的方法, 用于通过控制观察到的混杂因子来估计平均因果效应^[39], 例如线性模型.

$$Y = \alpha + \tau T + \beta X + \varepsilon \quad (4)$$

其中, α 表示截距项, T 为处理变量, Y 是结果变量, X 是包含混杂因子的协变量向量, β 为 X 的系数向量, ε 是误差项. 该模型被成功应用于单系统软件的缺陷定位任务中^[26]. 具体而言, Y 表示为测试结果变量, T 为处理语句, X 为 T 的混杂因子的协变量向量. 从测试过程中捕捉到的观察数据可以拟合出相应的回归模型^[40], 其系数 τ 即为 T 对 Y 的因果效应.

3 基于因果和频谱效应融合的高效 SPL 可变缺陷定位方法

本节首先简要介绍 SPL 系统中缺陷定位的问题描述, 接着详细介绍 FCS-FL 的基本框架及实现细节, 包括可疑特征交互识别、可疑语句隔离、可疑值计算 (因果效应和频谱效应的计算与融合) 及可疑语句排序.

3.1 问题描述

对于一个存在可变缺陷的 SPL 系统 $\Gamma = \langle S, F, \varphi \rangle$, 其采样产品的配置集合 $C = C_p \cup C_f$, 缺陷定位任务可以描述为识别出导致 C_f 的缺陷特征交互 B , 并输出该特征交互相关的语句集合 $S' \subseteq S$ 及每条语句所对应的可疑值. 形式上, 旨在构造一个映射函数 $\Phi(\cdot)$, 使得每条可疑语句 $s \in S'$ 映射为一个常数值 $\Phi(s)$, 即 s 的可疑值. 理论上, 一个好的定位方法应保证 S' 中所包含的语句数应尽可能少, 并且缺陷语句的可疑值尽可能高.

3.2 基本框架

本文提出的 FCS-FL 方法的基本框架如后文图 2 所示, 包含两个阶段: 特征级缺陷定位和语句级缺陷定位. 在特征级缺陷定位阶段, 基本的流程如下: 首先, 基于缺陷相关性计算每个未通过产品的潜在特征选择集合; 其次, 移除被包含的可疑特征选择集合以避免生成重复部分配置; 接着, 对于每个可疑特征选择集合, 生成相应的 1-7 阶潜在特征交互集合, 并使用缓存机制防止重复检查相同的特征交互. 最后, 保存满足缺陷相关性的可疑特征交互, 用于定位缺陷语句. 在语句级缺陷定位阶段, 基本的流程是: 首先, 对每个未通过产品中的可疑部分配置利用程序切片技术隔离出可疑语句. 接着, 对每一条可疑语句, 分别计算它们的频谱效应和因果效应, 并加权融合因果效应和频谱效应. 最后, 根据融合后的可疑值进行排序, 检查可疑性高的语句即可实现缺陷定位.

3.3 高效的可疑特征交互识别

如第 2.1 节中所述, 缺陷特征交互应同时满足缺陷相关性和最小性. 事实上, 可能存在多个特征交互同时满足这两点性质. 可利用未通过产品与通过产品在特征选择上的差异, 有效地识别出所有满足缺陷相关性和最小性的特征交互, 并将它们作为可疑特征交互^[8]. 为便于解释, 本文有以下关键定义.

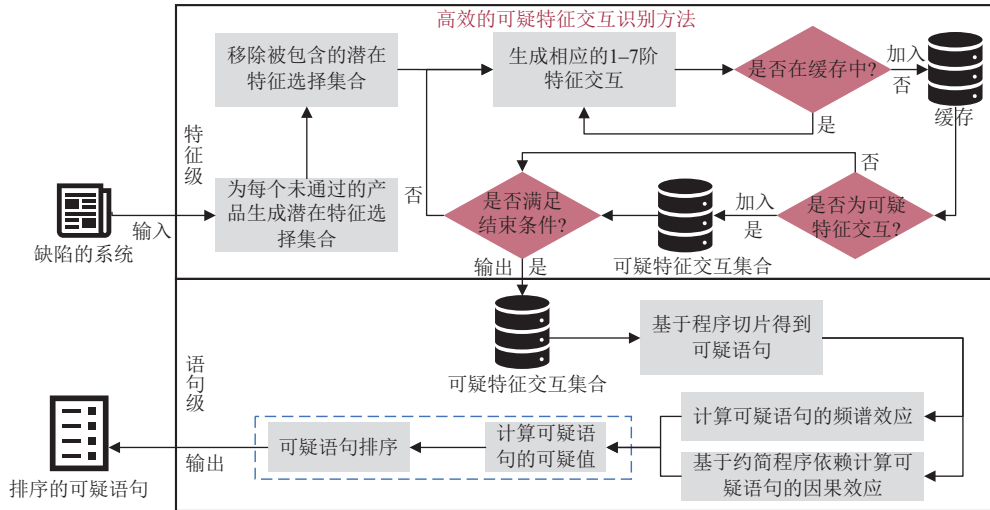


图2 FCS-FL 的框架图

定义 1. 特征选择差异. 若两个产品的配置在同一特征上有不同的选择, 则表示在该特征上这两个产品存在特征选择差异.

例如, 存在两个产品配置分别为 $c_1 = \{-f_1, +f_2, +f_3\}$ 和 $c_2 = \{+f_1, +f_2, -f_3\}$, 这两个配置在特征 f_1 和 f_3 上存在特征选择差异.

定义 2. 差异特征选择集合. 一个产品配置 c_1 相对于另一个产品配置 c_2 所有存在特征选择差异的特征集合, 即为 c_1 相对于 c_2 的差异特征选择集合 $D(c_1, c_2)$.

例如, 存在两个产品配置分别为 $c_1 = \{-f_1, +f_2, +f_3\}$ 和 $c_2 = \{+f_1, +f_2, -f_3\}$, 则 c_1 相对于 c_2 的差异特征选择集合为 $D(c_1, c_2) = \{-f_1, +f_3\}$.

定义 3. 可疑特征选择集合. 对于一个未通过的产品配置 c 和所有通过的产品配置集合 PCs , c 的可疑特征选择集合为 c 与 PCs 中所有产品的差异特征选择集合的并集, 表示为 $DS(c) = \bigcup_{c' \in PCs} D(c, c')$.

例如, 存在一个有 3 个产品配置的系统, 它包含一个未通过的产品配置 $c = \{-f_1, +f_2, +f_3, +f_4\}$ 和两个通过的产品配置 $PCs = \{c_1, c_2\}$, 其中 $c_1 = \{+f_1, +f_2, -f_3, +f_4\}$ 和 $c_2 = \{+f_1, +f_2, +f_3, -f_4\}$. 基于定义 2, c 较 c_1 的差异特征选择集合为 $D(c, c_1) = \{-f_1, +f_3\}$, c 较 c_2 的差异特征选择集合为 $D(c, c_2) = \{-f_1, +f_4\}$. 因此, c 的可疑特征选择集合为 $DS(c) = D(c, c_1) \cup D(c, c_2) = \{-f_1, +f_3, +f_4\}$.

定义 4. 潜在特征交互集合. 对于一个未通过的产品配置 c , 其可疑特征选择集合的所有 1-7 阶子集的并集即为 c 的潜在特征交互集合.

需要说明的是, 实践中大多数特征交互缺陷涉及的特征数都不超过 6 个^[14], 因此本文只取 1-7 阶子集以保证检查效率.

例如, 存在一个未通过的产品配置 c , 它的可疑特征选择集合 $DS(c) = \{-f_1, +f_3, +f_4\}$, 其 1 阶子集为 $\{(-f_1), (+f_3), (+f_4)\}$, 2 阶子集为 $\{(-f_1, +f_3), (+f_3, +f_4), (-f_1, +f_4)\}$, 3 阶子集为 $\{(-f_1, +f_3, +f_4)\}$. 因此, c 的潜在特征交互集合为 $\{(-f_1), (+f_3), (+f_4), (-f_1, +f_3), (+f_3, +f_4), (-f_1, +f_4), (-f_1, +f_3, +f_4)\}$.

定义 5. 可疑特征交互. 满足缺陷相关性和最小性的特征交互即为可疑特征交互.

在本文的研究中, 所有未通过产品的潜在特征交互集合即为搜索空间. 本文提出的高效的可疑特征交互识别方法即从搜索空间中识别出所有可疑的特征交互. 该方法通过对所有未通过的产品配置计算它们的可疑特征选择集合, 进而生成潜在特征交互集合并检查每个特征交互的可疑性 (即检查是否满足缺陷相关性和最小性).

所提方法基于以下两点特性来提升检查效率.

1) 未通过产品的可疑特征选择集合之间存在包含关系, 这会导致生成重复的特征交互. 例如, 存在两个未通过产品的可疑特征选择集合分别为 $DS_1 = \{-f_1, -f_2, +f_3, -f_4\}$ 和 $DS_2 = \{-f_1, -f_2, +f_3, -f_4, -f_5\}$. 显然 $DS_1 \subseteq DS_2$, 导致它们存在许多相同的子集 (特征交互). 若不进行处理, 这些相同的特征交互将被重复生成并检查它们的可疑性. 因此, 舍弃被包含的可疑特征选择集合, 以避免相同的特征交互被重复生成.

