基于模糊 C 均值聚类的软件多缺陷定位方法

王兴亚[1),2)] 姜淑娟[1)] 高鹏飞[1)] 陆 凯[1)] 薄莉莉[1)] 鞠小林[3)] 张艳梅[1),4)]

[1)](中国矿业大学计算机科学与技术学院矿山数字化教育部工程研究中心 江苏 徐州 221116)

[2)](南京大学计算机软件新技术国家重点实验室 南京 210023)

[3)](南通大学计算科学与技术学院 江苏 南通 226019)

[4)](桂林电子科技大学广西可信软件重点实验室 广西 桂林 541004)

摘 要 缺陷间的相互干扰会使程序的频谱信息和运行结果发生变化,进而影响基于频谱信息的缺陷定位方法(SBFL)的有效性.本文对缺陷干扰现象进行了研究,通过分析单缺陷程序与多缺陷程序在缺陷运行、感染和传播过程及程序运行结果间的差异定义了两类缺陷干扰,并根据干扰前后缺陷在互斥子集中的分布变化分析了缺陷干扰对 SBFL 方法有效性的影响.研究结果表明:与特定缺陷无关的失败测试用例是降低 SBFL 方法缺陷定位有效性的主要原因.在此基础上,本文提出了一种基于模糊 C 均值聚类的多缺陷定位方法 FCMFL: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,得到每个缺陷相关的失败测试信息;其次,基于隶属度矩阵加权计算每条语句的可疑度,并通过互斥子集优先级分析确定不同语句集合的检查顺序,最终生成一个语句检查序列指导开发人员进行程序调试.实验结果表明:(1)缺陷干扰会对 SBFL 方法产生影响,降低 SBFL 方法的缺陷定位有效性;(2)FCMFL 方法可以降低多缺陷对 SBFL 方法的影响,提高 SBFL 方法的缺陷定位有效性.

关键词 程序调试;缺陷定位;程序切片;缺陷干扰;模糊聚类

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2020.00206

Fuzzy C-Means Clustering Based Multi-Fault Localization

WANG Xing-Ya[1),2)] JIANG Shu-Juan[1)] GAO Peng-Fei[1)] LU Kai[1)] BO Li-Li[1)]
JU Xiao-Lin[3)] ZHANG Yan-Mei[1),4)]

[1)](*Engineering Research Center of Mini Digitalization of Ministry of Education, School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

[2)](*State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing* 210023)

[3)](*School of Computer Science and Technology, Nantong University, Nantong, Jiangsu* 226019)

[4)](*Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, Guangxi* 541004)

Abstract Debugging is a notoriously tedious, time-consuming, but an essential interactive process in software development and maintenance. When a program fails, developers first determine the regions that are likely to contain faults and then conduct a more serious examination in these regions to locate and fix the fault. It is not always easy to get suspicious areas in a short time, and the quality of the suspicious areas depends on many factors, such as the complexity of the fault and the developer's experience. Spectrum-Based Fault Localization (SBFL) provides a representative technique of narrowing down the possible scope of fault. Generally, a spectrum is composed of coverage information and executing results. Based on a given risk evaluation metric and the collected spectrum information, it can automatically compute the likelihood of containing

收稿日期:2018-07-26;在线出版日期:2019-05-05.本课题得到国家自然科学基金(61673384,61832009,61772260,61502497,61562015)、江苏省博士后科研资助计划(2018K028C)、计算机软件新技术国家重点实验室创新项目(ZZKT2018B02)、广西可信软件重点实验室研究课题(kx201609,kx201532)支持.**王兴亚**,博士,主要研究方向为程序调试、缺陷定位.E-mail:xingyawang@cumt.edu.cn;xingyawang@nju.edu.cn.**姜淑娟**(通信作者),博士,教授,博士生导师,主要研究领域为编译技术、软件工程等.E-mail:shjjiang@cumt.edu.cn.**高鹏飞**,硕士,主要研究方向为软件分析与测试.**陆 凯**,硕士,主要研究方向为缺陷定位.**薄莉莉**,博士,主要研究方向为并发程序分析与验证.**鞠小林**,博士,副教授,主要研究方向为软件分析与测试.**张艳梅**,博士,副教授,主要研究方向为软件分析与测试.

a fault (i.e., suspiciousness) for each statement. Therefore, developers can examine the statements in order of descending suspiciousness to reduce the debugging effort. Given a faulty program that contains only one fault, executing the fault is a necessary, but not a sufficient, condition for program failure, indicating that all failed executions must execute the fault, and the statements observed in all failed executions should be assigned with the highest suspiciousness. However, when the faulty program contains multiple faults, executing one fault is neither a necessary nor sufficient condition for program failure. For a multi-fault program, interferences among the faults may change the spectrum information as well as the executing results, and thus influence the effectiveness of SBFL. This paper investigates fault interference, which is defined by the comparison between the executing processes (i.e., execution, infection, propagation, result) of the single-fault program and the ones of multi-fault program. To accurately describe the execution condition and the infection & propagation condition for each fault in a multi-fault program, we propose the E-IP model (Execution-Infection Propagation) and use it to determine the interference between the multiple faults. Then, we provide an in-depth analysis of the influence of fault interference on SBFL. Our research shows that the set of failed test cases, which are irrelevant to a specific fault, is the main reason to decrease the effectiveness of SBFL. Moreover, to mitigate the negative influence of fault interference on SBFL, we propose FCMFL (Fuzzy C-Means Clustering based Multi-Fault Localization). First, using a Fuzzy C-Means clustering algorithm, we estimate the membership degree of each failed test cases in each fault and use a matrix to record the membership relations. Second, based on this matrix, we conduct a weighted calculation to estimate the suspiciousness of each statement and use a priority analysis to determine the check order of the disjoint subsets. Finally, a suspicious ranked list of statements in descending order is available to developers for debugging a program. Our empirical study on 12 Java programs (i.e., 6 Siemens programs and 6 SIR programs), together with 123 single-faulty versions and 3410 multi-faulty versions, shows that: (1) fault interference influences SBFL and decreases the effectiveness of SBFL; (2) FCMFL can effectively mitigate the interference, and significantly improve the effectiveness of SBFL.

Keywords　program debugging; fault localization; program slicing; fault interference; fuzzy clustering

1　引　言

当程序运行失败时，开发人员需要通过程序调试来理解程序失败的原因，进而定位并修复缺陷，确保软件可以正确运行[1]. 缺陷定位是程序调试过程中最为耗时和费力的过程[2]. 传统的缺陷定位方法一般采用设置断点、查看状态、输出日志等方式逐步分析、定位缺陷，整个过程效率低下，难以满足现代大型复杂软件的程序调试需求[3-4]. 因此，研究人员提出了多种缺陷定位方法来帮助开发人员理解缺陷发生的环境、识别缺陷的可疑位置，从而提高程序调试的效率和精度. 其中，基于频谱的缺陷定位方法(Spectrum-Based Fault Localization，SBFL)是一类主要的缺陷定位方法[5]，该方法通过分析程序在不同输入下覆盖信息和运行结果的差异来度量语句出错的可能性，帮助开发人员快速定位缺陷位置.

尽管在实证研究中，SBFL 已经取得了较好的缺陷定位效果，但是应用到实践中还存在一些问题[3-4]，其中一个就是多缺陷问题. 在理论分析和实证研究中，研究人员通常假设程序只包含单个缺陷，并采用单缺陷程序来验证 SBFL 的有效性[6]. 此时，程序中所有的运行失败均由某个特定的缺陷引发，语句覆盖信息的差异可以有效地反映语句出错的可能性. 然而，真实软件中缺陷的数量、位置和相互影响关系是未知的，程序的运行过程和运行结果会因其他缺陷的影响而发生改变. 此时，程序运行失败不再由某个特定的缺陷引发，语句覆盖信息的差异难

以有效地反映语句出错的可能性.因此,在多缺陷环境下直接应用 SBFL 难以取得理想的缺陷定位效果.

部分研究人员对多缺陷定位问题进行了研究.文献[1,7-13]采用聚类(如分层聚类[1,7]、C 均值聚类[8-10,13])、分类[11]、线性规划[12]等技术对不同程序实体(如语句、谓词、运行轨迹)的覆盖信息进行分析,用以实现对失败测试用例进行划分,期望获得与每个缺陷相关的测试用例来减少缺陷干扰对 SBFL 缺陷定位有效性的影响.然而,缺陷通常不是相互独立的,运行失败可能由多个缺陷共同引发.文献[14]将进化优化思想引入到多缺陷定位中,将多缺陷定位问题转化为语句的组合优化问题,并采用遗传算法进行求解,取得了较好的效果.然而该方法并未考虑缺陷间的相互干扰,语句组合可疑度的高低难以有效反映其出错的可能性.文献[15]分析了多缺陷对程序运行结果的干扰,但并未分析多缺陷对程序运行过程和缺陷定位有效性的影响,也未提出相应的多缺陷定位方法.文献[13]通过实证研究分析了多缺陷对基于覆盖的缺陷定位方法的影响,但也未提出相应多缺陷定位方法.因此,当前缺乏多缺陷对 SBFL 影响原因的系统性分析,也缺少相应的多缺陷定位方法.

针对上述问题,本文提出了基于模糊 C 均值聚类的多缺陷定位方法(Fuzzy C-Means Clustering based Multi-Fault Localization,FCMFL).首先,本文对缺陷干扰进行定义,并分析缺陷干扰对 SBFL 缺陷定位有效性的影响.同时,在构建 E-IP(Execution-Infection & Propagation)模型对程序运行过程进行抽象的基础上,根据缺陷增加后程序在运行过程和运行结果上的变化定义 E-IPI(E-IP Interference)和 RI(Result Interference)等两类干扰,进而通过干扰前后缺陷在程序互斥子集间的分布分析缺陷干扰对 SBFL 的影响.分析结果表明:与特定缺陷无关的失败测试用例是降低 SBFL 方法缺陷定位有效性的主要原因.其次,根据干扰影响分析结果,给出 FCMFL 方法的基本框架和具体步骤.针对给定的测试用例,收集程序的语句覆盖信息和运行结果.对于每个失败测试用例,以与程序运行失败相关的变量为切片准则进行动态程序切片,获取动态切片结果,接着通过模糊 C 均值(Fuzzy C-Means,FCM)聚类分析各个失败测试用例与每个缺陷的隶属关系,基于隶属度矩阵加权计算语句的可疑度,并通过互斥子集优先级分析确定不同语句集合的检查次序.最终,生成一个语句检查序列来指导开发人员进行程序调试.为了验证FCMFL的有效性,我们在 Siemens 程序[16]和 SIR 程序[17]上开展缺陷定位对比实验.实验结果表明:FCMFL 可以减少缺陷干扰的负面影响,有效提高 SBFL 方法的缺陷定位有效性.

论文的主要贡献如下:

(1)提出了描述程序运行过程的 E-IP 模型,简化了复杂的程序运行表示,便于开展缺陷干扰分析.

(2)在定义 E-IPI 干扰和 RI 干扰的基础上,分析了缺陷干扰对 SBFL 方法的影响.

(3)根据干扰影响分析结果,进一步提出一种基于 FCM 聚类的多缺陷定位方法 FCMFL.在失败执行的动态切片上进行模糊聚类,分析失败执行在不同缺陷的隶属度计算可疑度从而生成语句检查序列.

(4)设计并实现 FCMFL 方法,以 Siemens 数据集和六个 SIR 程序作为实验对象,通过实验验证了该方法的有效性.

本文第 2 节给出本文方法的技术背景;第 3 节和第 4 节详细分析缺陷间的干扰效应以及缺陷干扰对 SBFL 方法的影响;第 5 节首先描述 FCMFL 方法的整体流程,然后对每个处理步骤进行了介绍;第 6 节使用 Siemens 数据集和六个 SIR 程序对本文研究进行实验分析;第 7 节介绍缺陷定位等相关工作;最后总结全文并讨论进一步的研究工作.

2 技术背景

2.1 基于频谱的缺陷定位方法

基于频谱的缺陷定位方法(简称 SBFL 方法)通过度量每个程序实体出错的可能性(可疑度)来帮助开发人员定位缺陷[3-4].程序实体可以是语句[18-21]、谓词[22]、方法[23]等.不失一般性,本文选择语句作为本文研究的程序实体.通常,SBFL 方法包含两个步骤:测试信息收集和可疑度度量.

给定一个包含 m 条语句的程序 PG 和一个包含 n 条测试用例的测试用例集 TS.依据 TS 运行 PG 并记录程序的运行时信息,可以得到程序在不同测试用例下的覆盖信息和运行结果.其中,程序覆盖信息可以表示为一个整型矩阵 $\boldsymbol{M}_{cover}=\{c_{i,j}\mid i\in[1,m],j\in[1,n]\}$.若语句 s_i 被测试用例 t_j 运行,元素 $c_{i,j}$ 的值为 1;若语句 s_i 未被测试用例 t_j 运行,元素 $c_{i,j}$ 的值为 0.程序运行结果可以表示为一个整型向量 $\boldsymbol{V}_{result}=\{r_j\mid j\in[1,n]\}$.若测试用例 t_j 运行失败,元素 r_j 值为 1;若测试用例 t_j 运行成功,元素 r_j 值为

0. 基于 $\boldsymbol{M}_{cover}$ 和 $\boldsymbol{V}_{result}$，可将语句 s_i 相关的覆盖信息统计数据表示为一个四元组 $N(s_i)=\langle n_{cp}(s_i), n_{cf}(s_i), n_{up}(s_i), n_{uf}(s_i)\rangle$，其中，$n_{cp}(s_i)$ 和 $n_{cf}(s_i)$ 分别表示在覆盖 s_i 的前提下，运行成功和运行失败的测试用例数目；$n_{up}(s_i)$ 和 $n_{uf}(s_i)$ 分别表示在未覆盖 s_i 的前提下，运行成功和运行失败的测试用例数目. 各个 SBFL 方法的主要区别主要在于采用了不同的可疑度度量方法[3]. 图 1 给出了 4 种可疑度度量方法.

$$\text{Tarantula}=\frac{n_{cf}/n_f}{n_{cf}/n_f+n_{cs}/n_s}$$

(a) Tarantula

$$\text{Ochiai}=\frac{n_{cf}}{\sqrt{(n_{cf}+n_{uf})(n_{cf}+n_{cs})}}$$

(b) Ochiai

$$\text{Naish}=\begin{cases}-1, & \text{if } n_{cf}<n_f;\\ n_s-n_{cs}, & \text{if } n_{cf}=n_f\end{cases}$$

(c) Naish

$$\text{Wong}=n_{cf}$$

(d) Wong

图 1　4 个可疑度度量公式

其中，Tarantula[18] 和 Ochiai[19] 是两种经典的可疑度度量方法，Naish[20] 和 Wong[21] 则是 Xie 等人[6] 理论证明最优的可疑度度量方法. 所有语句度量结束后，按照可疑度将语句降序排列，最终生成一个可疑度语句序列指导开发人员定位缺陷位置.

2.2　动态程序切片

程序切片是源程序所包含语句的一个子集，该集合中的元素会通过程序内部的控制依赖和数据依赖关系直接或间接影响程序中某条语句中的单个或数个变量[24]. 在进行缺陷定位时，可采用程序切片技术识别与程序执行失败相关的语句，去除与程序执行失败无关的部分来缩小缺陷的搜索范围，从而起到降低缺陷定位代价的作用.

根据切片过程对程序输入的依赖程度，可以将程序切片分为静态切片和动态切片：静态切片不考虑程序输入，分析了程序中所有可能的执行路径，包含了所有与兴趣变量相关的语句；动态切片考虑程序具体的输入，分析了程序的某一条执行路径，包含了该执行中与兴趣变量相关的语句[25]. 与静态切片相比，动态切片充分利用了给定输入下程序的运行时信息，因而切片结果范围更小、更精确.

在进行程序切片时，程序切片结果需要基于一定的准则来进行计算. 切片准则不同，得到的切片结果也不尽相同. 动态切片准则 DSC 表示为一个三元组 $\langle t, s, V\rangle$. 其中，t 是切片对象的一个测试输入，s 是切片对象中的一个兴趣点（对应一条语句或一个语句块），V 是 s 中变量集合的一个子集.

进一步，以表 1 中 compute 程序为例来说明动态切片与语句覆盖的区别以及不同切片准则对动态切片结果的影响. compute 程序以三个整型变量 x、y、z 作为输入，用于计算它们的中位数 mid 和求和 sum 以及平均数 ave. 该程序包含两个缺陷 f_1 和 f_2，分别存在于语句 s_7 和 s_{10} 中，f_1 和 f_2 均可造成 mid 计算出错. 为验证程序的正确性，我们设计了 $t_1\sim t_4$ 四个测试用例进行测试. 对于每个测试用例 t_i，$stmt$ 列给出了程序在 t_i 下的覆盖结果，$slice_1$ 列给出了程序在 $\langle t_i, s_{14}, \{mid, sum, ave\}\rangle$ 准则下的动态切片结果，$slice_2$ 列给出了程序在 $\langle t_i, s_{14}, \{mid\}\rangle$ 准则下的动态切片结果. 对于每一列，值为“●”表示语句被覆盖或包含在切片中. 同时，表 1 还给出程序在不同测试用例下的执行结果：执行成功用 Passed 表示，执行失败用 Failed 表示.

