

# 软件多缺陷定位方法研究综述

李征<sup>1)</sup> 吴永豪<sup>1)</sup> 王海峰<sup>1)</sup> 陈翔<sup>2)</sup> 刘勇<sup>1)</sup>

<sup>1)</sup>(北京化工大学信息科学与技术学院 北京 100029)

<sup>2)</sup>(南通大学信息科学技术学院 江苏 南通 226019)

**摘要** 软件多缺陷定位(Multiple Fault Localization,简称 MFL)尝试在含有多个缺陷的软件程序中自动标识出这些缺陷所在的位置.传统的缺陷定位研究一般假设被测软件内仅含有一个缺陷,而实际情况下软件内往往包含多个缺陷,因此 MFL 问题更加贴近实际场景.当程序中存在多个缺陷时,由于缺陷数量难以准确估计,同时缺陷之间可能存在互相干扰,因此对 MFL 问题的研究更具挑战性.已有研究表明传统单缺陷假设下的缺陷定位技术会随着程序中缺陷数目的增多而出现定位效果下降的问题.因此,需要对已有缺陷定位技术加以改进使其在 MFL 问题中具有更好的缺陷定位效果.本文以 MFL 研究问题为核心,对相关研究成果进行了系统的梳理.首先将已有的 MFL 技术细分为三类,分别是基于缺陷干扰假设的多缺陷定位方法,基于缺陷独立假设的多缺陷定位方法和不基于任何假设的多缺陷定位方法;然后依次总结了每一类方法的主要设计思想和相关研究成果,随后分析了 MFL 研究中经常使用的评测指标和评测对象;最后,本文从扩大评测对象的编程语言范围、考虑更多的软件程序、寻找更多的工业应用场景等多个角度对 MFL 的未来研究方向进行了展望.

**关键词** 软件调试;多缺陷定位;缺陷干扰;缺陷独立

**中图法分类号** TP311 **DOI号** 10.11897/SP.J.1016.2022.00256

## Review of Software Multiple Fault Localization Approaches

LI Zheng<sup>1)</sup> WU Yong-Hao<sup>1)</sup> WANG Hai-Feng<sup>1)</sup> CHEN Xiang<sup>2)</sup> LIU Yong<sup>1)</sup>

<sup>1)</sup>(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029)

<sup>2)</sup>(School of Information Science and Technology, Nantong University, Nantong, Jiangsu 226019)

**Abstract** With the development of the computer and electronic information industry, computer software's functions and scale have become increasingly complex and large, bringing unprecedented challenges to software debugging. When defects occur in the software, developers need to spend an incalculable workload to debug the software, and the first step in software debugging is to find the location of the defect, that is, the fault localization. There has been a drastic growth of research in Multiple Fault Localization (MFL) in the past few years. MFL attempts to automatically identify multiple fault locations in a software program with multiple defects. Traditional fault localization techniques generally assume that the software programs under test only contain one single defect. But in reality, the software often contains multiple defects, so the MFL problem is closer to the debugging scenarios. However, compared with the fault localization of programs containing only a single defect, MFL research is much more difficult. When multiple defects in the program under test, it is difficult to estimate the accurate number of defects in this scenario, and the multiple faults may interfere with each other, which are the problems that will not be

收稿日期:2020-11-13;在线发布日期:2021-04-30.本课题得到国家自然科学基金(61902015,61872026,61672085)、南通市应用研究计划(JC2019106)资助.李征,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为软件测试、源代码分析和维护. E-mail: lizheng@buct.edu.cn.吴永豪,博士研究生,主要研究方向为软件测试和错误定位.王海峰,博士研究生,主要研究方向为软件测试、错误定位和软件缺陷预测.陈翔(通信作者),博士,副教授,中国计算机学会(CCF)会员,主要研究方向为软件维护和软件测试. E-mail: xchen@ntu.edu.cn.刘勇(通信作者),博士,副教授,中国计算机学会(CCF)会员,主要研究方向为源代码分析、变异测试和错误定位. E-mail: lyong@mail.buct.edu.cn.

encountered in the process of single defect fault localization; hence the MFL problem is quite a challenging problem. Studies have shown that the fault localization accuracy of techniques designed for single-fault localization will be decreased when there are multiple defects in the software programs under test. Therefore, it is necessary to improve the fault localization performance in MFL problem. This survey takes the MFL research problem as the core and offers a systematic overview of existing research achievements. In this survey, we firstly classified these MFL techniques into three groups, which are Defect Interference Hypothesis based MFL (INF-MFL), Defect Independence Hypothesis based MFL (IDP-MFL), and None Hypothesis based MFL (NOH-MFL). INF-MFL method only locates and repairs a single defect each time during debugging. When a single defect is repaired, all test cases are re-executed to collect coverage information and execution results until all defects are repaired. IDP-MFL method divides the MFL task into multiple single fault localization subtasks so that different developers can perform parallel debugging on different subtasks. NOH-MFL method attempts to locate multiple defects at the same time in a debugging process. Then, we summarized the design ideas and detailed research results of each MFL technique. Among them, the INF-MFL method is currently the most studied in MFL, and it is widely used because of its simple implementation. We further analyzed the evaluation metrics, statistical hypothesis test method, and subject programs used in MFL research. Specifically, In MFL research, manual defects are often used to simulate real defect behavior. However, in recent years, researchers believe that this kind of defect will impact the validity of empirical research conclusions and cause the industry to question the practicability of fault localization technology. Therefore, more and more researchers use real procedures with real defects to conduct empirical research to ensure the validity of research conclusions. Finally, we discussed future research directions of MFL, which include: Further study the granularity of fault localization, optimize fault localization technology from the perspective of time cost, consider projects implemented in more other programming languages, consider more software features, combine MFL with defect prediction, and find more industrial application scenarios.

**Keywords** software debugging; multiple fault localization; defect interference; defect independence

## 1 引言

软件调试是保障软件质量并使其正常运行的重要手段<sup>[1]</sup>. 软件调试过程中, 确定程序内的缺陷语句所在位置被称为软件缺陷定位 (Software Fault Localization), 是软件调试过程中最为费时费力的一个步骤<sup>[2]</sup>. 自动化软件缺陷定位方法旨在不需要或较少需要人为干预的前提下, 自动确定程序内缺陷所在位置, 以帮助开发人员更快地修复缺陷. 其中, 基于程序谱的缺陷定位技术 (Spectrum-Based Fault Localization, SBFL) 是一种常用的自动化缺陷定位技术<sup>[3]</sup>, 程序谱是指测试用例在执行期间产生的特征向量. 因为失败测试用例产生的特征向量不同于通过测试用例, 通过比对特征向量之间的差异, 可以计算出代码中各个语句包含缺陷的可能性

(即怀疑度), 怀疑度值越高的语句则越有可能包含缺陷.

在早期的自动化缺陷定位研究中, 绝大部分研究工作假设被测程序仅含有一个缺陷. 研究人员针对单缺陷程序提出了大量的缺陷定位技术, 并且取得了不错的定位效果<sup>[4-7]</sup>. 但是, 单缺陷假设并不符合实际的软件调试场景特征. 在商业项目或开源项目中, 通常会包含多个缺陷, 甚至部分缺陷之间还会存在相互干扰<sup>[8-9]</sup>. Xue 等人<sup>[10]</sup>的研究工作表明, 一些缺陷定位方法虽然在单缺陷程序上可以取得较好的定位效果, 但在多缺陷程序上, 其定位效果会显著下降. 因此本文重点关注的软件多缺陷定位 (Multiple Fault Localization, 简称 MFL) 问题, 比传统的软件单缺陷定位问题更具研究挑战性. 其面临的主要挑战包括: (1) 当被测程序内包括多个缺陷时, 难以精准地确定各个缺陷和与其关联的

失败测试用例,即无法判断单个失败测试用例是因为执行哪个缺陷所导致的;(2)程序内含有的实际缺陷数难以精确估计;(3)部分缺陷之间可能存在相互干扰的问题.例如,单缺陷下存在某个测试用例执行失败,但引入新的缺陷后,该测试用例可能就会执行通过,导致无法精确定位到各个缺陷的实际位置.因此 MFL 逐渐成为软件自动缺陷定位领域的一个热点问题,并引起研究人员的广泛关注.

近些年来,研究人员对 MFL 问题展开了深入的研究,重点集中于分析缺陷之间的相互影响和提升多缺陷定位的效果和效率.为了对该综述主题进行系统的梳理,我们在 IEEE Xplore、ACM、Science Direct、Wiley、Springer、中国知网等国内外相关论文数据库中搜索与综述主题相关的论文,查询的英文关键词包括“program fault localization”和“software bug detect”等;查询的中文关键词包括“缺陷定位”和“错误定位”等.但是在相关论文检索时,我们没有直接使用“multiple fault”或“multiple bug”这样的关键词,是因为我们发现有些论文虽然探讨了 MFL 问题,但在标题和摘要中并没有包含这些关键词,因此使用这些关键字会遗漏一些相关论文.在论文的搜索过程中,我们发现与缺陷定位相关的研究工作数量较多.因此,我们遵循以下原则来选择文献:

- (1) 研究论文与综述主题相关,排除与多缺陷研究无关的文献;
- (2) 只选择在同行评议期刊/会议上发表的研究论文;
- (3) 如果论文首先发表在会议上,随后扩充版

本发表在期刊上,则仅选择发表在期刊上的论文;

- (4) 仅考虑中文和英文的文献.

接下来将论文的搜索过程总结如下,我们首先下载了与上述关键词相关的所有论文,然后借助手工分析方式逐一筛选出与 MFL 问题相关的论文.随后查阅这些论文对应的相关工作和研究人员已发表论文列表来进一步补充论文.最终确定了与该综述主题相关的论文共 87 篇(截止到 2020 年 7 月).

图 1 统计了 MFL 领域每年发表的论文数量.从图中可以看出,与 MFL 相关的工作呈大致上升的趋势,在 2016 年至 2019 年,这种上升趋势尤其明显.表 1 中对 MFL 相关论文的发表数量进行了统计,并按照发表的论文总数从大到小进行排序(表中仅列出了中国计算机学会推荐的英文期刊/会议和中文核心期刊,且发表数量至少 2 篇)不难看出,与 MFL 相关的工作主要集中于软件工程领域的权威期刊和会议上,例如:在 JSS 期刊上发表 12 篇,在 TSE 期刊上发表 5 篇,在 ISSRE 会议上发表 5 篇,在 ASE 会议上发表 3 篇,在 ISSTA 会议上发表 2 篇.

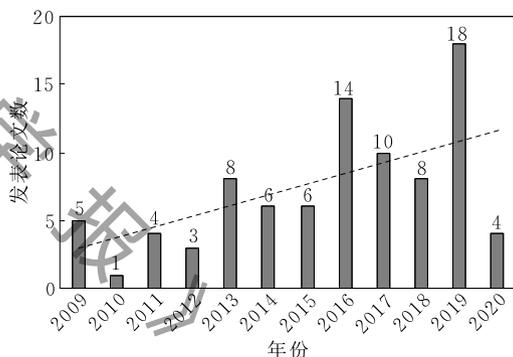


图 1 MFL 相关论文年度统计结果

表 1 MFL 相关论文发表源统计结果

发表源全程(简称)	发表源类型	期刊/会议级别	论文数
Journal of Systems and Software (JSS)	期刊	CCF B 类	12
IEEE Transactions on Software Engineering (TSE)	期刊	CCF A 类	5
International Symposium on Software Reliability Engineering (ISSRE)	会议	CCF B 类	5
Information and Software Technology (IST)	期刊	CCF B 类	3
International Conference on Automated Software Engineering (ASE)	会议	CCF A 类	3
Software Quality Journal (SQJ)	期刊	CCF C 类	3
Software: Practice and Experience (SPE)	期刊	CCF B 类	3
计算机学报	期刊	一级学报	2
计算机研究与发展	期刊	一级学报	2
Frontiers of Computer Science (FCS)	期刊	CCF C 类	2
International Conference on Software Maintenance and Evolution (ICSME)	会议	CCF B 类	2
International Conference on Software Quality, Reliability and Security (QRS)	会议	CCF C 类	2
International Conference on Software Testing, Verification and Validation (ICST)	会议	CCF C 类	2
International Symposium on Software Testing and Analysis (ISSTA)	会议	CCF A 类	2

软件自动缺陷定位是近些年来软件自动调试领域的一个重要研究问题,在 Wong 等人<sup>[1]</sup>2016 年发表在《IEEE Transactions on Software Engineering》上的综述中,他们将已有的缺陷定位方法细分为八类:基于切片的方法、基于程序频谱的方法、基于统计的方法、基于程序状态的方法、基于机器学习的方法、基于数据挖掘的方法、基于模型的方法和其他方法.国内研究人员针对该问题在《计算机学报》和《软件学报》上也先后发表了三篇综述论文<sup>[11-13]</sup>.但与上述综述不同,本文主要关注的是软件自动缺陷定位领域中的一个子问题,即多缺陷定位问题,并系统地分析了多缺陷定位问题在近 12 年(即 2009 年至 2020 年)内取得的重要研究成果.虽然 Zakari 等人<sup>[14]</sup>对多缺陷定位问题也进行过系统综述,但与该综述相比,本文覆盖了更多的与多缺陷定位相关的研究论文,累计新增论文 32 篇,并且本文对每一类型的方法进行了机理分析以及更为细致的分析和评点.除此之外,本文还额外统计整理了多缺陷定位研究中常用的性能评测指标、统计显著性分析方法和常用评测对象,以方便研究人员进行合理的实验设计.

本文的主要贡献可总结如下:

(1) 通过搜集分析在国内外权威期刊和会议上发表的相关文献,系统地分析了 MFL 领域近 12 年来取得的研究成果,提出 MFL 的研究框架并识别框架内的影响因素;

(2) 将已有的 MFL 方法细分为三类:基于缺陷干扰假设的多缺陷定位方法、基于缺陷独立假设的多缺陷定位方法以及不基于任何假设的多缺陷定位方法.详细分析了三种 MFL 方法的机理并举例说明其执行过程.针对每一类方法,我们依次分析了研究人员的解决思路,并对属于同一类的不同方法进行了系统的比较与分析;

(3) 总结了 MFL 方法在实证研究中经常使用的评测指标和评测对象,并对评测指标和评测对象的使用趋势进行了分析.上述分析能提供有价值的指导,有助于后续研究工作更好地进行实验研究设计.

本文第 2 节给出 MFL 方法的整体研究框架,对三种 MFL 方法的机理进行详细分析,并举例说明其执行过程;第 3 节对基于缺陷干扰假设的多缺陷定位方法进行详细分析;第 4 节对基于缺陷独立假设的多缺陷定位方法进行详细分析;第 5 节对不

基于任何假设的多缺陷定位方法进行详细分析;第 6 节和第 7 节分析并统计研究人员在 MFL 问题上经常使用的性能评测指标和显著性分析方法;第 8 节分析并统计经常使用的评测对象;最后总结全文,并对 MFL 领域未来值得关注的研究方向进行深入探讨.

## 2 研究框架

本文研究 MFL 方法的整体框架如图 2 所示,首先将已有 MFL 方法根据调试模式的不同细分为三类,随后总结已有 MFL 研究中经常使用的程序评测对象、性能评测指标和统计显著性分析方法.

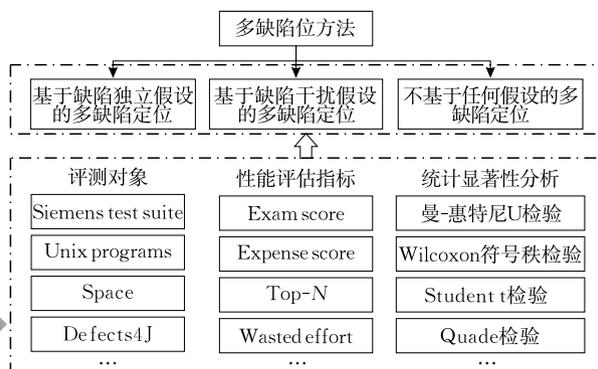


图 2 多缺陷定位方法的整体研究框架

### 2.1 多缺陷定位方法分类

根据开发人员对多缺陷行为的不同假设,我们将已有的 MFL 方法分为三类:

(1) 基于缺陷干扰假设的多缺陷定位 (Defect Interference Hypothesis based MFL, 简称 INF-MFL) 的调试方法,这类方法在调试中每次仅定位并修复单个缺陷,当修复完单个缺陷后,会重新执行所有测试用例来收集覆盖信息和执行结果,直至所有缺陷都被修复;

(2) 基于缺陷独立假设的多缺陷定位 (Defect Independence Hypothesis based MFL, 简称 IDP-MFL) 的调试方法,这类方法将 MFL 任务划分为多个单缺陷定位任务(即子任务),以方便不同开发人员可以在不同的子任务上进行并行调试;

(3) 不基于任何假设的多缺陷定位 (None Hypothesis based MFL, 简称 NOH-MFL) 的调试方法,这类方法在一次调试中尝试同时定位多个缺陷所在位置.

我们对不同类型的方法在已有工作中所占的比例进行了统计,统计结果如表 2 所示.

表 2 MFL 研究方法类型占比

方法类型	论文数	所占比例/%
基于缺陷干扰假设的多缺陷定位(INF-MFL)	57	65.52
基于缺陷独立假设的多缺陷定位(IDP-MFL)	20	22.99
不基于任何假设的多缺陷定位(NO-H-MFL)	10	11.49

其中有 65.52% 的研究工作采用的是 INF-MFL 调试方法, 22.99% 的研究工作采用的是 IDP-MFL 调试方法, 11.49% 的研究工作采用的是 NOH-MFL 调试方法. 可以看出, 基于缺陷干扰假设的多缺陷定位的 INF-MFL 方法是目前 MFL 研究领域内的主流方法, 基于缺陷独立假设的多缺陷定位方法的 IDP-MFL 方法次之, 不基于任何假设的多缺陷定位的 NOH-MFL 方法则相对较少. 本文随后在 2.1 节对三种 MFL 方法的机理进行详细分析, 并在 2.2 节通过一个简单示例来分析不同类型的多缺陷定位方法的主要思想, 之后在第 3 节、第 4 节和第 5 节分别整理和剖析了这三种 MFL 方法的研究动机、研究进展与现状, 以及潜在的问题与改进之处.

## 2.2 多缺陷定位机理分析

多缺陷定位的难点主要体现在分析多个缺陷如何交互以表现为新的缺陷行为, 针对新缺陷行为的分析方式对设计软件调试策略至关重要. 然而基于不同的测试环境、被测程序或假设检验方法, 研究人员会得出不同的分析结论. 本文汇总的三种软件调试策略本质上是针对不同的新缺陷行为的分析模式而提出的.

### 2.2.1 多缺陷行为

程序中的多个缺陷可能以多种方式相互影响, 从而产生仅包含其中任意一个缺陷的单缺陷程序都无法实现的新行为. 新的缺陷行为存在多种类型, 且对传统的单缺陷定位技术产生难以预料的负面影响, DiGiuseppe 等人<sup>[15]</sup>对缺陷相互影响进行了深入研究, 并把多缺陷行为分为四种类型:

(1) 缺陷协同. 与单缺陷程序相比, 通过测试用例检测的缺陷表现增多;

(2) 缺陷混淆. 与单缺陷程序相比, 通过测试用例检测的缺陷表现减少;

(3) 缺陷独立. 与单缺陷程序相比, 通过测试用例检测的缺陷表现无变化;

(4) 混合类型. 同时发生缺陷协同和缺陷混淆.

因此, 研究人员针对上述多缺陷行为提出了众

多的多缺陷程序定位方法, 这些方法根据其其对多缺陷行为的假设可被分为 3 类: 基于缺陷独立假设的多缺陷定位方法、基于缺陷干扰假设的多缺陷定位方法和不基于任何假设的多缺陷定位方法.

其中基于缺陷独立假设的多缺陷定位方法假设多缺陷程序中各缺陷相互独立; 基于缺陷干扰假设的多缺陷定位方法假设多缺陷程序中各缺陷相互影响, 从而可能产生缺陷混淆或缺陷协同现象; 不基于任何假设的多缺陷定位方法假设缺陷独立和缺陷相互影响两种情况都有可能存在. 下文将详细介绍上述三种方法的缺陷定位依据以及在其假设中的缺陷定位流程.

### 2.2.2 基于缺陷独立假设的多缺陷定位方法

基于缺陷独立假设的多缺陷定位方法假定同一程序中的不同缺陷相互独立, 即单个程序中的缺陷之间不会发生交互, 从而不会产生新的缺陷行为.