2) 不同潜在特征交互集合之间存在相同的子集, 将使算法重复检查相同的特征交互. 例如, 存在另一个差异特征集合 $D_3 = \{+f_1, +f_2, +f_3, -f_4\}$, 其子集集合为 $\{\dots, (+f_2, +f_3), (+f_3, -f_4), \dots\}$, 而 D_2 的子集集合为 $\{\dots, (-f_2, +f_3), (+f_3, -f_4), \dots\}$, 它与 D_3 的子集集合存在相同的子集 $(+f_3, -f_4)$. 这里仅展示了它们的部分子集, 事实上存在大量相同的特征交互, 意味着许多相同的特征交互会被重复检查. 为此, 利用缓存机制 (如哈希表) 保存被检查过的特征交互, 若当前特征交互在缓存中, 则不进行可疑性检查, 反之进行可疑性检查. 通过这种方法, 可以有效避免重复检查, 进而提高特征级缺陷定位的效率.

可疑特征交互识别方法的伪代码如算法 1 所示. 首先, 见算法 1 第 5 行, 对于每个未通过产品的配置, 计算它们的可疑特征选择集合. 对于不同未通过产品之间的可疑特征选择集合可能存在包含关系, 如第 6-8 行所示, 舍弃被包含的可疑特征选择集合. 接着, 算法 1 中第 9-19 行表示, 对未被舍弃的可疑特征选择集合, 基于缓存机制生成相应的特征交互并进行可疑性检查. 最后, 输出可疑的特征交互集合.

算法 1. 高效的可疑特征交互识别方法.

输入: 所有的产品配置集合 $C = PCs \cup FCs$, 其中 PCs 和 FCs 分别表示通过与未通过的产品配置集合;

输出: 可疑的特征交互集合 FIs .

```

1.  $FIs \leftarrow \emptyset$ 
2.  $Cache \leftarrow \emptyset$  //用于保存检查过的特征交互
3.  $\Delta \leftarrow \emptyset$  //保存未通过产品的可疑特征选择集合
4. for  $i \leftarrow 1$  to  $|FCs|$  do //遍历未通过产品集合
5.    $\Delta_i \leftarrow DS(FCs_i)$  //  $\Delta_i$  存储  $FCs_i$  的潜在特征交互集合
6.   if  $\Delta_i$  被包含于  $\Delta$  中任意一个集合
7.     continue
8.   end if
9.   for  $k \leftarrow 1$  to 7 do
10.     $\Omega \leftarrow KItemSet(\Delta_i, k)$  //  $KItemSet(\Delta_i, k)$  表示计算  $\Delta_i$  的  $k$  阶子集
11.    for  $j \leftarrow 1$  to  $|\Omega|$  do
12.      if  $\Omega_j \in Cache$ 
13.        continue
14.      else if  $\Omega_j$  满足缺陷相关性和最小性
15.         $FIs \leftarrow \Omega_j$ 
16.      end if
17.       $Cache \leftarrow \Omega_j$ 
18.    end for
19.  end for
20. end for
21. return  $FIs$ 

```

本文所提可疑特征交互识别方法的效率与被检查系统本身有关, 包括其特征的数量和测试的配置数. 在最坏情况下, 即可疑特征选择集合之间不存在包含关系以及没有相同的子集, 提出方法的时间为 $O(|FCs| \times |\Omega|)$, 其中

$|FC_S|$ 为未通过的配置数, $|\Omega|$ 为生成的 1-7 阶子集数且其随特征数指数式增长. 事实上, 绝大部分的案例下不存在上述两种情况 (见实验部分 RQ1), 提出方法的识别效率会大幅改善.

最后, 得到所有可疑特征交互之后, 根据文献 [11] 的实践, 运用程序切片技术隔离出可疑语句, 为后续可疑语句的可疑性评估奠定基础. 详细的隔离方法参见文献 [11].

3.4 基于约简程序依赖的因果效应评估

本节详细介绍所提出的约简因果模型, 其与现有模型的区别以及因果效应的计算方法.

程序的控制依赖和数据依赖关系一定程度上反映了语句间的因果关系^[41]. 此外, 在程序分析情境中, 对某条语句的干预 (如屏蔽或插桩) 在固定输入与运行环境下会产生确定且可重复的输出, 不存在“一次处理对应多种结果”的歧义, 因而满足因果推断中的一致性假设. 因此, 本文利用程序依赖图 (program dependence graph, PDG) 构造可用于因果效应评估的因果图模型, 进而计算可疑语句对测试结果的因果效应. 首先, 为所有可疑语句生成相应的变量节点, 节点的值是二元的, 表示在未通过产品的测试中是否覆盖了该语句. 对于一条可疑语句 s , 若覆盖了这条语句, 则 s 所对应的变量节点的值为 1, 否则为 0. 接着, 构建节点间的因果关系, 其中节点间的因果关系由语句间的依赖关系决定. 对于两条语句 s_1 和 s_2 , 若 s_1 对 s_2 存在数据或控制依赖关系, 则为 s_1 对应变量节点生成一条指向 s_2 对应节点的边. 最后, 为测试结果生成变量节点 Y , 其值为 1 或 0, 分别表示测试通过或未通过. 由于每条可疑语句都可能导致测试未通过, 因此为每条可疑语句变量节点连接一条指向测试结果节点的有向边.

然而, 基于 PDG 得到的完整因果图模型并不适合进行因果效应评估. 一方面, 混杂因子的存在会影响因果估计的准确性; 另一方面, 较长的因果链会降低方法的评估效率 (见实验部分 RQ4). 此外, 现有研究表明 PDG 中的路径可能存在环^[26], 导致相应的因果图无法直接用于因果效应评估. 为消除有向图中的环, Aho 等人^[42]提出了一种传递约简技术, 该技术通过找到一个最小的有向子图, 使其保留原始图中的所有可达性关系, 从而得到一个等价的可用于因果评估的有向无环图. 受他们工作的启发, 对于一个产品系统中任何一个可疑语句节点 (即处理节点), 仅保留它所有的父节点和子节点的因果关系, 以得到用于因果效应评估的因果图模型. 在该因果图模型中, 处理节点与其子节点之间的结构性依赖关系构成了从处理变量到结果变量的重要传导路径. 在此基础上, 所构建的约简因果图通过纳入处理节点的父节点以控制潜在混杂因素, 并通过去环策略切断无关路径, 确保了后门准则的适用性. 这一设计使得基于回归的效应估计既能消除混杂偏差, 又能获得一致的因果效应估计.

下面对所提出的约简因果图进行理论分析. 在此之前, 有必要对因果推断中若干核心概念进行说明.

1) 后门准则: 若一组协变量能够阻断所有从处理变量 T 到结果变量 Y 的后门路径, 且不包含任何交汇节点及其后代, 则该协变量集合满足后门准则, 可用于在因果效应估计中控制混杂影响^[39].

2) 后门: 在因果图中, 若一条连接 T 与 Y 的路径在 T 一端的第 1 条边指向 T , 则称该路径从 T 进入的方式为“后门”. 后门的存在通常意味着处理变量与结果变量之间可能存在共同原因或混杂因素.

3) 后门路径: 是指从 T 到 Y 的一条路径, 其第 1 条边指向 T , 并且该路径不是 $T \rightarrow Y$ 的直接因果路径. 这类路径可能传递混杂效应, 导致因果效应估计产生偏差.

4) 交汇节点: 在因果图中, 若某个节点同时接收来自两条或多条有向边的输入, 则该节点为交汇节点. 对交汇节点进行条件化可能开启原本阻断的路径, 引入额外的依赖关系, 因此在变量选择中应予以规避.

5) 条件化: 在概率计算或建模中, 将一组变量固定为特定值或纳入条件概率的已知部分. 在因果推断中, 对满足后门准则的变量集合进行条件化, 相当于在分析中控制这些变量, 从而阻断混杂路径, 获得无偏的因果效应估计.

命题 1. 后门可调性. 在约简因果图 G' 中, 处理语句 s 的父节点集合 $P(s)$ 构成阻断 $s \rightarrow Y$ 所有后门路径的最小调整集.

解释: 在 G' 中, 根据后门的定义, 每条后门路径 π 必含一个与 s 相邻且指向 s 的父节点 v . 对 $P(s)$ 条件化即阻断了 π 在该非交汇节点处的传播, 从而所有后门路径均被阻断. 若路径经由交汇节点重新开启, 需要该交汇节点或其后代被条件化才能打开路径; 由于 $P(s)$ 不包含交汇节点的后代且图经传递约简后无额外祖先-后代混杂, 路径仍

被阻断. 因此, $P(s)$ 阻断 s 的所有后门路径成立. 接着, 设不再对 $P(s)$ 中的一个节点 v 进行条件化, 这将导致暴露一条长度为 2 的后门路径 $s \leftarrow V \rightarrow Y$, 因此 $P(s)$ 中缺少任一节点都不再阻断全部后门路径, 证明最小性. 综上, $P(s)$ 是满足后门准则的最小调整集.