表 1　程序 compute 在测试用例 $t_1\sim t_4$ 下的覆盖信息与动态切片信息

		$t_1=(1,2,3)$			$t_2=(1,1,4)$			$t_3=(3,2,1)$			$t_4=(2,1,3)$		
	compute(int x, int y, int z){	$stmt$	$slice_1$	$slice_2$	$stmt$	$slice_1$	$slice_2$	$stmt$	$slice_1$	$slice_2$	$stmt$	$slice_1$	$slice_2$
s_1	int $mid=z$;	●			●			●			●		
s_2	if ($y<z$)	●	●	●	●	●	●	●	●	●	●	●	●
s_3	if ($x<y$)	●	●	●	●	●	●				●	●	●
s_4	$mid=y$;	●	●	●									
s_5	else if ($x<z$)				●	●	●				●	●	●
s_6	$mid=y$; //**Fault f_1: $mid=x$**				●	●	●				●	●	●
s_7	else							●	●	●			
s_8	if ($x>y$)							●	●	●			
s_9	$mid=z$; //**Fault f_2: $mid=y$**							●	●	●			
s_{10}	else if ($x>z$)												
s_{11}	$mid=x$;												
s_{12}	int $sum=x+y+z$;	●	●		●	●		●	●		●	●	
s_{13}	double $ave=sum/3$;	●	●		●	●		●	●		●	●	
s_{14}	print(mid, sum, ave);	●	●	●	●	●	●	●	●	●	●	●	●
}	Pass/Fail	Passed			Passed			Failed			Failed		

对比 $stmt$ 列与 $slice_1$ 列可以发现，语句覆盖包含了程序每次执行所涉及的语句，同时也包含了与程序输出结果无关的语句，而动态切片可以去除这部分无关语句(如 s_1)．对比列 $slice_1$ 与列 $slice_2$ 可以发现，以失败相关变量(如 mid)作为切片准则所得到的切片结果范围更小，因而可以缩小缺陷定位的范围，来帮助程序调试人员更快地更有效地判断缺陷位置．

2.3 模糊 C 均值聚类

聚类分析是一种无监督的学习过程，可以在没有任何先验信息的指导下，根据样本之间的相似性，从数据集中发现潜在的相似模式，自动完成样本对象的划分[26]．根据划分结果表示的不同，可将聚类分析技术分为硬聚类和软聚类[26]等两种聚类形式．其中，硬聚类方法的聚类结果界限分明，它强制地将每个对象互斥的分配到某一个聚类子集中；软聚类方法的聚类结果界限则存在中介性，它允许一个对象同时隶属于多个聚类子集．

模糊 C 均值聚类是一类典型的软聚类方法，该方法将经典的 C 均值(C-Means，CM)聚类方法[27]通过模糊集理论推广到模糊情形中，从而可以有效说明每个对象与聚类子集间的隶属关系[28]．

进一步，通过算法 1 来具体描述 FCM 方法，该算法以聚类数目 c、数据集合 D、停止阈值 ε、模糊因子 m 和最大迭代次数 $b_{\max}$ 作为输入，以聚类中心集合 $\boldsymbol{V}$ 和隶属度矩阵 $\boldsymbol{U}$ 作为输出．

算法 1. 模糊 C 均值聚类算法．

输入：聚类数目 c，数据集合 D，停止阈值 ε，模糊因子 m，最大迭代次数 $b_{\max}$

输出：聚类中心集合 $\boldsymbol{V}$，隶属度矩阵 $\boldsymbol{U}$

1. $b \leftarrow 0$
2. Initialize cluster centers $\boldsymbol{V}^b=\{v_1,v_2,\cdots,v_c\}$ and membership matrix $\boldsymbol{U}^b=\{\mu_{i,j}\mid i\in[1,c],j\in[1,n]\}$
3. REPEAT
4. FOR ALL $d_j\in D=\{d_1,d_2,\cdots,d_n\}$ DO
5. Compute the membership degree of d_j for each cluster center in $\boldsymbol{V}^b$, and update $\boldsymbol{U}^b$
6. END FOR
7. Compute and update $\boldsymbol{V}^b$ using $\boldsymbol{U}^b$ and D
8. $b \leftarrow b+1$
9. UNTIL $\mathrm{dis}(\boldsymbol{V}^{b-1},\boldsymbol{V}^b)\leq\varepsilon$ OR $b>b_{\max}$
10. Output $\boldsymbol{V}^b$ and $\boldsymbol{U}^b$

$$\mu_{i,j}=\left(\sum_{k=1}^{c}\left(\frac{dis(d_j,v_i)}{dis(d_j,v_k)}\right)^{\frac{2}{m-1}}\right)^{-1} \tag{1}$$

$$\mu_{i,j}=\begin{cases}1, & \text{if } dis(d_j,v_i)=0;\\ 0, & \text{if } dis(d_j,v_i)>0\end{cases} \tag{2}$$

$$v_i=\frac{\sum_{k=1}^{n}(\mu_{i,k})^m\times d_k}{\sum_{k=1}^{n}(\mu_{i,k})^m} \tag{3}$$

FCM 算法具体过程如下：首先，在 D 中随机选择 c 个对象作为初始聚类中心 $\boldsymbol{V}$(行 2)．其次，计算 D 中每个元素 d_j 在各个类别 c_i 中的隶属度 $\mu_{i,j}$，同时更新隶属度矩阵 $\boldsymbol{U}$(行 4～行 6)．在不同情况下，$\mu_{i,j}$ 的计算方法不同：(a)若满足 $\forall i\in[1,c],dis(d_j,v_i)>0$，$\mu_{i,j}$ 的值用式(1)计算；(b)若满足 $\exists i\in[1,c],dis(d_j,v_i)=0$，$\mu_{i,j}$ 的值用式(2)计算．其中，$dis(d_j,v_i)$ 表示数据对象 d_j 与聚类中心 v_i 的欧式距离．该公式也可用于表示两个数据对象或两个聚类中心的欧式距离．再次，计算并更新聚类中心 $\boldsymbol{V}$(行 7)．对于 $\boldsymbol{V}$ 中的每个聚类中心，根据新生成的隶属度矩阵 $\boldsymbol{U}$ 通过式(3)重新计算其位置．最后，判断聚类结果是否满足预设条件(行 9)．通过停止阈值 ε 和最大迭代次数 $b_{\max}$ 等两个方面来进行判断：(1)当前聚类中心集合 $\boldsymbol{V}^b$ 与前一次迭代的聚类中心集合 $\boldsymbol{V}^{b-1}$ 差异不大(变化幅度小于 ε)，则认为已经达到最优聚类结果；(2)当前迭代次数高于 $b_{\max}$，则认为应当结束聚类．如果聚类过程中满足上述条件之一，即可停止聚类．此时，FCM 聚类结束并输出聚类结果 $\boldsymbol{V}$ 和 $\boldsymbol{U}$．

在聚类完成后，通常需要对聚类结果的质量进行评价．根据是否存在可用基准，一般可将聚类评价方法分为外在方法和内在方法两类[26]：外在方法属于有监督方法，该类方法通过同质性、完全性、小簇保持性等标准来比较聚类结果和基准结果，从而评判聚类结果的优劣；内在方法属于无监督方法，该类方法通过聚类子集的分离情况来评判聚类结果的优劣．

$$XB(\boldsymbol{U},\boldsymbol{V},c)=\frac{\sum_{i=1}^{c}\sum_{j=1}^{n}(\mu_{i,j}^m\times dis(x_j,v_i))}{n\times\min(dis(v_i,v_j))},\ i\neq j \tag{4}$$

Xie-Beni 系数是一种代表性的内在评估方法[29]，该系数考虑数据集的几何结构和隶属关系，因此适用于评价模糊聚类方法的有效性．如式(4)所示，Xie-Beni 系数是一个比值型度量指标，该系数用各样本到聚类中心的距离之和(分子)表示类内的紧致性，用所有聚类中心之间的最小距离(分母)表示类间分离性．因此，该系数越大，聚类结果越差；该系数越小，聚类结果越好．

3　干扰效应分析

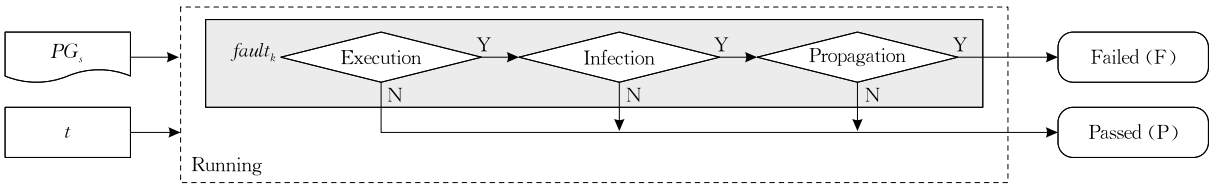
3.1　E-IP 模型

程序在运行时会产生大量的对象和复杂的逻辑关系.开发人员在进行程序理解和程序调试时需要花费大量的精力来收集和分析这类信息.对此,研究人员提出了一系列模型(如 PIE 模型[30-31])来简化和抽象程序的运行过程,用于减轻程序理解和程序调试的负担.

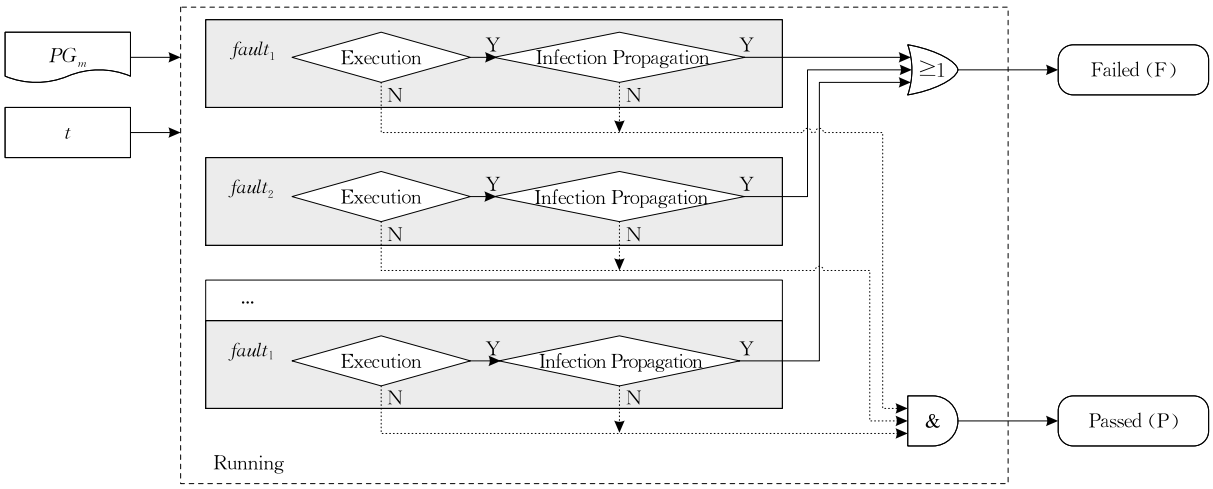
$$PIE(PG,t)=\begin{cases}\langle\rightarrow E,\rightarrow I,\rightarrow P\rangle, & \Rightarrow R(PG,t)=P;\\ \langle E,\rightarrow I,\rightarrow P\rangle, & \Rightarrow R(PG,t)=P;\\ \langle E,I,\rightarrow P\rangle, & \Rightarrow R(PG,t)=P;\\ \langle E,I,P\rangle, & \Rightarrow R(PG,t)=F\end{cases}\tag{5}$$

Voas 提出的 PIE 模型说明了程序运行失败所要满足的 3 个条件[30]:(1) Execution(E),运行了缺陷语句;(2) Infection(I),程序产生了错误状态;(3) Propagation(P),错误状态传播到程序输出.用 $R(PG,t)$ 表示程序 PG 在测试用例 t 下的运行结果.若 PG 运行成功,$R(PG,t)$ 值为 P,否则 $R(PG,t)$ 值为 F.如式(5)所示,当 Execution、Infection、Propagation 都满足时,程序运行失败,否则程序运行成功.需要注意的是,Execution、Infection、Propagation 不能任意组合,满足前一个条件是满足后一个条件的基础.

对于一个单缺陷程序 PG_s,PIE 模型可以有效地描述 PG_s 中缺陷 f_k 与程序运行结果间的关系.图 2(a)描述了程序 PG_s 在 PIE 模型下的运行过程与运行结果.可以看到,当关于缺陷 f_k 的 3 个条件(Execution、Infection 和 Propagation)都满足时,PG_s 运行失败(Failed);否则,PG_s 运行成功(Passed).对于多缺陷程序 PG_m,使用 PIE 模型同样可以描述所有缺陷 FS 与程序运行结果的关系.然而,该模型并不适用于描述 FS 中每个缺陷与运行结果的关系.对此,我们定义了一个新的模型(E-IP 模型)来对程序的运行过程和运行结果进行建模,该模型记录程序在一次运行时的运行结果以及每个缺陷的运行状态,因此适用于描述单缺陷程序和多缺陷程序的运行过程.



(a) 单缺陷程序PG_s



(b) 多缺陷程序PG_m

图 2　程序运行过程与运行结果

定义 1.　E-IP 模型.给定测试用例 t 和包含缺陷集 FS 的程序 PG,根据 PG 在运行 t 时产生的测试信息构建 E-IP 模型.$E\text{-}IP(PG,t)$ 是一个集合,集合中每个元素表示缺陷 f_k 的 E-IP 过程,其中 $f_k\in FS$.E-IP 过程 $E\text{-}IP\text{-}F(PG,t,f_k)$ 表示缺陷 f_k 在测试用例 t 下的运行时状态,包括 Execution:$E(PG,t,f_k)$ 和 Infection & Propagation:$IP(PG,t,f_k)$.如果 f_k 被测试用例 t 运行,$E(PG,t,f_k)$ 的值为 E,

否则值为 $\neg E$. 如果运行 f_k 导致程序产生缺陷状态且该状态被传播到程序输出位置，$IP(PG,t,f_k)$ 的值为 IP，否则值为 $\neg IP$.

$$E\text{-}IP(PG,t)=\{E\text{-}IP\text{-}F(PG,t,f_k)\mid f_k\in FS\} \quad (6)$$

$$E\text{-}IP\text{-}F(PG,t,f_k)=\langle E(PG,t,f_k),IP(PG,t,f_k)\rangle \quad (7)$$

E-IP 模型记录了 FS 中所有缺陷的运行时状态，即对于 FS 中的每一个缺陷 f_k，E-IP 模型分析了 f_k 的运行时状态（包括 Execution、Infection、Propagation）. 由于 Infection 对 PG 的运行结果以及 f_k 的覆盖信息无直接影响，建模时可以将 Infection 和 Propagation 一并考虑. 需要注意的是，$E\text{-}IP\text{-}F(PG,t,f_k)$ 不能任意取值，满足条件 Infection 和 Propagation 需要预先满足条件 Execution. 因此，$E\text{-}IP\text{-}F(PG,t,f_k)$ 取值包括以下 3 种：$\langle\neg E,\neg IP\rangle$、$\langle E,\neg IP\rangle$ 和 $\langle E,IP\rangle$.

进一步地，我们使用图 2(b) 来说明 E-IP 模型. 对于多缺陷程序 PG_m，该图描述了 FS 中每个缺陷的运行时状态以及 PG_m 在测试用例 t 下潜在的运行结果. 图 2(b) 表明：(1) 当 PG_m 运行失败时 ($R(PG_m,t)=F$)，FS 中至少存在一个缺陷 f_k 被 t 运行产生缺陷状态且该状态被传播到程序输出位置，即 $E\text{-}IP\text{-}F(PG_m,t,f_k)=\langle E,IP\rangle$；(2) 当 PG_m 运行成功时 ($R(PG_m,t)=P$)，对于 FS 中每个缺陷 f_k，f_k 未被 t 运行或 f_k 被 t 运行产生缺陷状态且该状态未被传播到程序输出位置，即 $E\text{-}IP\text{-}F(PG_m,t,f_k)=\langle\neg E,\neg IP\rangle\oplus\langle E,\neg IP\rangle$. 可以看到，E-IP 模型清楚地描述了每个缺陷与程序运行结果的关系. 这表明 E-IP 模型不仅适用于描述多缺陷程序，也适用于描述单缺陷程序.

3.2 缺陷干扰

当程序 PG 包含单个缺陷时，可以通过 E-IP 模型推断其在测试用例 t 下的运行结果. 如式 (8) 所示，当 $E\text{-}IP\text{-}F(PG,t,f_k)$ 的值为 $\langle\neg E,\neg IP\rangle$ 或 $\langle E,\neg IP\rangle$ 时，PG 运行成功；当 $E\text{-}IP\text{-}F(PG,t,f_k)$ 的值为 $\langle E,IP\rangle$ 时，PG 运行失败. 然而，当 PG 包含更多的缺陷时，式 (8) 不再成立，且 f_k 的 E-IP 过程和程序运行结果可能会发生变化. 例如，引入的缺陷会影响程序的控制流导致 f_k 不再被运行，或是 $E\text{-}IP\text{-}F(PG,t,f_k)$ 值为 $\langle\neg E,\neg IP\rangle$ 时 PG 依然运行失败. 由此，缺陷间至少存在两类干扰：其中一类干扰为新引入的缺陷导致程序的控制流、数据流发生变化，由此使得原缺陷的 E-IP 过程发生改变；另外一类则是新引入的缺陷对程序运行结果产生影响，由此使得程序的运行结果发生改变. 可以看到，第 1 类干扰可能会改变缺陷 f_k 的运行过程，因此影响 f_k 的覆盖信息；第 2 类干扰会影响程序的运行结果. 两类干扰都会对程序的频谱信息造成影响.

$$E\text{-}IP\text{-}F(PG,t,f_k)=\begin{cases}\langle\neg E,\neg IP\rangle, & \Rightarrow R(PG,t)=P;\\ \langle E,\neg IP\rangle, & \Rightarrow R(PG,t)=P;\\ \langle E,IP\rangle, & \Rightarrow R(PG,t)=F\end{cases} \quad (8)$$

定义 2. E-IPI 干扰 (E-IP Interference，E-IPI). 给定测试用例 t，若缺陷 f_k 在单缺陷程序 PG_s 上的 E-IP 过程不同于其在多缺陷程序 PG_m 上的 E-IP 过程，即 $E(PG_s,t,f_k)\neq E(PG_m,t,f_k)$ 或 $IP(PG_s,t,f_k)\neq IP(PG_m,t,f_k)$，则认为缺陷间存在 E-IPI 干扰.