如图 3 所示的是一个缺陷相互独立的程序中的测试用例-程序缺陷关系图,  $B_1$ 、 $B_2$  和  $B_3$  指的是 3 个程序缺陷,  $T_{1\sim 9}$  表示 9 个失败测试用例. 单个测试用例位于某个缺陷的影响范围内, 表示该测试用例被对应的缺陷影响而失败. 例如图中的  $T_{1\sim 4}$  被缺陷  $B_1$  影响而失败,  $T_{5\sim 7}$  被缺陷  $B_2$  影响而失败,  $T_{8\sim 9}$  被缺陷  $B_3$  影响而失败.

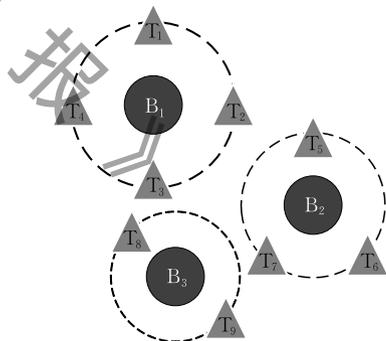


图 3 缺陷独立示意图

在缺陷相互独立的假设中, 程序的每一个缺陷行为都有其唯一对应的程序缺陷<sup>[16]</sup>. 因此, IDP-MFL 方法通常试图对失败的测试用例进行划分成组, 且使每组中测试用例仅对应于单个导致该测试用例失败的程序缺陷<sup>[17]</sup>. 以图中所示关系图为例, 测试用例应该被划分为三组:  $T_{1\sim 4}$  为一组(对应缺陷  $B_1$ )、 $T_{5\sim 7}$  为一组(对应缺陷  $B_2$ )、 $T_{8\sim 9}$  为一组(对应缺陷  $B_3$ ). 最后软件测试人员使用单组内的测试用例对其对应的程序缺陷进行定位.

综上所述, IDP-MFL 方法获得的同一组中的失

败测试用例均与同一个程序缺陷相关. 但在实践过程中, 多缺陷的真实行为并不一定完全符合该多缺陷定位方法的假设, 即程序中不可避免地会存在缺陷协同和缺陷混淆的情况<sup>[18]</sup>. 为此, Yan 等人进行了一个实证研究<sup>[19]</sup>, 他们的实验结果表明缺陷独立假设在大多数情况下成立, 且超过 50% 的失败测试用例仅执行了程序中的单个缺陷, 该结果为 IDP-MFL 方法的提出提供了支持.

除此之外, IDP-MFL 方法还受到另一个因素的挑战. 即该方法通常采用基于相似度的聚类算法进行测试用例划分, 而聚类算法不一定能够得到完全正确的结果. 因此可能存在测试用例划分之后, 分组数量少于程序真实缺陷数量的情况. 此时软件测试人员无法在单次迭代中通过每个分组定位所有程序缺陷. 为了解决这个问题, 研究人员提出了迭代的方法<sup>[20]</sup>进行 IDP-MFL 方法的缺陷定位, 即重复进行测试用例执行、分组和缺陷定位的过程, 直至不再出现失败的测试用例.

### 2.2.3 基于缺陷干扰假设的多缺陷定位方法

尽管实证研究表明超过 50% 的失败测试用例仅执行了程序中的单个缺陷<sup>[19]</sup>, 但是在多缺陷程序中依然存在多种缺陷相互影响的情况, 因此 INF-MFL 方法不完全认同单个程序中的不同缺陷相互独立的假设, 即不同缺陷之间可能会相互影响, 从而产生缺陷协同或混淆的现象.

当缺陷协同发生时, 程序出现了新的缺陷行为, 即同一个失败测试用例有可能执行多个程序缺陷<sup>[21]</sup>, 此时程序缺陷与测试用例无法一一对应. 图 4 中展示的是一个发生了缺陷协同的测试用例-程序缺陷关系图, 从图中可以看出, 程序缺陷  $B_1$  和  $B_2$  共同导致了测试用例  $T_{1\sim7}$  的失败, 此时  $T_{1\sim7}$  无法通过 INF-MFL 方法进行有效划分.

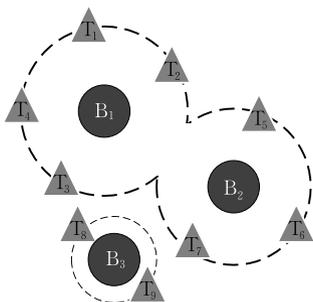


图 4 缺陷协同示意图

使用 INF-MFL 方法对图 4 中的协同缺陷进行缺陷定位, 能够在每次迭代中定位并修复单个程序

缺陷, 直至解决缺陷协同问题. 如图 5 所示的是使用 INF-MFL 方法对图 4 中的缺陷协同进行缺陷定位的效果示意图, 图中  $B_1$  被定位并修复,  $T_{1\sim3}$  变为通过测试用例, 从而解决缺陷协同问题.

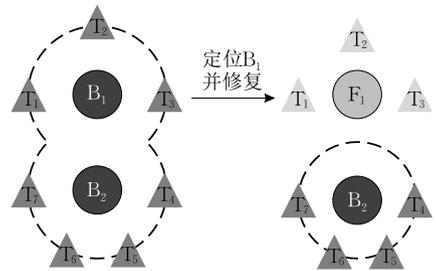


图 5 INF-MFL 方法解决缺陷协同

除此之外, 缺陷混淆对测试用例的影响更加显著, 因为缺陷混淆的存在会使得测试用例检错能力降低, 即测试用例能够在单缺陷程序中检测到缺陷(产生失败测试用例), 但是在多缺陷程序中却无法检测到缺陷<sup>[22]</sup>. 在这基础上, Debroy 和 Wang 对缺陷的相互干扰进行了进一步详细的分析<sup>[21]</sup>, 他们在实证研究中进一步提出了缺陷掩盖现象. 缺陷掩盖是一种缺陷混合类型相互影响的极端情况, 发生缺陷掩盖时, 某个程序缺陷相关的失败测试用例因缺陷混淆与该缺陷失去关联(即不再执行该缺陷), 而这部分缺陷却同时与另一个程序相关联, 即前者被后者完全掩盖, 此时前一个缺陷无法被测试用例检测. 图 6 中展示的是一个发生了缺陷混淆的测试用例-程序缺陷关系图, 从图中可以看出, 程序缺陷  $B_1$  部分掩盖了缺陷  $B_2$ , 且完全掩盖了缺陷  $B_3$ .

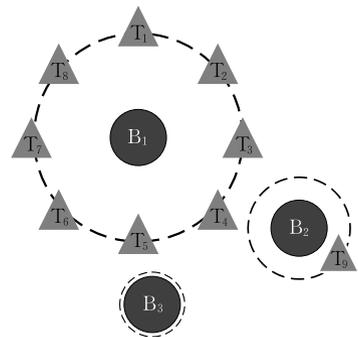


图 6 缺陷掩盖示意图

此时  $B_1$  已主导了程序的缺陷行为, 即大部分测试用例均与  $B_1$  相关联, 因此这种情况下使用 INF-MFL 方法进行缺陷定位能够实现与单缺陷程序缺陷定位类似的调试环境, 从而获得良好的缺陷定位效果. 当前已有研究表明, 缺陷掩盖的发生频率高于

其他类型的多缺陷行为<sup>[19,21]</sup>,因此当多缺陷程序中缺陷相互影响时,INF-MFL 方法依然能够有效地完成缺陷定位。

### 2.2.4 不基于任何假设的多缺陷定位方法

不基于任何假设的多缺陷定位方法创新性地规避了多缺陷行为差异的问题。具体来说,该方法会同时给多个(或单个)程序实体(变量、语句或实体)计算怀疑度并排序,从而实现把单个测试用例与多个(或单个)程序缺陷进行对应。因此该方法能够在缺陷协同和缺陷独立的情况下完成缺陷定位。

关于 NOH-MFL 方法的一种可行的策略是抽取程序实体进行排列组合,然后设计一个适应度函数对每个排列组合的选项进行排序<sup>[23]</sup>。图 7 中所示的是一个适应度分数的排名结果示例图,其中 B 表示包含缺陷的程序实体,N 表示不包含缺陷的程序实体。理想情况下排名越高的程序实体组合越有可能包含更多的程序缺陷,从而实现在单次迭代内定位出程序中的所有缺陷。

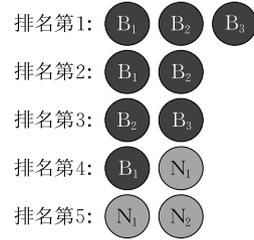


图 7 NOH-MFL 方法适应度分数排名示例图

### 2.3 多缺陷定位示例分析

图 8 给出了一个示例程序,该节将基于这个示例程序来简要分析三种 MFL 方法的主要执行过程。该示例程序 P 用于判断输入的年份是否为闰年,其中包含两个缺陷,分别是语句 4 和语句 6,该程序的配套测试套件(Test Suite)共包含 6 个测试用例{t1,t2,t3,t4,t5,t6},图中给出了每个测试用例的输入取值、代码覆盖情况和执行结果。需要说明的是,我们在这个示例程序中,以 Ochiai<sup>[24]</sup>公式为例来计算 INF-MFL 和 IDP-MFL 调试方法对应的语句怀疑度(即含有缺陷的可能性)。

def judge(year):	t1	t2	t3	t4	t5	t6	Sus (INF-MFL)	Sus (IDP-MFL)	
	2019	2020	2000	1600	1900	1800		1st cluster	2nd cluster
1: flag=false	●	●	●	●	●	●	0.82	0.71	0.71
2: if year % 100 == 0:	●	●	●	●	●	●	0.82	0.71	0.71
3: if year % 400 == 0:			●	●	●	●	1.00	0.71	0.71
4: flag=false #应该为 true			●	●			0.71	0.71	0.00
5: else:					●	●	0.71	0.00	0.71
6: flag=true #应该为 false					●	●	0.71	0.00	0.71
7: elif year % 4 == 0:	●	●					0.00	0.00	0.00
8: flag=true		●					0.00	0.00	0.00
9: print(flag)	●	●	●	●	●	●	0.82	0.71	0.71
Fail/Pass	P	P	F	F	F	F			

●表示该语句被对应的测试用例覆盖

P表示该测试用例执行通过,F表示该测试用例执行失败

图 8 MFL 示例

若使用 INF-MFL 调试方法,开发人员首先根据 Ochiai 公式计算出每个语句的怀疑度,并按照怀疑度取值从大到小依次检查,直至定位到第一个缺陷。对于图 8 所示的示例程序,开发人员定位到第一个缺陷语句需要检查 5 行语句。当修复完该缺陷语句后,开发人员需要重新编译程序并执行测试用例以搜集覆盖信息和执行结果,开发人员将根据 Ochiai 公式重新计算每个语句的怀疑度,按照怀疑度取值形成语句有序列表,并依次检查每个语句,直至定位到第二个缺陷语句。

若使用 IDP-MFL 调试方法,开发人员首先将所有失败测试用例按照其特征(例如覆盖信息)划分

到两个类簇(Cluster),随后分别基于两个类簇计算出相应的怀疑度并形成不同的语句有序列表。对于图 8 所示的示例程序,不难看出,第一个类簇主要针对缺陷语句 4,第二个类簇主要针对缺陷语句 6。最后,可以为两个开发人员分别提供不同的语句有序列表,以支持对两个缺陷语句的并行定位。

若使用 NOH-MFL 方法,其流程与 INF-MFL 方法相似,但不同之处是,开发人员在找到第一个缺陷语句之后,会继续检查语句来尝试找到第二个缺陷语句。因此 NOH-MFL 方式会尝试将多个缺陷语句都排在语句有序列表的前列,使得开发人员在单次迭代中能快速找到多个缺陷语句。以图 8 所示的

INF-MFL 方法怀疑度值为例, 当使用 NOH-MFL 方法进行缺陷定位时, 开发人员定位到第一个缺陷语句仍然需要检查 5 行语句. 但当修复完该缺陷语句后, 开发人员会按照怀疑度值继续检查程序语句, 直至找到第二个程序缺陷. 因此针对图 8 所示的示例程序, 采用 NOH-MFL 方法进行缺陷定位至多需要检查 7 行语句.

### 3 基于缺陷干扰假设的多缺陷定位方法

基于缺陷干扰假设的多缺陷定位的 INF-MFL 方法是目前多缺陷定位领域中研究最多的一类方法, 具有实现简单的特点. 从发表年份分析, 在我们收集的 53 篇与 INF-MFL 方法相关的研究工作中, 2019 年发表的数量最多(为 14 篇), 2016 年和 2017 年次之(分别为 11 篇和 8 篇). 需要注意的是, 搜集的部分论文的方法并没有针对多缺陷定位问题提出定制算法, 而仅考虑了含有多个缺陷的评测对象. 但这些实证研究对 MFL 问题的后续研究仍有指导和参考价值, 因此该综述也考虑了这些论文.

由于 INF-MFL 方法一次只定位一个缺陷, 假设一个程序包含两个缺陷, 那么使用 INF-MFL 方法定位这些缺陷则总共需要执行两次调试. 具体来说, 在第一次调试过程中, INF-MFL 方法将尝试定位其中一个缺陷, 随后开发者将其修复; 修复完第一个缺陷后, INF-MFL 方法将对修复完第一个缺陷后的程序重新执行测试用例并收集缺陷定位所需要的信息, 并尝试定位第二个缺陷. 不难看出, INF-MFL 方法需要在每次缺陷定位结束后, 由开发者对程序进行缺陷修复并重新执行测试用例, 直到找到并修复被测程序包含的所有缺陷. 研究人员尝试将 INF-MFL 方法与多种经典的缺陷定位技术结合使用, 如基于程序频谱的方法 (Spectrum-Based Fault Localization, 简称 SBFL)、基于统计的方法和基于机器学习的方法<sup>[7, 21, 25-27]</sup>等.

#### 3.1 INF-MFL 方法的提出与改进

##### 3.1.1 基于优化模型的方法

基于优化模型的方法通常将多缺陷定位问题转换为优化模型的求解问题, 研究人员通过提取程序中某种粒度(粒度可以设置为程序组件、程序类、函数、语句、变量等)的特征, 并使用该特性构造多种不同的优化问题模型, 从而在测试数据中对相同粒度的程序缺陷进行识别.

在近些年的研究中, 有研究人员将缺陷定位抽

象为一个数据挖掘问题, 并尝试使用约束规划模型求解该问题. 这种方法可以提取满足对最可疑语句建模的一组约束的数个最佳匹配模式, 该匹配结果可以为程序中的缺陷(如失败的测试用例、包含缺陷的程序语句), 或者为需要排除的无关因素(如通过的测试用例、正确的程序语句).

当匹配结果是以程序中的缺陷为目标, 则相应算法需要以尽可能高的效率定位到程序中缺陷的位置并报告给软件开发人员. 例如, Dean 等人<sup>[28]</sup>利用线性规划模型, 将多缺陷定位问题转化为查找最小覆盖所有失败测试用例的语句集合问题, 他们从 Tarantula 公式的定义出发, 分析认为被更多失败测试用例覆盖且被更少通过测试用例覆盖的语句集合有更大的可能性成为缺陷语句, 并将这些语句集合作为目标, 构建约束条件并进行求解. 他们在包含 2 到 4 个缺陷的被测程序上展开了实证研究, 结果表明这种方法在定位多个缺陷时, 比 AMPLE<sup>[24]</sup>和 Tarantula 等方法有更好的缺陷定位效果. 2011 年, Artho 基于 Zeller 等人<sup>[29]</sup>的工作, 提出了针对多缺陷程序的缺陷定位方法 IDD (Iterative Delta Debugging)<sup>[30]</sup>. 该方法首先使用失败的测试用例执行程序的历史版本, 从而找到使得这些测试用例通过的程序版本, 之后通过提取当前版本和历史版本之间的差异进行缺陷定位. 当程序中存在多个缺陷, 且缺陷互相干扰使得测试用例在引入某一缺陷之前的程序版本中依然无法执行通过时, 该方法将会迭代式向更为历史的版本进行追溯, 直至所有的程序缺陷都能被定位. Birch 等人<sup>[31]</sup>提出了一种基于符号执行的多缺陷定位方法, 可自动定位到程序的可修复片段. 这种方法提升了测试用例的搜索速度, 减少了模型中符号执行的开销. 在多缺陷程序上的实验结果表明, 使用该方法可以有效缩小可疑缺陷位置的搜索范围, 具有较好的缺陷定位效果. 2019 年, Ma 等人<sup>[32]</sup>提出了一种向量表模型 VTM (Vector Table Model), 用于系统分析和比较不同的 SBFL 公式. 基于该模型, 他们实现了一个统一的系统调查框架, 以同时考虑单缺陷程序和多缺陷程序. 通过定义不同类型的缺陷并基于 VTM 研究缺陷类型的数学表达, 可以系统地分析和比较不同缺陷定位技术的有效性. 他们的实验结果表明, 在单缺陷情况下, 怀疑度计算公式 Ochiai 的性能要优于 Dstar, 而在多缺陷情况下, Ochiai 的稳定性要优于 Dstar. Liu 等人<sup>[33]</sup>提出了一种基于 Simulink 模型的多缺陷定位方法. 该方法可以对测试用例进行聚类, 以帮助识

别 Simulink 模型中的多个缺陷,然后根据每个类簇内的测试用例计算怀疑度,进而生成相应的语句怀疑度排序列表,最后根据他们提出的评估准则选出具有最佳缺陷定位效果的排序列表用于缺陷定位. 尽管这种方法对失败测试用例进行聚类,但其仍然属于 INF-MFL 方法,即每次迭代仅定位单个缺陷并立即进行修复.

当匹配结果是以无关因素为目标,则该技术需要尽可能剔除缺陷定位过程中的无关或干扰因素,以提高开发人员定位缺陷的效率. 例如 Wang 等人<sup>[34]</sup>发现,对被测程序使用 SBFL 技术生成的语句怀疑度表中,存在一些排名较高的语句为正确语句,并导致缺陷定位精度不高. 为了进一步提高 SBFL 技术的有效性,他们依据软件缺陷传播模型 RIP (Reachability, Infection, Propagation) 来识别那些在怀疑度列表中排名较高的正确语句,并通过排除这些语句来提高缺陷语句的排名. 他们基于 Siemens 数据集进行了实证研究,实验结果表明他们所提的方法可以有效提升 SBFL 技术在多缺陷程序上的缺陷定位性能.

尽管研究人员提出了多种基于优化模型的方法,但是该方法在处理大规模程序时,因为方法本身的算法复杂度较高,因此会存在时间开销大的问题. 为了解决这个问题,Arabi 等人<sup>[35]</sup>利用全局约束和基于模式的怀疑度计算公式,提出了一种新的关联多缺陷上下文的约束规划模型,来挖掘怀疑度最高的数个语句,可以有效缓解因缺陷之间的复杂依赖关系所引起的问题. 实验结果表明,与 Tarantula、Ochiai 和 Jaccard 等传统缺陷定位方法相比,该方法在多缺陷定位时的性能有显著提升. 此外,他们<sup>[36]</sup>还提出了一种基于多标准层次分析 (Analytic Hierarchy Process, 简称 AHP) 的方法. AHP 方法将缺陷定位问题建模为多准则决策问题 (Multi-Criteria Decision Making Problem), 并利用加权线性公式将不同度量指标整合成单一度量指标,来作为程序语句怀疑度的排序依据. 他们在包含 2 个和 4 个缺陷的程序上进行了实证研究,发现 AHP 方法比常见的 SBFL 方法 (如 Tarantula、Ochiai 等) 有着更高的缺陷定位精度.

最近 Peng 等人<sup>[37]</sup>提出的一种基于自动编码的缺陷定位方法 ABFL (Auto-encoder Based Fault Localization). 该方法包括四个步骤,分别是编码器训练、特征提取、排名模型的训练和语句有序列表的生成. 首先标记所有程序语句以训练编码器,训练完

成的编码器可以为每个语句编码成一个固定长度的特征表示. 然后,将所有提取的特征表示输入到排序学习 (Learning to Rank) 算法,以训练排名模型. 最后,ABFL 基于排名模型生成语句怀疑度序列列表. 在 Defects4J 数据集上的实验结果表明,ABFL 方法在多缺陷上的定位效果要优于最新的 14 种 SBFL 方法 (例如 DStar、GP13、GP19 等).