命题 2. 顺序可忽略性. 约简因果图 G' 在固定输入与无并发环境下, 满足处理独立性和中介独立性.

解释: 显然, $P(s)$ 包含了全部可影响 s 与 Y 的上游状态变量. 不同于其他问题, 软件程序是确定性转移系统, 这意味着固定 $P(s)$ 后 s 的取值由控制-数据依赖唯一确定, 即处理独立性成立. 同理, 给定 s 和 $P(s)$, 子节点集合 $M(s)$ 的执行状态完全由控制-数据依赖决定, 且程序输出 Y 仅再受下游状态影响, 因此中介独立性亦成立.

命题 3. 无交互性. 对于处理变量 T 、中介变量 $R = (R_1, R_2, \dots, R_k)$ 与结果变量 Y , 在给定协变量 X 的条件下, $\frac{\partial^2 Y}{\partial T \partial R_j} = 0, j = 1, 2, \dots, k$, 即 T 对 Y 的边际效应不随任何单个中介变量 R_j 的取值而变化, 处理效应与中介效应可加而无交互项.

解释: 在 G' 中, s 与每个子节点 $m \in M(s)$ 的唯一交汇节点是测试结果节点 Y . 这意味着不存在既受 s 又受 m 共同影响的第 3 节点; 也不存在 $s \leftarrow m \rightarrow s$ 型的交汇节点. 意味着 s 与 m 之间无公共子后代. 此外, 所有 $s \rightarrow Y$ 的直接路径与 $s \rightarrow m \rightarrow Y$ 的间接路径仅在 Y 会合, 无其余交汇节点. 从图论意义上, 这说明两条路径可加而非相乘或相互调节, 意味着 $\frac{\partial^2 Y}{\partial T_s \partial R_m} = 0$. 为此, 无交互性假设在 G' 中成立, 可使用线性可加模型估计直接效应与间接效应, 故公式 (4) 可修改为:

$$Y = \alpha + \tau T + \beta X + \nu R + \varepsilon \quad (5)$$

其中, R 为中介变量向量, ν 表示中介通路系数.

命题 4. 可观测性. 约简因果图 G' 中的所有节点均对应可测随机变量.

解释: 语句节点可通过频谱信息记录其执行标志 $\{0, 1\}$, 其父节点与子节点皆来源于 PDG, 可在一次执行中被唯一确定. 此外, 结果节点由测试通过/失败输出直接给出. 因此, 提出的约简因果图中不存在任何未测量的潜在变量或缺失依赖, 满足因果推断所需的可观测性前提.

综上, 由于命题 1-4 同时成立, 则处理语句对测试结果的因果效应可由公式 (5) 得到. 并且基于公式 (5) 得到的因果模型相比于基于公式 (4) 的因果模型, 其误差项 ε 中的方差更小. 因为公式 (5) 考虑了缺陷在程序间的传递影响, 更适合软件的缺陷定位任务.

为更清晰地说明提出因果图模型的构造细节, 用程序 transfer (见第 3 节) 进行说明, 其完整的 PDG 如图 3(a) 所示, 每个节点的值对应程序中语句的行数, 两个节点间的有向边表示语句间的依赖关系. 以语句 11 作为处理节点为例, 提出的约简因果模型如图 3(b) 所示, 保留语句 11 及其父节点和子节点的依赖关系, 并新建测试结果变量节点 Y , 构建所有到 Y 的有向边. 要计算变量节点 11 对节点 Y 的因果效应, 首先利用后门准则确定混杂因子. 显然, 节点 8 阻断了节点 11 到 Y 的全部路径, 因此可得到后门路径 $11 \leftarrow 8 \rightarrow Y$. 接着, 将节点 12、13 和 15 作为中介变量纳入节点 11 对节点 Y 的因果效应评估中.

与现有技术不同的是, Baah 等人^[26]提出的因果模型如图 4 所示, 该模型仅保留节点 11 的父节点来约简因果关系. 尽管这种方式得到的因果图模型可以消除环, 并减轻混杂因子对因果效应评估带来的影响. 然而, 与其他领域不同的是, 缺陷定位任务中的观察数据由程序的执行过程得到. 因此, 由处理节点到测试结果节点的因果效应可能通过中间节点的传导. 换言之, 处理节点除了自身对结果节点的直接效应, 还可能通过子节点传递间接影响. 例如, 在 transfer 程序中, 语句 11 导致测试无法通过是因为它的执行进一步触发了语句 12 和 13 的执行. 因此, 本文提出的约简因果图模型将处理节点的子节点作为中介变量, 将其纳入因果效应计算中以减少误差项 ε 中的方差, 进而提升了评估的准确性.

综上所述, 对于每个未通过的产品系统, 每一条可疑语句均可以基于提出的因果图模型计算出对测试结果的因果效应. 然而, 由于不同产品系统具有不同的程序语句及依赖关系, 导致可疑语句在不同产品中会得到不同的因果效应值. 对于每一条可疑语句, 本文使用不同产品的算术平均值进行聚合, 聚合后的因果效应值即为最终的值.

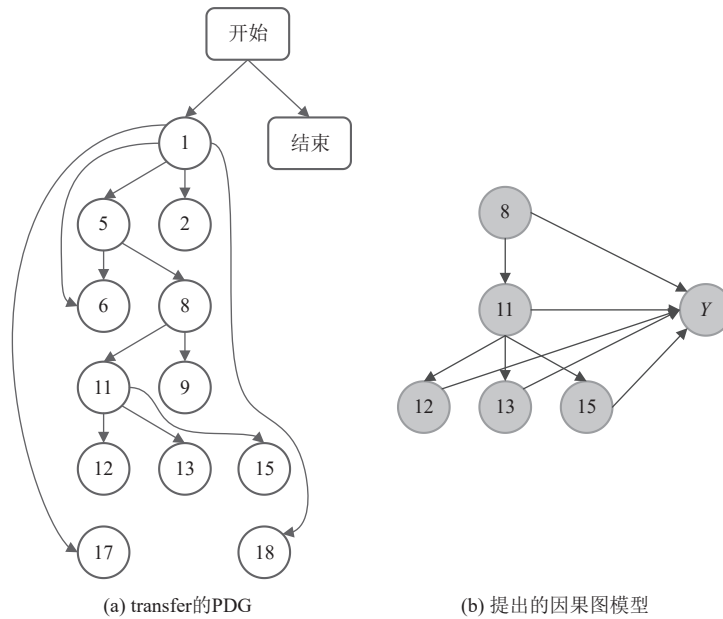


图3 程序 transfer 的 PDG 和提出的因果图模型

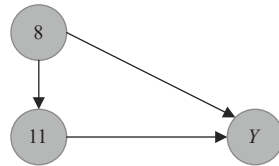


图4 Baah 等人^[26]提出的因果图模型

3.5 可疑语句的可疑值计算及排序

对于任意一条可疑语句 s , 同时考虑其对测试结果的因果效应和频谱效应. 线性加权聚合作为一种模型平均策略, 避免了单一度量的不足, 有利于降低评估的总体误差. 因此, 本文采用互补信息的线性加权聚合方式融合因果效应和频谱效应. 该方式既能保留直接与间接路径的因果效应, 又能兼顾频谱效应所提供的全局统计相关性信息. 融合后的值作为可疑语句的可疑值. 设 $E = [e_1, e_2, \dots, e_{|S|}]$ 表示所有可疑语句对测试结果的因果效应. 为突出更高因果效应的语句对导致测试未通过的影响, 在融合前, 采用 Softmax 函数对所有可疑语句的因果效应值进行归一化. 设归一化后所有可疑语句的因果效应值为 $E' = [e'_1, e'_2, \dots, e'_{|S|}]$. 接着, 对于每一条可疑语句 s , 融合其因果效应和频谱效应作为最终语句 s 的可疑值, 即:

$$suspiciousness(s) = \mu \times E'(s) + (1 - \mu) \times H(s) \tag{6}$$

其中, μ 是平衡因果效应和频谱效应的权重值, $H(s)$ 表示语句 s 的频谱效应. $suspiciousness(s)$ 值越大, 表示语句 s 的可疑性越高. 因果效应与频谱效应往往在不同场景下互补: 当因果效应因样本稀疏或模型简化出现偏差时, 频谱效应可提供冗余支持; 而当统计相关误导频谱效应时, 因果效应可加以修正. 得到所有可疑语句的可疑值后, 利用快速排序算法对可疑语句根据可疑值进行排序.

在本文的研究中, 可疑语句的频谱效应计算从基于产品和基于单元测试两个粒度进行评估. 基于产品的评估从产品测试通过或不通过的角度对可疑语句进行评估. 具体来说, 根据测试通过和未通过产品覆盖可疑语句 s 的数量, 计算语句 s 的可疑程度. 直观而言, s 在未通过的产品中出现得越多, 在通过的产品中出现得越少, 其可疑性越高. 具体的计算规则可以采用特定 SBFL 指标度量, 如 OP2. 基于单元测试的评估则仅针对未通过产品中的单元

测试. 同样地, 根据每条可疑语句在通过的单元测试和未通过的单元测试中的覆盖情况, 使用适合的 SBFL 指标进行度量.