定义 3. RI 干扰 (Result Interference，RI). 给定测试用例 t，若单缺陷程序 PG_s 的运行结果不同于其在多缺陷程序 PG_m 上的结果，即 $R(PG_s,t,f_k)\neq R(PG_m,t,f_k)$，则认为缺陷间存在 RI 干扰.

对于造成 E-IPI 干扰的测试用例，命名为 E-IPI 测试用例；对于造成 RI 干扰的测试用例，命名为 RI 测试用例.

表 2 给出了不同干扰下 f_k 的 E-IP 过程及程序运行结果的变化. 其中，列 1～2、3～4、5～6、7～8 分别给出了在没有干扰、仅 E-IPI、仅 RI 以及同时 E-IPI 和 RI 下的 E-IP 过程及程序运行结果. 表中数据表明：(1) 在缺陷干扰下，失败运行中 f_k 的 E-IP 过程不再是唯一的. 这是因为当程序中包含多个缺陷时，f_k 不再是唯一导致程序失败的原因，其他缺陷同样也可导致程序运行失败，此时 f_k 的 E-IP 过程可以为 $\langle\neg E,\neg IP\rangle$、$\langle E,\neg IP\rangle$ 和 $\langle E,IP\rangle$；(2) 当 f_k 在单缺陷程序 PG_s 上的 E-IP 过程为 $\langle E,IP\rangle$ 时，RI 干扰必然伴随着 E-IPI 干扰. 根据式 (8)，$E\text{-}IP\text{-}F(PG_s,t,f_k)$ 值为 $\langle E,IP\rangle$ 时程序运行失败. 如果存在 RI 干扰，程序则由失败变为成功，$IP(PG_m,t,f_k)$ 必定不能为 IP. 此时，$IP(PG_s,t,f_k)\neq IP(PG_m,t,f_k)$，因此 RI 干扰必然伴随着 E-IPI 干扰；(3) 所有的干扰必定会对频谱信息造成影响. E-IPI 干扰意味着 f_k 及缺陷状态传播路径上的语句受到影响，因此语句覆盖矩阵 $\boldsymbol{M}_{cover}$ 会发生变化；RI 干扰意味着程序运行结果发生变化，因此程序运行结果向量 $\boldsymbol{V}_{result}$ 会发生变化.

表 2　E-IPI 干扰和 RI 干扰

Non-Interference (PG_s)		E-IPI only (PG_m)		RI only (PG_m)		E-IPI and RI (PG_m)	
E-IP-F	R	E-IP-F	R	E-IP-F	R	E-IP-F	R
$\langle\neg E,\neg IP\rangle$	P	$\langle E,\neg IP\rangle$	P	$\langle\neg E,\neg IP\rangle$	F	$\langle E,IP\rangle$, $\langle E,\neg IP\rangle$	F
$\langle E,\neg IP\rangle$	P	$\langle\neg E,\neg IP\rangle$	P	$\langle E,\neg IP\rangle$	F	$\langle E,IP\rangle$, $\langle\neg E,\neg IP\rangle$	F
$\langle E,IP\rangle$	F	$\langle\neg E,\neg IP\rangle$, $\langle E,\neg IP\rangle$	F	—	—	$\langle E,\neg IP\rangle$, $\langle\neg E,\neg IP\rangle$	P

4　干扰影响分析

根据第 3 节的分析，可知 E-IPI 干扰和 RI 干扰都会对程序频谱信息造成影响. 因此，SBFL 方法的缺陷定位结果也会受到影响. 然而，缺陷干扰对 SBFL 方法缺陷定位有效性的影响是未知的. 本节通过缺陷语句在 DS 集合中的分布来分析缺陷干扰对缺陷定位的影响. 对于缺陷 f_k，如果缺陷干扰将其移入到一个更好的(Better)子集，表明其对 SBFL 有正面影响；如果缺陷干扰将其移入到一个更差的(Worse)子集，表明其对 SBFL 有负面影响；如果缺陷干扰将其移入到一个等价的(Equivalent)子集，表明其对 SBFL 没有影响；否则缺陷干扰对 SBFL 的影响是不确定的(Non-Deterministic).

4.1　*DS* 集合

定义 4.　DS 集合(Disjoint Subsets, DS). 给定一个包含 m 条语句的程序 PG 和一个包含 n 条测试用例的测试用例集 TS. 根据程序在成功运行以及失败运行中的语句覆盖特征，可将 PG 分为以下 9 个互斥子集：

- S_{I}：被所有失败运行覆盖且未被任何成功运行覆盖，即 $S_{\mathrm{I}}=\{s_i\mid n_{cp}(s_i)=0, n_{cf}(s_i)=n_f\}$；
- S_{II}：被所有失败运行覆盖，同时被部分成功运行覆盖，即 $S_{\mathrm{II}}=\{s_i\mid n_{cp}(s_i)\in[1,n_p), n_{cf}(s_i)=n_f\}$；
- S_{III}：被所有失败运行覆盖且被所有成功运行覆盖，即 $S_{\mathrm{III}}=\{s_i\mid n_{cp}(s_i)=n_p, n_{cf}(s_i)=n_f\}$；
- S_{IV}：被部分失败运行覆盖且未被任何成功运行覆盖，即 $S_{\mathrm{IV}}=\{s_i\mid n_{cp}(s_i)=0, n_{cf}(s_i)\in[1,n_f)\}$；
- S_{V}：被部分失败运行覆盖且被部分成功运行覆盖，即 $S_{\mathrm{V}}=\{s_i\mid n_{cp}(s_i)\in[1,n_p), n_{cf}(s_i)\in[1,n_f)\}$；
- S_{VI}：被部分失败运行覆盖，且被所有成功运行覆盖，即 $S_{\mathrm{VI}}=\{s_i\mid n_{cp}(s_i)=n_p, n_{cf}(s_i)\in[1,n_f)\}$；
- S_{VII}：未被任何失败运行覆盖，但未被任何成功运行覆盖，即 $S_{\mathrm{VII}}=\{s_i\mid n_{cp}(s_i)=0, n_{cf}(s_i)=0\}$；
- S_{VIII}：未被任何失败运行覆盖，但被部分成功运行覆盖，即 $S_{\mathrm{VIII}}=\{s_i\mid n_{cp}(s_i)\in[1,n_p), n_{cf}(s_i)=0\}$；
- S_{IX}：未被任何失败运行覆盖，但被所有成功运行覆盖，即 $S_{\mathrm{IX}}=\{s_i\mid n_{cp}(s_i)=n_p, n_{cf}(s_i)=0\}$.

表 3 和表 4 分别给出了在 PG_s 和 PG_m 中 f_k 属于不同 DS 子集时 E-IP 过程的数目. 其中，n_p^s 和 n_f^s 分别表示 PG_s 中成功和失败测试用例的数目，n_p^m 和 n_f^m 分别表示 PG_m 中成功测试用例和失败测试用例的数目. 由于 PG_s 仅包含 f_k，对于 TS 中的每一个失败测试用例 t_f，其 E-IP 过程是确定的，即 $E\text{-}IP\text{-}F(PG_s,t_f,f_k)=\langle E,IP\rangle$. 因此，$\langle E,IP\rangle$ 的数量为 n_f^s，f_k 属于 $S_{\mathrm{I}}\cup S_{\mathrm{II}}\cup S_{\mathrm{III}}$. 然而，在 PG_m 中，新引入的缺陷可能带来 E-IPI 干扰和 RI 干扰，使得失败运行中出现新的 E-IP 过程，即 $\langle\neg E,\neg IP\rangle$ 和 $\langle E,\neg IP\rangle$. 因此，$\langle E,IP\rangle$ 的数量不一定为 n_f^s，f_k 可能分布在任一 DS 子集中.

表 3　PG_s 中缺陷属于不同 *DS* 子集时的 E-IP 过程

DS	Passed		Failed		
	$\langle\neg E,\neg IP\rangle$	$\langle E,\neg IP\rangle$	$\langle\neg E,\neg IP\rangle$	$\langle E,\neg IP\rangle$	$\langle E,IP\rangle$
S_{I}	n_p^s	0	0	0	n_f^s
S_{II}	$(0,n_p^s)$	$(0,n_p^s)$	0	0	n_f^s
S_{III}	0	n_p^s	0	0	n_f^s

表 4　PG_m 中缺陷属于不同 *DS* 子集时的 E-IP 过程

DS	Passed		Failed		
	$\langle\neg E,\neg IP\rangle$	$\langle E,\neg IP\rangle$	$\langle\neg E,\neg IP\rangle$	$\langle E,\neg IP\rangle$	$\langle E,IP\rangle$
S_{I}	n_p^m	0	0	$[0,n_f^m]$	$[0,n_f^m]$
S_{II}	$(0,n_p^m)$	$(0,n_p^m)$	0	$[0,n_f^m]$	$[0,n_f^m]$
S_{III}	0	n_p^m	0	$[0,n_f^m]$	$[0,n_f^m]$
S_{IV}	n_p^m	0	$(0,n_f^m)$	$[0,n_f^m)$	$[0,n_f^m)$
S_{V}	$(0,n_p^m)$	$(0,n_p^m)$	$(0,n_f^m)$	$[0,n_f^m)$	$[0,n_f^m)$
S_{VI}	0	n_p^m	$(0,n_f^m)$	$[0,n_f^m)$	$[0,n_f^m)$
S_{VII}	n_p^m	0	n_f^m	0	0
S_{VIII}	$(0,n_p^m)$	$(0,n_p^m)$	n_f^m	0	0
S_{IX}	0	n_p^m	n_f^m	0	0

4.2　*DS* 子集间关系

给定一个可疑度度量公式 $sus(s)$，用 $sus(s_i)$ 和 $sus(s_j)$ 分别表示语句 s_i 和 s_j 的可疑度，其中 $s_i\in S_x$，$s_j\in S_y$，$S_x, S_y\in DS$. 由此，根据集合间元素可疑度高低，可以定义 4 种类型的 DS 子集间关系.

定义 5.　优于(Better). 如果 S_x 和 S_y 满足：

$\forall s_i \in S_x, \forall s_j \in S_y, sus(s_i) \geq sus(s_j)$，则称 S_x 优于 S_y，记为 $S_x \geq S_y$.

定义 6. 差于(Worse). 如果 S_x 和 S_y 满足：$\forall s_i \in S_x, \forall s_j \in S_y, sus(s_i) < sus(s_j)$，则称 S_x 差于 S_y，记为 $S_x < S_y$.

定义 7. 等价于(Equivalent). 如果 S_x 和 S_y 满足：$\forall s_i \in S_x, \forall s_j \in S_y, sus(s_i) = sus(s_j)$，则称 S_x 等价于 S_y，记为 $S_x = S_y$.

定义 8. 不确定(Non-Deterministic). 如果 S_x 和 S_y 满足：$\exists s_{i1}, s_{i2} \in S_x, \exists s_{j1}, s_{j2} \in S_y, sus(s_{i1}) \geq sus(s_{j1}), sus(s_{i2}) < sus(s_{j2})$，则称 S_x 与 S_y 的关系是不确定的.

可以看到，关系 Better 满足自反性和传递性，即：(1) $S_x \geq S_x$；(2) 如果 $S_x \geq S_y, S_y \geq S_z$ 成立，$S_x \geq S_z$ 同样成立. 关系 Worse 满足传递性，即如果 $S_x < S_y, S_y < S_z$ 成立，$S_x < S_z$ 同样成立. 关系 Equivalent 满足自反性、对称性和传递性，即：(1) $S_x = S_x$；(2) 如果 $S_x = S_y$ 成立，$S_y = S_x$ 同样成立；(3) 如果 $S_x = S_y$，$S_y = S_z$ 成立，$S_x = S_z$ 同样成立.

程序语句在不同的可疑度度量方法下具有不同的可疑度及排序结果. 表 5 给出了各个互斥子集中语句在 4 个可疑度度量方法(Tarantula[18]、Ochiai[19]、Naish[20] 和 Wong[21])下的可疑度(或可疑度范围)，据此可以判断各个互斥子集间关系. 可以看到，不同可疑度度量方法下语句的可疑度(或可疑范围)是不同的. 因此，需要对 *DS* 子集间关系依据可疑度度量方法分别进行分析.

表 5 不同可疑度度量下各个 *DS* 子集的可疑度范围

DS	Tarantula	Ochiai	Naish	Wong
S_{I}	1	1	n_p	n_f
S_{II}	(1/2,1)	(0,1)	$(1, n_p-1)$	n_f
S_{III}	1/2	(0,1)	0	n_f
S_{IV}	1	(0,1)	-1	$(1, n_f-1)$
S_{V}	(0,1)	(0,1)	-1	$(1, n_f-1)$
S_{VI}	(0,1/2)	(0,1)	-1	$(1, n_f-1)$
$S_{\mathrm{VII}}, S_{\mathrm{VIII}}, S_{\mathrm{IX}}$	0	0	-1	0

对于 Naish 方法，*DS* 中元素间可疑度的高低是确定的，因而各个互斥子集的关系也是确定的. 例如，集合 S_{I} 中语句的可疑度始终大于集合 S_{II}，因此 $S_{\mathrm{I}} \geq S_{\mathrm{II}}$，同理 $S_{\mathrm{II}} \geq S_{\mathrm{III}}$、$S_{\mathrm{III}} \geq S_{\mathrm{IV}}$，而 S_{IV} 中语句的可疑度始终等于 S_{V}，因此 $S_{\mathrm{IV}} = S_{\mathrm{V}}$，同理 $S_{\mathrm{V}} = S_{\mathrm{VI}}$、$S_{\mathrm{VI}} = S_{\mathrm{VII}}$、$S_{\mathrm{VII}} = S_{\mathrm{VIII}}$、$S_{\mathrm{VIII}} = S_{\mathrm{IX}}$. 综上，Naish 方法下 *DS* 中元素间关系为：$S_{\mathrm{I}} \geq S_{\mathrm{II}} \geq S_{\mathrm{III}} \geq S_{\mathrm{IV}} = S_{\mathrm{V}} = S_{\mathrm{VI}} = S_{\mathrm{VII}} = S_{\mathrm{VIII}} = S_{\mathrm{IX}}$.

对于其他方法，部分互斥子集中语句的可疑度并不是唯一的且彼此间的数值大小是不确定的. 例如，当 $sus(s)$ 为 Tarantula 时，S_{II} 中语句的可疑度值介于 (1/2,1) 间，S_{V} 中语句的可疑度值介于 (0,1) 间，满足条件 $\exists s_{i1}, s_{i2} \in S_x, \exists s_{j1}, s_{j2} \in S_y, sus(s_{i1}) > sus(s_{j1}), sus(s_{i2}) < sus(s_{j2})$. 因此，$S_{\mathrm{II}}$ 与 S_{V} 无法判断优劣. 同理，S_{III} 与 S_{V}、S_{V} 与 S_{VI} 也无法判断优劣. 其它元素间关系则是确定的. 综上，Tarantula 方法下 *DS* 中元素间关系为 $S_{\mathrm{I}} = S_{\mathrm{IV}} \geq S_{\mathrm{II}} \geq S_{\mathrm{III}} \geq S_{\mathrm{VI}} \geq S_{\mathrm{VII}} = S_{\mathrm{VIII}} = S_{\mathrm{IX}}$，$S_{\mathrm{I}} = S_{\mathrm{IV}} \geq S_{\mathrm{V}} \geq S_{\mathrm{VII}} = S_{\mathrm{VIII}} = S_{\mathrm{IX}}$.

$$sus(s_i) = \frac{n_f}{\sqrt{n_f \times (n_f + n_{cp}^{\mathrm{II}})}} = \left(\frac{n_f}{n_f + n_{cp}^{\mathrm{II}}}\right)^{1/2}, \ s_i \in S_{\mathrm{II}} \tag{9}$$

$$sus(s_j) = \frac{n_f}{\sqrt{n_f \times (n_f + n_p)}} = \left(\frac{n_f}{n_f + n_p}\right)^{1/2}, \ s_j \in S_{\mathrm{III}} \tag{10}$$

$$sus(s_k) = \frac{n_{cf}^{\mathrm{IV}}}{\sqrt{n_f \times (n_{cf}^{\mathrm{IV}} + 0)}} = \left(\frac{n_{cf}^{\mathrm{IV}}}{n_f}\right)^{1/2}, \ s_k \in S_{\mathrm{IV}} \tag{11}$$

$$sus(s_l) = \frac{n_{cf}^{\mathrm{VI}}}{\sqrt{n_f \times (n_{cf}^{\mathrm{VI}} + n_p)}} = \left(\frac{n_{cf}^{\mathrm{VI}} \times n_{cf}^{\mathrm{VI}}}{n_f \times (n_{cf}^{\mathrm{VI}} + n_p)}\right)^{1/2}, \ s_l \in S_{\mathrm{VI}} \tag{12}$$

当两个互斥子集 S_x 和 S_y 中语句可疑度相同时，S_x 和 S_y 满足条件 $\forall s_i \in S_x, \forall s_j \in S_y, sus(s_i) = sus(s_j)$，这两个子集是等价的. 然而，当两个互斥子集具有相同的可疑度范围时，该条件不再满足，这两个子集的关系是不确定的. 此时，需要具体分析比较两个互斥子集中语句间的可疑度. 例如，当 $sus(s)$ 为 Ochiai 时，S_{II}、S_{III}、S_{IV}、S_{V}、S_{VI} 中语句的可疑度均介于 (0,1) 间，需要对它们进行两两分析. 因为 S_{II}、S_{III}、S_{IV}、S_{V}、S_{VI} 中语句的可疑度均大于 0，因此可以通过可疑度的平方值来比较大小. 以 S_{II} 和 S_{III}、S_{II} 和 S_{IV}、S_{III} 和 S_{VI} 为例进行分析. 式(9)～式(12)分别给出了 Ochiai 度量方法下，互斥子集 S_{II}、S_{III}、S_{IV}、S_{VI} 的可疑度表达式.