### 3.1.2 基于程序切片的方法

基于程序切片的方法一般通过分析程序信息,缩小缺陷定位的语句搜索范围,以提高缺陷定位效果. 近些年来,研究人员提出了多种结合程序切片分析的 MFL 方法,改进了多缺陷定位精度. 程序切片可简单分为静态切片和动态切片两类,动态切片通过分析测试用例的覆盖信息和执行结果等获得切片内容,而静态切片仅通过分析程序源代码获得切片内容.

静态切片是软件工程中常用的程序分析方法,但是静态切片通常会返回范围较大的程序集合,难以精确定位到缺陷所在位置. 为了解决上述问题,Zhang 等人<sup>[38]</sup>提出了一种基于静态程序切片的方法 PRIOSLICE,该方法建立程序中数据依赖的概率模型,为切片中的每一个语句计算范围为 $[0,1]$ 的权重,用以表示该语句属于切片的可能性,最后依照权重对静态切片内的语句进行优先级排序,这种排序结果能提高静态切片的准确度,从而达到提高缺陷定位效果的目的. 实验结果表明 PRIOSLICE 方法比现有的静态切片方法能更有效提升多缺陷定位效果.

与静态切片相比,动态切片执行过程更为复杂且耗时,但是其切片范围更小且更精准,文万志等人<sup>[39]</sup>通过 codecover 工具收集测试用例的语句或语句块覆盖信息,提出了一种基于条件执行切片谱 (Conditioned Execution Slicing Spectrum) 的多缺陷定位技术. 该技术主要分为 4 步: (1) 计算缺陷相关条件执行切片,缩小缺陷搜索范围; (2) 构造条件执行切片谱矩阵; (3) 根据条件执行切片谱,计算缺陷相关条件执行切片内每个元素的怀疑度; (4) 根据怀疑度大小依次定位程序中的缺陷,并生成怀疑度报告. 在 3 个面向对象程序 (Tetris、SimpleJavaApp 和 JHSA) 上的实验结果表明该技术比基于程序谱的 Tarantula 技术、基于程序切片的 Intersection 技术和 Union 技术等具有更高的多缺陷定位精度.

将 SBFL 技术与其他技术结合,可以保留两种技术的优势,因此这通常被视为一种可行的改进缺

陷定位效果的方法. 现有研究<sup>[40]</sup>表明, 将 SBFL 与切片命中集计算(Slicing Hitting Set Computation)技术相结合的 Sendys 方法具有较好的缺陷定位效果. 更进一步, Tu 等人<sup>[41]</sup>从理论层面对 Ochiai 公式与切片命中集计算结合的定位方法 Sendys 方法进行了分析, 并提出 Sendys 方法的多个改进版本, 该方法具有严格的理论支撑, 同时, 在多缺陷定位上具有更好的定位效果. Parsa 等人<sup>[42]</sup>从缺陷代码与失败测试用例输出的依赖关系角度出发, 提出了一种结合 INF-MFL 调试方法和程序切片技术的方法 Stat-Slice. 该方法能够查找到更多类型的缺陷, 如代码遗漏缺陷、头文件缺陷(如头文件中包含缺陷的宏命令)等. 在 grep 和 gzip 程序上的实验结果表明, Stat-Slice 方法能有效减少缺陷定位的开销, 并在定位两个缺陷和三个缺陷时有更好的定位效果.

### 3.1.3 基于怀疑度计算公式改进的方法

SBFL 和基于变异的缺陷定位(Mutation-Based Fault Localization, MBFL)技术因其较好的缺陷定位效果, 近年来被研究人员广泛运用. SBFL 和 MBFL 技术的核心是怀疑度值计算公式, 然而传统的怀疑度计算公式(Tarantula, Ochiai, Jaccard 等)并未针对多缺陷问题进行优化. 因此, 基于 SBFL 和 MBFL 怀疑度计算公式改进的方法通常通过改进现有的怀疑度计算公式, 或提出新的怀疑度计算公式, 从而提高缺陷定位效果.

针对上述问题, 一部分研究人员尝试通过加权、组合等方法, 对传统的怀疑度计算公式进行改进. 例如, Abreu 等人在传统的测试用例覆盖频谱信息(仅记录程序语句是否被测试用例执行)的基础上, 进一步考虑了程序语句的执行次数信息<sup>[43]</sup>, 并提出了 Zoltar-C 方法. 该方法同时考虑到程序语句的执行覆盖频谱以及程序语句被执行的次数, 并综合利用这些信息来改进贝叶斯方法, 最终计算出程序语句的出错概率. 他们通过理论分析验证了 Zoltar-C 方法能够提高多缺陷定位效果. 但实证研究的结果表明 Zoltar-C 方法在多缺陷定位精度上提升效果有限. Lee 等人<sup>[44]</sup>基于实证研究发现覆盖多缺陷的测试用例所执行的语句比仅覆盖单个缺陷的测试用例所执行的语句有更高的怀疑度. 因此, 他们提出一种加权的测试用例分类方法来提升多缺陷定位精度. 实验结果表明, 这种加权方法能够有效提高多缺陷定位的精度. Zhang 等人<sup>[45]</sup>提出一种轻量的缺陷定位技术 PRFL, 该方法通过使用 PageRank 算法来增强现有的 SBFL 技术. 具体而言, PRFL 根据每个测

试用例的贡献程度, 使用 PageRank 算法对原始程序的覆盖信息进行加权, 然后使用传统的 SBFL 技术重新计算语句的怀疑度值, 以实现更有效的缺陷定位. 在 Defects4J 程序集和 87 个 GitHub 项目的多缺陷版本上的实验结果表明, PRFL 的缺陷定位效果要显著优于传统的 SBFL 技术(Tarantula, Ochiai 等). Zou 等人<sup>[46]</sup>针对来自 Defects4J 评测程序集的真实程序缺陷, 系统比较了不同缺陷定位方法(基于频谱的方法, 基于变异的方法, 基于切片的方法等)的多缺陷定位效果. 在 Defects4J 上的实验结果表明, SBFL 的独立定位效果是最好的, 并优于 MBFL. 同时他们提出一种组合定位方法, 结果表明该组合方法要明显优于任何单独的方法.

除了改进已有的计算公式, 也有一部分研究人员尝试通过构造更加丰富的程序特征信息、设计出一些全新的怀疑度计算公式. 例如, Wong 等人<sup>[25]</sup>对 Kulczynski 相关系数<sup>[47]</sup>进行研究, 在此基础上提出了一种全新的语句怀疑度值的计算方法 Dstar. 实验结果表明, 与其他 38 种缺陷定位公式相比, Dstar 在单缺陷程序和多缺陷程序上都能有更好的缺陷定位效果. 通过对比覆盖多缺陷的失败测试用例与覆盖单缺陷的失败测试用例, Naish 等人<sup>[48-49]</sup>基于遗传编程(Genetic Programming), 提出一种新的怀疑度计算方法, 即双曲度量(Hyperbolic Metric). 这种度量方法考虑到程序多缺陷位置的不确定性, 通过机器学习从训练数据中获取度量多缺陷的最优参数. 在 Siemens 和 Unix 数据集上的实验结果表明, 与 Ochiai, Tarantula 和 GP13 等常用的怀疑度计算方法相比, 他们提出的双曲度量方法在多缺陷程序上有更好的定位效果.

同时, Laghari 等人<sup>[50]</sup>改进了传统的 SBFL 方法, 提出了一种基于模式化频谱(Patterned Spectrum)的缺陷定位方法. 与传统的 SBFL 方法主要使用测试用例的频谱信息不同, 基于模式化频谱的缺陷定位方法通过记录程序中的函数调用关系和次数来构造出一种新的程序频谱. 实验结果表明该方法比传统 SBFL 方法有更好的多缺陷定位效果.

除此之外, Li 等人<sup>[51]</sup>提出一种迭代用户驱动(Iterative User-Driven)的半自动缺陷定位方法 Swift. 该方法基于函数粒度, 借助统计缺陷定位方法来识别出可疑程序函数, 然后对这些函数的正确性生成相应的查询, 最后结合开发人员的反馈信息来改进缺陷定位结果. 实证评估结果显示 Swift 在多缺陷程序调试上对开发者能产生一定程度的帮

助,可以减少查找缺陷所需的时间开销并提高调试效率.

### 3.1.4 基于测试套件改进的方法

在缺陷定位的过程中,除了人工检查被测程序以外,执行测试套件内的测试用例也会产生巨大的时间开销,因此提高测试用例的执行效率也可以有效提高缺陷定位的效率.基于测试套件改进的方法就是指通过改进测试套件内测试用例生成、选择、执行顺序等过程,或改进测试用例的覆盖频谱信息,从而提高多缺陷定位的效果和效率.根据方法的不同,基于测试套件改进的方法可以分别对测试用例执行的三个阶段进行优化(执行前、执行中和执行后).在执行测试用例前,开发人员可以对测试用例进行选择以及通过组合测试的方法来减少测试用例生成的数量,从而控制测试用例的执行流程和效率;在执行测试用例中,开发人员可以对测试用例的进行排序,或者调整测试用例的覆盖路径来提高执行效率;在执行测试用例后,开发人员通过降低测试预言和偶然正确用例对缺陷定位的负面影响来提高缺陷的定位效果.

测试用例的选取是指人为判断测试套件内各个测试用例是否被保留,通常开发人员会保留或复制对缺陷定位有价值的测试用例,并删除冗余的测试用例,从而实现提高缺陷定位效率且不降低缺陷定位精度.例如,为了优化测试套件内测试用例之间的关系,Gong 等人<sup>[52]</sup>将通过测试用例的数量与失败测试用例的数量之比称为测试套件的平衡率,其中平衡率为 1 的测试套件被称为平衡测试套件.他们证明了测试套件的平衡率越接近 1,则 SBFL 技术的缺陷定位效果越好.但在实际的缺陷定位中,失败测试用例的数量通常要低于通过测试用例的数量.因此,针对缺陷定位中的测试用例不平衡问题,Zhang 等人<sup>[53]</sup>调查针对类不平衡问题(即失败测试用例数量远小于通过测试用例的数量),研究简单复制失败测试用例,从而扩充失败测试用例数量对 SBFL 方法定位精度的影响.他们的研究表明对失败测试用例的复制,能有效提升多缺陷定位效果.除此之外,与常用的 SBFL 技术相比,MBFL 技术的时间开销更大,MBFL 通过变异的方法植入人工缺陷来模拟真实的程序缺陷,并通过计算两者之间的相似度来进行缺陷定位,MBFL 是一种精度较高的缺陷定位技术.然而该技术需要对语句进行变异操作产生大量的变异体,同时对每个变异体执行所有的测试用例,其时间开销极大,因此很少被工业界采

用.针对这个问题,de Oliveira 等人<sup>[54]</sup>通过对变异体只执行失败测试用例而忽略通过测试用例,来减少变异执行开销.基于 Defects4J 上的实验结果表明,这种执行策略可以减少约 90% 的执行开销,且多缺陷定位精度降低并不显著.

组合测试(Combinatorial Testing)通过关注部分参数间的组合覆盖,来减少需要生成的测试用例.使用部分测试用例对不同的程序模块进行调试,可以有效缩减需要执行的测试用例数量.Ghandehari 等人<sup>[55]</sup>利用组合测试思想提出了一种缺陷定位方法 BEN.组合测试用例中不同模块有不同的测试用例集合,BEN 方法首先确定可能执行缺陷的测试用例组合,然后根据执行缺陷语句数量对测试用例集合进行排序,最后从测试用例集合的排序结果中选择排名靠前的测试用例集合用于缺陷定位.该方法在单缺陷程序和多缺陷程序上均进行了性能评估.与 Tarantula 和 Ochiai 方法相比,BEN 能有效地定位到缺陷所在位置.除此之外,研究人员已证明组合测试可以有效地揭示由影响系统行为的因素之间的相互作用所引起的缺陷,从而提出了最小失效原因架构 MFS(Minimal Failure-Causing Schema)理论以隔离缺陷原因<sup>[56]</sup>.但是大多数旨在识别 MFS 的算法都集中于处理被测程序中的单个缺陷,因此这些方法可能被多缺陷程序中的缺陷混淆情况所影响,从而无法观察到某些缺陷.为此,Niu 等人<sup>[57]</sup>提出了一种新的 MFS 模型,该模型考虑了多缺陷的情况,会分别处理被测程序中的每个缺陷.具体而言,针对某一个程序缺陷,该模型仅会关注在该缺陷中通过或失败的测试用例.触发其他不同缺陷的测试用例将被新生成的测试用例替换.因此,该方法可以在不受缺陷混淆干扰的情况下正常工作.

在执行测试用例的过程中,开发人员可以对测试用例进行排序,或者调整测试用例的覆盖路径,及组合测试中的模块组合,从而提高后续工作的效率.开发人员对测试套件内的测试用例进行排序,会优先执行对缺陷定位有价值的测试用例,并延后执行其余测试用例.例如,为降低 INF-MFL 调试成本,Fu 等人<sup>[58]</sup>从测试用例角度,基于程序元素怀疑度值的变化,提出了一种测试用例优先级排序方法.实验结果表明该方法可以有效减少调试的成本.开发人员通过调整测试用例的覆盖路径或者组合测试中的模块组合来提升执行效率.例如,Perez 等人<sup>[59]</sup>提出一种动态代码覆盖的方法 DCC(Dynamic Code Coverage),该方法会动态改变测试套件对被测程序

的覆盖频谱,旨在减少缺陷定位过程中的执行开销。DCC方法首先分析测试用例针对被测程序中系统组件(子程序)的覆盖路径,然后从文件粒度开始进行逐步细化,直至达到语句级别。实验结果表明,相比SBFL方法,DCC能够平均降低27%的多缺陷定位开销。Yu等人<sup>[60]</sup>在2015年提出一种分类方法,可以区分失败测试用例与单缺陷有关还是与多缺陷有关。他们的实验结果验证了上述分类方法的有效性,从测试用例角度为提升多缺陷定位效果提供了新的研究思路。除此之外,开发人员可以在程序测试过程中,根据程序的状态生成新的测试用例,从而实现更高精度的缺陷定位。例如,Yilmaz等人<sup>[61]</sup>为了解决缺陷混淆问题,提出了一种反馈驱动的自适应组合交互测试过程FDA-CIT(Feedback Driven Adaptive Combinatorial Interaction Testing Process)。在此过程的每次迭代中,他们首先执行测试用例并分析结果,通过计算缺陷特征模型(即通过识别可能的缺陷相互作用)来检测潜在的缺陷混淆情况,然后生成可避免该缺陷混淆情况的新的测试用例,从而减轻缺陷混淆对缺陷定位产生的负面影响。基于两个大型开源软件系统(Apache和MySQL)的实验结果表明,他们提出的FDA-CIT方法能够有效减轻缺陷混淆的影响,从而提高缺陷定位精度。

在测试用例执行结束之后,开发人员需要根据测试预言(Test Oracle)判断测试用例的执行结果,测试预言通常是指某个测试用例的预期输出,通过对比测试用例的实际输出和预期输出,可以判断测试用例是否执行通过。尽管大多数缺陷定位技术仅使用少量测试用例也可以相对准确地定位缺陷,但是选择合适的测试用例并为其创建测试预言费时费力。因为创建测试预言需要花费大量的人工成本。为了解决上述挑战,Xia等人<sup>[62]</sup>提出了一种多样性最大化加速的方法DMS(Diversity Maximization Speedup)。DMS方法试图仅优选少量高质量测试用例进行缺陷定位,开发人员只需要人工给选出的少量测试用例创建测试预言,即可通过缺陷定位技术取得较好的缺陷定位结果。其中可行的优选方法包括:选择执行更多程序语句的测试用例、选择执行了当前已选的测试用例未执行语句的测试用例、选择覆盖路径相似度较低的测试用例。实验结果表明,DMS方法能有效约减测试用例的数量,加速多缺陷定位过程,并同时保证了缺陷定位的精度。而Gao等人<sup>[63]</sup>提出了一种基于汉明距离和K-Means聚类的方法来预测测试用例的执行结果。该方法根据未

标记测试用例与失败测试用例之间的距离来判断其标签(通过或失败),当未标记测试用例与失败测试用例的汉明距离小于某一阈值或与失败测试用例划分至同一类簇时,则判定该未标记测试用例为失败,否则为通过。基于Unix程序集的实验结果表明使用预测结果的定位效果与原始执行结果的定位效果同样有效,有些甚至定位性能更好。Zhang等人<sup>[64]</sup>使用了INF-MFL调试方法,提出了一种测试用例分类方法,以帮助在定位缺陷时使用未标记的测试用例。他们提出一种基于测试分类的方法来利用未标记的测试用例。该分类器称为基于可疑概率(Suspiciousness Probability-based)的分类器,它利用SBFL领域知识(Domain Knowledge)为每个未标记的测试用例分配一个估计的标签(通过或失败)。分类后,新标记的测试用例将被用于语句怀疑度值的计算。基于多缺陷程序的实证研究表明,该方法有助于提高缺陷定位的效果。

需要注意的是,在缺陷定位过程中,存在一类通过的测试用例,这些通过的测试用例执行了包含缺陷的语句,但是执行结果与预期一致,这一类测试用例被称为偶然正确的测试用例(Coincidental Correct Test Cases),当前大量研究表明偶然正确测试用例对缺陷定位效果有负面影响<sup>[65]</sup>。针对上述问题,Liu等人<sup>[66]</sup>提出了一种加权模糊分类的方法FW-KNN(Fuzzy Weighted K-Nearest+Neighbor),来识别并处理偶然正确的测试用例。该方法基于模糊KNN分类算法,其将失败的测试用例作为训练数据,通过的测试用例作为测试数据进行判断。与失败测试用例相似度较高的通过测试用例将会被判定为偶然正确的测试用例。在单缺陷和多缺陷程序的实验结果表明,与传统的Jaccard、Ochiai和Dstar等SBFL缺陷定位方法相比,应用FW-KNN方法能有效提升缺陷定位效果。

### 3.1.5 基于程序语义分析的方法

基于程序语义分析的方法通常通过分析程序内部或程序与程序之间的相关性(如变量相关性、谓词相关性、抽象语法树相关性、控制流图相关性等)进行缺陷定位。基于变量相关性的方法通常会分析程序中变量之间的关系,然后根据变量之间的依赖关系计算语句怀疑度。考虑到程序中变量的变化,Kim等人<sup>[67]</sup>提出一种基于变量的缺陷定位方法VFL(Variable-based Fault Localization),用于解决SBFL方法在测试用例覆盖信息相近时定位效果差的问题。VFL通过识别程序中的可疑变量,来指导程序

语句的排名. 基于 Defects4J 程序集的实验结果表明 VFL 的定位效果较好, 且具有轻量级和可扩展的优点, 并可以与其他方法进行结合以进一步提升缺陷定位效果. Wong 等人<sup>[26]</sup>提出一种基于交叉表的统计方法 CBT(Crosstab-Based Technique). 该方法首先对每条执行语句构建交叉表, 然后使用统计方法来原因确定对应语句的怀疑度. 但实验结果表明 CBT 方法在多缺陷定位上的效果提升并不明显, 同时他们发现 CBT 方法的定位效果容易受到测试套件规模的影响. Gong 等人<sup>[68]</sup>在分析程序控制流图(Control Flow Graph)间的状态相关性信息的基础上, 研究使用状态依赖概率模型来描述程序语句之间的控制依赖关系, 随后提出一种基于通过和失败测试用例状态依赖的缺陷定位方法. 基于 Siemens 和 Unix 程序集中的多缺陷版本的结果表明, 他们提出的方法与 SOBER<sup>[69]</sup>、Tarantula 和 CP<sup>[70]</sup>等缺陷定位方法相比, 在少量测试用例的场景下, 能取得更好的缺陷定位效果.

Feyzi 等人<sup>[71]</sup>基于信息理论分析和统计因果推理, 提出了一种缺陷定位方法 Infarence. 该方法采用基于互信息的特征选择策略, 通过计算语句间的依赖关系来确定导致程序失败的语句组合. 实验结果表明, Infarence 在多缺陷定位效果上要优于 DStar 和 GP19 等缺陷定位方法.

Wang 等人<sup>[72]</sup>提出了一种基于动态不变量(Dynamic Invariant)差异的方法 FDDI. 动态不变量指的是变量之间的关系. 具体来说, 程序在执行过程中, 存在部分变量的值在变化过程中有一定的相关性. 因此研究人员认为, 相关性被破坏的动态不变量可以为缺陷定位提供线索. 当前已经存在通过分析动态不变量来检测程序异常的技术, 但是这些技术通常会产生较大的时间开销. 为了解决上述问题, FDDI 方法首先选择怀疑度较高的函数, 然后检测这些函数中存在的动态不变量. 随后, FDDI 方法会剔除存在于通过测试用例和失败测试用例之间的动态不变量, 从而降低时间复杂度. 最后, 他们使用所有测试用例, 并基于 INF-MFL 调试模式进行缺陷定位. 实验结果表明, FDDI 方法不仅能够提高缺陷定位效果, 而且可以平均减少算法 87.2% 以上的执行时间.