实际上, 基于产品的评估是从全局视角对可疑语句进行度量. 该评估方式既考虑了通过产品的运行情况, 也考虑了未通过产品的运行情况. 而基于单元测试的评估只考虑未通过的产品, 是从局部度量可疑语句. 对于每一条可疑语句 s , 同样采用线性加权聚合的方式平衡其基于产品评估 $ps(s)$ 和基于单元测试评估 $ts(s)$, 进而最小化总体误差, 即:

$$H(s) = \eta \times ps(s) + (1 - \eta) \times ts(s) \quad (7)$$

其中, η 表示权重值常数.

4 实验设置

本节详细地介绍实验实施细节, 包括数据集、研究问题和评价指标及基准方法. 所有实验均在一台硬件条件为 Intel Core i5-12450H CPU (2.00 GHz) 和 16.00 GB RAM 的 Linux 服务器上完成. 对于编程语言, 采用 Python 实现可疑特征交互识别和语句可疑性评估, 程序切片利用 IBM 公司开源的基于 Java 的 Wala 框架实现. 此外, 所有的实验结果以及 FCS-FL 的源代码公布在 <https://github.com/Songluhaining/FCS-FL>.

4.1 数据集

本文选择了一个包含可变缺陷的大型公开数据集用于评估 FCS-FL, 该数据集包含了 6 个真实的 SPL 系统, 其中共包含 260 个只有单个缺陷的单缺陷案例和 999 个具有 2 个及以上缺陷的多缺陷案例. 此外, 选择的产品线系统的特征数从 6 至 27, 语句覆盖率为 42.9%–99.9%, 更详细的信息如表 1 所示. 值得说明的是, 据我们所知, 这是目前唯一公开的用于 SPL 缺陷定位且包含测试信息的数据集^[43].

表 1 实验数据集

系统	特征数	产品数	单缺陷的案例数	多缺陷的案例数	测试语句覆盖率 (%)	代码规模
Elevator-FH-JML	6	18	13	26	92.9	854
BankAccountTP	8	34	56	298	99.9	143
ExamDB	8	8	48	214	99.5	513
Email-FH-JML	9	27	20	55	97.7	439
ZipMe	13	25	20	139	42.9	3460
GPL	27	99	103	267	99.4	1944

4.2 研究问题

本文通过回答以下几个研究问题 (RQ) 以评估提出方法的有效性.

RQ1: FCS-FL 在特征级缺陷定位中效率提升的幅度有多大?

RQ2: 在单缺陷情形下, FCS-FL 的语句级缺陷定位准确性如何?

RQ3: 在多缺陷情形下, FCS-FL 的根因定位能力如何?

RQ4: 在 FCS-FL 中, 不同因果图模型和公式 (7) 中 μ 的不同取值对缺陷定位效果的影响如何?

在本研究中, RQ1 旨在验证所提方法在特征级缺陷定位中的效率, 这是后续研究问题的基础. 基于该验证结果, RQ2 进一步评估方法在单缺陷场景下语句级定位的准确性; 而 RQ3 则将这一评估扩展至多缺陷场景, 考察其在更复杂条件下的适用性. 最后, RQ4 针对方法的超参数设置, 系统分析其对算法性能的影响.

为回答 RQ1, 选择每个 SPL 系统的所有单缺陷案例, 在计算未通过产品的可疑特征选择集合时, 统计被包含可疑特征选择集合的比例, 以及不同可疑特征选择集合生成相同特征交互数的比例. 基于这两个比例在不同 SPL 系统的分布情况, 分析基于 FCS-FL 节约生成和检查特征交互的数量. 此外, 通过与先进方法比较在每个系统所有单缺陷案例上的平均运行时间, 进而说明提出方法在识别可疑特征交互时的高效性.

对于 RQ2, 选用 5 种流行的 SBFL 指标, 分别为 Tarantula^[21]、Ochiai^[44]、OP2^[45]、Barinel^[22]和 Dstar^[10], 并将它们应用于 FCS-FL 和基准方法中. 在单缺陷案例上, 利用评估指标 Rank、EXAM^[46]和 Hit@X^[47] (具体定义见第 5.3 节) 比较 FCS-FL 与基准方法的定位性能. 首先, 基于 Rank 和 EXAM 比较提出方法与基准方法的整体定位性能. 接着, 根据结果选择更适合的 SBFL 指标计算所有方法的 Hit@1–Hit@3, 以评估提出方法的根因定位能力. 综合 3 个评估指标分析 FCS-FL 在单缺陷案例上的性能表现.

为处理 RQ3, 将 FCS-FL 与 3 个先进方法在多缺陷案例上进行实验比较. 基于 RQ2 中选择的 SBFL 指标, 利用 Hit@1–Hit@3 和 PBL (见第 5.3 节) 比较 FCS-FL 与基准方法的根因定位能力.

为回答 RQ4, 首先, 分别使用未约简的因果图模型、Baah 等人^[26]提出的因果模型和本文提出的因果图模型进行因果效应评估. 在单缺陷案例上, 基于相同的因果效应评估方法计算各自因果效应, 比较不同因果图所得到的性能, 以说明 FCS-FL 的有效性. 接着, 分别取 μ 为 $\{0, 0.1, 0.2, \dots, 0.9, 1\}$, 比较 FCS-FL 在单缺陷案例上的 Hit@1 以说明融合因果效应和频谱效应的必要性.

4.3 评价指标及基准方法

选用 Hit@X、Rank、EXAM 和 PBL^[48]这 4 个指标评估提出方法的有效性, 这些指标在缺陷定位任务中被广泛使用^[4,8,22,49], 它们的具体介绍如下.

1) Hit@X 表示检查到第 X 条语句检查到缺陷语句的样例数, 例如, Hit@1 统计在第 1 次检查即为缺陷语句的样例数. Hit@1 可以体现方法的根因定位能力. 一般而言, 开发者应仅需检查少量的可疑语句即可定位到缺陷, 因此本文着重关注 $X \in \{1, 2, 3\}$.

2) Rank 为排序后缺陷语句在可疑语句中的排名, 排名越靠前表示算法的定位效果越好.

3) EXAM 是指检测到缺陷语句时被检测语句的数量占总的可疑语句的比例, 检查语句的比例越少越好.

4) PBL 常用于多缺陷案例的评估, 表示检查一定数量语句中缺陷语句所占的比例, 占比越高表示算法的定位能力越优.

选择了 4 个基准算法, 分别为 FB^[5]、SBFL^[45,48–51]、S-SBFL^[30,52]和 VarCop^[11], 与所提方法进行实验对比. 这些方法是用于 SPL 缺陷定位的先进方法, 它们的简要介绍如下.

1) FB^[5]是一种基于 SBFL 技术的特征级缺陷定位方法.

2) SBFL^[45,48–51]是一种基于频谱的缺陷定位技术, 广泛应用于单系统软件的缺陷定位. 为了保证有效评估 SBFL, 每个 SPL 系统都被认为是一个不可配置的, 利用未通过的单元测试对所有的语句进行评估.

3) S-SBFL^[30,52]是一种改进的 SBFL 方法, 其在使用 SBFL 指标评估前使用切片技术隔离出与缺陷相关的语句.

4) VarCop^[11]是一种基于特征级和语句级的层级式缺陷定位方法: 在特征级上利用缺陷相关性和最小性定位缺陷特征交互; 在语句级上基于程序切片和 SBFL 技术定位缺陷语句. 据我们所知, 这是目前 SPL 缺陷定位中性能最优的技术.

5 实验结果与分析

本节展示了在 6 个 SPL 系统上的一系列实验, 并通过分析实验结果来回答相应的研究问题.

5.1 可疑特征交互识别中效率的提升 (RQ1)

图 5 用箱线图展示了在单缺陷案例上执行特征级缺陷定位时, 每个案例中被包含的可疑特征选择集合占所有未通过产品数的比例 (表示为包含率). 如图所示, 6 个 SPL 系统中均存在被包含的可疑特征选择集合. 以 Bank-AccountTP 为例, 该系统包含 56 个单缺陷案例, 因此得到 56 个数据点 (组合为箱型), 其中每个数据点表示该样例中包含率的值. 除 ExamDB 系统外, 其余 5 个系统的平均包含率均较高 (即位于 12.9% 与 62.8% 之间), 意味着在不处理的情况下将生成大量重复的特征交互. 对于 ExamDB 系统, 其大部分案例中的包含率为 0. 一个可能的解释是包含率对测试配置的数量很敏感. 如表 1 所示, ExamDB 系统仅有 8 个测试配置, 相比于其他系统更少. 当测试

配置的数量较少时,可疑特征选择集合的总数比较稀少,这使得包含关系难以存在.因此,在大多数情况下,未通过产品对应的可疑特征选择集合是被包含的,去除这些集合可以有效提升后续可疑特征交互识别的效率.