命题 1. $S_{\mathrm{II}} \geq S_{\mathrm{III}}$

根据定义 5，$S_{\mathrm{II}} \geq S_{\mathrm{III}}$ 的前提为 S_{II} 中所有语句的可疑度均高于或等于 S_{III} 中所有语句的可疑度. 由于 S_{II} 和 S_{III} 中语句的可疑度均大于 0，因此可以用两者的平方差来分析两个 *DS* 子集的关系. 因此，需证明对于 $\forall s_i \in S_{\mathrm{II}}, \forall s_j \in S_{\mathrm{III}}, sus(s_i)^2 - sus(s_j)^2 \geq 0$ 成立.

证明. 根据式(9)和式(10)，$sus(s_i)^2$ 与 $sus(s_j)^2$

的差为

$$sus(s_i)^2-sus(s_j)^2=\frac{n_f}{n_f+n_{cp}^{\mathrm{II}}}-\frac{n_f}{n_f+n_p}=\frac{n_f\times(n_p-n_{cp}^{\mathrm{II}})}{(n_f+n_{cp}^{\mathrm{II}})\times(n_f+n_p)}.$$

由于 n_f、$n_f+n_{cp}^{\mathrm{II}}$、n_f+n_p 和 $n_p-n_{cp}^{\mathrm{II}}$ 均大于 0，且 $sus(s_i)^2-sus(s_j)^2>0$ 是恒成立的. 因此，$S_{\mathrm{II}}\geq S_{\mathrm{III}}$ 成立.

命题 2.　S_{II} 与 S_{IV} 的关系为 Non-Deterministic.

根据定义 8，S_{II} 与 S_{IV} 的关系为 Non-Deterministic 的前提是 S_{II} 中语句的可疑度并不总是高于 S_{IV} 中语句且并不总是低于 S_{IV} 中语句. 因此，需要证明在 S_{II} 中存在一条可疑度高于 S_{IV} 最小值的语句，同时 S_{II} 中存在一条可疑度低于 S_{IV} 最高值的语句，即 $\exists s_{i1},s_{i2}\in S_{\mathrm{II}}$，$\exists s_{j1},s_{j2}\in S_{\mathrm{IV}}$，$sus(s_{i1})^2-sus(s_{j1})^2>0$，$sus(s_{i2})^2-sus(s_{j2})^2<0$ 成立.

证明. 根据式(9)和式(12)，$sus(s_i)^2$ 与 $sus(s_j)^2$ 的差为

$$sus(s_i)^2-sus(s_j)^2=\frac{n_f}{n_f+n_{cp}^{\mathrm{II}}}-\frac{n_{cf}^{\mathrm{IV}}}{n_f}=\frac{1}{n_f}\times\left(\frac{n_f^2}{(n_f+n_{cp}^{\mathrm{II}})}-n_{cf}^{\mathrm{IV}}\right).$$

由于 $1/n_f$ 大于 0 恒成立，多项式 $n_f^2/(n_f+n_{cp}^{\mathrm{II}})-n_{cf}^{\mathrm{IV}}$ 决定 $sus(s_i)^2-sus(s_j)^2$ 是否大于 0. 当 $n_{cf}^{\mathrm{IV}}<n_f^2/(n_f+n_{cp}^{\mathrm{II}})$ 时，$n_f^2/(n_f+n_{cp}^{\mathrm{II}})-n_{cf}^{\mathrm{IV}}$ 大于 0，此时 $sus(s_i)^2$ 与 $sus(s_j)^2$ 的差大于 0；当 $n_{cf}^{\mathrm{IV}}>n_f^2/(n_f+n_{cp}^{\mathrm{II}})$ 时，$n_f^2/(n_f+n_{cp}^{\mathrm{II}})-n_{cf}^{\mathrm{IV}}<0$，此时 $sus(s_i)^2$ 与 $sus(s_j)^2$ 的差小于 0. 因此，S_{II} 中存在一些语句可疑度高于 S_{IV} 中语句，也存在一些语句可疑度低于 S_{IV} 中语句. 由此，我们证明 S_{II} 与 S_{IV} 的关系为 Non-Deterministic.

命题 3.　$S_{\mathrm{III}}\geq S_{\mathrm{VI}}$.

证明. 根据式(10)和式(12)，$sus(s_i)^2$ 与 $sus(s_j)^2$ 的差为

$$sus(s_i)^2-sus(s_j)^2=\frac{n_f}{n_f+n_p}-\frac{n_{cf}^{\mathrm{VI}}\times n_{cf}^{\mathrm{VI}}}{n_f\times(n_{cf}^{\mathrm{VI}}+n_p)}=\frac{n_{cf}^{\mathrm{VI}}\times(n_f-n_{cf}^{\mathrm{VI}})+n_p\times(n_f^2-n_{cf}^{\mathrm{VI}}\times n_{cf}^{\mathrm{VI}})}{(n_f+n_p)\times(n_{cf}^{\mathrm{VI}}+n_p)}.$$

由于 $n_{cf}^{\mathrm{VI}}\times(n_f-n_{cf}^{\mathrm{VI}})$、$n_p\times(n_f^2-n_{cf}^{\mathrm{VI}}\times n_{cf}^{\mathrm{VI}})$ 和 $(n_f+n_p)\times(n_{cf}^{\mathrm{VI}}+n_p)$ 大于 0，$sus(s_i)^2-sus(s_j)^2>0$ 恒成立. 因此，$S_{\mathrm{III}}\geq S_{\mathrm{VI}}$ 成立. 由于关系 Better 满足传递性，且命题 1 成立，因此 $S_{\mathrm{II}}\geq S_{\mathrm{VI}}$ 也成立. 类似地，可以证明在 Ochiai 方法下其他各个 *DS* 子集间关系. 由此在 Ochiai 方法下，*DS* 子集间关系为 $S_{\mathrm{I}}\geq S_{\mathrm{II}}\geq S_{\mathrm{III}}\geq S_{\mathrm{VI}}\geq S_{\mathrm{VII}}=S_{\mathrm{VIII}}=S_{\mathrm{IX}}$，$S_{\mathrm{I}}\geq S_{\mathrm{IV}}\geq S_{\mathrm{V}}\geq S_{\mathrm{VII}}=S_{\mathrm{VIII}}=S_{\mathrm{IX}}$.

类似地，可以证明在 Wong 方法下，*DS* 子集间关系为 $S_{\mathrm{I}}=S_{\mathrm{II}}=S_{\mathrm{III}}\geq S_{\mathrm{IV}}\geq S_{\mathrm{VII}}=S_{\mathrm{VIII}}=S_{\mathrm{IX}}$，$S_{\mathrm{I}}=S_{\mathrm{II}}=S_{\mathrm{III}}\geq S_{\mathrm{V}}\geq S_{\mathrm{VII}}=S_{\mathrm{VIII}}=S_{\mathrm{IX}}$，$S_{\mathrm{I}}=S_{\mathrm{II}}=S_{\mathrm{III}}\geq S_{\mathrm{VI}}\geq S_{\mathrm{VII}}=S_{\mathrm{VIII}}=S_{\mathrm{IX}}$.

4.3　缺陷干扰对 SBFL 方法的影响

本节通过缺陷受到干扰后在 *DS* 集合中的分布来分析缺陷干扰对 SBFL 方法的影响：(1) Better 集合意味着正面影响(POSitive Impact，POSI)；(2) Worse 集合意味着负面影响(NEGative Impact，NEGI)；(3) Equivalent 集合意味着没有影响(No Impact，NI)；(4) Non-Deterministic 集合意味着影响是不确定的(Non-Deterministic，ND).

表 6 给出缺陷干扰对 Tarantula、Ochiai、Naish 和 Wong 等 4 种SBFL方法的影响结果. 根据表 3 和表 4，在未受到缺陷干扰时(单缺陷情况下)，缺陷 f_k 可能存在于集合 S_{I}、S_{II} 和 S_{III} 中. 对于每一个集合，列 3～列 6 分别给出其受到正面影响、无影响、负面影响以及不确定时的 *DS* 分布结果. 可以看到，在某些情况下缺陷干扰没有影响甚至会提高 SBFL 方法的有效性. 然而在多数情况下，缺陷干扰对 SBFL 存在负面影响. 因此，缺陷干扰很有可能降低 SBFL 方法的有效性.

表 6　缺陷干扰对 SBFL 方法的影响

SBFL	*DS*	POSI	NI	NEGI	ND
Tarantula	S_{I}	—	$S_{\mathrm{I}},S_{\mathrm{IV}}$	$S_{\mathrm{II}},S_{\mathrm{III}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{II}	$S_{\mathrm{I}},S_{\mathrm{IV}}$	S_{II}	$S_{\mathrm{III}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	S_{V}
	S_{III}	$S_{\mathrm{I}},S_{\mathrm{II}},S_{\mathrm{IV}}$	S_{III}	$S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	S_{V}
Ochiai	S_{I}	—	S_{I}	$S_{\mathrm{II}},S_{\mathrm{III}},S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{II}	S_{I}	S_{II}	$S_{\mathrm{III}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	$S_{\mathrm{IV}},S_{\mathrm{V}}$
	S_{III}	$S_{\mathrm{I}},S_{\mathrm{II}}$	S_{III}	$S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	$S_{\mathrm{I}},S_{\mathrm{V}}$
Naish	S_{I}	—	S_{I}	$S_{\mathrm{II}},S_{\mathrm{III}},S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{II}	S_{I}	S_{II}	$S_{\mathrm{III}},S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{III}	$S_{\mathrm{I}},S_{\mathrm{II}}$	S_{III}	$S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
Wong	S_{I}	—	$S_{\mathrm{I}},S_{\mathrm{II}},S_{\mathrm{III}}$	$S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{II}	—	$S_{\mathrm{I}},S_{\mathrm{II}},S_{\mathrm{III}}$	$S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—
	S_{III}	—	$S_{\mathrm{I}},S_{\mathrm{II}},S_{\mathrm{III}}$	$S_{\mathrm{IV}},S_{\mathrm{V}},S_{\mathrm{VI}},S_{\mathrm{VII}},S_{\mathrm{VIII}},S_{\mathrm{IX}}$	—

进一步，对缺陷干扰造成正面影响和负面影响的原因进行分析. 在特定输入下，分别用 S_s 和 S_m 表示缺陷 f_k 在单缺陷程序 PG_s 和多缺陷程序 PG_m 下的分布结果. 由于不同可疑度度量下互斥子集间的

关系是不同的,影响结果也是不同的.例如,令 $S_s=S_{\mathrm{I}}, S_m=S_{\mathrm{IV}}$.当采用 Tarantula 方法时,$S_s=S_m$,缺陷干扰对缺陷定位没有影响;当采用 Ochiai 方法时,$S_s\geq S_m$,缺陷定位对缺陷有负面影响.因此,需要根据可疑度度量方法分别进行分析.

分析 Tarantula 和 Ochiai 可知:(1)增加成功 E-IP 过程$\langle E,IP\rangle$对缺陷定位没有负面影响;(2)增加成功 E-IP 过程$\langle E,\rightarrow IP\rangle$对缺陷定位没有正面影响;(3)增加失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$对缺陷定位没有正面影响;(4)增加失败 E-IP 过程$\langle E,IP\rangle$和$\langle E,\rightarrow IP\rangle$对缺陷定位没有负面影响.对于 Naish 方法,当测试结果中不包含失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$时,可以看到:(1)增加成功 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$对缺陷定位存在正面影响;(2)增加成功 E-IP 过程$\langle E,\rightarrow IP\rangle$对缺陷定位存在负面影响.在其它的情况下 Naish 方法定位效果最差.对于 Wong 方法,可以看到:(1)成功 E-IP 过程$\langle E,IP\rangle$和$\langle E,\rightarrow IP\rangle$对缺陷定位没有影响;(2)增加失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$对缺陷定位存在负面影响;(3)增加失败 E-IP 过程$\langle E,IP\rangle$和$\langle E,\rightarrow IP\rangle$对缺陷定位存在正面影响.上述分析表明,在每一种度量方法下:(1)增加成功 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$对缺陷定位没有负面影响;(2)增加成功 E-IP 过程$\langle E,\rightarrow IP\rangle$对缺陷定位没有正面影响;(3)增加失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$对缺陷定位没有正面影响;(4)增加失败 E-IP 过程$\langle E,IP\rangle$和$\langle E,\rightarrow IP\rangle$对缺陷定位没有负面影响.可以发现:缺陷干扰造成的成功 E-IP 过程$\langle E,\rightarrow IP\rangle$和失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$是降低 SBFL 方法缺陷定位有效性的原因.

无论是单缺陷程序还是多缺陷程序,均存在偶然正确运行[32-33],都会对 SBFL 方法的有效性带来负面影响.然而,缺陷干扰可能会导致成功 E-IP 过程$\langle E,\rightarrow IP\rangle$,但并不是其产生的必要条件,而失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$则是由缺陷干扰所导致的.因此在多缺陷程序中,与成功 E-IP 过程$\langle E,\rightarrow IP\rangle$相比,缺陷干扰所导致的失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$可能是导致 SBFL 方法有效性进一步降低的直接原因.此外,在可疑度计算时,语句在失败运行中被覆盖的频次具有更高的权重[34],缺陷语句在失败运行中出现频次的降低也会降低其可疑度.因此可以得出结论:与成功 E-IP 过程$\langle E,\rightarrow IP\rangle$相比,缺陷干扰造成的失败 E-IP 过程$\langle\rightarrow E,\rightarrow IP\rangle$是降低 SBFL 方法缺陷定位有效性的主要原因.

5 FCMFL 方法

本节首先概述基于 FCM 聚类的软件多缺陷定位方法 FCMFL,然后详细描述 FCMFL 中的两个重要步骤:基于 FCM 聚类的失败测试用例集划分以及基于隶属度矩阵的语句检查序列生成.

5.1 方法概述

给定测试用例,监控源程序运行,获取程序的语句覆盖信息和运行结果,并将其作为下一步缺陷定位分析的结果.FCMFL 方法框架如图 3 所示.

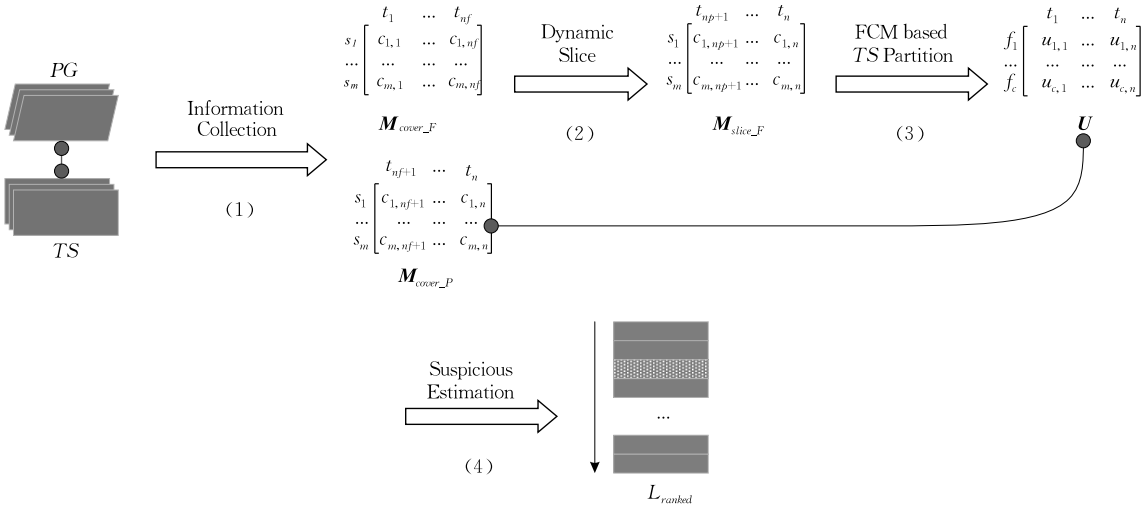


图 3 基于 FCM 聚类的软件多缺陷定位方法框架

第 1 步:信息收集(Information Collection).给定测试集 $TS=\{t_1,\cdots,t_n\}$,运行程序 $PG=\{s_1,\cdots,s_m\}$并记录其在 TS 下的语句覆盖信息和运行结果向量,进而生成成功运行的语句覆盖矩阵 $\boldsymbol{M}_{cover_P}=\{c_{i,j}\mid i\in[1,m],j\in[1,n_p]\}$和失败运行的语句覆盖矩阵 $\boldsymbol{M}_{cover\_F}=\{c_{i,j}\mid i\in[1,m],j\in[1,n_f]\}$.其中,$n_p$

和 n_f 分别表示 TS 中成功测试用例和失败测试用例的数目；

第 2 步：动态切片(Dynamic Slice). $\boldsymbol{M}_{cover\_F}$ 记录了 TS 中的失败测试用例. 对于每个失败测试用例，以 PG 输出语句中与程序运行失败相关的变量作为切片准则进行动态程序切片，从而生成 PG 在所有失败运行下的动态切片矩阵 $\boldsymbol{M}_{slice\_F}=\{x_{i,j}\mid i\in[1,m],j\in[1,n_f]\}$. 其中，元素 $x_{i,j}$ 值为 1 表示语句 s_i 包含于失败测试用例 t_j 的动态切片，$x_{i,j}$ 值为 0 表示 s_i 未被包含. 此外，用 ds_j 表示 PG 在测试用例 t_j 下的动态切片结果，因而 $\boldsymbol{M}_{slice\_F}=\{ds_j\mid j\in[1,n_f]\}$，$ds_j=\{s_i\mid i\in[1,m]\}$；

第 3 步：测试用例划分(FCM based TS Partition). 根据第 2 步得到的切片结果 $\boldsymbol{M}_{slice\_F}$，基于 FCM 聚类算法对失败测试用例进行聚类分析，以获得每个失败测试用例在不同缺陷上的隶属度，从而构建隶属度矩阵；

第 4 步：检查序列生成(Suspiciousness Estimation). 开展互斥子集检查优先级分析. 在此基础上，基于第 3 步得到的隶属度矩阵对语句进行可疑度加权计算，并针对不同互斥子集中的语句按照可疑度由高到低进行排序和组合，最终生成一个语句检查序列指导开发人员进行程序调试.