基于程序谓词分析的缺陷定位方法也受到了研究人员的关注. 程序中的谓词(Predicate)是一种返回值为布尔类型的函数, 通常用于控制 for、if、while 等语句块. 研究人员提出的方法一般通过对单个谓

词的切换, 从而强制更改谓词实例的状态(返回结果从 true 更改为 false, 或从 false 更改为 true), 然后通过检查谓词状态切换能够改变测试用例的执行结果, 从而来确定缺陷是否包含于该谓词中. 但是, 切换一个谓词实例有其局限性, 在 Wang 等人<sup>[73]</sup>的实验中, 他们发现单个谓词切换只能识别少数关键谓词. 要克服这个局限性, 可能需要切换多个谓词实例. 但是若考虑所有的谓词实例组合, 则可能存在指数爆炸问题. 因此, Wang 等人<sup>[73]</sup>提出一种基于分层的多谓词切换方法 HMPS. HMPS 方法仅在怀疑度较高的语句内搜索可能包含缺陷的谓词. 实验结果表明 HMPS 方法的多缺陷定位效果要优于部分已有的缺陷定位方法(例如 Barinel、Ochiai).

基于谓词的统计缺陷定位方法以谓词作为研究对象, 度量谓词与程序缺陷间的关联程度, 关联程度越高, 谓词的怀疑度值越高. 但是, 高关联度的谓词并不意味着它是程序执行失败的原因. 软件缺陷定位过程是寻找引发程序失效的原因的过程, 可理解为针对程序执行失效这一现象, 来寻找引发缺陷的原因的因果推理过程. 在因果推理过程中, 在分析处理变量与输出变量之间的因果关系时, 需要考虑其他变量的影响, 否则推理过程会受到混杂偏倚效应(Confounding Effect)的影响, 并导致推理结果不准确. 针对上述问题, 王兴亚等人<sup>[74]</sup>对谓词怀疑度量过程中的混杂偏倚效应的消除进行了研究, 在此基础上提出了 PBSFL(Predicate-based Statistical Fault Localization)方法. 该方法包括 5 个步骤, 分别是程序静态切片、缺陷候选谓词筛选、混杂偏倚元素识别、监控执行与动态约减、回归分析及排序. 具体来说: 首先对源程序执行静态切片, 将影响输出变量的语句集合作为可疑语句集合, 然后根据程序中变量的类型对其进行缺陷候选谓词筛选, 生成可疑谓词集合, 对其中的每一个元素通过程序依赖分析构建因果图, 并推理识别混杂偏倚元素, 最后通过监控记录可疑谓词取值结果、混杂偏倚元素信息进行回归分析, 度量谓词导致失败的贡献来进行排序, 生成谓词序列来帮助缺陷定位. 基于 Siemens 程序集的多缺陷版本的实验结果表明: PBSFL 可以有效提高传统缺陷定位方法(Tarantula、Ochiai、Naish1 等)的效果.

Liu 等人<sup>[75]</sup>提出了一种称为多谓词切换的有界调试(Bounded debugging via Multiple Predicate Switching, 简称 BMPS)方法, 该方法试图通过切换谓词状态来使测试用例通过, 从而定位存在于谓

词中的程序缺陷. BMPS 专注于由控制流引起的程序缺陷. 他们在后续的工作中<sup>[76]</sup>认为这种类型缺陷仅占很小的比例. 因此, 他们进一步提出了一种将 BMPS 与基于语义的调试方法 (Semantic based Debugging Method) 相结合的方法来定位更多类型的缺陷. 基于语义的调试算法从失败的测试用例的覆盖信息中生成一系列方程, 并根据方程的解为开发人员提供包含缺陷代码的最小程序段. 通过迭代和交互的过程, 他们的方法以 INF-MFL 调试方法定位程序中多个缺陷. 基于 Siemens 程序集的实验结果表明: 他们的方法的定位效果要显著优于 BMPS 方法.

除此之外, 基于程序语义分析的方法还能够通过提取程序执行过程中的特征, 分析与缺陷相关的语句或测试用例, 从而提高 SBFL 等缺陷定位技术的效率. 在软件测试过程中, 动态分析是指根据程序的运行过程产生的信息对程序进行分析, SBFL 就是一种动态分析方法. 静态分析是指根据程序非运行时获得的信息对程序进行分析, 其可行的实现方式包括提取抽象语法树、控制流图等. Neelofar 等人<sup>[77]</sup>同时考虑了动态分析方法和静态分析方法, 提出一种结合动静态分析的加权方法来辅助缺陷定位. 其中静态分析方法根据语句的怀疑度分配不同的权重后, 对这些语句进行分类, 动态分析则将计算语句加权之后的分数作为新的怀疑度, 并生成 INF-MFL 调试中使用的语句有序列表. 实验结果表明, 该方法能将单缺陷程序和多缺陷程序上的定位效果分别提升 20% 和 42%. Xu 等人<sup>[78]</sup>将未执行某个缺陷语句而失败的测试用例称为噪声测试用例, 噪声测试用例的存在会导致已有方法在定位该缺陷语句时效率降低. 因此, 他们提出一种可减少这类噪声测试用例的方法框架<sup>[79]</sup>, 该框架通过分析代码块之间的关系, 构建程序基本块之间的关系链, 来改进 Jaccard、Anderberg 和 Dice 等怀疑度计算公式, 从而降低噪声测试用例的影响. 实验结果表明, 他们提出的多缺陷定位方法与 SBFL 技术相比, 其定位精度至少能提高 10%. Li 等人<sup>[80]</sup>将深度学习 (Deep Learning) 引入到缺陷定位, 提出 DeepFL 定位方法. DeepFL 通过抽取多缺陷程序的四类特征来构建怀疑度预测模型, 这四类特征分别为: SBFL 技术计算的语句怀疑度、MBFL 技术计算的语句怀疑度、用于缺陷预测的程序复杂度指标、基于信息检索的文本相似度信息. 实验结果表明: DeepFL 的多缺陷定位效果要明显优于最新的 TraPT 和 FLUCCS

缺陷定位技术.

最后我们针对已有的 INF-MFL 方法进行分类统计, 最终统计结果如表 3 所示. 从表中可以看出, 大部分的研究工作都旨在提高单缺陷定位的准确度. 因为基于 INF-MFL 调试方法的多缺陷定位技术在每次迭代中仅会定位单个缺陷, 这意味着每次迭代都被视为单缺陷定位, 所以提高单缺陷定位准确度就是提高每次迭代定位的准确度, 从而提高基于多缺陷定位流程整体的效果. 除此之外, 还有一些研究工作试图降低缺陷定位的成本, 即减少缺陷定位算法执行过程中的时间消耗.

表 3 INF-MFL 方法相关论文统计结果

分类	论文	数量
基于优化模型的方法	[28,30,31,32,33,34,35,36,37,81]	10
基于切片的方法	[38,39,41,42]	4
基于怀疑度计算公式改进的方法	[25,43,44,45,46,48,49,50,51]	9
基于测试套件进行改进的方法	[53,54,55,57,58,59,60,61,62,63,64,66]	12
基于程序语义分析的方法	[26,67,68,71,72,73,74,76,77,79,80,82]	12

### 3.2 针对 INF-MFL 方法的实证研究

INF-MFL 调试方法是 MFL 领域中研究最多的一类方法, 因此也有大量研究人员对该调试方法展开实证研究. 本节对基于 INF-MFL 调试方法的 MFL 相关研究论文进行了概述, 其中部分论文研究了多缺陷的存在对缺陷定位技术有效性的影响.

在多缺陷定位研究当中, 多缺陷的存在对现有缺陷定位技术的影响方式备受研究人员关注. 例如, DiGiuseppe 等人<sup>[9]</sup>进行了缺陷数量对 SBFL 技术影响的实证研究. 传统的多缺陷定位研究认为, 缺陷定位技术的有效性与缺陷数量成反比. 为了验证该推测, 他们对来自 Unix 程序集的三个程序 (Gzip、Space、Replace) 进行了研究, 这些程序的大小各不相同, 涉及超过 13 000 个多缺陷版本. 他们的研究结果表明: 多缺陷的存在对 SBFL 技术的影响不如预期的那样大, 对基于 INF-MFL 调试方法的 SBFL 技术影响可忽略不计. Sun 等人<sup>[83]</sup>分析了部分缺陷定位公式 (Symmetric Klosgen<sup>[84]</sup>、Ochiai、Jaccard 等) 的共性, 以分析某些定位公式比其他公式更有效的原因. 基于多缺陷程序的实验结果表明: 相较于其他缺陷定位方法, Symmetric Klosgen、relative-Ochiai、relative F1 和 enhanced Tarantula 更适用于多缺陷定位问题. Yan 等人从缺陷数量对 SBFL 定位效果的影响进行了实证研究<sup>[85]</sup>, 研究结果表明:

多缺陷的存在对缺陷定位效果普遍存在负面影响。Xie 等人<sup>[3]</sup>为了克服传统实证研究经验化的局限性,提出了针对怀疑度计算公式的理论研究框架。该框架基于简单的直觉来识别不同公式之间的关系,直觉认为怀疑度值高于缺陷语句的正确语句数量决定了缺陷语句的排名。因此该框架将所有程序语句划分为三个不相交的集合,分别为怀疑度值高于、等于和低于缺陷语句。通过比较不同怀疑度计算公式产生的集合大小,该框架可以比较得出不同怀疑度公式的缺陷定位有效性。上述评估方法是基于单缺陷程序情况提出,因此 Xie 等人的理论研究框架也适用于 INF-MFL 方法。

SBFL 是单缺陷定位研究中的一种经典缺陷定位方法。Lucia 等人比较了 40 种 SBFL 定位公式的实际定位效果<sup>[84]</sup>,他们采用 INF-MFL 调试方法对定位效果进行评估,发现没有一种 SBFL 公式在单缺陷和多缺陷的情况下都能达到最好的定位效果。Xia 等人<sup>[86]</sup>在包含多缺陷版本的大规模程序上调查研究 SBFL 方法是否能够有效辅助开发人员实现缺陷的自动定位。研究结果表明:SBFL 方法的差异性对程序调试的结果和效率有着重要的影响,同时 SBFL 方法能有效节约开发人员的调试时间。

传统的 SBFL 在对小规模程序进行缺陷定位时通常能够取得较好的定位结果。Heiden 等人<sup>[87]</sup>认为传统的 SBFL 方法难以适用于真实的大规模程序。因此,他们使用包含多缺陷版本的 Defects4J 评测程序和 AspectJ 评测程序(包含超过 500 000 行代码),对传统的 SBFL 方法进行验证。他们的实证研究结果表明:SBFL 方法在定位 90% 的缺陷时,需要开发人员平均检查约 450 行代码。因此他们认为,需要考虑更多配套软件制品以提高 SBFL 的有效性,例如通过考虑缺陷报告信息或代码历史变更信息,来进一步提高缺陷定位效果。

基于上述研究结果,我们发现:多缺陷通常会对传统缺陷定位技术的精度产生负面影响,因此一些研究人员试图进一步分析多缺陷产生负面影响的原因,从而希望后续研究,可以提出更加有效的多缺陷定位策略。Perez 等人<sup>[88]</sup>认为即使程序中存在多个缺陷,但测试时也仅有一个缺陷会影响程序的运行,因此 INF-MFL 调试模式相较于其他调试模式更加适用于多缺陷定位。为了验证 INF-MFL 调试方法是否适用于真实程序,他们对真实软件维护过程中的 INF-MFL 调试模式使用情况进行了分析,通过挖掘软件仓库,查找缺陷修复以及根据修复数量对缺陷

进行分类,以评估 INF-MFL 调试方式在实际软件调试过程中使用的普遍性。结果表明,在实际软件调试时,在 82% 的情况下,开发人员会使用 INF-MFL 调试方式,这表明基于 INF-MFL 调试方式的研究具有很高的实用价值。Yan 等人<sup>[19]</sup>基于 4 个真实工业软件系统,展开了多缺陷定位方法的实证研究,研究结果表明,多缺陷程序的异常行为主要来源于程序缺陷之间的相互作用。Zhang 等人研究了程序频谱的分布特征对缺陷定位的影响<sup>[89]</sup>,重点探讨了不同类型的语句与程序频谱之间的关系。其中,他们引入了三种概念以描述不同类型的语句:与缺陷无关、与缺陷有关和与缺陷排除有关。其中与缺陷排除有关指的是执行该类型语句会使得测试用例失败率降低。基于多缺陷程序的实验结果表明:不同类型的语句具有不同的测试用例覆盖路径特征,但是多缺陷的存在会削弱每种类型语句的原始特征。

## 4 基于缺陷独立假设的多缺陷定位方法

基于缺陷干扰假设的多缺陷定位的 INF-MFL 方法需要开发人员多次迭代并逐个定位并修复缺陷,而基于缺陷独立假设的多缺陷定位方法的 IDP-MFL 方法则将多缺陷定位任务划分为多个单缺陷定位子任务,以便允许多个开发人员在不同的子任务上展开并行调试。当被测程序内含有多个缺陷的时候,使用并行调试方法可以简化调试过程,并缩减软件的交付时间。IDP-MFL 一般会对失败测试用例进行聚类分析,并将每个类簇内的失败测试用例与所有的通过测试用例进行组合,以创建针对单个缺陷的测试套件。构建出的不同测试套件最终被分配给不同的开发人员以进行并行调试。

### 4.1 IDP-MFL 方法的提出与改进

针对多缺陷定位问题, Jones 等人<sup>[17]</sup>首次提出了并行调试的思想,即对失败的测试用例进行聚类分析,并将每个类簇内的失败测试用例与所有的通过测试用例进行组合,以生成针对单个缺陷的测试套件。随后,将这些针对缺陷的类簇提供给开发人员以进行并行调试,他们的工作假设每个类簇都能够快速定位出被测程序内的某个缺陷。然而 Hogerle 等人<sup>[8]</sup>通过研究发现:上述假设并不一定成立,因此他们认为 Jones 等人提出的方法<sup>[17]</sup>存在不足。例如,如果某一个开发人员完成调试任务后,其他开发人员仍在进行调试,那么这个开发人员的修复动作可能会影响到其他人员的调试工作,从而降低其他

开发人员的缺陷定位效率. 同时, 他们发现不同聚类算法对缺陷分离的效果影响很大. 随后, Steimann 和 Frenkel<sup>[90]</sup> 提出了基于整数线性规划 (Integer Linear Programming) 的程序频谱分割方法, 他们试图将多缺陷定位问题分解为规模较小且可独立解决的子问题. 因此, 他们采用整数线性规划算法将程序频谱分割成多个独立的程序频谱分区, 每个分区内都包含部分测试用例的频谱信息, 且可以由单个开发人员进行缺陷定位. 结果表明, 与未划分程序频谱的缺陷定位方法相比, 他们提出的方法能够在每一个分区内获得更高的缺陷定位精度.

使用聚类算法对多缺陷进行分离 (Bug Isolation) 是 IDP-MFL 方法中常用的策略之一, 研究人员从这个角度入手开展了多项研究. 聚类的效果直接关系到后续的缺陷定位精度, 如果聚类结果不准确 (即类簇数量不等于程序中的缺陷数量), 则开发人员可能会花费更多的时间进行 IDP-MFL, 甚至通过多次迭代才能完全修复程序中的所有缺陷.

考虑到聚类效果对定位精度的显著影响, 研究人员从多个角度提出聚类策略, 试图得到精度较高的聚类结果. Wang 等人<sup>[91]</sup> 提出了一种基于加权特征的聚类选择策略 WAS (Weighted Attribute-based Strategy). 传统的 SBFL 技术使用的程序谱信息仅通过“1”或“0”来表示对应的语句被某一测试用例“执行”或“未执行”, 而 WAS 策略则对不同类型的程序实体 (例如语句、语句块、函数等) 计算怀疑度, 并使用该怀疑度对测试用例覆盖信息进行加权, 即使用怀疑度对程序谱中的“1”或“0”进行加权, 并根据加权后覆盖信息的相似度对测试用例进行聚类. 基于多缺陷程序上的实验结果表明, 使用 WAS 策略在缺陷定位中的聚类效果要优于其他聚类算法. Wei 和 Han<sup>[92]</sup> 提出了一种基于参数组合的方法来帮助开发人员快速定位多个缺陷. 该方法使用二分法对失败测试用例进行聚类, 进而生成针对单个缺陷的测试套件. 实验结果证明, 该方法的多缺陷定位效果要优于使用 Tarantula 怀疑度计算公式的 INF-MFL 方法, 且优于其他几种主流的缺陷定位技术. Parsa 等人<sup>[93]</sup> 基于程序层次聚类角度, 提出了 Hierarchy-Debug 方法. Hierarchy-Debug 方法旨在分析程序谓词之间的影响, 从而将分层聚类算法应用于聚类谓词, 从而实现定位多个包含缺陷的谓词语句. 实验结果表明, 该方法可以帮助开发人员查找与缺陷相关的谓词.

曹鹤玲等人<sup>[94]</sup> 提出了一种基于 Chameleon 聚

类分析的多缺陷定位方法. 该方法首先选择每一个失败测试用例和所有通过测试用例的覆盖路径来计算怀疑度值, 并选择怀疑度值较高的数个语句作为该失败测试用例的特征向量. 然后使用 Chameleon 聚类算法通过上述特征向量对失败测试用例进行聚类. 最后将每个类簇内的失败测试用例和所有通过测试用例的覆盖路径计算得到多个怀疑度排序列表. 他们的方法假设每个类簇均对应程序中一个缺陷, 因此他们采用并行调试模式, 来同时定位程序中的多个缺陷. 基于 SIR 程序集上的实验结果表明: 其方法比 Jones 等人<sup>[17]</sup> 提出的 INF-MFL 方法在效率有所提升, 且调试迭代次数下降.

为了进一步提高聚类的精度, 部分研究人员对传统的聚类算法进行了改进, 其中的一种常用策略是提出更为精确的测试用例距离度量方法. 例如, Gao 和 Wong 等人提出了一种并行定位多个缺陷语句的方法 MSeer<sup>[20]</sup>, 该方法基于一种改进的  $k$ -medoids 算法对失败测试用例进行聚类, 然后使用聚类后的测试套件进行缺陷定位. 实验结果表明: MSeer 比 INF-MFL 方法和 Jones 等人的方法在缺陷定位效率和效果上均表现得更好. Zakari 等人<sup>[95]</sup> 使用了基于边缘间距 (edge-betweenness) 的距离公式来度量程序语句 (即节点) 执行的距离, 并提出一种网络社区聚类算法. 该算法将失败测试用例划分至多个类簇中, 其中每个类簇仅针对单个缺陷. 实验结果表明, 网络社区聚类算法可以有效地将不同的缺陷分离至以缺陷为中心的不同类簇中. 但是, Zakari 等人<sup>[96]</sup> 认为现有的聚类算法无法有效地将失败的测试与其引起的缺陷进行有效划分, 这对缺陷定位存在负面影响. 随后, 他们考虑到类簇中连接越密集的程序语句越有可能包含相同的缺陷, 在原始网络社区聚类 (Divisive Network Community Clustering) 结果中通过删除相似度较低的关联, 以进一步精简类簇. 实验结果表明: 与 MSeer 和原始方法相比, 他们提出的改进方法具有更好的多缺陷定位效果.