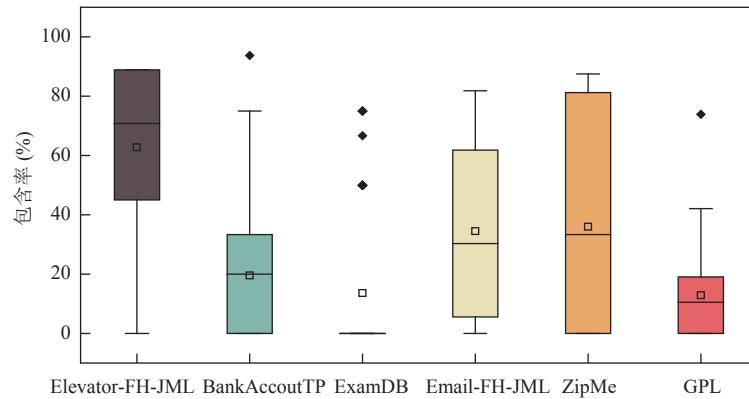


图5 不同 SPL 系统中被包含可疑特征选择集合的占比率

图6进一步地说明了,在特征交互可疑性检查时,使用缓存机制可以节约的检查数占总特征交互数的比例(表示为节约率).如图6所示,ExamDB系统的平均节约率为0,这是因为节约率同样受测试配置数量的影响.对于其余5个系统,平均的节约率分别为7.9%、27.3%、15.6%、35.2%和40.3%,这意味着不同的可疑特征选择集合会生成大量重复的特征交互.因此,本文使用缓存机制可以节约大量重复的可疑性检查,尤其对于更大规模的SPL系统.

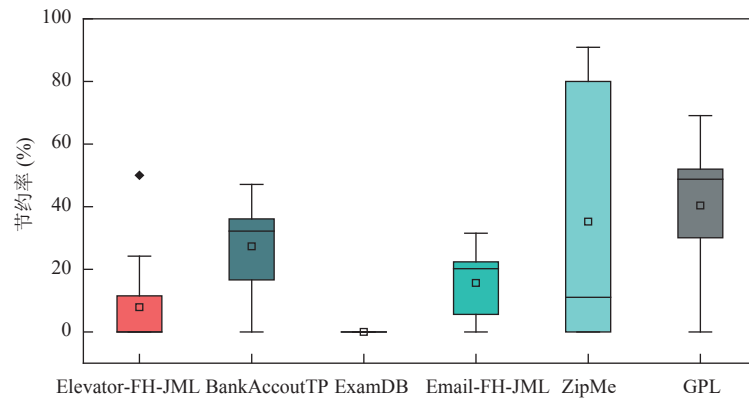


图6 不同 SPL 系统中特征交互检查的节约率

为更直观地说明FCS-FL在特征级缺陷定位上效率的提升,表2分别展示了FCS-FL与VarCop在6个系统上特征交互检查数和平均运行时间的比较,以及FCS-FL相比于VarCop的减少比例.如表2所示,FCS-FL在6个系统上的检查数相比于VarCop均有所下降,其中在Elevator-FH-JML、ZipMe和GPL上的下降比例均超过了50%.对于ExamDB,FCS-FL与VarCop的检查数接近,其原因是在该系统中包含率和节约率都接近0.对于其余5个系统,FCS-FL相比于VarCop能减少至少33.9%的检查次数.对于更大规模的系统,如GPL,在组合数指数式增长的情况下,减少这样比例的特征交互数是极其有意义的.对于运行时间,除了在ExamDB上无明显变化,在其余系统上均显著减少.此外,FCS-FL在特征级缺陷定位上的效率优势对特征数更多的系统中更为明显.如表2所示,FCS-FL在GPL上的平均特征级缺陷定位时间为2481.130 s,较VarCop下降了41%.这说明了所提方法在更大规模的系统中能更快地完成可疑特征交互的识别.后续表格中用加粗表示最优结果.

表2 FCS-FL与VarCop检查的特征交互数和平均运行时间比较

系统	特征数	检查特征数			运行时间		
		VarCop	FCS-FL	减少比例 (%)	VarCop (s)	FCS-FL (s)	减少比例 (%)
Elevator-FH-JML	6	73	36	50.7	0.002	0.001	50
BankAccountTP	8	731	443	39.4	0.038	0.031	18
ExamDB	8	117	115	0	0.002	0.002	0
Email-FH-JML	9	1103	729	33.9	0.039	0.034	13
ZipMe	13	3079	1471	52.2	0.094	0.087	7
GPL	27	9646128	3875392	59.8	4216.437	2481.130	41

综上,实验结果说明本文提出的可疑特征交互识别方法相较于基准方法在效率上具有明显提升.相比于现有方法,基于可疑特征交互去重策略可以避免生成和检查大量重复的特征交互,实现在大多数系统中少检查33.9%–59.8%的特征交互.此外,对于更大规模的SPL系统,如GPL,在特征交互数随特征数指数式增长的情况下,FCS-FL减少的运行时间比例仍有41%,这很好地回答了RQ1.

5.2 单缺陷情形下定位能力的比较 (RQ2)

表3和表4分别展示了FCS-FL与基准算法在单缺陷案例下的Rank和EXAM值.结果显示,FCS-FL和基准算法都在使用OP2指标时表现出更优的定位能力,这意味着OP2指标更适合用于选择的SPL系统.无论使用哪种SBFL指标,FCS-FL的Rank和EXAM相比于基准方法都是最优的.在OP2指标中,FCS-FL显著优于FB,因为FB仅关注二阶特征交互,无法准确地实现特征层缺陷定位.此外,提出的方法在5种SBFL指标下的平均Rank值相比于SBFL和S-SBFL分别提升了3.94和2.04,这表明SBFL直接应用于SPL系统无法获得较优的定位性能.相比于VarCop,FCS-FL在Rank和EXAM上分别提升了0.25和0.17,这是因为提出的因果模型可以提升可疑语句评估的准确性.因此,FCS-FL在单缺陷案例上的缺陷定位性能相比于基准方法具有足够的优势.

表3 FCS-FL与基准方法使用不同SBFL指标下的Rank比较

SBFL指标	FCS-FL	VarCop	S-SBFL	SBFL	FB
Tarantula	5.65	5.97	7.79	12.27	104.56
Ochiai	4.85	5.07	6.89	7.95	90.19
OP2	4.17	4.26	5.24	5.33	77.93
Barinel	6.71	7.11	9.73	12.27	104.56
Dstar	4.63	4.83	6.54	7.89	87.94

表4 FCS-FL与基准方法在不同SBFL指标下的EXAM比较

SBFL指标	FCS-FL	VarCop	S-SBFL	SBFL	FB
Tarantula	1.49	1.69	2.45	3.25	17.58
Ochiai	1.29	1.42	1.73	1.92	14.01
OP2	1.15	1.36	1.56	1.56	12.72
Barinel	1.74	1.94	2.44	3.25	17.58
Dstar	1.27	1.40	1.68	1.90	13.65

图7展示了所提方法与基准方法在单缺陷案例中Hit@1–Hit@3所占案例数的比例.从图中可以看出,在单缺陷的情况下,FCS-FL的Hit@1–Hit@3相比于基准方法均有提升,尤其是Hit@1.由于因果效应和频谱效应的融合,FCS-FL能够在更多的案例中将缺陷语句排序至第1的位置,因此Hit@1的表现更为优越.此外,FCS-FL在超过46%的案例中首次检查即可发现缺陷语句,并且在超过80%的案例中仅需检查前3条语句即可定位到缺陷语句.

对于RQ2,实验结果说明了在单缺陷案例中,FCS-FL的缺陷定位能力更加准确.无论使用哪种SBFL指标,FCS-FL都可以得到最优的定位性能.对于根因定位能力,FCS-FL较基准方法有明显提升,使用FCS-FL在超过一半的单缺陷案例中一次检查即可定位到缺陷语句.

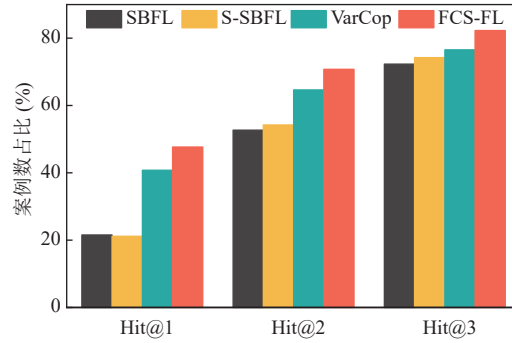


图7 FCS-FL与基准方法在单缺陷案例中的Hit@1-Hit@3比较(案例总数: 260)

5.3 多缺陷情形下根因定位能力的比较 (RQ3)

图8(a)展示了FCS-FL与基准方法使用OP2指标在所有多缺陷案例下的Hit@1-Hit@3。从图中可以看出, FCS-FL在Hit@1上展现出了明显优势,即FCS-FL能够在超过35%的案例中将其中一条缺陷语句排序至第1的位置,而基准方法均低于25%,尤其是SBFL和S-SBFL低于5%。这是因为提出的因果图模型约简了因果关系,减轻多个缺陷语句之间产生的混杂影响,从而提升了根因定位的准确性。相比于VarCop, FCS-FL在Hit@1的优势较Hit@2和Hit@3更明显,因为FCS-FL在更多的案例中可以将缺陷语句排序至第1,而VarCop在更多的案例中只能将缺陷语句排序至第2或第3。此外,尽管多缺陷案例的定位更加复杂且困难, FCS-FL仍能在超过70%的案例中在前3条可疑语句中找到缺陷语句。