5.2　基于 FCM 聚类的失败测试用例划分

失败测试用例划分是 FCMFL 方法的重要步骤. 在定位某个缺陷时，与该缺陷无关的失败测试用例会影响缺陷语句的可疑度评价，从而降低 SBFL 方法的缺陷定位有效性. 因此，需要分析并识别与每个缺陷相关的失败测试用例，以避免或减轻无关失败测试用例的影响. 由于程序中缺陷的类型、数量和位置是未知的，难以获得充分的先验信息来指导失败测试用例划分，因此直接对失败测试用例进行分类是不合适的. 对此，本文采用聚类分析来实现失败测试用例的自动划分.

缺陷 f_k 的失败运行 E-IP 过程 $\langle E,IP\rangle$ 表示程序运行缺陷 f_k，同时产生并传播缺陷状态到程序输出位置，造成程序运行结果与期望结果不一致，导致程序运行失败. 缺陷状态是由程序运行缺陷语句产生的，缺陷状态则是通过程序内部的数据流和控制流关系传播的. 如果程序在不同的失败运行中具有相同的或相似的数据流、控制流信息，则程序运行失败很有可能由同一缺陷引发. 动态依赖分析技术可以识别程序在运行时产生的动态数据依赖和动态控制依赖关系，这些依赖关系有效地反映了程序中的状态传播和语句的运行关联. 如果以程序输出变量作为分析准则，并以此开展后向依赖分析，则可以识别程序中所有影响程序输出结果的语句(称为动态后向切片[35]，简称动态切片)，这些语句嵌入了与程序失败相关的数据流和控制流知识. 因此，本文将程序在不同失败运行中的动态切片结果作为聚类对象，指导失败测试用例的划分.

对失败测试用例进行聚类分析时，我们期望针对程序中的每一个缺陷生成一个聚类子集，该子集包含所有由该缺陷引发的失败测试用例. 然而，以下两个问题制约着聚类分析在失败测试用例划分中的应用：(1) 在多缺陷程序中，程序运行失败可能由多个缺陷共同引发. 此时，该失败用例不应当限定于某一个聚类子集中，而应当存在于所有与其相关的聚类子集中. 也就是说，聚类子集的划分界限不应当是分明的；(2) 多缺陷程序中缺陷的数量是未知的，聚类子集的数目也因此是未知的. 此时，应当提供一种聚类数目的预测方法.

对于第 1 个问题，基于第 2.3 节的分析，本节采用 FCM 聚类来划分失败测试用例. FCM 聚类为每一个失败测试用例在不同的聚类子集中分配一个隶属度，而不是将其强制划分到某一个类别中，因此可以更客观地描述失败测试用例与不同缺陷的关系. 对于第 2 个问题，本节通过度量不同聚类数目下聚类结果的质量来预测缺陷数目[36]. 聚类质量越高，则表明聚类结果与客观事实更加相符. 基于第 2.3 节的分析，本节采用 Xie-Beni 系数来评价 FCM 方法的聚类结果，并选择 Xie-Beni 系数最低的聚类结果作为最终聚类结果.

算法 2.　基于 FCM 聚类的失败测试用例划分算法.

输入：最大聚类数目 $c_{\max}$，失败测试用例切片矩阵 $\boldsymbol{M}_{slice\_F}$，停止阈值 ε，模糊因子 m，最大迭代次数 $b_{\max}$

输出：最优隶属度矩阵 $\boldsymbol{U}_{best}$，最优聚类中心集合 $\boldsymbol{V}_{best}$，最优聚类数目 c_{best}

```
1.  compute_Ubest() {
2.    FOR c=2 to c_max DO
3.      (U,V)←FCM(c)
4.      IF XB(U,V,c)<XB(U_best,V_best,c_best) THEN
5.        U_best←U, V_best←V, c_best←c
6.      END IF
7.    END FOR
8.    Output U,V,c
9.  }
```

```
10. FCM(c) {
11.   b←0
12.   Initialize cluster centers V^b = {v_1, v_2, ···, v_c} and
      membership matrix U^b = {μ_{i,j} | i∈[1,c], j∈[1,n]}
13.   REPEAT
14.     FOR ALL ds_j ∈ M_{slice_F} DO
15.       Compute the membership degree of ds_j for each
          cluster center in V^b, and update U^b
16.     END FOR
17.     Compute and update V^b using M_{slice_F}
18.     b←b+1
19.   UNTIL dis(V^{b-1}, V^b) ≤ ε or b > b_max
20.   Output U^b and V^b
21. }
```

基于上述分析，我们设计了一个基于 FCM 聚类的失败测试用例划分算法. 如算法 2 所示，算法以最大聚类数目 $c_{\max}$、失败测试用例切片矩阵 $\boldsymbol{M}_{slice\_F}$、停止阈值 ε、模糊因子 m 和最大迭代次数 $b_{\max}$ 作为输入，以最优隶属度矩阵 $\boldsymbol{U}_{\text{best}}$、最优聚类中心集合 $\boldsymbol{V}_{\text{best}}$ 和最优聚类数目 c_{best} 作为输出.

该算法主要包含两个部分：第 1 部分，根据 Xie-Beni 系数识别最优划分结果 $\boldsymbol{U}_{\text{best}}$、$\boldsymbol{V}_{\text{best}}$ 和 c_{best}（行 1～行 9）. 由于缺陷数量未知，聚类中心的数目也是未知的. 行 2～行 7 计算 $\boldsymbol{M}_{slice\_F}$ 在不同聚类数下划分结果的 Xie-Beni 系数（式(4)），并选择 Xie-Beni 系数最小的聚类结果作为最终的划分结果. 第 2 部分，根据 FCM 算法在给定聚类数目 c 下划分失败测试用例切片矩阵 $\boldsymbol{M}_{slice\_F}$（行 10～行 21）. 首先，在 $\boldsymbol{M}_{slice\_F}$ 中随机选择 c 个对象作为初始聚类中心（行 12）. 其次，更新隶属度矩阵 $\boldsymbol{U}$（行 14～行 16）. $\boldsymbol{U}$ 中元素 $\mu_{i,j}$ 表示第 j 个失败测试用例的切片结果 ds_j 在子集 v_i 中的隶属度. 基于 2.3 节的分析，根据 ds_j 与 $\boldsymbol{V}$ 中各个聚类中心的距离选择合适的隶属度计算方法（式(1)和式(2)）. 再次更新聚类中心集合 $\boldsymbol{V}$（行 17）. 对于每个聚类中心，根据新生成的隶属度矩阵通过式(3)重新计算聚类中心的位置. 最后，判断聚类结果是否满足预设条件（行 19）. 若不满足，则重复计算并更新隶属度矩阵和聚类中心；若满足，不再进行聚类操作. 此时，FCM 聚类结束并输出聚类结果 $\boldsymbol{V}$ 和 $\boldsymbol{U}$.

5.3 基于隶属度矩阵的语句检查序列生成

在单缺陷程序中，所有失败运行均由同一缺陷引发. 此时，缺陷语句肯定包含在所有失败测试用例的覆盖信息中，开发人员检查互斥子集 S_{I}、S_{II}、S_{III} 即可定位缺陷. 在 5.2 节中，我们通过 FCM 聚类分析了失败测试用例与不同缺陷间的隶属关系. 隶属度越高，表明测试用例运行失败由该缺陷引发的可能性越高，反之可能性越低. 因此在定位某个缺陷时，应选择高隶属于该缺陷的失败测试用例和所有的成功测试用例来开展缺陷定位，该缺陷有较大可能性存在于缺陷支配测试用例集的互斥子集 S_{I}、S_{II}、S_{III} 中. 然而，高隶属度的失败测试用例并不一定全部由该缺陷引发，缺陷也可能存在于互斥子集 S_{IV}、S_{V}、S_{VI} 中. 因此，在多缺陷程序中开展缺陷定位时，应当首先检查 S_{I}、S_{II}、S_{III} 中的语句，再检查 S_{IV}、S_{V}、S_{VI} 中的语句，最后检查 S_{VII}、S_{VIII}、S_{IX} 中的语句.

$$S_{\text{I}} \geqslant S_{\text{II}} \geqslant S_{\text{III}} \geqslant S_{\text{IV}} \geqslant S_{\text{V}} \geqslant S_{\text{VI}} \geqslant S_{\text{VII}} \geqslant S_{\text{VIII}} \geqslant S_{\text{IX}} \quad (13)$$

进一步地，根据 Wong 等人[37]实证验证的 4 个假设，我们对这三类集合中元素的检查优先顺序分别进行排列，最终得到如式(13)所示的多缺陷程序中互斥子集的检查优先序列.

对于同一互斥集合中的语句，按照可疑度由高到低进行检查. 可以根据已有的可疑度度量方法（如 Tarantula[18]、Ochiai[19]、Naish[20] 和 Wong[21]）计算语句的可疑度. 由于每个失败测试用例与缺陷的隶属程度不同，其对于度量结果的贡献度也应是不同的. 因此，在可疑度计算时需对 n_{cf} 和 n_{uf} 做加权处理. 如式(14)和式(15)所示，对于每个失败测试用例，应根据其对缺陷的隶属度对每条语句在失败测试用例上的覆盖结果做加权处理，处理结果表示为 ωn_{cf}^{ic} 和 ωn_{uf}^{ic}. 然而，由于隶属度普遍小于 1，失败测试用例在可疑度度量时起到的影响过小. 因此，如式(16)和式(17)所示，同样需要对每条语句在成功测试用例上的覆盖结果做加权处理，来平衡成功测试用例对可疑度度量的影响. 特别地，用失败测试用例的平均隶属度对其做加权处理，处理结果表示为 ωn_{cs} 和 ωn_{us}. 此时，每条语句的可疑度可利用加权四元组 $\omega N_s^{ic} = (\omega n_{cs}, \omega n_{cf}^{ic}, \omega n_{us}, \omega n_{uf}^{ic})$ 来计算.

$$\omega n_{cf}^{ic}(s_i) = \sum_{j=n_s+1}^{n_s+n_{cf}^{ic}} (\mu_{j-n_s} \times c_{i,j}^{ic} \times r_j^{ic}) \quad (14)$$

$$\omega n_{uf}^{ic}(s_i) = \sum_{j=n_s+1}^{n_s+n_{cf}^{ic}} (\mu_{j-n_s} \times (1 - c_{i,j}^{ic}) \times r_j^{ic}) \quad (15)$$

$$\omega n_{cs}(s_i) = \frac{\sum_{jf=1}^{n_f^{ic}} \mu_{jf}}{n_f^{ic}} \times \sum_{j=1}^{n_s} (c_{i,j}^{ic} \times (1 - r_j^{ic})) \quad (16)$$

$$\omega n_{us}(s_i)=\frac{\sum_{jf=1}^{n_f^{ic}}\mu_{jf}}{n_f^{ic}}\times\sum_{j=1}^{n_s}((1-c_{i,j}^{ic})\times(1-r_j^{ic}))\quad(17)$$

基于上述分析，我们设计了一个基于隶属度矩阵的语句检查序列生成算法. 如算法 3 所示，算法以成功和失败测试用例覆盖信息矩阵 $\boldsymbol{M}_{cover\_P}$ 和 $\boldsymbol{M}_{cover\_F}$、隶属度矩阵 $\boldsymbol{U}$ 和可疑度度量方法 sus 作为输入，以语句检查序列 L_{ranked} 作为输出.

算法 3.　基于隶属度矩阵的语句检查序列生成算法.

输入：成功测试用例覆盖信息矩阵 $\boldsymbol{M}_{cover\_P}$，失败测试用例覆盖信息矩阵 $\boldsymbol{M}_{cover\_F}$，隶属度矩阵 $\boldsymbol{U}$，可疑度度量方法 sus

输出：语句检查序列 L_{ranked}

1. //Phase 1: suspiciousness estimation.
2. FOR $ic=1$ to c DO
3. FOR $jf=1$ to n_f DO
4. IF $\mu_{if,ic}\geq 1/c$ THEN
5. $\boldsymbol{M}_{cover\_F}^{ic}=[\boldsymbol{M}_{cover\_F}^{ic},\boldsymbol{M}_{cover}^{ic}[,if]]$
6. END IF
7. END FOR
8. $\boldsymbol{M}_{cover}^{ic}=[\boldsymbol{M}_{cover\_P},\boldsymbol{M}_{cover\_F}^{ic}]$
9. Generate $N_s^{ic}(s_i)$ for each statement s_i using $\boldsymbol{M}_{cover}^{ic}$
10. Generate $DS^{ic}=\{S_x^{ic}\mid x\in[\mathrm{I},\mathrm{IX}]\}$ using $\boldsymbol{M}_{cover}^{ic}$
11. Generate $\omega N^{ic}(s_i)$ for each statement s_i using $\boldsymbol{M}_{cover}^{ic}$ and $\boldsymbol{U}$
12. Compute the suspiciousness for each statement s_i using $\omega N_s^{ic}(s_i)$ and sus
13. END FOR
14. //Phase 2: checked list generation.
15. $DS^u=\{S_x^u\mid x\in[\mathrm{I},\mathrm{IX}]\}$
16. FOR $x=\mathrm{I}$ to IX DO
17. IF $x=\mathrm{I}$ THEN
18. $S_x^u\leftarrow(S_x^1\cup S_x^2\cup\cdots\cup S_x^c)$
19. ELSE
20. $S_x^u\leftarrow(S_x^1\cup S_x^2\cup\cdots\cup S_x^c)-(S_1\cup S_2\cup\cdots\cup S_{x-1})$
21. END IF
22. Update suspiciousness for each statement s_i in S_x with the max value
23. Sort statements in S_x by their suspiciousness in descending order
24. Add sort results to L_{ranked}
25. END FOR
26. Output L_{ranked}

该算法主要分为 2 步：第 1 步，可疑度度量(行 1～行 13). 首先，对于每个聚类子集，选择该子集下所有高隶属度(大于等于平均隶属度 $1/c$)的失败测试用例生成程序在该子集下的失败测试用例覆盖信息矩阵 $\boldsymbol{M}_{cover\_F}^{ic}$，进而生成程序在该子集下的程序覆盖信息矩阵 $\boldsymbol{M}_{cover}^{ic}$(行 3～行 8). 其次，根据 $\boldsymbol{M}_{cover}^{ic}$ 计算程序中每条语句的四元组信息 N_s^{ic} 及互斥子集 DS^{ic}(行 9 和行 10). 再次，根据隶属度矩阵 $\boldsymbol{U}$ 计算程序中每条语句的加权四元组信息 ωN_s^{ic}(行 11). 最后，基于 ωN_s^{ic}，根据给定的可疑度度量方法 sus 计算程序中每条语句的可疑度(行 12). 第 2 步，语句检查序列生成(行 14～行 26). 根据互斥子集间关系分析，应当首先检查高优先级互斥子集中的语句，再检查低优先级互斥子集中的语句. 因此，对于存在于多个级别互斥子集中的语句，应当首先将其置入于高优先级互斥子集中(行 17～行 21). 对于互斥子集中的语句，选择其在该级别中的最高可疑度值作为语句的最终可疑度值并进行降序排列，将其添加到检查序列 L_{ranked} 中(行 22～行 24). 所有互斥子集的语句序列添加完成后，检查序列 L_{ranked} 生成.