王兴亚等人<sup>[97]</sup> 认为与特定缺陷无关的失败测试用例是 SBFL 方法缺陷定位有效性降低的主要原因. 因此, 他们提出了一种基于模糊 C 均值聚类 (Fuzzy C-Means Clustering) 的多缺陷定位方法 FCMFL. 该方法首先通过模糊 C 均值聚类分析失败测试用例与不同缺陷间的隶属关系, 得到与每个缺陷关联的失败测试用例, 然后根据隶属度作为权重计算每条语句的怀疑度, 最终每个类簇将生成一个语句检查序列, 以指导开发人员进行程序调试. 基

于 6 个 Siemens 程序和 6 个 SIR 库中的程序的实验结果表明:与 Tarantula、Ochiai、Naish 和 Wong 等方法相比,FCMFL 方法可以降低多缺陷对 SBFL 方法的负面影响,从而提高 SBFL 方法的缺陷定位精度。

除了聚类方法,研究人员还从其他不同角度实现 IDP-MFL。例如,Jeffrey 等人<sup>[98]</sup>提出一种基于值替换(Value Replacement)的缺陷定位方法。该方法在 INF-MFL 过程中可以反复更改执行程序的状态,通过搜索程序运行中可能导致出现异常的赋值语句来定位缺陷语句。但是反复更改程序状态在多缺陷程序中会带来更多的时间开销<sup>[99]</sup>。因此,他们<sup>[99]</sup>将并行调试引入到值替换方法中,以减少定位多个缺陷所需的总时间消耗。实验结果表明,并行调试下的值替换方法在定位单个版本中的所有缺陷仅需数分钟,因此可以有效减少缺陷定位的时间开销。Sun 等人提出了一种基于迭代的方法 IPSETFUL,用于选择更有效的测试用例<sup>[100]</sup>。IPSETFUL 方法首先依据执行结果生成程序谱概念格(Concept Lattice of Program Spectrum,简称 CLPS),并将程序语句划分为危险、敏感和安全三个级别,随后开发人员检查危险级别的语句以判断是否含有缺陷。然后,IPSETFUL 选择部分测试用例覆盖危险和敏感的语句,在下一迭代中生成新的 CLPS,直到所有测试通过则迭代终止。实验结果表明:IPSETFUL 方法的定位效果要优于一些传统 SBFL 方法(例如 Jaccard、Ochiai、Tarantula)。

本节对已有的 IDP-MFL 相关论文进行了分类统计,最终统计结果如表 4 所示。从表 4 可以看出,为了实现并行调试任务,研究人员提出了多种方法将单个多缺陷定位任务划分为多个子任务,其中基于聚类的划分方法被最广泛使用。基于聚类的多缺陷定位方法会将执行失败的测试用例划分至多个类簇中,每一个类簇内的失败测试用例均由同一个缺陷导致,因此多个开发人员可以并行地使用多个类簇,分别定位到程序中的每一个缺陷。基于聚类的多缺陷定位方法的结果准确度与聚类的精度有较强的相关性,因此这部分研究工作都试图提出更高聚类精度的聚类方法,使得多缺陷定位效果更好。

表 4 IDP-MFL 方法相关论文统计结果

分类	论文	数量
缺陷分离	[8,20,90,91,92,93,94,95,96,97]	10
其他方法	[98,99,100]	3

## 4.2 针对 IDP-MFL 方法的实证研究

除了提出并行调试场景下多缺陷定位的新方法,研究人员还针对 IDP-MFL 方法的有效性展开了实证研究。

为了从 IDP-MFL 角度上验证多缺陷对缺陷定位是否存在负面影响,Li 等人<sup>[101]</sup>对多缺陷划分进行了实证研究。考察不同精度的聚类结果对多缺陷分离后的缺陷定位效果的影响。实证结果表明,缺陷定位的效果与聚类算法的聚类效果密不可分,聚类算法的聚类效果越差,其定位效果也越差。因此划分的准确度对于后续步骤来说至关重要。Huang 等人<sup>[102]</sup>对缺陷定位中的缺陷分离进行了实证研究,他们分析了 6 种缺陷定位方法(Naish2、Jaccard、Tarantula、Wong2、Wong1 和 Rogot1)和两种聚类方法(即 K-means 聚类和层次聚类)对缺陷分离效果的影响。结果表明,Wong1 能实现最好的缺陷定位效果,K-means 聚类算法缺陷分离上要优于层次聚类算法。Zakari 等人<sup>[103]</sup>通过调查研究,他们发现:使用基于失败测试用例的覆盖相似性与距离度量进行划分的聚类算法并不合适。他们调查了并行调试方法的有效性,实验中通过 K-means 聚类算法和三个基于相似性的距离度量公式来评估缺陷定位的效果。同时,他们比较了最新的并行调试方法 MSeer 和 INF-MFL 调试方法的缺陷定位效果。实验结果表明:基于失败测试用例执行路径相似度的聚类算法并不适用于缺陷分离,同时使用该聚类算法会降低多缺陷定位的效果。

为了分析部分情况下聚类算法效果较差的原因,Debrov 等人<sup>[21]</sup>对多缺陷程序内部的缺陷相互影响情况进行了研究,因为他们认为当程序内含有多个缺陷时,可能部分缺陷之间会存在相互影响,进而对多缺陷定位效果产生一定的负面影响。针对上述问题,DiGiuseppe 等人<sup>[15]</sup>将缺陷间的干扰现象分为四类(详见 2.2 节),在他们的研究中,缺陷混淆是最普遍存在的一种现象,应该受到更多的关注与研究。

随后,Xue 等人<sup>[10]</sup>发现,在面向对象的程序设计语言中也存在缺陷干扰现象,但这种现象对缺陷定位性能的影响可以忽略不计。DiGiuseppe 等人<sup>[104]</sup>从缺陷数量、缺陷类型和缺陷定位方法等方面展开了大规模实证研究。他们的实验结果表明,缺陷数量对缺陷定位有效性影响显著。同时,他们发现缺陷类型与缺陷定位干扰之间不存在相关性。该研究结果对软件开发的实践人员和研究人员具有一定的指导意

义. 除此之外, Yan 等人研究了缺陷数量对基于频谱的缺陷定位方法的影响<sup>[105]</sup>. 他们基于 14 个大规模开源程序进行了评估, 实验结果表明, 尽管多缺陷确实会对缺陷定位效果产生负面影响, 但是不同的缺陷定位技术的负面影响程度并不相同.

## 5 不基于任何假设的多缺陷定位方法

不基于任何假设的多缺陷定位的 NOH-MFL 方法与 INF-MFL 方法和 IDP-MFL 方法不同, 这一类方法通过单次迭代来定位多个缺陷语句, 且不把多缺陷定位任务进行拆分. NOH-MFL 方法缩减了其他 INF-MFL 方法需要多次编译被测项目和执行测试; 用例所产生的时间开销, 且避免了聚类算法产生的额外开销, 和聚类过程可能产生的聚类结果不准确的问题. 现有的 NOH-MFL 方法<sup>[106-107]</sup> 通常使用人为限制或机器学习的方法来设置缺陷语句搜索上限, 以减少人工调试成本和提高缺陷定位效果为研究目标. 本文在调研时发现, 相比前两类方法, NOH-MFL 方法较少受到研究人员的关注, 在最近 12 年中仅有 10 篇相关论文发表. 这些论文根据研究思路的不同主要分为两类: (1) 多个缺陷语句排列在同一个语句有序列表中; (2) 借助机器学习算法实现多缺陷语句的一次定位.

第一类方法与传统缺陷定位流程相似, 不同的是研究人员通过改进怀疑度计算方法, 尝试将程序内的多个缺陷语句都排在有序列表的前列, 从而让开发人员可以快速定位到多个缺陷语句. Abreu 等人<sup>[108]</sup> 结合了 SBFL 方法和 MBD (Model-Based Diagnosis) 方法提出了一种缺陷定位方法 BARINEL. 该方法基于测试用例的程序频谱对被测程序进行建模, 随后基于 Ochiai 公式计算每个程序模块含有缺陷的概率. 作为其组成部分的 MBD 方法主要针对多缺陷定位问题提出, MBD 方法首先依据频谱信息构建程序语句组合, 然后采用贝叶斯推理 (Bayesian Reasoning) 计算不同程序语句组合的出错概率, 最终排在前列的程序语句组合被认为是多缺陷语句. BARINEL 方法充分利用了程序频谱所包含的信息, 并考虑到多条语句对程序输出的影响. 基于 Siemens 数据集上的实验结果表明: BARINEL 方法在多缺陷定位性能上要优于 SBFL 方法 (Ochiai 和 Tarantula) 和仅使用贝叶斯推理的方法. Abreu 等人<sup>[109-110]</sup> 还提出一种基于逻辑推理的缺陷定位方法 Zoltar-M. 该方法将程序语句视为组件, 采用贝叶斯

方法分别计算出组件对输出失败和输出通过的概率, 然后对程序候选语句计算出错概率, 最后按照出错概率的大小进行降序排列. 实验结果表明: Zoltar-M 方法在多缺陷定位效果上要优于 Ochiai 和 Tarantula 等传统缺陷定位方法.

Lamraoui 和 Nakajima<sup>[111]</sup> 采用全流敏感追踪公式 (Full Flow-Sensitive Trace Formula) 对程序进行编码, 可以更为有效地识别出程序缺陷位置. 他们的方法结合基于满意度的公式验证技术和基于模型的诊断理论, 因此能够定位程序中多个缺陷. 然而, 他们的方法仅在 Siemens 评测程序中一个相对较小的程序 (即 Tcas) 上进行了验证. 此外, 对于包含多个缺陷的大规模程序来说, 他们的方法在缺陷定位时仅考虑了通过测试用例, 而未考虑失败测试用例. 而失败测试用例在已有的 MFL 研究中<sup>[10, 15, 104]</sup> 则被经常使用. Zakari 等人<sup>[107]</sup> 基于复杂网络理论, 提出了一种缺陷定位方法 FLCN. FLCN 方法采用度中心性 (Degree Centrality) 与结构洞 (Structural Hole) 来查找缺陷相关语句. FLCN 方法能够在一次调试迭代中同时定位多个缺陷. 实验结果表明: FLCN 方法要优于现有的缺陷定位方法 (例如 Tarantula、Ochiai 等).

第二类方法主要借助基于搜索的方法, 该方法将程序中的各个实体视为对象, 然后通过学习算法找到多个缺陷语句的位置. Wong 等人<sup>[5]</sup> 结合神经网络算法, 提出了一种基于径向基函数的多缺陷定位方法. 该方法把测试用例的覆盖路径视为特征向量, 并把测试用例的执行结果 (即成功/失败) 作为神经网络的输出数据. 然后为每一个程序语句构造虚拟覆盖路径作为训练数据. 其中每个虚拟覆盖路径都只会覆盖一个语句, 则神经网络的输出就是该语句的怀疑度值. 例如, 为包含 5 行语句的程序中的第 3 行语句构造虚拟覆盖路径, 则路径为“00100”. 将该虚拟路径作为特征向量输入神经网络, 输出数值即为第 3 行语句的怀疑度值. 实验结果表明, 该方法的多缺陷定位效果要优于使用 Crosstab 或 Tarantula 等公式的基准方法. 何加浪等人<sup>[112]</sup> 认为 INF-MFL 和 IDP-MFL 方法的效果都不够理想, 其原因是由于缺陷之间的复杂关系使多缺陷定位问题变得十分复杂. 因此, 通过深入分析缺陷间的征兆相关信息, 计算出输入对各个缺陷的支持度分量, 构造合适的神经网络模型来学习输入与缺陷位置的关系. 神经网络训练完成后, 针对每个语句构造一个虚拟测试作为已训练出的神经网络的输入, 神经网络的输出

结果即为该语句的怀疑度值. 该方法的网络模型第一层选用概率神经网络 (Probabilistic Neural Network, 简称 PNN), 用于判断输入对各缺陷的支持度, 第二层选用径向基函数网, 利用 RBF 的逼近能力, 计算每个语句为缺陷语句的概率, 然后生成有序列表, 并最终完成多缺陷定位. 实验结果表明: 他们的方法效果优于现有的 RBF-FL<sup>[5]</sup> 和 PARA-FL<sup>[17]</sup> 方法. 王赞等人<sup>[23]</sup> 提出一种基于遗传算法的多缺陷定位方法 GAMFal. GAMFal 方法包括三个步骤: 首先对多缺陷定位问题进行建模, 利用遗传算法搜索最优种群, 确定缺陷分布的染色体编码方式和选定合适的适应度函数; 随后使用遗传算法在解空间中搜索具有最高适应度值的候选缺陷分布, 在满足终止条件后返回最优解种群; 最后依据最优解种群对程序实体排序, 最终开发人员可以依次检查程序实体并确定多个缺陷的具体位置. 基于 Siemens 程序集中的 7 个程序和 Unix 程序集中的 3 个程序 (gzip、grep 和 sed) 作为评测对象, GAMFal 方法在整体定位效率方面优于其他经典的缺陷定位方法 (Tarantula、Improved Tarantula 及 Ochiai), 且需要更少的人工交互. 除此之外, GAMFal 的执行时间也在可接受的范围之内. Zheng 等人<sup>[106]</sup> 提出了一种基于遗传算法的 FSMFL 方法. FSMFL 方法将所有语句按照是否含有缺陷编码成染色体, 例如, “0100100000” 表示一个 10 行代码的程序中第 2 行和第 5 行代码包含缺陷. 使用操作算子 (例如选择、交叉、变异) 进化种群, 直到个体的适应度函数达到预定的阈值, 最终个体内为 “1” 的语句即为缺陷语句. 实验结果表明, FSMFL 在多缺陷定位的效果和精度上, 比较传统的 SBFL 方法 (Tarantula、Ochiai、DStar 等) 都具有更好的定位效果. Wang 等人<sup>[113]</sup> 提出了一种通过分析缺陷传播环境来确定缺陷的方法, 缺陷传播环境是指测试用例执行缺陷语句之后的程序异常状态, 其通常体现于变量的异常值或异常的谓词判断结果. 他们的方法通过识别缺陷程序和示例程序之间的执行状态和结构语义上的差异来定位可疑语句. 通过对值序列和结构语义进行交互分析, 可以减少代码变化和缺陷传播的影响. 实验结果表明, 该方法可以有效地定位可疑语句, 并在有足够的示例程序的情况下可以为缺陷修复提供帮助.

本节对 NOH-MFL 相关方法进行了分类统计, 统计结果如表 5 所示. 从表 5 中可以看出, NOH-MFL 定位方法主要分为两种策略. “所有缺陷语句在单个

怀疑度排序列表内排名靠前的位置”这一策略旨在把所有的缺陷语句排名至怀疑度排序列表的靠前位置, 因此开发人员在根据怀疑度排序列表依次检查语句时, 能在较短的时间内找到多个缺陷语句. “通过搜索算法一次性识别程序中的所有缺陷”是一个理想化的策略, 旨在使用如遗传算法等元启发式搜索算法, 或神经网络等机器学习算法, 一次性识别并修复程序中的缺陷语句.

表 5 NOH-MFL 方法相关论文统计结果

分类	论文	数量
多个缺陷在单个怀疑度排序列表内排名靠前	[107,108,109,110,111]	5
机器学习算法一次性识别程序中的多个缺陷	[5,23,106,112,113]	5

## 6 性能评测指标分析

这一节我们对 MFL 研究中经常使用的评测指标进行了分类, 并统计了不同指标的累计使用频率, 结果如表 6 所示, 表中仅列出了使用频率超过 1 次的性能评测指标. 从表中可以看出, Exam Score 指标的使用次数最多, 累计使用次数达到了 20 次. 其次分别是 Expense Score 指标和 Wasted effort 指标, 分别为 19 次和 12 次. 根据不同的代码审查策略, 本文将表 6 中的性能评测指标分为两类, 即基于代价的评测指标和基于精度的评测指标. 不同的评测指标适用于不同的调试模式, 接下来将依次介绍这些评测指标并举例说明.

表 6 性能评测指标使用次数统计结果

指标类别	指标名称	累计使用次数
基于代价的 评测指标	EXAM score	20
	Expense score	19
	Wasted effort	12
	Average/Cumulative number of statements examined	11
	T-score	5
基于精度的 评测指标	Efficiency	3
	Top-N	8
	Proportion of bugs localized	2

### 6.1 基于代价的评测指标

基于代价的评测指标, 通过统计开发人员定位出程序中真实缺陷位置所需的时间成本来评估缺陷定位技术有效性. 常见的基于代价的评测指标有: EXAM score 及其相关指标、检查语句总和及均值.

#### 6.1.1 EXAM score 及其相关指标

EXAM score 是缺陷定位研究中最常使用的评

测指标之一<sup>[37,95-96,105,114]</sup>,这是一种基于语句怀疑度排序的评价指标,研究人员将程序实体按怀疑度取值从高到低进行排序,并依次审查,直到找到缺陷语句,EXAM score 指标衡量开发人员发现第一个缺陷语句所需要检查代码的百分比<sup>[67]</sup>,其计算公式为

$$\text{EXAM score} = \frac{\text{rank of the faulty entity}}{\text{number of the executable entities}} \quad (1)$$

在式(1)中,分子是缺陷语句在怀疑度排序列表中所处的位置(即 Rank 值),分母是被执行的程序语句总数.当存在多条语句的怀疑度取值相等的时候,那么 rank 值将无法确定.目前有三种方式可以计算其对应的 rank 值.假设存在缺陷语句  $e_f$ ,怀疑度取值高于  $e_f$  的正确语句数量为  $A$ ,怀疑度取值与  $e_f$  相同的正确语句数量为  $B$ ,则:

(1)最好情况.在检查所有怀疑度值与缺陷语句  $e_f$  相同的语句时,开发人员第一个检查的就是语句  $e_f$ ,即开发人员能够以最快的速度定位真实缺陷语句的位置.此时 rank 值的计算公式为

$$\text{rank}(e_f) = A + 1 \quad (2)$$

(2)最坏情况.与“最好情况”相反,在检查所有怀疑度值与缺陷语句  $e_f$  相同的语句时,开发人员最后一个检查的才是  $e_f$ ,即开发人员需要耗费最多的时间定位缺陷语句.此时 rank 值的计算公式为

$$\text{rank}(e_f) = A + B \quad (3)$$

(3)平均情况.介于最好情况与最坏情况之间,即开发人员会检查一部分怀疑度值与缺陷语句  $e_f$  相同的正确语句,但是不及“最坏情况”那么多.此时 rank 值的计算公式为

$$\text{rank}(e_f) = \frac{(A+1) + (A+B)}{2} \quad (4)$$

包括 EXAM score 在内的基于检查语句百分比的评估方法是缺陷定位领域最常用的评估方法.在本文统计的 MFL 相关论文中,Expense Score、Wasted effort、T-score 和 Efficiency 的计算公式与 EXAM score 的计算公式(式(1))完全相同,所评估的内容也完全一致.

对于单缺陷程序定位,EXAM score 值越小表明开发人员只需要检查较少的语句就能找到真正的缺陷语句,则相应的缺陷定位方法有更好的缺陷定位效果.但是 EXAM score 指标仅能评估单个缺陷的定位时间成本,不能完整统计 IDP-MFL 和 NOH-MFL 过程产生的总体时间成本,因此传统的 EXAM score 更适用于基于缺陷干扰假设的多缺陷

定位技术的评测.

为了使 EXAM score 能够评测更多类型的缺陷定位方法,研究人员对原有的 EXAM score 评测指标进行了拓展.例如:

(1)Gao 等人<sup>[20]</sup>在他们的研究工作中针对 IDP-MFL 调试方法提出了 T-EXAM 评测指标.T-EXAM 通过计算所有子任务中定位首个缺陷语句的 EXAM score 值的总和来评估多缺陷定位方法的效率;

(2)Zheng 等人<sup>[106]</sup>在他们的研究工作中针对 NOH-MFL 调试方法提出了  $\text{EXAM}_F$  和  $\text{EXAM}_L$  评测指标. $\text{EXAM}_F$  和  $\text{EXAM}_L$  分别表示单次迭代中定位第一个缺陷和最后一个缺陷所需要检查的程序语句占总程序语句的百分比.

### 6.1.2 检查语句总和及均值

EXAM score 及其相关指标针对是单个缺陷版本内的缺陷定位时间成本.除此之外,研究人员还提出了针对多个缺陷版本的评测指标:检查语句总和(Cumulative Number of Statements Examined, CNSE)和检查语句均值(Average Number of Statements Examined, ANSE).

CNSE 评测指标<sup>[66]</sup>是指对于给定的被测程序, $n$  表示缺陷版本的数量, $M$  和  $N$  表示两种不同的缺陷定位技术. $M(i)$  和  $N(i)$  指的是通过  $M$  和  $N$  进行缺陷定位后,在第  $i$  个缺陷版本定位所有缺陷语句所需检查语句数量的总和.由此可知,如果  $\sum_{i=1}^n M(i) < \sum_{i=1}^n N(i)$ ,则  $M$  比  $N$  更有效.越低的 CNSE 值意味着更好的缺陷定位技术. ANSE 评测指标<sup>[20]</sup>与 CNSE 评测指标相似,如果  $\frac{\sum_{i=1}^n M(i)}{n} < \frac{\sum_{i=1}^n N(i)}{n}$ ,则  $M$  比  $N$  更有效.因为 CNSE 和 ANSE 指标统计了每个缺陷版本的所有缺陷语句的定位时间成本,所以这两个指标适用于单缺陷和多缺陷定位问题,且同时适用于三种不同多缺陷调试模式.