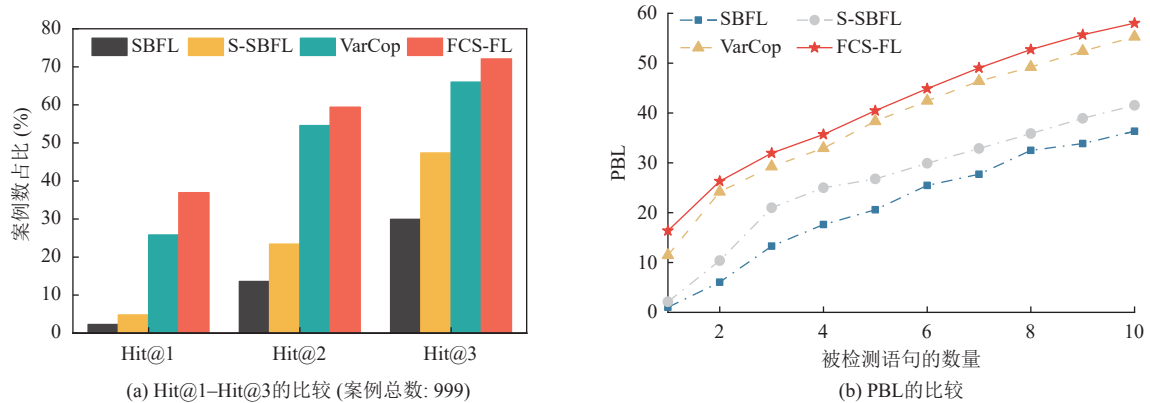


图8 FCS-FL与基准方法在多缺陷案例上性能的比较

图8(b)展示了FCS-FL与基准方法在PBL指标上的比较。结果显示, FCS-FL在所有的多缺陷案例中检查第一条语句即可检查出约17%的缺陷语句,明显优于基准方法,这也说明了FCS-FL具有更优的根因定位性能。SBFL和S-SBFL由于未考虑特征级定位,更多不相关的程序语句会影响缺陷语句的排名,因此它们在检查一条语句时的PBL接近0。提出的方法的PBL曲线与VarCop较为接近,但检查前10条语句时, FCS-FL的PBL均更优。这是因为两种方法计算语句可疑性都考虑了基于产品和单元测试的频谱效应,而FCS-FL额外考虑了程序的因果关系,从而提升了方法缺陷定位的性能。

因此,对于RQ3可得到以下答案。相比于基准方法, FCS-FL在多缺陷案例中具有更优的根因定位能力。此外,实验结果说明了提出的约简的因果图模型能够准确计算语句对测试结果的因果效应,也说明了与频谱效应融合后可以更准确地定位到缺陷语句。

5.4 不同因果模型和融合权重的不同取值对定位能力的影响 (RQ4)

为验证提出因果图模型(用 CG_1 表示)的有效性,将其与另外两种因果图模型在6个产品线系统的单缺陷案例上进行实验比较,这两种因果图模型分别为 Baah 等人^[26]构建的因果图(用 CG_2 表示)和未约简的因果图模型(用 CG_3 表示)。实际上,由于 CG_3 可能存在环,无法直接作为因果图进行因果效应评估。因此,本文使用一种简单的方式去除环,同时尽可能保留已有的因果关系,即找到环中出度最小的节点,并删除以它为先驱节点的边,直至环消失。

表5展示了在相同因果效应评估方法下,分别使用 CG_1 、 CG_2 和 CG_3 时对缺陷定位的性能比较。表中“—”表示由于因果图模型过大,无法在有限的时间内计算出因果效应。当使用 CG_3 作为因果图模型时,由于 Elevator、ZipMe 和 GPL 的语句节点数较多,逐一计算每个节点对测试结果的因果效应极为费时,因此无法获得结果。这表明过长的因果路径会严重降低因果推理的效率,尤其在更复杂的系统中。对于 ExamDB 和 Email,使用 CG_3 进行因果推断可以获得比 CG_1 和 CG_2 更优的定位性能,因为它考虑了语句间更多的因果关系,在混淆因素少的情况下可以得到更准确的因果效应。然而,在 BankAccountTP 中,使用 CG_3 取得了最差的定位性能,这是因为未约简的因果图模型可能包含更多的潜在因子,增加了模型的复杂性和混淆因素的数量,这些因素同时影响因变量和自变量,导致观察到的效应并不是真正的因果效应。因此,对基于程序依赖得到的因果图模型进行因果关系约简是有必要的。此外,从表5中可知,使用 CG_2 进行因果推断时,定位性能略低于使用 CG_1 。这是因为 CG_1 将中介变量纳入因果效应计算,以减少误差项的方差,使因果效应评估更精确。

表5 FCS-FL 使用不同因果图的性能比较

系统	平均Rank			平均EXAM		
	FCS-FL (CG_1)	FCS-FL (CG_2)	FCS-FL (CG_3)	FCS-FL (CG_1)	FCS-FL (CG_2)	FCS-FL (CG_3)
Elevator-FH-JML	1.54	1.54	—	0.34	0.34	—
BankAccountTP	2.71	2.73	2.80	3.53	3.55	3.66
ExamDB	2.46	2.56	2.27	0.98	1.02	0.90
Email-FH-JML	2.30	2.35	2.20	0.93	0.95	0.89
ZipMe	9.20	9.35	—	0.40	0.40	—
GPL	6.79	6.82	—	0.70	0.71	—

图9展示了 μ 和 η 在不同取值下, FCS-FL 的 Hit@1 表现,反映其定位缺陷语句的能力。从图中可以看出,当因果效应的权重显著高于频谱效应的权重(例如, μ 取值为 0.9 或 1 且 η 取值为 0.5 时),缺陷定位能力有所下降。然而,当 μ 取值为 0 或 1 时,所获得的 Hit@1 显著低于取值在 0.1–0.9 范围内的 Hit@1。这表明, FCS-FL 仅使用频谱效应或因果效应时,其缺陷定位能力较差。造成这一现象的原因在于,频谱效应能够更准确地定位缺陷的代码块,而因果效应则更有效地识别代码块中的具体缺陷语句,两种效应的优势相辅相成。因此,融合这两种效应显得尤为重要。此外,图9同样说明当频谱效应仅由基于产品评估或基于单元测试评估决定时(例如, η 取值为 0 或 1 且 μ 取值为 0.1),算法定位缺陷语句的能力明显下降。这表明,基于产品评估和基于单元测试评估能从不同粒度上反应可疑语句的可疑性,从而降低频谱效应的误差。

值得注意的是,当 μ 取值在 0.1–0.8 之间时, Hit@1 的值相对接近,因为因果效应在通过 Softmax 函数归一化后对频谱效应的影响变化较小。此外, μ 和 η 都在取非极端值(如 0 和 1)时对缺陷定位性能的影响较为平稳,主要原因是中间权重区域同时利用了频谱效应和因果效应,两种信息的优势叠加、劣势互补,使得组合模型的误差接近最小且对权重扰动不敏感;只有在极端权重下忽略了一整类信息时,性能才明显下降。需要说明的是,对于不同的软件产品,超参数的取值应依据其测试充分性进行设定。当测试较为充分,即语句覆盖率较高时,可适当提高频谱效应的权重,如 μ 设置为 0.2 或 0.3;反之,则应适当提高因果效应的权重。同理,当被测产品的采样覆盖率较高时,可增大基于产品的频谱信息权重,如 η 设置为 0.7–0.9;反之可增大基于单元测试的频谱信息权重。

基于以上分析,对 RQ4 的回答如下:本文提出的因果图模型,相较于现有基于程序依赖的图模型,更适合用于程序缺陷定位任务。与未约简的因果图模型相比,该模型不仅能够保证更高的效率,还能实现更为精确的因果评估。对于所使用的数据集,在测试较充分且采样覆盖率较高时减小 μ 且增大 η 能更精确地定位缺陷。