6　实证研究

6.1　实验对象

表 7 对本文实验对象进行描述. 对于每个实验

表 7　实验对象

程序	描述	代码行数	测试用例数	单缺陷版本数	多缺陷版本数								
					2	3	4	5	6	7	8	9	10
JTcas (JT)	碰撞检测器	181	1608	39	100	100	100	100	100	100	100	100	100
Tot_info (TI)	信息测量	283	1079	21	100	100	100	100	100	100	100	100	100
Schedule1 (S1)	优先排队	290	2650	8	27	50	55	36	13	2	0	0	0
Schedule2 (S2)	优先排队	317	2710	7	20	30	25	11	2	0	0	0	0
Print_tokens1 (PT1)	词法分析器	478	4140	7	20	30	25	11	2	0	0	0	0
Print_tokens2 (PT2)	词法分析器	410	4140	10	43	100	100	100	85	28	4	0	0
NanoXML v1 (N1)	XML 解析器	4351	235	6	15	20	15	6	1	0	0	0	0
NanoXML v2 (N2)	XML 解析器	5671	235	4	6	4	1	0	0	0	0	0	0
NanoXML v3 (N3)	XML 解析器	6838	237	10	45	100	100	100	100	100	45	10	1
NanoXML v5 (N5)	XML 解析器	7160	237	7	21	35	35	21	7	1	0	0	0
Siena v5 (S5)	发布/订阅系统	6098	567	2	1	0	0	0	0	0	0	0	0
Siena v7 (S7)	发布/订阅系统	6034	567	2	1	0	0	0	0	0	0	0	0
			—	123	399	569	556	485	410	331	249	210	201

对象，列 1～列 5 分别给出了程序名称、程序功能、可运行代码行数、测试用例数和可用的单缺陷版本数[38]等信息. 其中，前 6 个程序是由 Santelices 等人[16]翻译成 Java 版本的 Siemens 程序，后 6 个 Java 程序由 SIR 提供[17]. 上述程序在缺陷定位研究中被广泛使用，因此具有一定的代表性[16,39]. 列 6～列 14 给出了不同程序在不同缺陷数下的多缺陷版本数. 由此，本文利用 123 个单缺陷版本和 3410 个多缺陷版本开展缺陷干扰研究.

6.1.1 多缺陷版本生成

由于这些数据集并未提供多缺陷版本，因此我们按照文献[12]的方法，将多个单缺陷版本组合生成一个多缺陷版本进行实证研究，每个多缺陷版本包含的缺陷数目为 2～10. 例如，图 4 给出 JTcas 程序(tcas.java)片段，数据集提供了 3 个与该片段相关的缺陷，缺陷 ID 分别为 F_T_HD_1、F_T_HD_2 和 F_T_HD_40. 图 5 给出一个包含缺陷 F_T_HD_1 和 F_T_HD_2 的双缺陷版本，通过替换源程序 64 行和 84 行即可完成.

```
s63   if(Climb_Inhibit>0)
s64     return Up_Separation+NOZCROSS;
      //F_T_HD_2: return Up_Separation+MINSEP;
...
s84     result=!(Own_Below_Threat())||((Own_Below_Threat())
      &&(!(Down_Separation>=ALIM())));
      //F_T_HD_1: result=!(Own_Below_Threat())||
      ((Own_Below_Threat())&&(!(Down_Separation>ALIM())));
      //F_T_HD_40: result=((Own_Below_Threat()) &&
      (!(Down_Separation >=ALIM())));
s85   }
```

图 4 一个 JTcas 源程序(tcas.java)片段

```
s63   if (Climb_Inhibit>0)
s64     return Up_Separation+MINSEP;
...
s84     result=!(Own_Below_Threat())||((Own_Below_Threat())
      &&(!(Down_Separation>ALIM())));
s85   }
```

图 5 一个 JTcas 双缺陷版本(F_T_HD_1 和 F_T_HD_2)

由于每个程序单缺陷版本的数目是不同的，因此在理论上，每个程序可能的多缺陷版本的数目也是不同的. 例如，JTcas 包含 39 个单缺陷版本，因此理论上可以生成 C_{39}^{2}，即 741 个双缺陷版本. 然而，并非所有的缺陷都可组合形成多缺陷版本. 例如，缺陷 F_T_HD_1 和 F_T_HD_40 均与语句 84 相关. 此时，这两个缺陷可认为是冲突的，无法同时存在. 在多缺陷版本生成时，应避免冲突缺陷的存在，造成期望缺陷数目与实际缺陷数目不符的结果.

此外，还需对每个缺陷中的行号信息进行修正，使源程序与缺陷程序各个语句的行号保持一致，以便于覆盖统计. 例如，图 6 中的 JTcas 源程序片段存在一个相关的缺陷 F_T_HD_37(如图 7). 若直接将 F_T_HD_37 注入到 JTcas 中，方法 ALIM 内部及其后续语句的行号都会发生变化. 此时，准备记录并对比程序在不同测试用例下的覆盖信息变得十分困难. 对此，需要在缺陷植入前对每个缺陷在行号上做修正，确保缺陷注入后各个语句的行号信息保持一致. 如图 8，可通过“注释”而不是“删除”来保证缺陷注入后语句行号的一致性.

```
s49   int ALIM ()
s50   {
s51       if (Alt_Layer_Value==0)
s52         return Positive_RA_Alt_Thresh_0;
s53       else if (Alt_Layer_Value==1)
s54         return Positive_RA_Alt_Thresh_1;
s55       else if (Alt_Layer_Value==2)
s56         return Positive_RA_Alt_Thresh_2;
s57       else
s58         return Positive_RA_Alt_Thresh_3;
s59   }
```

图 6 一个 JTcas 源程序(tcas.java)片段

```
s49   int ALIM ()
s50   {
s51     return Positive_RA_Alt_Thresh_0;
s52   }
```

图 7 缺陷 F_T_HD_37

```
s49   int ALIM ()
s50   {
s51   //  if(Alt_Layer_Value==0)
s52         return Positive_RA_Alt_Thresh_0;
s53   //  else if(Alt_Layer_Value==1)
s54   //      return Positive_RA_Alt_Thresh_1;
s55   //  else if(Alt_Layer_Value==2)
s56   //      return Positive_RA_Alt_Thresh_2;
s57   //  else
s58   //      return Positive_RA_Alt_Thresh_3;
s59   }
```

图 8 修正后的缺陷 F_T_HD_37

由上可知，若要生成多缺陷版本：一方面，需要识别冲突缺陷，避免因缺陷数量不一致而造成定位结果分析不准确的问题；另一方面，在缺陷注入前应当修正缺陷的行号信息，确保缺陷注入前后语句行号的一致性. 除此之外，生成数千个多缺陷版本也是一项繁重的工作. 对此，我们实现并发布了一个多缺陷版本自动生成工具 xPoppy①，用以实现多缺陷版

① https://github.com/xingyawang/xPoppy

本的快速自动生成.

6.1.2　E-IP 建模

对于每个缺陷版本，我们记录其在给定测试用例集下的语句覆盖信息和程序运行结果，并利用这些信息来构建 E-IP 模型. 给定测试用例 t，对于单缺陷程序 PG_s，缺陷 f_k 的 E-IP 过程是确定的：(1) 如果 f_k 未被 t 覆盖且 PG_s 运行成功，$E\text{-}IP\text{-}F(PG_s,t,f_k)=\langle\neg E,\neg IP\rangle$；(2) 如果 f_k 被 t 覆盖但 PG_s 运行成功，$E\text{-}IP\text{-}F(PG_s,t,f_k)=\langle E,\neg IP\rangle$；(3) 如果 f_k 被 t 覆盖且 PG_s 运行失败，$E\text{-}IP\text{-}F(PG_s,t,f_k)=\langle E,IP\rangle$. 然而，对于多缺陷程序 PG_m，缺陷 f_k 的 E-IP 过程并非均可确定：(1) 如果 f_k 未被 t 覆盖且 PG_m 运行成功，$E\text{-}IP\text{-}F(PG_m,t,f_k)=\langle\neg E,\neg IP\rangle$；(2) 如果 f_k 被 t 覆盖但 PG_m 运行成功，$E\text{-}IP\text{-}F(PG_m,t,f_k)=\langle E,\neg IP\rangle$；(3) 如果只有缺陷 f_k 被 t 覆盖且 PG_m 运行失败，$E\text{-}IP\text{-}F(PG_m,t,f_k)=\langle E,IP\rangle$；(4) 如果多个缺陷(包括 f_k)被 t 覆盖且 PG_m 运行失败，$E\text{-}IP\text{-}F(PG_m,t,f_k)$ 的值为 $\langle E,IP\rangle$ 或 $\langle E,\neg IP\rangle$. 此时，针对所有构建好的 E-IP 模型，根据定义 2 和定义 3 来识别 E-IPI 干扰和 RI 干扰以及 E-IPI 测试用例和 RI 测试用例. 需要注意的是，并非所有的 E-IPI 干扰都可被直接识别. 例如，当 $E(PG_m,t,f_k)=E$、$R(PG_m,t)=F$ 时，仅通过覆盖信息难以判断 f_k 是否是导致 PG_m 运行失败的原因，即 $IP(PG_m,t,f_k)$ 无法得知. 此时，无法判断是否发生了 E-IPI 干扰.

6.1.3　实验对比方法选择

C 均值(C-Means，CM)聚类[27]是经典硬聚类算法，也是多缺陷定位中划分失败测试用例集的常用方法[8-10,13]. 本文所采用的 FCM 聚类方法由 CM 聚类方法[27]通过模糊集理论推广到模糊情形中而来. 为了验证所提 FCMFL 方法的有效性，本文选择了不使用聚类的方法(No Clustering based Multi-Fault Localization，NCMFL)和基于 CM 聚类的方法(CM Clustering based Multi-Fault Localization，CMFL)作为实验对比对象. 其中，NCMFL 方法以覆盖矩阵和结果向量作为输入，不加入聚类结果，直接计算语句可疑度并排序. CMFL 方法则进一步加入 CM 聚类结果来进行可疑度计算及排序. 同时，以面向硬聚类的 Silhouette 系数[40]来评价 CM 方法的聚类结果质量，并选择最优聚类结果来进行缺陷定位. FCMFL 加入 FCM 聚类结果来进行可疑度计算及排序. 以 Xie-Beni 系数[40]来评价 FCM 聚类结果质量，并选择最优聚类结果来进行缺陷定位.

目前存在多种可疑度度量方法，Naish 等人[20]和 Xie 等人[6]均从理论分析的角度对它们的有效性做了分析，研究结果表明 Naish[20]、Wong[21]等方法在理论上具有最好的缺陷定位效果；Le 等人[41]则从实证研究的角度对这些可疑度度量方法的有效性做了分析，研究结果表明 Ochiai[19]和 Tarantula[18]在实践中具有更好的缺陷定位效果. 因此，本文分别应用 Tarantula、Ochiai、Naish、Wong 等方法来计算语句可疑度. 综上，本文应用 3 种聚类方法和 4 种缺陷定位方法等 12 种组合开展实证研究. 其中，NCMFL 方法分别表示为 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$，CMFL 方法分别表示为 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$，FCMFL 方法分别表示为 T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$.

6.1.4　运行失败相关变量的识别

给定源程序 PG、缺陷程序 PG_f 和测试用例集 TS，自动确定运行失败相关变量的步骤如下：首先，静态分析 PG 和 PG_f 来识别程序中每一个与控制台显示、数据转储等程序输出相关的变量对 $\langle var,var_f\rangle$，将其作为运行失败相关变量的候选集；其次，基于 TS 运行 PG 和 PG_f，同时记录它们在每一个测试用例上的输出结果以及候选集中变量的取值结果；再次，通过比较 PG 和 PG_f 在不同测试用例上的输出结果将 TS 分为成功测试用例集 TS_P 和失败测试用例集 TS_F；最后，对 TS_F 中的每一个失败测试用例，比较候选集中每一个变量对 $\langle var,var_f\rangle$ 的取值是否相同. 若 var 与 var_f 取值不同，则将 var_f 标记为程序运行失败相关变量. 对于每次程序失败执行，所有程序运行失败相关变量构建为动态切片准则.

6.1.5　参数设置

算法 2 需要配置 $c_{\max}$、ε、m、$b_{\max}$ 等各个参数的取值. 在本文实验中，各个参数的取值结果如下：$c_{\max}$ 为最大聚类数目，本文实验对象程序最大缺陷数目为 10，因而选择 10 作为其取值结果；ε 为停止阈值，取值为 0，表示在聚类结果稳定、不再发生变化时满足停止条件；m 为模糊因子，取值为 2，Pal 等人[42]的实证研究表明 m 的最佳候选区间为 $[1.5,2.5]$，在无特殊要求下可取区间中值 2；$b_{\max}$ 为最大迭代次数，本文实证研究表明当迭代次数为 100 时，聚类结果趋于稳定，因此 $b_{\max}$ 取值为 100.

6.2　实验设计

本文基于 E-IP 模型定义了两类缺陷干扰并分析了其对 SBFL 方法缺陷定位有效性的影响，并在此基础上提出了基于 FCM 聚类的多缺陷定位方法

FCMFL.

本节设计了 7 个实验来验证分析结果的正确性及 FCMFL 方法的有效性.

实验问题 1. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,计算 E-IPI 干扰和 RI 干扰在不同缺陷数目下的频繁度.

实验问题 2. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,分析不同测试用例集下缺陷在 *DS* 集合中的分布.

实验问题 3. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,分析缺陷数目的预测结果与真实结果的差异.

实验问题 4. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,评估 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 等 4 种 NCMFL 方法的缺陷定位代价.

实验问题 5. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,评估 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$ 等 4 种 CMFL 方法的缺陷定位代价.

实验问题 6. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,评估 T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 等 4 种 FCMFL 方法的缺陷定位代价.

实验问题 7. 使用单缺陷和多缺陷版本的 Siemens 和 SIR 程序,比较 NCMFL、CMFL 与 FCMFL 方法的缺陷定位代价.

实验问题 1 的目的在于研究缺陷干扰的普遍程度;实验问题 2 的目的在于分析缺陷干扰对缺陷在 *DS* 集合中分布的影响;实验问题 3 的目的在于分析 Xie-Beni 系数用于缺陷数目预测时的有效性;实验问题 4 的目的在于分析缺陷干扰对 SBFL 方法缺陷定位有效性的影响;实验问题 5 和实验问题 6 的目的在于分析 CMFL 和 FCMFL 方法是否会降低缺陷干扰的影响;实验问题 7 的目的在于分析 FCMFL 方法是否可以改善 SBFL 方法的缺陷定位效果.

6.3 评价指标

采用频繁度(*Frequency*)来评估缺陷干扰发生的频繁程度,*Frequency* 表示发生缺陷干扰时测试用例 $num_{interfered}$ 占所有测试用例 num_{test} 的百分比,使用式(18)计算. *Frequency* 值越高,缺陷干扰发生的可能性越高.

$$Frequency = \frac{num_{interfered}}{num_{test}} \tag{18}$$

采用定位代价(*Expense*)评估缺陷定位有效性,*Expense* 表示定位到缺陷时需要检查的语句数 $num_{examined}$ 占所有语句数 num_{stmt} 的百分比,使用式(19)计算. 定位到缺陷时需要检查的语句数 $num_{examined}$ 指检测到所有缺陷语句时平均所需检查语句数. *Expense* 值越低,定位缺陷时需要检查的语句数量越少,定位精度越高. 此外,检查结果中可能存在多条语句检查等级相同,"最好"情况是检查第 1 条语句就定位到错误,"最坏"情况为直到检查到最后 1 条语句才定位到错误. 与文献[43]相同,本文选取"最好"和"最坏"情况平均值进行评估.

$$Expense = \frac{num_{examined}}{num_{stmt}} \tag{19}$$

6.4 实验结果与分析

(1)针对实验 1 的结果分析

表 8 分别给出 E-IPI 干扰和 RI 干扰在不同程序中的频繁度. 对于每个程序,列 2～列 10 列出了缺陷数量为 2～10 时的 *Frequency* 值. 需要注意的是 E-IPI 干扰的 *Frequency* 值要低于其实际值. 对比两表数据可知,对于每个程序,在相同缺陷数目下 E-IPI 干扰的 *Frequency* 值总是要低于 RI 干扰的 *Frequency* 值. 因此,E-IPI 干扰的平均 *Frequency* 值(3.50%)

表 8 E-IPI 干扰和 RI 干扰的频繁度

PG	E-IPI (≥%)									RI (%)								
	2	3	4	5	6	7	8	9	10	2	3	4	5	6	7	8	9	10
JT	1.94	3.46	3.81	4.71	6.58	7.56	8.24	9.54	9.86	2.45	7.83	7.72	9.88	12.11	13.15	16.89	16.54	18.93
TI	0.51	0.43	1.99	1.83	1.91	2.45	2.86	3.00	3.81	7.90	12.70	17.18	21.51	21.78	24.99	25.24	26.76	29.32
S1	1.11	2.25	3.39	4.53	5.64	6.73	—	—	—	3.66	7.25	10.63	13.75	16.60	19.18	—	—	—
S2	0.08	0.08	0.11	0.13	0.15	—	—	—	—	1.32	2.44	3.38	4.17	4.86	—	—	—	—
PT1	0.30	0.57	0.83	1.08	1.33	—	—	—	—	3.43	6.68	9.72	12.56	15.21	—	—	—	—
PT2	0.51	0.52	0.61	1.06	1.60	1.82	2.15	—	—	4.55	8.54	12.22	14.92	16.99	19.15	21.13	—	—
N1	2.94	4.50	5.95	7.37	8.43	—	—	—	—	7.64	13.53	18.80	22.30	26.13	—	—	—	—
N2	0.47	0.57	0.28	—	—	—	—	—	—	5.25	7.61	9.22	—	—	—	—	—	—
N3	1.94	3.46	3.81	4.71	6.58	7.56	8.24	9.54	9.86	2.49	3.92	2.87	3.52	6.39	11.46	13.63	15.26	16.88
N5	0.70	1.41	2.09	2.78	3.46	4.14	—	—	—	6.08	11.10	15.56	19.78	23.97	28.10	—	—	—
S5	0	—	—	—	—	—	—	—	—	0	—	—	—	—	—	—	—	—
S7	0	—	—	—	—	—	—	—	—	17.11	—	—	—	—	—	—	—	—
Ave	3.50									10.04								

要低于 RI 干扰的平均 *Frequency* 值(10.04%). 这表明与 E-IPI 干扰相比,RI 干扰更容易被引发. 该差异产生的原因在于 E-IPI 干扰的发生要求其他缺陷对另一个缺陷的运行和传播产生影响,而对于 RI 干扰则不是必须的. 尽管两类干扰的频繁度有较大的差异,但依然存在一些类似的趋势. 例如,随着缺陷数量增加,E-IPI 干扰和 RI 干扰的频繁度也在逐渐增加. 这表明缺陷数量的增多增加了缺陷交互的可能性,由此使得缺陷的覆盖结果、程序运行结果更易发生改变.