## 6.2 基于精度的评测指标

基于精度的评测指标,通过统计经过缺陷定位技术处理之后达到某一定位精度要求的程序版本数量,从而评估缺陷定位技术有效性.

### 6.2.1 TOP-N

TOP-N 评测指标同时适用于单缺陷和多缺陷定位问题.在软件单缺陷定位问题中,Top-N 是指开发人员在检查排名表前  $N$  个程序实体(如程序语

句、函数或文件)内,就能发现缺陷的程序版本数量.该指标在部分文献中也称为  $\text{acc}@n$ . Top- $N$  值越高表示开发人员在检查相同数量的语句行,对应的缺陷定位技术能检测到缺陷版本数量越多,相应的缺陷定位技术就有着更好的定位精度.针对多缺陷定位问题,Top- $N$  为前  $N$  个实体内成功检测到真实缺陷的数量,同理,Top- $N$  值越高表明相应的技术能定位到更多的缺陷.

据 Kochhar 等人<sup>[115]</sup>的研究统计,大多数开发人员在缺陷定位时仅会检查怀疑度排序列表中靠前的一些程序实体.其中,9.43%的开发人员只检查第一个程序实体,而 73.58%的开发人员会检查前 5 个程序实体,接近 98%的开发人员认为检查前 10 个程序实体是较为合理的.因此 TOP-1、TOP-5 和 TOP-10 这三个指标具有重要的意义.

### 6.2.2 定位缺陷比率

定位缺陷比率(Proportion of bugs localized)评测指标首先会设定数个比率,并假定开发人员在缺陷定位的过程中只愿意检查给定比率的代码,然后针对每个比率计算能够被定位的程序缺陷的数量.因此,在相同比率下,能够定位到更多程序缺陷的缺陷定位技术效率更高.

Sun 等人<sup>[83]</sup>在他们的研究中分析了部分缺陷定位公式能够取得更好缺陷定位效率的原因.他们的实验结果表明,Tarantula 和 Ochiai 公式在仅检查 10%的程序元素时可以定位 39%和 45%的缺陷.Klogsen 和 Added Value 公式相比可以定位更多的缺陷,分别为 47%和 48%.

## 7 统计显著性分析

统计假设检验(Statistical hypothesis test)是用来判断样本与样本,样本与总体的差异是由抽样引起还是本质差别造成的统计推断方法.其基本原理是先对总体的特征作出某种假设,然后通过抽样研究的统计推理,对此假设应该被拒绝还是接受作出推断.假设检验可分为正态分布检验、正态总体均值分布检验、非参数检验三类.考虑到缺陷定位领域的怀疑度无法满足参数检验要求的假定,因此研究人员大量采用的非参数检验,例如曼-惠特尼 U 检验(Mann-Whitney U test),Wilcoxon 符号秩检验(Wilcoxon signed-rank test),其使用次数统计结果如表 7 所示.

表 7 显著性分析方法使用次数统计结果

指标	累计使用次数
Wilcoxon 符号秩检验	6
曼-惠特尼 U 检验	2
student t 检验	1
Quade 检验	1

Wilcoxon 符号秩检验(Wilcoxon Signed Rank Test)是由 Wilcoxon 于 1945 年提出,是缺陷领域中研究人员最常使用的指标<sup>[80,95,103]</sup>. Wilcoxon 符号秩检验是一种非参数假设检验方法,不受样本总体分布的影响,但严格要求进行检验的两组样本数量一致.通常研究人员在比较不同缺陷定位技术的定位效果,都是在相同的数据集下进行实验,因此最终得到的定位结果都是匹配的,符合 Wilcoxon 符号秩检验的条件.例如,Zakari 等人<sup>[95]</sup>使用 Wilcoxon 检验基准方法在多缺陷程序中定位的缺陷数量是否大于比提出的方法定位的缺陷数量,实验显示提出的方法能有效定位更多的缺陷.实验中常采用置信水平为 95%的 Wilcoxon 符号秩检验进行显著性分析<sup>[20,25,48,63,88,95,103,106]</sup>,即假设检验水准为 0.05.该种检验方法适用于 INF-MFL、IDP-MFL 和 NOH-MFL 三类缺陷定位方法,是检验不同缺陷定位方法效果是否存在显著性差异的首选指标

曼-惠特尼 U 检验(Mann-Whitney U test)是由 Mann 和 Whitney 于 1947 年提出,是一种非参数的假设检验方法,适用于检验两组数据是否来自同一样本总体.其假设基础是,若两组样本存在差异,则其中心位置将不同,因此这种检验方法不要求两组数据必须严格匹配.在缺陷定位领域,曼-惠特尼 U 检验常用于检验不同缺陷定位技术的定位效果是否存在显著性差异.例如,DiGiuseppe 等人<sup>[104]</sup>使用曼-惠特尼 U 检验来检验不同因素对多缺陷定位的效果是否存在显著性差异.由于曼-惠特尼 U 检验可适用于两组样本数量不同的情况,因此建议研究人员在实际实验中对非匹配样本数据检验时可采用该种假设检验方法

Student t 检验(Student's t test),简称 t 检验,最早是由 William 于 1908 年发表于《Biometrika》<sup>[116]</sup>. t 检验主要用于样本含量较小,总体标准差未知的正态分布. t 检验通过 t 分布理论来推论差异发生的概率,从而比较两组样本的均值是否存在显著性差异.例如,DiGiuseppe 等人<sup>[104]</sup>在他们的研究中同样使用了 student t 检验,他们使用 t 检验评估缺陷数量对缺陷定位花费的影响是否显著.最终结果表明

他们不能拒绝原假设,即缺陷数量对缺陷定位花费影响显著.但是 t 检验的使用前提是总体样本需要服从正态分布,因此在使用该假设检验方法前需要先检验定位结果是否服从正态分布.然而很难保证定位结果能够满足该要求,因此不建议研究人员使用 t-假设检验方法.

Quade 检验一种非参数的假设检验方法,是 Wilcoxon 符号秩和检验的多重采样扩充,由 Quade 提出<sup>[117]</sup>,Quade 检验适用于检验多组数据是否来自同一样本总体.例如 Hogerle 等人使用 Quade 检验,研究不同程序划分算法对缺陷划分效果的影响.他们提出假设:实验中采用的五个划分算法产生的数据结果是有关联的.最终实验结果表明该假设被拒绝,即不同程序划分算法对缺陷划分的效果影响有差异.这种假设检验方法适用于 IDP-MFL 方法不同程序划分算法的比较.

## 8 评测对象分析

在多缺陷定位的研究工作中,研究人员一般使用开源软件作为评测对象进行实证研究.我们对论文中使用的评测对象进行了系统整理,统计了累计使用次数,结果如表 8 所示.需要说明的是在表中我们仅列出了使用频率最高的 10 个评测对象,因此该表并未包含使用次数较少的评测对象.表 8 列出了评测程序的名称、编程语言、累计使用次数和初次使用时间以及程序规模分类等信息.从表 8 中可以看出,被采用次数最多的评测对象是 C 语言的 Siemens 程序集、Unix 程序集以及 Space 程序,其次是 Java 语言的 Defects4J 程序集、NanoXML 程序和 Ant 程序,除此之外还有 Linux 编译程序 Make 和其他开源程序.

表 8 评测对象使用次数统计结果

评测对象	编程语言	累计使用次数	首次使用时间	程序规模
Siemens test suite <sup>[119]</sup>	C	40	1994	小规模
Unix programs <sup>[118]</sup>	C	35	2004	gzip 为中等规模,其余为大规模
Defects4J <sup>[120]</sup>	Java	13	2014	大规模
Space <sup>[121]</sup>	C	23	1998	中等规模
NanoXML <sup>[122]</sup>	Java	7	2005	中等规模
Ant <sup>[123]</sup>	Java	7	2008	大规模
Make <sup>[124]</sup>	C	6	2010	大规模
XML-Security <sup>[125]</sup>	Java	5	2008	大规模
Jmeter <sup>[126]</sup>	Java	4	2010	大规模
Jtopas <sup>[127]</sup>	Java	3	2009	中等规模

### 8.1 常用评测对象介绍

如表 8 所示,研究人员广泛使用的评测程序主要分为 C 语言程序和 Java 语言程序这两类.论文将各个评测程序进一步细分为大、中、小三种规模,其中代码行数小于 1000 行的程序被称为小规模程序,行数在 1000 行至 10000 行之间的程序被称为中等规模程序,行数超过 10000 行的程序被称为大规模程序.除了 Defects4J 以外的所有评测程序都来自于软件代码仓库 SIR(Software-artifact Infrastructure Repository)<sup>[118]</sup>,SIR 提供了丰富的软件源代码、配套的测试用例、人工植入或真实的软件缺陷,以及多种辅助脚本工具方便研究人员使用.这些有利条件使得 SIR 成为了研究人员使用最广泛的评测程序来源之一.

在 SIR 库中,Siemens 测试套件是使用时间最早,使用次数最多的小规模被测程序,累计有 40 项研究使用了该程序集.Siemens 测试套件是由西门子子公司研究部的 Ostrand 及其同事搜集而成,用于

研究控制流和数据流覆盖标准的缺陷检测能力,Ostrand 将其共享给 SIR 库.Siemens 测试套件中包含 7 个程序,其中 tcas 是飞机防撞系统,schedule2 和 schedule 是优先级调度程序,tot\_info 是在输入数据给定的情况下计算统计信息,print\_tokens 和 print tokens2 是词法分析器,而 replace 执行模式匹配和替换程序.

Siemens 测试套件因其规模小、运行速度快等特点被大量缺陷定位相关研究所采用<sup>[66,101]</sup>,但是 Siemens 测试套件包含的这 7 个程序的代码长度均未超过 600 行,其代码规模远小于真实生产环境中的大规模程序.为了使实验结论更具普遍性和实际指导意义,研究人员随后进一步在 SIR 库中添加了中大规模程序,其中常用的包括 Unix 程序集(包含程序 grep、gzip、sed 和 flex)和 Space 程序.

在 Unix 程序集中,flex、grep、gzip 和 make 均为来自 Gnu 站点的 Unix 实用程序.其中,flex 是一个词法解析程序,包含 11000 多行代码;Grep 是一

个文本搜索工具,能使用特定模式匹配搜索文本,该程序中包含 13000 多行代码;Gzip 是一个 GNU 自由软件的文件压缩程序,包含 6000 多行代码.相比 Siemens 程序集,Unix 程序集的代码规模更大,可以模拟真实生产环境中的中大型程序,因此 Unix 程序集也被众多研究人员采用<sup>[9,63,68,74,75,97,108,111]</sup>.但是 Unix 程序集中的缺陷大都采用人工植入的方式来产生,因此即使其规模接近生产环境中的程序,其缺陷类型依然与生产环境中的缺陷存在差异.为了进一步提高研究工作的可信度,研究人员随后在 SIR 库中添加了来自真实生产环境的大型缺陷程序 Space. Space 程序是一个针对数组定义语言(Array Definition Language,简称 ADL)的解释器,包含 9000 多行代码.该程序读取含有多个 ADL 语句的文件,并检查文件内容是否符合 ADL 语法和特定的一致性规则.Space 程序的每一个缺陷均来自真实的生产过程,能在一定程度上代表真实生产环境中的缺陷程序.

上述评测对象均基于 C 编程语言实现,随着近年来面向对象语言程序的日益流行以及 Java 语言程序占比的逐步升高,研究人员逐渐在实证研究中考虑基于 Java 语言实现的评测对象.其中以 Defects4J、NanoXML 和 Ant 为主. Defects4J 程序集是一个开源软件缺陷存储库,可以帮助软件相关从业人员和研究人员评估缺陷定位效果;NanoXML 是一个轻量级 XML 解析程序,包含 8000 多行代码;Ant 是 Apache 项目提供的基于 Java 的开源构建工具,类似于 Unix 的 Make 程序.

在上述 Java 评测对象中,Defects4J 最初是由 Martinez 等人开发<sup>[128]</sup>,现在已广泛用于缺陷自动定位和修复方法的评估<sup>[129-130]</sup>. Defects4J 经过了同行评审和结构化调整,被认为是当下具有良好组织的最大真实 Java 缺陷公开数据库<sup>[131]</sup>,因此近年来 Defects4J 程序集成为了缺陷定位领域的主流评测对象. Defects4J 属于大规模 Java 程序,截至 2020 年 7 月最新版本的 Defects4J 中共包含 17 个中大规模 Java 程序,共 835 个真实的缺陷版本.但是 Defects4J 的版本仍在不断迭代,且规模不断扩大,因此表 8 中只列出了 Defects4J 更新之前最常用的 6 个程序.

从表 8 中可以看出,目前的多缺陷定位研究采用的评测对象基本上都是集中于强类型编程语言(例如 C++ 和 Java 等).但当前的智能软件系统需要使用机器学习算法,其中大部分借助弱类型编程语言(例如 Python 等)实现.针对这类系统,我们在

后续工作中需要深入分析.例如,多缺陷的干扰是否更为严重,以及现有的多缺陷定位方法是否能够在这类智能软件系统中取得好的定位效果.

## 8.2 缺陷植入分析趋势分析

缺陷植入是指通过代码变异或人工修改等方式在正确的程序中植入缺陷,并且该过程仅适用于人工植入缺陷类型的评测程序.

在来自 SIR 库的评测程序中(如 Siemens 程序集、Unix 程序集等),许多程序仅提供了植入单个缺陷的版本,因此这样的评测程序不适用于多缺陷定位研究.为了解决上述问题,研究人员通过自行制作多缺陷版本的方式构造被测程序<sup>[8,20,60,99-100,102,108,111]</sup>.研究人员把多个来自不同单缺陷版本的缺陷语句植入到同一个评测程序中,在此过程中,如果同一行代码在不同版本中具有不同的缺陷,则在植入过程中,一行代码内仅可以包含其中一个缺陷,因为当前的缺陷定位方法不能在单行语句中识别不同类型的缺陷.此外,由于基于频谱的缺陷定位需要至少一个失败的测试用例,因此在植入的过程中通常会排除没有失败测试用例的缺陷版本.最后,由于只有可执行语句才能影响程序的执行或产生覆盖率信息,因此还会排除位于头文件、变量定义或声明的缺陷.

图 9 给出了不同缺陷植入方式的增长趋势图.图中横轴表示论文的发表年份,纵轴表示对应年份中的不同缺陷类型评测对象的数量统计.其中“植入缺陷”是指该论文只使用了包含人工植入缺陷的评测程序,“真实缺陷”是指该论文只使用了包含真实缺陷的评测程序,“混合”是指该论文同时使用了包含人工植入缺陷的评测程序以及包含真实缺陷的评测程序.

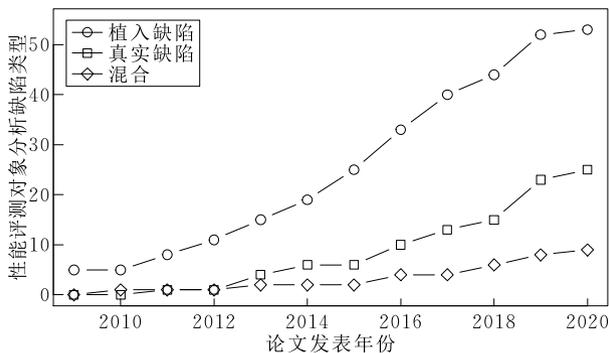


图 9 缺陷植入方式统计

通过对图 9 中评测程序的使用趋势进行分析,可以看出 2019 年以前采用真实缺陷的研究数量增长趋势比较平稳,但 2019 年当年有显著的提升,采

用混合缺陷的研究数量也有小幅度提升. 因此我们对 2019 年前后的论文发表数量进行划分, 并进行深入分析:

(1) 早期(2009~2018 年)的多缺陷定位研究的评测程序缺陷类型统计结果如图 10(a)所示. 其中, 68%的研究使用人工植入缺陷的数据集. 因为这段时期内研究人员主要使用的是 Siemens 测试套件和 Unix 程序集等多个 SIR 库内的评测程序, 这些程序中的缺陷都是人工植入的, 且多缺陷程序的生成通常是将多个单缺陷版本的缺陷注入到同一程序版本中. 例如, 可以将 5 个单缺陷版本中的缺陷同时植入到程序中来创建 5 个缺陷的程序. 目前, 通过注入多个单缺陷版本的缺陷来创建多缺陷程序的方法已被用于许多已有研究工作中<sup>[8,20,60,102,108,111]</sup>;

(2) 近两年(2019~2020 年)的多缺陷定位研究的评测程序缺陷类型统计结果如图 10(b)所示. 从图中可以看出使用包含真实缺陷评测程序的论文数量明显增加. 因为有研究表明, 同一缺陷定位技术在植入缺陷和真实缺陷程序中的表现差异较大, 且原本在使用植入缺陷的研究中获得的结论并不能在真实缺陷上成立<sup>[132]</sup>. 因此, 近年来大量缺陷定位领域的研究工作使用了真实缺陷程序作为评测程序. 其中, 表 8 中排名第四的 Defects4J 数据集成为近些年来被研究人员普遍认可的多缺陷定位研究领域的数据集, 它由规模较大且含有实际缺陷的程序组成.

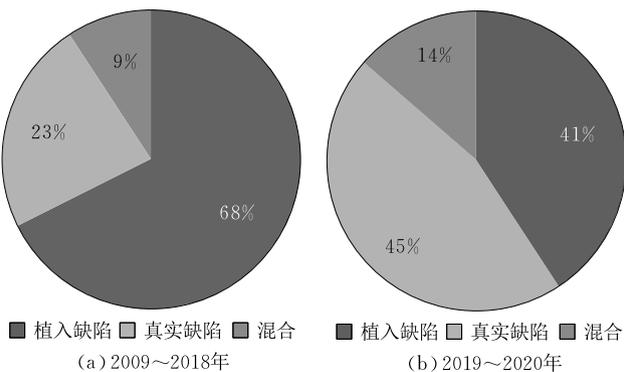


图 10 缺陷植入方式

在多缺陷定位研究中, 植入缺陷经常被用于模拟真实的缺陷行为, 这些植入缺陷由人工编写或插入变异体所生成. 如图 9 所示, 大多数的研究工作都在使用人为植入的缺陷程序, 但是这种趋势可能与所选的评测对象程序(例如 Siemens 测试套件程序)的高使用率有关, 因为表 8 表明 Siemens 测试套件是最常用的评测对象程序, 且 Siemens 测试套件在默认情况下, 程序仅包含单个缺陷<sup>[110]</sup>. 因此, 研究

人员必须人为植入缺陷从而创建多缺陷程序. 研究人员通常使用基于变异的缺陷植入方法, 来创建更多的缺陷程序版本<sup>[20,60,111]</sup>, 基于变异产生的缺陷可用于模拟真实的程序缺陷, 并为缺陷定位研究提供可靠的实验数据. 例如 Gao 和 Wong<sup>[20]</sup>在他们的研究中使用了两类变异算子执行变异操作:

(1) 替换算术、关系、逻辑、递增、递减或赋值运算符(使用同类运算符进行替换), 例如把代码中的 + 变异为一、> 变异为 <、++ 变异为 -- 等;

(2) if 或 while 语句中的决策否定, 例如把 if 语句的判断条件变异为 False.

近些年来, 研究人员认为该过程会为实证研究的结论有效性产生影响, 并引起工业界对缺陷定位技术实用性的质疑. 因此, 越来越多的研究人员采用带有真实缺陷的真实程序进行实证研究, 从而确保实证研究结论的有效性, 并进一步鼓励在工业界中使用缺陷定位技术已经取得的研究成果.

## 9 展望与总结

基于上述分析可以看出, 多缺陷定位问题已经受到了学术界和工业界的广泛关注, 并取得了大量研究成果. 本文对国内外针对多缺陷定位问题的研究成果进行了系统的回顾. 本文给出了当前多缺陷定位问题研究的整体研究框架、研究方法、性能评测指标以及评测对象. 基于本文的总结, 可以看出多缺陷定位问题是当前软件调试中的一个重要研究热点, 具有丰富的理论价值和应用前景. 但通过分析目前的研究现状来看, 尽管研究者已经针对多缺陷定位提出了一些策略, 但是当前方法的结论主要基于中小规模的开源程序, 因而难以适用于实际的大规模复杂软件或工业生产环境软件. 同时, 现有的方法主要基于主流程程序语言(例如 C/C++、Java 或 Python), 而针对其他语言或由多种语言混合编写的程序的研究较少. 为了使多缺陷定位方法实现更高的缺陷定位的精度和效率, 研究人员需要研究和实现更加新颖或增强的策略来解决当前面临的问题和挑战. 论文依次从扩大评测对象的编程语言范围、考虑更多程序信息来源、寻找更多的工业运用场景等多个角度对未来研究工作进行展望.