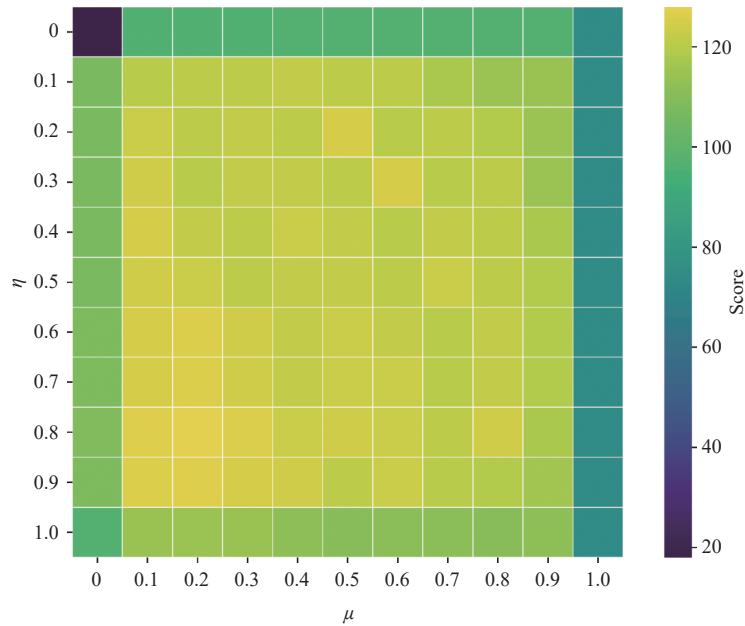


图9 μ 和 η 的不同取值对根因定位能力的影响

5.5 讨论

尽管 FCS-FL 能够在多缺陷场景下实现有效的缺陷定位,并在多个真实项目中展现了较强的泛化性能,但对于缺陷间交互作用(如特征冲突、协同效应等)的因果建模与理论分析仍有待进一步深入。目前模型主要关注单一缺陷对结果的直接与间接因果路径,将多个缺陷视为相互独立的处理变量进行建模,未对多缺陷共存时可能出现的复杂交互关系进行系统分析。这在实验中也表现为,在部分存在显著特征冲突或协同作用的多缺陷场景下,模型的怀疑度排序存在一定偏差,影响了部分案例的定位准确率。

尽管如此,提出的约简因果图模型依然为多缺陷场景下的缺陷定位提供了结构化、可解释的因果分析框架。该模型通过对程序依赖关系的简化与父、子节点的合理保留,有效刻画了主效应及其与直接相关中介路径的影响机制,使得在缺陷相对独立或交互关系不显著时,能够准确识别出主要的致因语句。

然而,针对多缺陷场景中的实际复杂性,仍有部分交互机制未被当前模型完全刻画。例如以下情况。

- 1) 不同缺陷依赖的配置选项存在互斥,可能导致某些缺陷在联合出现时其怀疑度被低估或高估。
- 2) 多个缺陷共同作用于同一执行路径或状态变量时,可能产生协同增强或互相掩蔽的现象,影响因果路径的可辨识性。
- 3) 一个缺陷的激活可能通过状态变量影响下游缺陷的暴露,形成多级传播,增加因果关系的复杂度。

这些现象表明,缺陷间的高阶交互效应不仅影响缺陷定位的排序准确性,也对模型的因果解释提出了更高要求。因此,后续我们将进一步扩展约简因果图,显式引入缺陷间冲突或协同的交互变量,并结合多级中介分析与正则化建模策略,实现对复杂多缺陷场景下因果效应的更精确识别与解释。

6 总结

针对 SPL,本文提出了一种融合因果和频谱效应的高效缺陷定位方法,名为 FCS-FL。该方法相比于现有方法提升了特征级缺陷定位的效率和语句级缺陷定位的准确性。FCS-FL 在特征级缺陷定位时,基于可疑特征交互去重策略提升定位效率,即可疑特征选择集合间的包含关系会导致重复生成相同的特征交互;以及不同特征选择集合间存在相同的子集会重复检查相同的特征交互。实验结果也说明了可疑特征交互去重策略可以明显提升识别

可疑特征交互的效率。在语句级缺陷定位中,本文提出了一种约简的因果图模型,该模型可以实现更精准的因果效应评估。此外,通过在6个真实的SPL系统上与基准方法进行实验对比,结果显示提出方法在单缺陷和多缺陷案例中均表现出更优的缺陷定位效果,说明了因果效应和频谱效应融合必要性。

然而,FCS-FL在处理大规模SPL系统时,特征交互的搜索空间仍然是巨大的,并且有必要在更大规模的系统上来验证方法的有效性。此外,针对SPL缺陷的智能化修复同样也是一项重要且艰巨的任务,但相关研究同样稀缺。在未来的工作中,将考虑内部特征交互对特征级缺陷定位的影响,以进一步缓和“组合爆炸”的影响。此外,计划针对SPL缺陷定位任务,构建更多高维且复杂的数据集,尤其是复杂的多缺陷案例,结合多级中介分析与正则化建模策略以扩展FCS-FL,并在这些系统中进行验证。最后,拟针对SPL缺陷修复提出更有效的方法,实现智能化测试和调试的闭环。

References

- [1] Clements PC, Northrop LM. *Software Product Lines: Practices and Patterns*. Boston: Addison-Wesley, 2002.
- [2] Nie KM, Zhang L, Fan ZQ. Systematic literature review of software product line variability modeling techniques. *Ruan Jian Xue Bao/Journal of Software*, 2013, 24(9): 2001–2019 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/4433.htm> [doi: 10.3724/SP.J.1001.2013.04433]
- [3] Xiang Y, Zhou YR, Cai SW. Integrating preference in many-objective optimal software product selection algorithm. *Ruan Jian Xue Bao/Journal of Software*, 2020, 31(2): 282–301 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/5637.htm> [doi: 10.13328/j.cnki.jos.005637]
- [4] Bagheri E, Gasevic D. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 2011, 19(3): 579–612. [doi: 10.1007/s11219-010-9127-2]
- [5] Arrieta A, Segura S, Markiegi U, Sagardui G, Etxeberria L. Spectrum-based fault localization in software product lines. *Information and Software Technology*, 2018, 100: 18–31. [doi: 10.1016/j.infsof.2018.03.008]
- [6] Nguyen TT, Ngo KT, Nguyen S, Vo HD. Detecting false-passing products and mitigating their impact on variability fault localization in software product lines. *Information and Software Technology*, 2023, 153: 107080. [doi: 10.1016/j.infsof.2022.107080]
- [7] Nguyen TT, Zhang XY, Arcaini P, Ishikawa F, Vo HD. Automated program repair for variability bugs in software product line systems. *Journal of Systems and Software*, 2025, 221: 112152. [doi: 10.1016/j.jss.2024.112152]
- [8] Wong WE, Gao RZ, Li YH, Abreu R, Wotawa F. A survey on software fault localization. *IEEE Trans. on Software Engineering*, 2016, 42(8): 707–740. [doi: 10.1109/TSE.2016.2521368]
- [9] Li XL, Wong WE, Gao RZ, Hu LH, Hosono S. Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. *Empirical Software Engineering*, 2018, 23(1): 1–51. [doi: 10.1007/s10664-016-9494-9]
- [10] Wong WE, Debroy V, Gao RZ, Li YH. The DStar method for effective software fault localization. *IEEE Trans. on Reliability*, 2014, 63(1): 290–308. [doi: 10.1109/TR.2013.2285319]
- [11] Nguyen TT, Ngo KT, Nguyen S, Vo HD. A variability fault localization approach for software product lines. *IEEE Trans. on Software Engineering*, 2022, 48(10): 4100–4118. [doi: 10.1109/TSE.2021.3113859]
- [12] Zeng MH, Wu YQ, Ye ZT, Xiong YF, Zhang X, Zhang L. Fault localization via efficient probabilistic modeling of program semantics. In: *Proc. of the 44th Int'l Conf. on Software Engineering*. Pittsburgh: ACM, 2022. 958–969. [doi: 10.1145/3510003.3510073]
- [13] Cleve H, Zeller A. Locating causes of program failures. In: *Proc. of the 27th Int'l Conf. on Software Engineering*. St. Louis: IEEE, 2005. 342–351. [doi: 10.1109/ICSE.2005.1553577]
- [14] Garvin BJ, Cohen MB. Feature interaction faults revisited: An exploratory study. In: *Proc. of the 22nd IEEE Int'l Symp. on Software Reliability Engineering*. Hiroshima: IEEE, 2011. 90–99. [doi: 10.1109/ISSRE.2011.25]
- [15] Kuhn DR, Wallace DR, Gallo AM. Software fault interactions and implications for software testing. *IEEE Trans. on Software Engineering*, 2004, 30(6): 418–421. [doi: 10.1109/TSE.2004.24]
- [16] Baah GK, Podgurski A, Harrold MJ. Causal inference for statistical fault localization. In: *Proc. of the 19th Int'l Symp. on Software Testing and Analysis*. Trento: ACM, 2010. 73–84. [doi: 10.1145/1831708.1831717]
- [17] Siebert J. Applications of statistical causal inference in software engineering. *Information and Software Technology*, 2023, 159: 107198. [doi: 10.1016/j.infsof.2023.107198]
- [18] Leszak M, Perry DE, Stoll D. A case study in root cause defect analysis. In: *Proc. of the 22nd Int'l Conf. on Software Engineering*. Limerick: ACM, 2000. 428–437. [doi: 10.1145/337180.337232]