(2)针对实验 2 的结果分析

表 9 给出了单缺陷程序(列 2)和多缺陷程序(列 3～列 11)下缺陷在 *DS* 集合中的分布结果. 通过列 2 数据可以看到,对于单缺陷程序,所有的缺陷都被分配到集合 $S_{\mathrm{I}} \cup S_{\mathrm{II}} \cup S_{\mathrm{III}}$ 中,也就说缺陷 f_k 会被所有的失败测试用例覆盖到. 其中,与 S_{I} 相比,S_{II} 和 S_{III} 占据了更大的比例(89.74%),这表明偶然正确运行是相当常见的,也就是说,运行缺陷 f_k 但没有造成程序运行失败是一种普遍现象. 与列 2 数据相比,列 3～列 11 中的数据发生了较大的变化. 可以看到,虽然集合 $S_{\mathrm{I}} \cup S_{\mathrm{II}} \cup S_{\mathrm{III}}$ 占据了一部分比例(36.67%～57.93%),但更多情况下缺陷被分配在集合 $S_{\mathrm{IV}} \cup S_{\mathrm{V}} \cup S_{\mathrm{VI}}$ 中(41.41%～62.49%). 这是因为当程序包含多个缺陷时,f_k 不再是唯一导致程序出错的原因,其他的缺陷同样也会导致程序运行失败,由此使得 f_k 不再分布在所有的失败运行中. 此外,我们还可以发现缺陷在 *DS* 集合中的分布是不均匀的:多数缺陷分布在集合 S_{II}(35.95%～51.55%)和 S_{V}(32.90%～51.06%)中,少部分分布在集合 S_{I}(0～1.80%)、S_{VI}(0～0.07%)、S_{VII}(0.29%～2.59%)、S_{III}(0～0.33%)和 S_{IX}(0)中. 这表明:(1)运行 f_k 不一定会导致程序运行失败;(2)多数情况下,f_k 不再是导致程序运行失败的原因;(3)缺陷干扰难以影响 f_k 的运行和传播.

表 9　缺陷在单缺陷程序 *DS* 集合和多缺陷程序 *DS* 集合中的分布

DS 集合	单缺陷程序(%)	多缺陷程序(%)								
		2	3	4	5	6	7	8	9	10
S_{I}	10.26	1.80	0.36	0.12	0	0	0	0	0	0
S_{II}	85.47	51.55	43.43	40.31	40.56	39.52	35.49	34.95	41.83	37.65
S_{III}	4.27	4.58	4.28	4.63	3.34	2.04	1.33	1.72	2.00	2.96
S_{IV}	0	8.51	11.04	10.64	13.34	17.84	22.69	17.56	8.94	5.74
S_{V}	0	32.90	40.38	43.84	42.33	40.03	39.80	44.71	45.15	51.06
S_{VI}	0	0	0.07	0	0	0	0	0	0	0
S_{VII}	0	0.33	0.29	0.29	0.43	0.57	0.70	1.07	2.09	2.59
S_{VIII}	0	0.33	0.15	0.17	0	0	0	0	0	0
S_{IX}	0	0	0	0	0	0	0	0	0	0

(3)针对实验 3 的结果分析

表 10 给出了各个程序中缺陷数目的真实值 num_{actual} 与平均预测值 $num_{predicated}$. 其中,列 2～列 13 分别给出了不同缺陷数目下的平均预测结果. 分析结果表明:当真实缺陷数目较少时,在多数情形下预测结果数目要高于真实结果,这是由于程序内在的逻辑结构复杂性及输出位置多样性,使得错误状态可以通过多条输出途径,由此导致同一缺陷的失败测试用例呈现出多种覆盖特征;然而,随着缺陷数量的持续增加,预测结果与真实结果的差异逐渐

表 10　缺陷数目的真实结果与平均预测结果对比

num_{actual}	$num_{predicated}$											
	JT	TI	S1	S2	PT1	PT2	N1	N2	N3	N5	S5	S7
2	4.69	3.63	3.89	3.80	2.05	2.07	3.87	3.66	4.36	2.95	2.00	2.00
3	5.71	4.40	4.22	4.27	2.23	2.26	4.25	4.00	5.32	3.54	—	—
4	5.80	5.00	4.38	4.56	2.80	2.61	4.53	7.00	5.83	4.66	—	—
5	6.15	5.78	4.71	5.36	3.36	3.10	5.17	—	6.20	5.57	—	—
6	7.44	6.68	5.92	7.00	4.00	3.73	5.00	—	7.07	6.14	—	—
7	7.95	7.23	6.50	—	—	4.32	—	—	7.56	6.00	—	—
8	8.23	8.05	—	—	—	4.75	—	—	7.84	—	—	—
9	8.53	8.53	—	—	—	—	—	—	—	—	—	—
10	8.54	9.13	—	—	—	—	—	—	—	—	—	—
相关系数	0.97	0.99	0.95	0.94	0.98	0.99	0.94	0.99	0.99	0.96	—	—
P-value	<0.001	<0.001	0.003	0.016	0.002	<0.001	0.016	0.044	<0.001	0.002	—	—

缩小，甚至存在预测结果低于真实结果的现象，这是因为随着缺陷数目增加，不同缺陷可能存在于同一缺陷模块或处于同一数据流、控制流中，此时它们具备相同的覆盖特征，因此会隶属于相同的聚类集合，从而降低了真实结果与预测结果的差异.

进一步，本实验还采用了 Pearson 相关系数度量了缺陷数目真实结果与预测结果的相关性. 表 10 列 2～列 11 最后 2 行分别给出了多缺陷版本数多于一的程序中真实结果与预测结果的相关强度与相关显著度. 可以看到，对于每个程序，其相关系数均高于 0.8，P-value 均低于 0.05，由此表明真实结果与预测结果存在显著正相关性[44].

(4)针对实验 4 的结果分析

本实验给出了 NCMFL 在单缺陷和多缺陷程序上的缺陷定位结果. 由于 Siemens 和 SIR 代码规模不同(Siemens 为小规模 Java 程序，SIR 为中等规模 Java 程序)[4]、测试用例数量不同(Siemens 测试用例数量较多，SIR 测试用例数量较少)，本文对 Siemens 程序和 SIR 程序分别进行分析.

表 11 和表 12 分别给出了不同缺陷定位方法在单缺陷和多缺陷 Siemens 程序上的平均缺陷定位代价. 其中，列 2～列 5 分别表示 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 等方法的平均缺陷代价. 与单缺陷定位结果相比，T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 在多缺陷定位过程中平均缺陷定位代价分别增加了 4.42%、6.2%、34.02% 和 3.64%. 其中，$N1^{NCM}$ 的缺陷定位代价增加幅度最大，这是因 Naish 默认缺陷应当被所有失败运行覆盖，对未出现在所有失败运行中的语句强制赋予一个最低的可疑度. 在单缺陷程序中，由于缺陷确实处于所有失败运行中，$N1^{NCM}$ 是适用的. 然而，在多缺陷程序中，由于存在 E-IPI、RI 等干扰，缺陷语句未必会出现在所有失败运行中，此时 $N1^{NCM}$ 不再适用. 因此，虽然 $N1^{NCM}$ 在单缺陷定位过程中平均缺陷定位代价最低，但在多缺陷定位过程中平均缺陷定位代价最高. Wong 以语句被失败测试用例覆盖的频次作为语句可疑度，该方法只考虑了失败测试用例对语句可疑度的影响，而未考虑成功测试用例对语句可疑度的影响. 因此，$W1^{NCM}$ 在定位单缺陷 Siemens 程序时效果较差，在定位多缺陷 Siemens 程序时效果依然较差.

表 11 不同方法在单缺陷 Siemens 程序上的缺陷定位代价

程序	T^{NCM}	O^{NCM}	$N1^{NCM}$	$W1^{NCM}$	T^{CM}	O^{CM}	$N1^{CM}$	$W1^{CM}$	T^{FCM}	O^{FCM}	$N1^{FCM}$	$W1^{FCM}$
JT	39.13	35.60	28.21	64.82	36.45	34.78	27.50	56.66	34.10	36.62	28.70	55.42
TI	28.62	18.26	11.75	46.57	31.80	30.40	21.61	53.99	33.54	24.04	13.52	51.91
S1	7.44	6.08	5.21	61.04	12.97	12.44	7.86	70.53	13.84	13.97	8.78	64.29
S2	51.09	47.11	39.96	60.54	46.67	41.17	41.56	59.79	48.55	38.01	36.32	54.39
PT1	28.57	19.78	11.83	30.94	22.57	18.02	9.83	28.15	22.40	16.25	17.32	16.49
PT2	25.35	18.78	8.86	41.61	12.93	14.41	9.79	31.70	13.47	14.01	12.90	34.91
ave	30.03	24.27	17.64	50.92	27.23	25.20	19.69	50.14	27.65	23.82	19.59	46.23

表 12 不同方法在多缺陷 Siemens 程序上的缺陷定位代价

程序	T^{NCM}	O^{NCM}	$N1^{NCM}$	$W1^{NCM}$	T^{CM}	O^{CM}	$N1^{CM}$	$W1^{CM}$	T^{FCM}	O^{FCM}	$N1^{FCM}$	$W1^{FCM}$
JT	41.51	37.57	39.92	69.81	41.59	37.88	41.36	55.20	35.91	35.41	39.13	54.27
TI	33.94	28.83	61.30	51.34	39.06	37.26	41.23	60.01	33.80	24.30	35.12	61.47
S1	16.39	16.07	63.63	64.47	29.80	38.61	30.62	71.17	13.01	16.34	22.40	72.62
S2	52.17	49.78	46.56	67.24	50.68	52.82	49.99	69.71	48.59	44.68	44.12	59.13
PT1	30.48	24.55	59.59	31.85	27.53	30.90	38.70	31.12	24.08	19.84	38.39	28.19
PT2	32.24	26.01	38.93	42.66	28.63	31.58	32.86	32.32	15.84	15.31	21.54	30.42
ave	34.46	30.47	51.66	54.56	36.22	38.17	39.13	53.26	28.54	25.98	33.45	51.02

表 13 和表 14 分别给出了不同缺陷定位方法在单缺陷和多缺陷 SIR 程序上的平均缺陷定位代价. 其中，列 2～列 5 分别表示 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 等方法的平均缺陷代价. 可以看到，与 Siemens 程序相比，T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 在 SIR 程序上的平均缺陷定位代价较低. 这是因为 SIR 程序规模较大，在程序运行过程中不会覆盖大量的程序语句. 此外，NanoXML 和 Siena 程序的测试用例均为单元测试用例，程序每次运行涉及的功能并不多. 因此，成功运行和失败运行的语句覆盖结果存在较大的差异，缺陷定位代价相对较低. 而 Siemens 程序规模较小，在运行时可能会运行大量的程序语句，由此造成成功运行和失败运行的语句覆盖结果差异较小，缺陷定位代价相对较高. 与单缺陷定位结果相比，T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$ 在多缺陷定位过程中平均缺陷定位代价分别增加了 3.30%、3.97%、42.12%、2.58%. 可以看到，$N1^{NCM}$ 的缺陷定位代价增加幅度依然最大，其原因与其在 Siemens 程序上的定位结

表 13　不同方法在单缺陷 SIR 程序上的缺陷定位代价

程序	T^{NCM}	O^{NCM}	$N1^{NCM}$	$W1^{NCM}$	T^{CM}	O^{CM}	$N1^{CM}$	$W1^{CM}$	T^{FCM}	O^{FCM}	$N1^{FCM}$	$W1^{FCM}$
N1	6.50	3.02	2.12	13.55	5.60	3.41	3.29	14.66	5.76	4.38	2.70	13.76
N2	4.16	1.56	0.92	7.57	3.98	3.05	4.22	8.32	2.50	2.60	2.18	7.70
N3	4.57	4.07	3.49	17.37	6.59	4.04	3.62	10.37	5.78	4.03	3.21	15.19
N5	9.26	2.12	1.35	11.30	5.37	2.76	2.67	13.53	6.13	4.63	1.26	1.60
S3	6.20	1.84	1.84	8.08	6.01	2.95	4.63	6.71	4.10	1.29	2.13	4.80
S5	5.10	3.43	3.43	6.05	4.54	4.50	3.51	5.97	3.51	2.68	3.08	4.13
ave	5.97	2.67	2.19	10.65	5.35	3.45	3.66	9.93	4.63	3.27	2.43	7.86

表 14　不同方法在多缺陷 SIR 程序上的缺陷定位代价

程序	T^{NCM}	O^{NCM}	$N1^{NCM}$	$W1^{NCM}$	T^{CM}	O^{CM}	$N1^{CM}$	$W1^{CM}$	T^{FCM}	O^{FCM}	$N1^{FCM}$	$W1^{FCM}$
N1	18.79	14.16	48.66	20.04	14.66	14.83	14.96	18.26	12.56	11.91	13.81	16.09
N2	5.71	2.61	55.92	8.39	5.25	3.69	8.25	13.85	4.32	2.81	9.30	8.88
N3	6.62	7.96	68.66	19.21	8.65	7.61	11.91	19.44	6.57	5.52	8.80	18.34
N5	13.04	9.32	86.87	17.04	12.62	10.13	13.98	18.10	6.01	6.86	8.77	16.54
S3	6.21	1.86	1.86	8.10	5.81	4.95	5.81	7.87	4.46	1.61	4.10	4.10
S5	5.22	3.97	3.90	6.59	5.59	4.54	4.54	6.96	4.82	3.51	3.51	5.86
ave	9.27	6.65	44.31	13.23	8.76	7.63	9.91	14.08	6.46	5.37	8.05	11.63

果不好的原因是相同的.

为进一步分析缺陷干扰对 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$等 4 种方法的影响,本文对这些方法在单缺陷和多缺陷程序上的平均缺陷定位代价开展 Wilcoxon 配对检验,给出的原假设为:在单缺陷程序上的缺陷定位平均代价与在多缺陷程序上的缺陷定位平均代价无显著差异. 在 Siemens 程序上,Wilcoxon 检验所得 p 值分别为 0.009、0.004、0.006、0.006;在 SIR 程序上,Wilcoxon 检验所得 p 值分别为 0.070、0.039、0.016、0.038. 可以看到,在多数情况下 p 值低于 0.05,此时应当拒绝原假设,单缺陷程序上的缺陷定位平均代价与多缺陷程序上的缺陷定位平均代价存在显著性的差异. 也就是说,缺陷干扰造成 T^{NCM}、O^{NCM}、$N1^{NCM}$、$W1^{NCM}$等方法的缺陷定位代价显著增加.

(5)针对实验 5 的结果分析

本实验给出了 CMFL 在单缺陷和多缺陷程序上的缺陷定位结果. 与实验 4 相同,我们对 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$方法在 Siemens 和 SIR 程序上的缺陷定位结果分别进行分析.

表 11 和表 12 中列 6～列 9 分别给出了 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$等方法在单缺陷和多缺陷 Siemens 程序上的平均缺陷定位代价. 可以看到,与单缺陷结果相比,T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$在多缺陷定位过程中平均缺陷定位代价分别增加了 8.99%、12.97%、19.43%、3.12%. 表 13 和表 14 中列 6～列 9 分别给出了 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$等方法在单缺陷和多缺陷 SIR 程序上的平均缺陷定位代价. 可以看到,与单缺陷结果相比,T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$在多缺陷定位过程中平均缺陷定位代价分别增加了 3.41%、4.17%、6.25%、4.15%.

总体上,与单缺陷定位结果相比,T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$在多缺陷定位过程中平均缺陷定位代价均有增加. 同实验 4,对这些方法在单缺陷和多缺陷程序上的平均缺陷定位代价开展 Wilcoxon 配对检验,给出的原假设为:在单缺陷程序上的缺陷定位平均代价与在多缺陷程序上的缺陷定位平均代价无显著差异. 在 Siemens 程序上,Wilcoxon 检验所得 p 值分别为 0.006、0.006、0.001、0.064;在 SIR 程序上,Wilcoxon 检验所得 p 值分别为 0.039、0.034、0.013、0.010. 可以看到,在多数情况下 p 值低于 0.05,此时应当拒绝原假设,单缺陷程序上的缺陷定位平均代价与多缺陷程序上的缺陷定位平均代价存在显著性的差异. 也就是说,缺陷干扰造成 T^{CM}、O^{CM}、$N1^{CM}$、$W1^{CM}$等方法的缺陷定位代价显著增加.

(6)针对实验 6 的结果分析

本实验给出 FCMFL 在单缺陷和多缺陷程序上的缺陷定位结果. 与实验 4 相同,我们对 T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$方法在 Siemens 和 SIR 程序上的缺陷定位结果分别进行分析.

表 11 和表 12 中列 10～列 13 分别给出了 T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 等方法在单缺陷和多缺陷 Siemens 程序上的平均缺陷定位代价. 可以看到,与单缺陷结果相比,T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$在多缺陷定位过程中平均缺陷定位代价分别增加了 0.89%、

2.16%、13.86%、4.78%. 表 13 和表 14 中列 10～列 13 分别给出了 T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 等方法在单缺陷和多缺陷 SIR 程序上的平均缺陷定位代价. 可以看到,与单缺陷结果相比,T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 在多缺陷定位过程中平均缺陷定位代价分别增加了 1.83%、2.10%、5.62%、3.77%.