(1) 进一步研究缺陷定位粒度

当程序中出现多个缺陷语句时, 其中部分缺陷语句可能存在关联. 例如某一函数中出现了大量紧

挨着的缺陷语句,或者程序中多处缺陷语句的缺陷原因相同.在传统的缺陷定位技术中,定位结果排序列表只能简单地罗列各个函数或语句的排名,当粒度较大时(如函数级),开发人员无法得知函数内的缺陷情况;但当粒度较小时(如语句级),开发人员无法通过排序列表得知其他可能由同一原因导致的缺陷语句的位置.

例如,在针对 Defects4J 数据集进行缺陷定位时,现有的工作通常采用函数级别的缺陷定位(即缺陷定位方法会反馈包含缺陷的函数信息)<sup>[45-46,50]</sup>,因为 Defects4J 数据集的单个程序规模较大,使用细粒度的缺陷定位策略(如语句级或变量级)时会导致较差的定位结果.而针对 SIR 数据集进行缺陷定位时,研究人员通常采用语句级的缺陷定位方法<sup>[58,79,100]</sup>,因为 SIR 数据集的单个程序规模适中,使用函数级的缺陷定位策略会使得定位结果范围过大,而语句级则恰到好处.上述缺陷定位方法的粒度虽然没有强制的统一标准,但是可以看出不同的数据集有其适用的缺陷定位粒度级别,而且针对某个数据集的缺陷定位方法不一定适用于其他数据集,从而降低缺陷定位方法的普遍性.

综上所述,当前的缺陷定位方法中,预设不变的粒度有可能会忽略缺陷语句之间存在的关联性,而这个关联性能有效改善开发人员检查语句时的工作体验.因此,相关研究可以尝试从这个角度入手,来提高多缺陷定位的效果.

### (2) 从时间开销角度优化缺陷定位技术

目前已有的绝大多数工作是从缺陷定位技术的定位效果上分析,而很少从时间开销上进行对比.然而,时间开销是软件测试的核心指标之一,因此在评判缺陷定位技术优劣的时候,需要将缺陷定位技术所需的时间开销纳入比较范围.例如基于贝叶斯推理的方法<sup>[108]</sup>和基于变异的方法<sup>[133-134]</sup>,这些方法虽然有较好的缺陷定位效果,但因其存在额外的计算和程序执行,造成较大的时间开销.

当前基于变异的方法的时间开销过大是这类方法的一个核心问题,研究人员从不同角度提出了一些优化方法来减少执行成本<sup>[54,131-135]</sup>.例如 Liu 等人提出了基于变异的缺陷定位方法的约简策略<sup>[132]</sup>,他们通过约简测试用例和变异体的数量,对该缺陷定位方法的整体耗时进行约简.

近年来,基于变异的缺陷定位方法因其显著的时间花费,吸引了众多研究人员为其提出优化策略<sup>[54,131-132]</sup>.然而除了基于变异的缺陷定位方法,其

他的缺陷定位方法也会产生巨大的时间消耗.尤其是基于搜索或基于模型的缺陷定位方法,在面对较大规模的缺陷程序时,其时间复杂度会显著增长.因此,为了进一步提高缺陷定位方法的实用价值,在未来的研究中考虑优化时间开销很有必要.

### (3) 考虑其他编程语言实现的项目

基于表 8 的结果表明,当前大多数多缺陷定位领域的研究仅针对 C 和 Java 编程语言,这些研究工作证实了其所提出方法在所选编程语言上的有效性.TIOBE 统计了三个年份(2020 年、2015 年、2010 年)的编程语言热门程度排名,其中 Java 和 C 语言一直是过去 10 年来最热门的两种编程语言,这也是大多数研究工作针对 Java 和 C 语言进行研究的原因.

然而针对某种语言提出缺陷定位方法之后,该方法将有可能局限于该语言的特性而无法得到拓展.例如众多方法针对编译型语言(如 Java 和 C 语言)提出了改进后的 SBFL 方法,但是它们只能对编译通过的程序进行缺陷定位,因为现有方法只能获取编译通过的 Java 或 C 程序的测试用例执行结果和覆盖信息.但是解释型语言(如 Python 和 JavaScript)则不存在这个局限,因为由这些语言编写的程序即使在语法或句法存在缺陷时也能部分执行,从而获取更多编译型语言无法获取的信息.此时如果不把针对编译型语言提出的缺陷定位先进方法进行改进和拓展,使其能够充分利用解释型语言的特性进一步提高缺陷定位效果,将会非常遗憾.

因此,为了使多缺陷定位的实证研究结论更具有普遍性,且促进论文提出的缺陷定位技术能应用于真实生产环境,研究人员在后续研究中需要考虑更多的编程语言.例如,随着 Python 等脚本语言的流行,针对初级开发人员的脚本语言的缺陷定位需求日益增长. Cosman 等人<sup>[136]</sup>提出一种面向脚本语言的缺陷定位方法,通过学习开发人员过去修复的真实实例,训练得出一套启发式方法,从而实现与人类行为和判断相符的正确修复答案.这种方法有助于协助调试,并可以帮助初级开发人员处理在编程时遇到的各种缺陷.

通过对包含多个缺陷的程序进行实验,结果表明这种方法能有效帮助初级开发人员修复缺陷程序.因此,研究人员可以将包括 Python 在内的更多程序语言作为评测对象,从而使论文中提出的方法适用于其他日益流行的编程语言.

### (4) 考虑更多的软件制品

现有的缺陷定位方法(如 SBFL、MBFL 或基于

切片的缺陷定位)一般使用测试用例的覆盖路径及测试用例的执行结果进行缺陷定位.但是,一些研究人员认为这些信息并不足以进行高精度的多缺陷定位,因此众多的研究工作在其缺陷定位框架中结合了来自被测程序的其他信息<sup>[55,92,137]</sup>.这些额外的信息包括代码详细文本、代码的修改历史、软件运行时输出的系统日志以及开发人员的反馈意见等,结合了这些信息的缺陷定位方法在相关研究的实证研究中具有更高的缺陷定位效率.

例如,在真实的工业软件开发环境或代码开源社区中,存在大量对理解和分析代码有帮助的额外信息(例如项目文档、代码注释、代码的修改历史和用户反馈等),妥善使用这些信息能够帮助开发人员提出更加高效且精准的缺陷定位方法.Wen 等人就通过分析代码的修改历史,识别出在程序迭代更新过程中更有可能包含缺陷的语句,从而进行缺陷定位<sup>[138]</sup>.

因此,未来的工作可以考虑在现有的缺陷定位框架中整合更多程序相关信息,这将有助于提出更高质量的解决方案.

#### (5) 将多缺陷定位与缺陷预测相结合

软件缺陷预测技术能够帮助软件开发人员提前发现程序中潜在的缺陷,并降低检测和修复缺陷所需的开销.现有的软件缺陷预测技术通常使用软件的代码复杂度信息、软件开发过程特征等来构建模型,并使用该模型来预测程序中的缺陷程序模块<sup>[139]</sup>.通过缺陷预测技术,开发人员可以识别出程序中包含缺陷概率更高的代码块,针对这些代码块设置更高的缺陷定位权重,将有助于提高多缺陷定位的效果.

#### (6) 将多缺陷定位与缺陷自动修复结合

程序缺陷自动修复技术能够拓展多缺陷定位技术的应用场景,目前无论是基于缺陷干扰假设的多缺陷定位方法、基于缺陷独立假设的多缺陷定位方法或不基于任何假设的多缺陷定位方法,在缺陷修复的时候都离不开人工介入,当前大部分多缺陷定位技术的最终目的是辅助开发人员进行缺陷修复.因此在基于缺陷干扰假设的多缺陷定位方法的调试过程中,需要开发人员进行缺陷修复后才能进入下一次迭代;在并行调试方法中,更是需要多名开发人员协同工作才能修复程序中的所有缺陷.

例如 Luo 等人<sup>[140]</sup>尝试使用现有的缺陷自动修复工具作为反馈对缺陷定位结果进行优化.具体来说,他们先对缺陷程序进行缺陷定位,然后使用所获得的怀疑度排行指导缺陷修复工具 PraPR<sup>[141]</sup>生成

程序补丁,之后再根据缺陷修复的情况进一步对怀疑度排行进行调整,从而提高缺陷定位的精度.因此,吸收缺陷自动修复领域的最新研究成果,将自动化缺陷定位与缺陷自动修复结合技术进行有效结合值得深入研究.

#### (7) 寻找更多的工业应用场景

当前大部分的实验使用来自 SIR 库或 Defects4J 库内的开源程序进行实验,其中包含人工植入的缺陷和真实的缺陷.但是很少有研究工作使用工业界真实生产环境中程序进行实验,并且很多论文在未来展望中均谈到在后续工作中需要进一步考虑工业界程序或更大规模的程序<sup>[95,103,113]</sup>.因此,如果将真实工业界的大规模程序作为多缺陷定位研究的被测程序,并提出切实可行的方案,这将有利于多缺陷定位技术的推广.除此之外,缺陷定位方法不仅能针对 C、C++ 或 Java 语言帮助开发人员提升测试效率并降低测试成本,研究人员还将该方法应用于语法缺陷规则查找、SQL 语句的缺陷定位等新型应用中.

例如, Raselimo 等人<sup>[81]</sup>将 SBFL 方法应用于查找上下文无关语法中的缺陷规则,他们模拟 SBFL 的原理将测试套件和语法分析器作为输入,收集相应的语法谱(GrammarSpectra),他们将语法谱非正式地定义为成功解析测试套件中的单词时所应用的所有规则的集合,然后得到一个基于怀疑度值的排序列表.针对 SQL 语句进行缺陷定位,现有缺陷定位方法将每个 SQL 语句视为一个程序实体,因此它们无法定位出存在于单个 SQL 语句内部的缺陷元素.为此,Guo 等人<sup>[82]</sup>提出了一种新颖的缺陷定位方法,该方法可以对 SQL 谓词中各个子句内的缺陷进行定位.他们称其为基于免除(Exoneration)的缺陷定位方法,该方法通过免除一些程序元素来识别缺陷所在位置.在后续工作中,还需要寻找更多应用场景.Yilmaz 等人<sup>[142]</sup>针对现代化的可高度配置软件系统提出了调试方法,他们使用分类树分析(Classification Tree Analysis),可对给定的软件配置进行建模,并对配置的正确性(是否会导致软件失效)进行预测,从而对特定条件下的程序缺陷进行准确定位.

如果国内外研究人员能够针对上述挑战提出有效的解决方案,将多缺陷定位方法拓展至更多语言的评测程序,在缺陷定位技术研究中结合更多的被测程序相关信息,并且在工业领域中寻找运用多缺陷定位方法使用的场景,有效提高软件缺陷定位的

精度和执行效率,将有助于减少软件调试成本,对提高软件产品质量起到促进作用。

### 参 考 文 献

- [1] Wong W E, Gao R Z, Li Y H, et al. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 2016, 42(8): 707-740
- [2] Collofello J S, Woodfield S N. Evaluating the effectiveness of reliability-assurance techniques. *Journal of Systems and Software*, 1989, 9(3): 191-195
- [3] Xie X, Chen T Y, Kuo F C, Xu B W. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on Software Engineering and Methodology*, 2013, 22(4): 1-40
- [4] Zheng W, Hu D S, Wang J. Fault localization analysis based on deep neural network. *Mathematical Problems in Engineering*, 2016, 2016(pt. 4): 1-11
- [5] Wong W E, Debroy V, Golden R, et al. Effective software fault localization using an RBF neural network. *IEEE Transactions on Reliability*, 2011, 61(1): 149-169
- [6] Wong W E, Qi Y. BP neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 2009, 19(4): 573-597
- [7] Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization//*Proceedings of the 24th IEEE International Conference on Software Engineering*. Orlando, USA, 2002: 467-477
- [8] Hogerle W, Steimann F, Frenkel M. More debugging in parallel //*Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering*. Naples, Italy, 2014: 133-143
- [9] DiGiuseppe N, Jones J A. On the influence of multiple faults on coverage-based fault localization//*Proceedings of the International Symposium on Software Testing and Analysis*. New York, USA, 2011: 210-220
- [10] Xue X Z, Namin A S. How significant is the effect of fault interactions on coverage-based fault localizations?//*Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Baltimore, USA, 2013: 113-122
- [11] Yu Kai, Lin Meng-Xiang. Advances in automatic fault localization techniques. *Chinese Journal of Computers*, 2011, 34(8): 1411-1422(in Chinese)  
(虞凯, 林梦香. 自动化软件错误定位技术研究进展. *计算机学报*, 2011, 34(8): 1411-1422)
- [12] Chen Xiang, Ju Xiao-Lin, Wen Wan-Zhi, Gu Qing. Review of dynamic fault localization approaches based on program spectrum. *Journal of Software*, 2015, 26(2): 390-412 (in Chinese)  
(陈翔, 鞠小林, 文万志, 顾庆. 基于程序频谱的动态缺陷定位方法研究. *软件学报*, 2015, 26(2): 390-412)
- [13] Wang Ke-Chao, Wang Tian-Tian, Su Xiao-Hong, Ma Pei-Jun. Key scientific issues and state-art of automatic software fault localization. *Chinese Journal of Computers*, 2015, 38(11): 2262-2278(in Chinese)  
(王克朝, 王甜甜, 苏小红, 马培军. 软件错误自动定位关键科学问题及研究进展. *计算机学报*, 2015, 38(11): 2262-2278)
- [14] Zakari A, Lee S P, Abreu R, et al. Multiple fault localization of software programs: A systematic literature review. *Elsevier Information and Software Technology*, 2020, 124: 106312
- [15] DiGiuseppe N, Jones J A. Fault interaction and its repercussions //*Proceedings of the 27th IEEE International Conference on Software Maintenance*. Williamsburg, USA, 2011: 3-12
- [16] Liu C, Zhang X, Han J. A systematic study of failure proximity. *IEEE Transactions on Software Engineering*, 2008, 34(6): 826-843
- [17] Jones J A, Bowring J F, Harrold M J. Debugging in parallel//*Proceedings of the International Symposium on Software Testing and Analysis*. New York, USA, 2007: 16-26
- [18] Morell L J. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 1990, 16(8): 844-857
- [19] Yan X B, Liu B, Li Jianxing. The failure behaviors of multi-faults programs: An empirical study//*Proceedings of the IEEE International Conference on Software Maintenance and Evolution*. Prague, Czech Republic, 2017: 1-7
- [20] Gao R Z, Wong W E. MSeer — An advanced technique for locating multiple bugs in parallel. *IEEE Transactions on Software Engineering*, 2017, 45(3): 301-318
- [21] Debroy V, Wong W E. Insights on fault interference for programs with multiple bugs//*Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering*. Mysuru, India, 2009: 165-174
- [22] Morell L J. A model for code-based testing schemes//*Proceedings of the 5th Annual Pacific Northwest Software Quality Conference*. Portland, USA, 1987: 309-326
- [23] Wang Zan, Fan Xiang-Yu, Zou Yu-Guo, Chen Xiang. Genetic algorithm based multiple faults localization technique. *Journal of Software*, 2016, 27(4): 879-900(in Chinese)  
(王赞, 樊向宇, 邹雨果, 陈翔. 一种基于遗传算法的多缺陷定位方法. *软件学报*, 2016, 27(4): 879-900)
- [24] Abreu R, Zoetewij P, Gemund A J V. An evaluation of similarity coefficients for software fault localization//*Proceedings of the 12th IEEE Pacific Rim International Symposium on Dependable Computing*. Riverside, USA, 2006: 39-46
- [25] Wong W E, Debroy V, Gao R Z, Li Y H. The DStar method for effective software fault localization. *IEEE Transactions on Reliability*, 2013, 63(1): 290-308
- [26] Wong W E, Debroy V, Xu D X. Towards better fault localization: A crosstab-based statistical approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2011, 42(3): 378-396

- [27] Jones J A, Harrold M J. Empirical evaluation of the Tarantula automatic fault-localization technique//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2005: 273-282
- [28] Dean B C, Presly W B, Malloy B A, Whitley A A. A linear programming approach for automated localization of multiple faults//Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 640-644
- [29] Zeller A, Hildebrandt R. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 2002, 28(2): 183-200
- [30] Artho C. Iterative delta debugging. *International Journal on Software Tools for Technology Transfer*, 2011, 13(3): 223-246
- [31] Birch G, Fischer B, Poppleton M. Fast test suite-driven model-based fault localisation with application to pinpointing defects in student programs. *Software and Systems Modeling*, 2019, 18(1): 445-471
- [32] Ma C, Nie C, Chao W, Zhang B. A vector table model-based systematic analysis of spectral fault localization techniques. *Software Quality Journal*, 2019, 27(1): 43-78
- [33] Liu B, Nejati S, Briand L, et al. Localizing multiple faults in Simulink models//Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering. Osaka, Japan, 2016: 146-156
- [34] Wang Y, Huang Z Q, Fang B W, Li Y. Spectrum-based fault localization via enlarging non-fault region to improve fault absolute ranking. *IEEE Access*, 2018, 6: 8925-8933
- [35] Aribi N, Maamar M, Lazaar N, et al. Multiple fault localization using constraint programming and pattern mining//Proceedings of the 29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI). Boston, USA, 2017: 860-867
- [36] Aribi N, Lazaar N, Lebbah Y, et al. A multiple fault localization approach based on multicriteria analytical hierarchy process//Proceedings of the IEEE International Conference on Artificial Intelligence Testing. Newark, USA, 2019: 1-8
- [37] Peng Z D, Xiao X, Hu G W, et al. ABFL: An autoencoder based practical approach for software fault localization. *Information Sciences*, 2020, 510: 108-121
- [38] Zhang Y J, Santelices R. Prioritized static slicing and its application to fault localization. *Journal of Systems and Software*, 2016, 114: 38-53
- [39] Wen Wan-Zhi, Li Bi-Xin, Sun Xiao-Bing, Qi Shan-Shan. A technique of multiple fault localization based on conditioned execution slicing spectrum. *Journal of Computer Research and Development*, 2013, 50(5): 1030-1043(in Chinese)  
(文万志, 李必信, 孙小兵, 齐珊珊. 基于条件执行切片谱的多错误定位. *计算机研究与发展*, 2013, 50(5): 1030-1043)
- [40] Hofer B, Wotawa F. Spectrum enhanced dynamic slicing for better fault localization//Proceedings of the 20th European Conference on Artificial Intelligence. Montpellier, France, 2012: 420-425
- [41] Tu J X, Xie X Y, Chen T Y, Xu B. On the analysis of spectrum based fault localization using hitting sets. *Journal of Systems and Software*, 2019, 147: 106-123
- [42] Parsa S, Vahidi-Asl M, Farzaneh Z. Statistical based slicing method for prioritizing program fault relevant statements. *Computing and Informatics*, 2016, 34(4): 823-857
- [43] Abreu R, Gonzalez-Sanchez A, Gemund A J V. Exploiting count spectra for Bayesian fault localization//Proceedings of the 6th International Conference on Predictive Models in Software Engineering. New York, USA, 2010: 1-10
- [44] Lee J, Kim J, Lee E. Enhanced fault localization by weighting test cases with multiple faults//Proceedings of the 14th International Conference on Software Engineering Research and Practice. Las Vegas, USA, 2016: 116
- [45] Zhang M, Li X, Zhang L M, Khurshid S. Boosting spectrum-based fault localization using PageRank//Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, USA, 2017: 261-272
- [46] Zou D, Liang J, Xiong Y, et al. An empirical study of fault localization families and their combinations. *IEEE Transactions on Software Engineering*, 2021, 47(2): 1-16
- [47] Choi S, Cha S, Tappert C C. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 2010, 8(1): 43-48
- [48] Naish L, Ramamohanarao K, Neelofar N. Multiple bug spectral fault localization using genetic programming//Proceedings of the 24th Australasian Software Engineering Conference. Adelaide, Australia, 2015: 11-17
- [49] Neelofar N, Naish L, Ramamohanarao K. Spectral-based fault localization using hyperbolic function. *Software: Practice and Experience*, 2018, 48(3): 641-664
- [50] Laghari G, Murgia A, Demeyer S. Fine-tuning spectrum based fault localisation with frequent method item sets//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2016: 274-285
- [51] Li X Y, d'Amorim M, Orso A. Iterative user-driven fault localization//Proceedings of the 12th Springer Haifa Verification Conference. Haifa, Israel, 2016: 82-98
- [52] Gong C, Zheng Z, Li W, Hao P. Effects of class imbalance in test suites: An empirical study of spectrum-based fault localization//Proceedings of the 36th IEEE Annual Computer Software and Applications Conference Workshops. Izmir, Turkey, 2012: 470-475
- [53] Zhang L, Yan L F, Zhang Z Y, et al. A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization. *Journal of Systems and Software*, 2017, 129: 35-57