- [19] Johnson B, Brun Y, Meliou A. Causal testing: Understanding defects' root causes. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 87–99. [doi: [10.1145/3377811.3380377](https://doi.org/10.1145/3377811.3380377)]
- [20] Wong WE, Gao RZ, Li YH, Abreu R, Wotawa F, Li DC. Software fault localization: An overview of research, techniques, and tools. In: Wong WE, Tse TH, eds. Handbook of Software Fault Localization: Foundations and Advances. Hoboken: Wiley-IEEE Press, 2023. 1–117.
- [21] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering. Long Beach: ACM, 2005. 273–282. [doi: [10.1145/1101908.1101949](https://doi.org/10.1145/1101908.1101949)]
- [22] Abreu R, Zoeteweyj P, Golsteijn R, Van Gemund AJC. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 2009, 82(11): 1780–1792. [doi: [10.1016/j.jss.2009.06.035](https://doi.org/10.1016/j.jss.2009.06.035)]
- [23] Papadakis M, Le Traon Y. Metallaxis-FL: Mutation-based fault localization. *Software Testing, Verification and Reliability*, 2015, 25(5-7): 605–628. [doi: [10.1002/stvr.1509](https://doi.org/10.1002/stvr.1509)]
- [24] Li ZJ, Yan LF, Liu YZ, Zhang ZY, Jiang B. MURE: Making use of mutations to refine spectrum-based fault localization. In: Proc. of the 2018 IEEE Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Lisbon: IEEE, 2018. 56–63. [doi: [10.1109/QRS-C.2018.00024](https://doi.org/10.1109/QRS-C.2018.00024)]
- [25] Boehm B, Basili VR. Software defect reduction top 10 list. *Computer*, 2001, 34(1): 135–137. [doi: [10.1109/2.962984](https://doi.org/10.1109/2.962984)]
- [26] Baah GK, Podgurski A, Harrold MJ. Mitigating the confounding effects of program dependences for effective fault localization. In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering. Szeged: ACM, 2011. 146–156. [doi: [10.1145/2025113.2025136](https://doi.org/10.1145/2025113.2025136)]
- [27] Zakari A, Lee SP, Abreu R, Ahmed BH, Rasheed RA. Multiple fault localization of software programs: A systematic literature review. *Information and Software Technology*, 2020, 124: 106312. [doi: [10.1016/j.infsof.2020.106312](https://doi.org/10.1016/j.infsof.2020.106312)]
- [28] Weiser M. Program slicing. *IEEE Trans. on Software Engineering*, 1984, SE-10(4): 352–357. [doi: [10.1109/TSE.1984.5010248](https://doi.org/10.1109/TSE.1984.5010248)]
- [29] Agrawal H, Horgan JR. Dynamic program slicing. *ACM SIGPLAN Notices*, 1990, 25(6): 246–256. [doi: [10.1145/93548.93576](https://doi.org/10.1145/93548.93576)]
- [30] Chaleshtari NB, Parsa S. SMBFL: Slice-based cost reduction of mutation-based fault localization. *Empirical Software Engineering*, 2020, 25(5): 4282–4314. [doi: [10.1007/s10664-020-09845-4](https://doi.org/10.1007/s10664-020-09845-4)]
- [31] Sinnema M, Deelstra S, Nijhuis J, Bosch J. COVAMOF: A framework for modeling variability in software product families. In: Proc. of the 3rd Int'l Conf. on Software Product Lines. Boston: Springer, 2004. 197–213. [doi: [10.1007/978-3-540-28630-1_12](https://doi.org/10.1007/978-3-540-28630-1_12)]
- [32] Xiang Y, Huang H, Li SZ, Li MQ, Luo C, Yang XW. Automated test suite generation for software product lines based on quality-diversity optimization. *ACM Trans. on Software Engineering and Methodology*, 2023, 33(2): 46. [doi: [10.1145/3628158](https://doi.org/10.1145/3628158)]
- [33] Nguyen S, Nguyen H, Tran N, Tran H, Nguyen T. Feature-interaction aware configuration prioritization for configurable code. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 489–501. [doi: [10.1109/ASE.2019.00053](https://doi.org/10.1109/ASE.2019.00053)]
- [34] Kolesnikov S. Feature interactions in configurable software systems [Ph.D. Thesis]. Passau: Universität Passau, 2019.
- [35] Dubslaff C, Weis K, Baier C, Apel S. Causality in configurable software systems. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 325–337. [doi: [10.1145/3510003.3510200](https://doi.org/10.1145/3510003.3510200)]
- [36] Yao LY, Chu ZX, Li S, Li YL, Gao J, Zhang AD. A survey on causal inference. *ACM Trans. on Knowledge Discovery from Data (TKDD)*, 2021, 15(5): 74. [doi: [10.1145/3444944](https://doi.org/10.1145/3444944)]
- [37] Imbens GW, Angrist JD. Identification and estimation of local average treatment effects. *Econometrica*, 1994, 62(2): 467–475. [doi: [10.2307/2951620](https://doi.org/10.2307/2951620)]
- [38] De Chaisemartin C, D'Haultfœuille X. Two-way fixed effects estimators with heterogeneous treatment effects. *American Economic Review*, 2020, 110(9): 2964–2996. [doi: [10.1257/aer.20181169](https://doi.org/10.1257/aer.20181169)]
- [39] Pearl J. *Causality: Models, Reasoning, and Inference*. Cambridge: Cambridge University Press, 2000.
- [40] Musco V, Monperrus M, Preux P. Mutation-based graph inference for fault localization. In: Proc. of the 16th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). Raleigh: IEEE, 2016. 97–106. [doi: [10.1109/SCAM.2016.24](https://doi.org/10.1109/SCAM.2016.24)]
- [41] Furia CA, Torkar R, Feldt R. Towards causal analysis of empirical software engineering data: The impact of programming languages on coding competitions. *ACM Trans. on Software Engineering and Methodology*, 2023, 33(1): 13. [doi: [10.1145/3611667](https://doi.org/10.1145/3611667)]
- [42] Aho AV, Garey MR, Ullman JD. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1972, 1(2): 131–137. [doi: [10.1137/0201008](https://doi.org/10.1137/0201008)]
- [43] Ngo KT, Nguyen TT, Nguyen S, Vo HD. Variability fault localization: A benchmark. In: Proc. of the 25th ACM Int'l Systems and Software Product Line Conf.—Vol. A. Leicester: ACM, 2021. 120–125. [doi: [10.1145/3461001.3473058](https://doi.org/10.1145/3461001.3473058)]
- [44] Abreu R, Zoeteweyj P, Van Gemund AJC. An evaluation of similarity coefficients for software fault localization. In: Proc. of the 12th

- Pacific Rim Int'l Symp. on Dependable Computing (PRDC 2006). Riverside: IEEE, 2006. 39–46. [doi: 10.1109/PRDC.2006.18]
- [45] Abreu R, Zoetewij P, Van Gemund AJC. Spectrum-based multiple fault localization. In: 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. Auckland: IEEE, 2009. 88–99. [doi: 10.1109/ASE.2009.25]
- [46] Wong E, Wei TT, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: Proc. of the 1st Int'l Conf. on Software Testing, Verification, and Validation. Lillehammer: IEEE, 2008. 42–51. [doi: 10.1109/ICST.2008.65]
- [47] Lucia, Lo D, Xia X. Fusion fault localizers. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 127–138. [doi: 10.1145/2642937.2642983]
- [48] Keller F, Grunske L, Heiden S, Filieri A, van Hoorn A, Lo D. A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. In: Proc. of the 2017 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Prague: IEEE, 2017. 114–125. [doi: 10.1109/QRS.2017.22]
- [49] Pearson S, Campos J, Just R, Fraser G, Abreu R, Ernst MD, Pang D, Keller B. Evaluating and improving fault localization. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Buenos Aires: IEEE, 2017. 609–620. [doi: 10.1109/ICSE.2017.62]
- [50] Naish L, Lee HJ, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Trans. on Software Engineering and Methodology, 2011, 20(3): 11. [doi: 10.1145/2000791.2000795]
- [51] Abreu R, Zoetewij P, van Gemund AJC. On the accuracy of spectrum-based fault localization. In: Proc. of the 2007 Testing: Academic and Industrial Conf. Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007). Windsor: IEEE, 2007. 89–98. [doi: 10.1109/TAIC.PART.2007.13]
- [52] Li XY, Orso A. More accurate dynamic slicing for better supporting software debugging. In: Proc. of the 13th IEEE Int'l Conf. on Software Testing, Validation and Verification (ICST). Porto: IEEE, 2020. 28–38. [doi: 10.1109/ICST46399.2020.00014]

附中文参考文献

- [2] 聂坤明, 张莉, 樊志强. 软件产品线可变更建模技术系统综述. 软件学报, 2013, 24(9): 2001–2019. <http://www.jos.org.cn/1000-9825/4433.htm> [doi: 10.3724/SP.J.1001.2013.04433]
- [3] 向毅, 周育人, 蔡少伟. 集成偏好的高维多目标最优软件产品选择算法. 软件学报, 2020, 31(2): 282–301. <http://www.jos.org.cn/1000-9825/5637.htm> [doi: 10.13328/j.cnki.jos.005637]

作者简介

王海宁, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多目标优化.

向毅, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为智能化软件工程, 演化计算, 多目标优化.

黄翰, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为微计算理论与方法, 智能化软件工程, 生化反应计算机.

吴春国, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为演化计算, 机器学习, 神经架构搜索.

冯夫健, 博士, 教授, CCF 专业会员, 主要研究领域为智能优化算法, 深度学习.

杨晓伟, 博士, 教授, 博士生导师, 主要研究领域为机器学习, 模式识别, 智能化软件工程.