- [54] de Oliveira A A L, Camilo-Junior C G, de Andrade Freitas E N, Vincenzi A M R. FTMES: A failed-test-oriented mutant execution strategy for mutation-based fault localization// Proceedings of the IEEE 29th International Symposium on Software Reliability Engineering, Memphis, USA, 2018: 155-165
- [55] Ghandehari L S, Lei Y, Kacker R, et al. A combinatorial testing-based approach to fault localization. *IEEE Transactions on Software Engineering*, 2018, 46(6): 616-645
- [56] Nie C, Leung H. The minimal failure-causing schema of combinatorial testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2011, 20(4): 1-38
- [57] Niu X, Nie C, Lei J Y, et al. Identifying failure-causing schemas in the presence of multiple faults. *IEEE Transactions on Software Engineering*, 2018, 46(2): 141-162
- [58] Fu W H, Yu H Q, Fan G S, Ji X. Test case prioritization approach to improving the effectiveness of fault localization// Proceedings of the IEEE International Conference on Software Analysis, Testing and Evolution. Kunming, China, 2016: 60-65
- [59] Perez A, Abreu R, Riboira A. A dynamic code coverage approach to maximize fault localization efficiency. *Journal of Systems and Software*, 2014, 90: 18-28
- [60] Yu Z X, Bai C G, Cai K Y. Does the failing test execute a single or multiple faults? An approach to classifying failing tests// Proceedings of the 37th IEEE International Conference on Software Engineering. Florence, Italy, 2015, 1: 924-935
- [61] Yilmaz C, Dumlu E, Cohen M B, Porter A. Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach. *IEEE Transactions on Software Engineering*, 2013, 40(1): 43-66
- [62] Xia X, Gong L, Le T B, et al. Diversity maximization speedup for localizing faults in single-fault and multi-fault programs. *Automated Software Engineering*, 2016, 23(1): 43-75
- [63] Gao R, Wong W E, Chen Z Y, Wang Y B. Effective software fault localization using predicted execution results. *Software Quality Journal*, 2017, 25(1): 131-169
- [64] Zhang X Y, Zheng Z, Cai K Y. Exploring the usefulness of unlabelled test cases in software fault localization. *Journal of Systems and Software*, 2018, 136: 278-290
- [65] Wes M, Rawad A A. Cleansing test suites from coincidental correctness to enhance fault-localization// Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation. Paris, France, 2010: 165-174
- [66] Liu Y, Li M Y, Wu Y H, Li Z. A weighted fuzzy classification approach to identify and manipulate coincidental correct test cases for fault localization. *Journal of Systems and Software*, 2019, 151(5): 20-37
- [67] Kim J, Kim J, Lee E. VFL: Variable-based fault localization. *Information and Software Technology*, 2019, 107: 179-191
- [68] Gong D D, Su X H, Wang T T, et al. State dependency probabilistic model for fault localization. *Information and Software Technology*, 2015, 57(1): 430-445
- [69] Liu C, Yan X F, Fei L, et al. SOBER: Statistical model-based bug localization. *ACM SIGSOFT Software Engineering Notes*, 2005, 30(5): 286-295
- [70] Zhang Z Y, Chan W K, Tse T H, et al. Capturing propagation of infected program states// Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA, 2009: 43-52
- [71] Feyzi F, Parsa S. Infarence: Effective fault localization based on information-theoretic analysis and statistical causal inference. *Frontiers of Computer Science*, 2019, 13(4): 735-759
- [72] Wang X Y, Liu Y M. Fault localization using disparities of dynamic invariants. *Journal of Systems and Software*, 2016, 122: 144-154
- [73] Wang X Y, Liu Y M. Automated fault localization via hierarchical multiple predicate switching. *Journal of Systems and Software*, 2015, 104(6): 69-81
- [74] Wang Xing-Ya, Jiang Shu-Juan, Ju Xiao-Lin, Cao He-Ling. Predicate-level statistical fault localization based on confounding bias mitigation. *Chinese Journal of Computers*, 2017, 40(12): 2671-2687 (in Chinese)  
(王兴亚, 姜淑娟, 鞠小林, 曹鹤玲. 基于混杂偏倚消除的谓词统计错误定位方法. *计算机学报*, 2017, 40(12): 2671-2687)
- [75] Liu Y M, Li B. Automated program debugging via multiple predicate switching// Proceedings of the 24th AAAI Conference on Artificial Intelligence. Atlanta, USA, 2010, 24(1): 327-332
- [76] Liu A S, Li L, Luo J. Automated program debugging for multiple bugs based on semantic analysis// Proceedings of the 5th International Workshop on Structured Object-Oriented Formal Language and Method; Springer. Paris, France, 2015: 86-100
- [77] Neelofar N, Naish L, Lee J, Kotagiri R. Improving spectral-based fault localization using static analysis. *Software Practice and Experience*, 2017, 47(11): 1633-1655
- [78] Xu J, Chan W K, Zhang Z Y, et al. A dynamic fault localization technique with noise reduction for Java programs// Proceedings of the 11th IEEE International Conference on Quality Software. Madrid, Spain, 2011: 11-20
- [79] Xu J, Zhang Z Y, Chan W K, et al. A general noise-reduction framework for fault localization of Java programs. *Information and Software Technology*, 2013, 55(5): 880-896
- [80] Li X, Li W, Zhang Y Q, Zhang L M. DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization // Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, USA, 2019: 169-180

- [81] Raselimo M, Fischer B. Spectrum-based fault localization for context-free grammars//Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering. New York, USA, 2019: 15-28
- [82] Guo Y, Li N, Offutt J, Motro A. Exoneration-based fault localization for SQL predicates. *Journal of Systems and Software*, 2019, 147: 230-245
- [83] Sun S F, Andy P. Properties of effective metrics for coverage-based statistical fault localization//Proceedings of the IEEE International Conference on Software Testing, Verification and Validation. Chicago, USA, 2016: 124-134
- [84] Lucia L, David L, Jiang L X, et al. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 2014, 26(2): 172-219
- [85] Yan X B, Liu B, Wang S H. How negative effects a multiple-fault program can do to spectrum-based fault localization//Proceedings of the IEEE Prognostics and System Health Management Conference. Qingdao, China, 2019: 1-6
- [86] Xia X, Bao L F, Lo D, Li S P. "Automated debugging considered harmful" considered harmful: A user study revisiting the usefulness of spectra-based fault localization techniques with professionals using real bugs from large systems//Proceedings of the IEEE International Conference on Software Maintenance and Evolution. Raleigh, USA, 2016: 267-278
- [87] Heiden S, Grunske S, Kehrer T, et al. An evaluation of pure spectrum based fault localization techniques for large-scale software systems. *Software: Practice and Experience*, 2019, 49(8): 1197-1224
- [88] Perez A, Abreu R, D'Amorim M. Prevalence of single-fault fixes and its impact on fault localization//Proceedings of the IEEE International Conference on Software Testing, Verification and Validation. Tokyo, Japan, 2017: 12-22
- [89] Zhang X Y, Zheng Z. Exploring the characteristics of spectra distribution and their impacts on fault localization//Proceedings of the Evaluation and Assessment in Software Engineering. New York, USA, 2020: 100-109
- [90] Steimann F, Frenkel M. Improving coverage-based localization of multiple faults using algorithms from integer linear programming//Proceedings of the 23rd IEEE International Symposium on Software Reliability Engineering. Dallas, USA, 2012: 121-130
- [91] Wang Y B, Gao R Z, Chen Z Y, Wong W E, Luo B. WAS: A weighted attribute-based strategy for cluster test selection. *Journal of Systems and Software*, 2014, 98: 44-58
- [92] Wei Z, Han B. Multiple-bug oriented fault localization: A parameter-based combination approach//Proceedings of the 7th IEEE International Conference on Software Security and Reliability Companion. Gaithersburg, USA, 2013: 125-130
- [93] Parsa S, Vahidi-Asl M, Asadi-Aghbolaghi M. Hierarchy-debug: A scalable statistical technique for fault localization. *Software Quality Journal*, 2014, 22(3): 427-466
- [94] Cao He-Ling, Jiang Shu-Juan. Multiple-fault localization based on Chameleon clustering. *Acta Electronica Sinica*, 2017, 45(2): 394-400(in Chinese)  
(曹鹤玲, 姜淑娟. 基于 Chameleon 聚类分析的多错误定位方法. *电子学报*, 2017, 45(2): 394-400)
- [95] Zakari A, Lee S P, Hashem IAT. A community-based fault isolation approach for effective simultaneous localization of faults. *IEEE Access*, 2019, 7(7): 50012-50030
- [96] Zakari A, Lee S P. Simultaneous isolation of software faults for effective fault localization//Proceedings of the 15th IEEE International Colloquium on Signal Processing B Its Applications. Penang, Malaysia, 2019: 16-20
- [97] Wang Xing-Ya, Jiang Shu-Juan, Gao Peng-Fei, et al. Fuzzy C-Means clustering based multi-fault localization. *Chinese Journal of Computers*, 2020, 43(2): 206-232(in Chinese)  
(王兴亚, 姜淑娟, 高鹏飞等. 基于模糊 C 均值聚类的软件多缺陷定位方法. *计算机学报*, 2020, 43(2): 206-232)
- [98] Jeffrey D, Gupta N, Gupta R. Fault localization using value replacement//Proceedings of the International Symposium on Software Testing and Analysis. New York, USA, 2008: 167-178
- [99] Jeffrey D, Gupta N, Gupta R. Effective and efficient localization of multiple faults using value replacement//Proceedings of the IEEE International Conference on Software Maintenance. Edmonton, Canada, 2009: 221-230
- [100] Sun X B, Peng X, Li B, et al. IPSETFUL: An iterative process of selecting test cases for effective fault localization by exploring concept lattice of program spectra. *Frontiers of Computer Science*, 2016, 10(5): 812-831
- [101] Li Z, Wu Y H, Liu Y. An empirical study of bug isolation on the effectiveness of multiple fault localization//Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security. Sofia, Bulgaria, 2019: 18-25
- [102] Huang Y Q, Wu J H, Feng Y, et al. An empirical study on clustering for isolating bugs in fault localization//Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops. Pasadena, USA, 2013: 138-143
- [103] Zakari A, Lee S P. Parallel debugging: An investigative study. *Journal of Software: Evolution and Process*, 2019, 31(11): e2178
- [104] DiGiuseppe N, Jones J A. Fault density, fault types, and spectra-based fault localization. *Empirical Software Engineering*, 2015, 20(4): 928-967
- [105] Yan X B, Liu B, Wang S H. An analysis on the negative effect of multiple-faults for spectrum-based fault localization. *IEEE Access*, 2018, 7(7): 2327-2347
- [106] Zheng Y, Wang Z, Fan X Y, et al. Localizing multiple software faults based on evolution algorithm. *Journal of Systems and Software*, 2018, 139: 107-123
- [107] Zakari A, Lee S P, Chong C Y. Simultaneous localization of software faults based on complex network theory. *IEEE Access*, 2018, 6: 23990-24002

- [108] Abreu R, Zoetewij P, Gemund A J V. Spectrum-based multiple fault localization//Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 88-99
- [109] Abreu R, Zoetewij P, Gemund A J V. Localizing software faults simultaneously//Proceedings of the 9th IEEE International Conference on Quality Software. Jeju, Korea, 2009: 367-376
- [110] Abreu R, Zoetewij P, Gemund A J V. Simultaneous debugging of software faults. *Journal of Systems and Software*, 2011, 84(4): 573-586
- [111] Lamraoui S M, Nakajima S. A formula-based approach for automatic fault localization of multi fault programs. *Journal of Information Processing*, 2016, 24(1): 88-98
- [112] He Jia-Lang, Zhang Hong. Application of artificial neural network in software multi-faults location. *Journal of Computer Research and Development*, 2013, 50(3): 619-625 (in Chinese)  
(何加浪, 张宏. 神经网络在软件多故障定位中的应用研究. *计算机研究与发展*, 2013, 50(3): 619-625)
- [113] Wang T T, Wang K C, Su X H. Fault localization by analyzing failure propagation with samples in cloud computing environment. *Journal of Cloud Computing*, 2020, 9(1): 1-13
- [114] Wang T T, Xu J H, Su X H, et al. Automatic debugging of operator errors based on efficient mutation analysis. *Multi-media Tools and Applications*, 2019, 78(21): 29881-29898
- [115] Kochhar P S, Xia X, Lo D, Li S. Practitioners' expectations on automated fault localization//Proceedings of the 25th International Symposium on Software Testing and Analysis. New York, USA, 2016: 165-176
- [116] Student. The probable error of a mean. *Biometrika*, 1908, 6(1): 1-25
- [117] Quade D. Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, 1979, 74(367): 680-683
- [118] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 2005, 10(4): 405-435
- [119] Hutchins M, Foster H, Goradia T, Ostrand T. Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria//Proceedings of the 16th IEEE International Conference on Software Engineering. Sorrento, Italy, 1994: 191-200
- [120] Just R, Jalali D, Ernst M D. Defects4J: A database of existing faults to enable controlled testing studies for Java programs//Proceedings of the International Symposium on Software Testing and Analysis. New York, USA, 2014: 437-440
- [121] Vokolos F I, Frankl P G. Empirical evaluation of the textual differencing regression testing technique//Proceedings of the IEEE International Conference on Software Maintenance. Bethesda, USA, 1998: 44-53
- [122] Dallmeier V, Ludig C, Zeller A. Lightweight defect localization for Java//Proceedings of the 19th Springer European Conference on Object-Oriented Programming. Glasgow, UK, 2005: 528-550
- [123] Apache. Ant. <http://ant.apache.org/>
- [124] GNU. Make. <https://www.gnu.org/software/make/>
- [125] Apache. XML-security. <http://xml.apache.org/security/>
- [126] Apache. Jmeter. <https://jmeter.apache.org/>
- [127] Schuler D, Zeller A. Javalanche: Efficient mutation testing for Java//Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA, 2009: 297-298
- [128] Martinez M, Durieux T, Sommerard R, et al. Automatic repair of real bugs in Java: A large-scale experiment on the Defects4J dataset. *Empirical Software Engineering*, 2017, 22(4): 1936-1961
- [129] Wen M, Chen J J, Wu R X, et al. Context-aware patch generation for better automated program repair//Proceedings of the 40th International Conference on Software Engineering. Gothenburg, Sweden, 2018: 1-11
- [130] Hua J, Zhang M S, Wang K Y, Khurshid S. Towards practical program repair with on demand candidate generation //Proceedings of the 40th International Conference on Software Engineering. New York, USA, 2018: 12-23
- [131] Campos E C, de Almeida Maia M. Common bug-fix patterns: A large-scale observational study//Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Toronto, Canada, 2017: 404-413
- [132] Pearson S, Campos J, Just R, et al. Evaluating and improving fault localization//Proceedings of the 9th IEEE International Conference on Software Engineering. Buenos Aires, Argentina, 2017: 609-620
- [133] Papadakis M, Le T Y. Using mutants to locate "unknown" faults//Proceedings of the 5th IEEE International Conference on Software Testing. Montreal, Canada, 2012: 691-700
- [134] Moon S, Kim Y, Kim M, Yoo S. Ask the mutants: Mutating faulty programs for fault localization//Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation. Cleveland, USA, 2014: 153-162
- [135] Papadakis M, Le T Y. Metallaxis-FL: Mutation-based fault localization. *Software Testing, Verification and Reliability*, 2015, 25(5-7): 605-628
- [136] Liu Y, Li Z, Wang L X, et al. Statement-oriented mutant reduction strategy for mutation based fault localization//Proceedings of the IEEE International Conference on Software Quality, Reliability and Security. Prague, Czech Republic, 2017: 126-137

- [137] Liu Y, Li Z, Zhao R L, Gong P. An optimal mutation execution strategy for cost reduction of mutation-based fault localization. *Information Sciences*, 2018, 422: 572-596
- [138] Li Z, Wang H F, Liu Y. HMER: A hybrid mutation execution reduction approach for mutation-based fault localization. *Journal of Systems and Software*, 2020, 168: 110661
- [139] Wang H F, Du B, He J, et al. IETCR: An information entropy based test case reduction strategy for mutation-based fault localization. *IEEE Access*, 2020, 8: 124297-124310
- [140] Cosman B, Endres M, Sakkas G, et al. PABLO: Helping novices debug Python code through data-driven fault, localization//Proceedings of the 51st ACM Technical Symposium on Computer Science Education. New York, USA, 2020: 1047-1053
- [141] Li Y, Wang S H, Nguyen T N, Van Nguyen S. Improving bug detection via context-based code representation learning and attention-based neural networks//Proceedings of the ACM on Programming Languages. New York, USA, 2019, 3(OOPSLA): 1-30
- [142] Wen M, Chen J J, Tian Y Q, et al. Historical spectrum based fault localization. *IEEE Transactions on Software Engineering*, 2019, 1: 1-1
- [143] Chen Xiang, Wang Li-Ping, Gu Qing, et al. A survey on cross-project software defect prediction methods. *Chinese Journal of Computers*, 2018, 41(1): 254-274(in Chinese)  
(陈翔, 王莉萍, 顾庆等. 跨项目软件缺陷预测方法研究综述. *计算机学报*, 2018, 41(1): 254-274)
- [144] Lou Y, Ghanbari A, Li X, et al. Can automated program repair refine fault localization? A unified debugging approach//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, USA, 2020: 75-87
- [145] Ghanbari A, Benton S, Zhang L. Practical program repair via bytecode mutation//Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, USA, 2019: 19-30
- [146] Yilmaz C, Cohen M B, Porter A A. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 2006, 32(1): 20-34



**LI Zheng**, Ph. D. , professor, Ph. D. supervisor. His main research interests include program testing, source code analysis and maintenance.

**WU Yong-Hao**, Ph. D. candidate. His main research interests include software testing and fault localization.

**WANG Hai-Feng**, Ph. D. candidate. His research interests include software testing, fault localization and software defect prediction.

**CHEN Xiang**, Ph. D. , associate professor. His research interests include software maintenance and software testing.

**LIU Yong**, Ph. D. His research interests include source code analysis, mutation testing and program fault localization.

## Background

Software fault localization is the most time-consuming and laborious step in the software debugging process. The automatic software fault localization method is designed to automatically determine the defect's location in the program without or less human intervention to help developers fix the defect faster.

In the early automated fault localization research, most research works assumed that the program under test contained only one defect. Researchers have proposed many fault localization techniques for single defect programs and achieved good localization results. However, the single defect assumption does not conform to the characteristics of existing software debugging scenarios. There are usually multiple defects in commercial projects or open-source

projects, and even some defects will interfere with each other. Therefore, although some defect locating methods can better locate defects on single defect programs, their locating effects will be significantly reduced on multiple defect programs.

This survey focuses on the software multiple fault localization (MFL) problem, which is more challenging than the traditional software single fault localization problem. Therefore, MFL has gradually become a hot issue in software automatic fault localization and has attracted researchers' widespread attention.

This survey takes the research problem of multi-fault localization as the core and systematically sorts out the related research results. Firstly, the existing multi-fault localization

technology is subdivided into three categories, namely, Defect Interference Hypothesis based MFL (INF-MFL), Defect Independence Hypothesis based MFL (IDP-MFL), and None Hypothesis based MFL (NOH-MFL). The main design ideas and related techniques of each type of method are then summarized in turn, followed by the analysis of the evaluation indicators and evaluation objects frequently used in multi-defect positioning research. Finally, we put forward a prospect for the future research direction of multi-fault localization, including expanding the range of programming languages to be evaluated, considering more software programs, and

discovering more industrial application scenarios.

Besides, we analyzed 87 papers from 2009 to 2020 and conducted a systematic literature review of multiple fault localization research. Our analysis shows that the field is developing rapidly while becoming more and more mature. At the same time, there are a series of challenges for future research.

In addition, this research is funded by the National Natural Science Foundation of China (61902015, 61872026 and 61672085) and the Nantong Application Research Plan (JC2019106).

《计算机学报》