总体上,与单缺陷定位结果相比,T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 在多缺陷定位过程中平均缺陷定位代价均有增加,但与 NCMFL、CMFL 等方法相比,增加幅度较低. 同实验 4 和实验 5,对这些方法在单缺陷程序和多缺陷程序上的平均缺陷定位代价开展 Wilcoxon 配对检验,给出的原假设为:在单缺陷程序上的缺陷定位平均代价与在多缺陷程序上的缺陷定位平均代价无显著差异. 在 Siemens 程序上,Wilcoxon 检验所得 p 值分别为 0.070、0.057、0.001、0.063;在 SIR 程序上,Wilcoxon 检验所得 p 值分别为 0.069、0.061、0.008、0.081. 可以看到在多数情况下 p 值高于 0.05,此时应当接受原假设,在单缺陷程序上缺陷定位平均代价与在多缺陷程序上的缺陷定位平均代价不存在显著性的差异. 也就是说,与单缺陷程序相比,T^{FCM}、O^{FCM}、$N1^{FCM}$、$W1^{FCM}$ 在多缺陷程序上的缺陷定位代价并未显著增加.

(7)针对实验 7 的结果分析

本实验对 NCMFL、CMFL 与在本文提出的 FCMFL 方法进行对比. 与实验 4 相同,我们对不同缺陷定位方法在 Siemens 和 SIR 程序上的缺陷定位结果分别进行分析.

表 11～表 14 分别给出了各个方法在单缺陷 Siemens、多缺陷 Siemens、单缺陷 SIR、多缺陷 SIR 程序上的缺陷定位结果. 对于单缺陷程序,$N1^{NCM}$、$N1^{CM}$、$N1^{FCM}$ 方法始终具有最好的缺陷定位效果. 对于多缺陷程序,$N1^{NCM}$ 的缺陷定位效果显著降低,而 $N1^{CM}$ 和 $N1^{FCM}$ 的缺陷定位能力虽然也有大幅降低,但依然具有一定的缺陷定位效果. O^{NCM}、O^{CM}、O^{FCM} 方法始终具有良好的缺陷定位效果,特别是在多缺陷程序中,O^{NCM}、O^{CM}、O^{FCM} 在多数情况下优于其他方法.

对于单缺陷程序,FCMFL 与 NCMFL、CMFL 的定位结果并不存在明显差异:对于 Siemens 程序,$N1^{NCM}$ 优于 $N1^{CM}$、$N1^{FCM}$ 方法,T^{CM} 优于 T^{NCM}、T^{FCM} 方法,O^{FCM} 和 $W1^{FCM}$ 方法分别优于 O^{NCM}、O^{CM} 方法和 $W1^{NCM}$、$W1^{CM}$ 方法;对于 SIR 程序,O^{NCM} 和 $N1^{NCM}$ 方法分别优于 O^{CM}、O^{FCM} 方法和 $N1^{CM}$、$N1^{NCM}$ 方法,T^{FCM} 方法和 $W1^{FCM}$ 方法分别优于 T^{NCM}、T^{CM} 方法和 $W1^{NCM}$、$W1^{CM}$ 方法. 为进一步比较 FCMFL 与 NCMFL、CMFL 方法的缺陷定位效果,对上述方法开展 Wilcoxon 配对检验. 给出的原假设为:方法 T^{FCM} 与 T^{CM} 及 T^{NCM}、方法 O^{FCM} 与 O^{CM} 及 O^{NCM}、方法 $N1^{FCM}$ 与 $N1^{CM}$ 及 $N1^{NCM}$、方法 $W1^{FCM}$ 与 $W1^{CM}$ 及 $W1^{NCM}$ 在单缺陷程序上的缺陷定位代价无显著性差异. 表 15 给出了各个方法的 Wilcoxon 配对检验结果. 其中列 2 和列 3 分别给出了各个方法在单缺陷 Siemens 和 SIR 程序上的配对检验结果. 可以看到,在多数情况下 FCMFL 与对比方法的 p 值均高于 0.05,此时应当接受原假设,即 FCMFL 方法与对比方法在单缺陷程序上的缺陷定位代价不存在显著差异.

表 15 Wilcoxon 配对检验结果(P-value)

FL^1 vs. FL^2	单缺陷		多缺陷	
	Siemens	SIR	Siemens	SIR
T^{NCM} vs. T^{CM}	0.169	0.232	0.268	0.285
O^{NCM} vs. O^{CM}	0.377	0.009	0.030	0.045
$N1^{NCM}$ vs. $N1^{CM}$	0.141	0.022	0.043	0.021
$W1^{NCM}$ vs. $W1^{CM}$	0.409	0.306	0.373	0.217
T^{NCM} vs. T^{FCM}	0.220	0.038	0.024	0.036
O^{NCM} vs. O^{FCM}	0.436	0.150	0.015	0.026
$N1^{NCM}$ vs. $N1^{FCM}$	0.101	0.196	0.016	0.020
$W1^{NCM}$ vs. $W1^{FCM}$	0.095	0.060	0.225	0.048
T^{CM} vs. T^{FCM}	0.270	0.070	0.011	0.024
O^{CM} vs. O^{FCM}	0.161	0.385	0.004	0.002
$N1^{CM}$ vs. $N1^{FCM}$	0.483	0.010	0.009	0.042
$W1^{CM}$ vs. $W1^{FCM}$	0.059	0.198	0.087	0.007

对于多缺陷程序,与 NCMFL 和 CMFL 方法相比,本文 FCMFL 方法均具有更好的平均缺陷定位效果. 同样地,对上述方法开展 Wilcoxon 配对检验. 给出的原假设为:方法 T^{FCM} 与 T^{CM} 及 T^{NCM}、方法 O^{FCM} 与 O^{CM} 及 O^{NCM}、方法 $N1^{FCM}$ 与 $N1^{CM}$ 及 $N1^{NCM}$、方法 $W1^{FCM}$ 与 $W1^{CM}$ 及 $W1^{NCM}$ 在多缺陷程序上的缺陷定位代价无显著性差异. 表 15 列 4 和列 5 分别给出了各个方法在多缺陷 Siemens 和 SIR 程序上的配对检验结果. 可以看到,在多数情况下 FCMFL 与对比方法的 p 值低于 0.05,此时应当拒绝原假设,即 FCMFL 方法的缺陷定位效果显著优于对比方法. 在其他情况下(除多缺陷 Siemens 程序下 $W1^{NCM}$ vs. $W1^{FCM}$),我们可以在置信度水平 10%的情况下拒绝原假设. 由此可知,本文 FCMFL 方法在多缺陷程序上具有更好的缺陷定位效果.

以上实验结果表明 FCMFL 方法显著优于 CMFL 和 NCMFL 方法. NCMFL 直接以覆盖矩阵和结果向量作为输入计算语句可疑度并排序,未对

测试用例进行聚类，忽略了缺陷干扰，因此定位效果不佳. CMFL 加入 CM 聚类结果来进行可疑度计算及排序. 然而现实对象没有严格属性，在性态和类属方面存在中介性，因而使用 CM 聚类方法对测试用例聚类并将其强制划分到某一类别不能客观描述失败测试用例与不同缺陷的关系，导致定位效果不佳. FCMFL 加入 FCM 聚类结果来进行可疑度计算及排序，FCM 聚类将 CM 聚类方法通过模糊集理论推广到模糊情形中，有效说明每个对象与聚类子集间的隶属关系，因此在 3 种方法中效果最好.

6.5　实验结论

通过第 6.4 节中对实验结果的观察与分析，我们初步得到 7 点结论.

结论 1. 通过实验 1 的分析，我们观察到：RI 干扰(10.04%)较 E-IPI 干扰(3.50%)更为频繁且随着缺陷数量的增加两类干扰的频繁度均逐渐增高. 因此得出结论：缺陷干扰在多缺陷程序中较为普遍.

结论 2. 通过实验 2 的分析，我们观察到：与在单缺陷程序中相比，多缺陷程序中缺陷在 DS 集合中的分布发生了较大的变化. 部分缺陷(42.02%～63.33%)被分配到新的 DS 子集中，即 S_{IV}、S_{V}、S_{VI}、S_{VII}、S_{VIII}和 S_{IX}. 因此得出结论：缺陷干扰对缺陷在 DS 集合中的分布有较大的影响.

结论 3. 通过实验 3 的分析，我们观察到：缺陷真实数目与预测结果存在相同的变化趋势，具有显著性的强相关性. 因此得出结论：本文提出的基于 Xie-Beni 系数的预测方法可以有效判断失败测试用例集的划分类别.

结论 4. 通过实验 4 的分析，我们观察到：与单缺陷程序相比，$\mathrm{T}^{\mathrm{NCM}}$、$\mathrm{O}^{\mathrm{NCM}}$、$\mathrm{N1}^{\mathrm{NCM}}$、$\mathrm{W1}^{\mathrm{NCM}}$等 4 种方法在多缺陷程序中的缺陷定位代价显著增高. 因此得出结论：多缺陷对 NCMFL 方法存在较大的负面影响，显著增加了缺陷定位代价.

结论 5. 通过实验 5 的分析，我们观察到：与单缺陷程序相比，T^{CM}、O^{CM}、$\mathrm{N1}^{\mathrm{CM}}$、$\mathrm{W1}^{\mathrm{CM}}$等 4 种方法在多缺陷程序中的缺陷定位代价显著增高. 因此得出结论：多缺陷对 CMFL 方法存在较大的负面影响，显著增加了缺陷定位代价.

结论 6. 通过实验 6 的分析，我们观察到：与单缺陷程序相比，$\mathrm{T}^{\mathrm{FCM}}$、$\mathrm{O}^{\mathrm{FCM}}$、$\mathrm{N1}^{\mathrm{FCM}}$、$\mathrm{W1}^{\mathrm{FCM}}$等 4 种方法在多缺陷程序中的缺陷定位代价更高，但在多数情况下不存在显著性的差异. 因此得出结论：多缺陷对 FCMFL 方法存在一定的负面影响，但不会显著增加缺陷定位代价.

结论 7. 通过实验 7 的分析，我们观察到：与 NCMFL 和 CMFL 方法相比，FCMFL 方法在单缺陷程序缺陷定位中具有较为相似的缺陷定位效果，在多缺陷程序缺陷定位中具有更好的缺陷定位效果. 因此得出结论：本文提出的 FCMFL 方法具有更好的缺陷定位效果.

6.6　有效性分析

本节从内部效度、外部效度、构造效度等方面来分析本文内容的有效性.

影响本文内容内部效度的因素主要包括 4 个方面. 首先，聚类分析算法和缺陷定位方法的实现准确度对实验结果存在影响. 为此，本文选择经典的 FCM 聚类方法[28]开展聚类分析，选用 4 种经典的 SBFL 方法(Tarantula[18]、Ochiai[19]、Naish[20]和 Wong[21])作为实验对比对象，并严格依据相关文献给出的算法描述进行方法复现，因此方法实现是准确的. 其次，实验所用的多缺陷版本对本文结论有影响. 同一缺陷在不同多缺陷版本下在 DS 集合中的分布和缺陷定位结果是不同的. 对此，针对每个程序我们利用已有缺陷进行组合生成数十至数百个不同数量的多缺陷版本来进行缺陷定位研究. 这在一定程度上降低了对该因素的威胁. 再次，测试用例集的构成对可疑度计算结果存在影响[4]. 对此，本文按照文献[6]的要求，在实证研究中选择实验对象自带的且满足语句覆盖的测试用例集，保证实验程序中的每一条语句都参与到评价过程中，避免缺陷的遗漏. 最后，E-IPI 干扰的识别结果对实验结果有影响. 本文第 6.1 节分析表明在多缺陷程序中并非所有的失败 E-IP 过程都可被直接识别，因此部分 E-IPI 干扰也不能被直接识别. 然而，该结果并不影响干扰效应对 SBFL 方法的影响分析. 后续研究中，我们将进一步通过依赖性分析、人工检查等手段判断这部分失败 E-IP 过程.

影响本文内容外部效度的因素主要包括 4 个方面. 首先，实验对象规模及数量等因素对本文结论的可推广性有影响. 本文选用 6 个小规模程序和 6 个中等规模程序作为实验对象[16-17]，缺乏大规模实验对象，因此难以保证本文结论的推广性. 为此，本文选择在缺陷定位研究领域被广泛使用的真实缺陷程序作为实验对象[4,32]，这在一定程度上弥补了不足. 实验结果表明：本文方法在中小规模的 Java 程序上具有较好的缺陷定位效果. 其次，人工注入缺陷对

本文结论的可推广性同样有影响.本文研究的单缺陷程序和多缺陷程序中的缺陷均为人工注入缺陷,人工注入的缺陷均为实验程序中真实存在的缺陷.尽管这些缺陷并非原发缺陷,但是这些缺陷均为研究人员精心设计[17],与原发缺陷对缺陷定位方法的影响一致[45].在接下来的研究中,我们将引入更多真实的缺陷来进一步验证本文分析的有效性.再次,最大聚类数目的设定对最优聚类结果的生成以及所提方法的有效性存在影响.本文从实证研究的角度出发选择 10 作为最大聚类数目,而对于更大更复杂的程序则应设置更高的最大聚类数目.未来我们将结合软件缺陷预测技术[46]提出更合理的最大聚类数目设置方法.最后,动态切片的时间成本对本文方法的可扩展性存在影响.对此,本文选择并实现了一种基于前向计算的动态切片方法[43,47].实验结果表明[39]:在中大规模程序[4]上该算法依然具有较低的时间成本,这在一定程度上降低了该因素的影响.

影响本文方法构造效度的因素主要在于本文采用的缺陷定位评价方法.本文采用 2 个缺陷定位研究中常用的评价指标[3-4,48]定位代价 Expense 来分别评价某个缺陷定位方法在某个缺陷上和所有缺陷上的缺陷定位效果,因此可以有效评价不同方法的缺陷定位效果.

7 相关工作

本文相关工作主要包括基于程序切片的缺陷定位方法和基于频谱信息的缺陷定位方法.

基于程序切片的方法通过分析程序内部数据流和控制流信息,识别引发程序运行失败的语句集合来缩小缺陷的搜索范围.Weiser[24]首先提出了程序切片的概念并提出了基于数据流方程的静态切片方法.然而,静态切片结果规模庞大,难以适用于真实的程序调试环境.对此,Korel[25]提出了动态切片方法,有效减小了切片规模.Zhang 等人[49]提出了带有置信度的动态切片剪枝策略,进一步缩小了动态切片的规模.Mao 等人[50]结合静态切片与运行切片提出了近似动态切片,有效地提高了切片计算的效率.许高阳等人[51]提出了一种适用于面向对象程序的层次切片技术,拓展了切片技术的应用范围.Gong 等人[52]利用程序中依赖信息构建状态依赖分析模型,有效提高了缺陷定位的精度.程序运行失效表明程序运行了缺陷语句并传播了缺陷状态,不同缺陷引发的程序运行失败传播途径不同.动态切片可以分析和识别程序中状态的传播和语句的运行情况.因此,本文采用动态切片技术识别对程序运行失败存在影响的语句集合,并对其进行聚类分析识别每个缺陷的影响测试用例集合.具体地,本文采用了一种基于前向计算的动态切片技术[43,47]完成失败测试执行的动态切片工作.

基于频谱信息的方法分析程序实体在成功运行和失败运行中的状态差异,生成供软件开发人员参照的可疑序列.Jones 等人[18]认为语句包含缺陷的可疑度与其在失败运行中被覆盖的次数正相关,由此提出了 Tarantula 方法.随后,研究人员提出了一系列可疑度度量方法(如 Ochiai[19]、Naish 等人[20]、Wong 等人[21])来提高语句可疑度度量的精度.Xie 等人[6]对这些度量方法进行了理论分析,并证明部分方法定位结果最优.文献[32-33]则分析了偶然正确性对 SBFL 方法缺陷定位有效性的负面影响,并分别提出了基于硬聚类算法和软聚类算法的偶然正确测试用例识别方法.研究及实证结果表明:上述方法可以提供有效的指导信息,帮助开发人员快速定位程序缺陷.然而,当程序中包含多个缺陷时,缺陷语句的覆盖结果可能会受到其它缺陷的干扰,降低缺陷语句的可疑度和检查排名.

部分研究人员对基于频谱信息的多缺陷定位技术进行了研究.Podgurski 等人[7,11]采用多变量映射对失败的测试用例进行分析,从而识别每个缺陷的相关测试用例集.Liu 等人[8]采用距离度量完成聚类分析.DiGiuseppe 等人[53-54]实验验证了缺陷数量和缺陷类型对缺陷定位效果的影响.实验结果表明多个缺陷会降低 SBFL 方法的缺陷定位效率,但影响程度较低.此外,缺陷类型不会影响 SBFL 方法的缺陷定位效率.Zhang 等人[55]则通过示例分析了缺陷干扰对基于谓词的 SBFL 方法缺陷定位有效性的负面影响,但并未提出相应的处置策略.Lamraoui 等人[56]基于 SBFL 方法实现了 SINPER 工具帮助识别多缺陷.Feyzi 等人[57]综合考虑了程序的静态与动态特征,根据语句对程序达到终止状态的影响力对语句排序,提出了 FPA-FL 方法.在多缺陷程序上的实验结果表明该方法能有效识别导致程序执行失败的因果链.Ju 等人[58]提出了一种新的在多缺陷情况下评价 SBFL 中可疑度计算公式的理论分析方法,同时结合实例分析多缺陷情况下 SBFL 方法的效率.上述研究给出了多缺陷程序缺陷定位方法,

并取得了较好的缺陷定位效果.然而,上述研究并未对缺陷间的干扰现象进行分析.本文通过分析缺陷语句在单缺陷程序 DS 集合和多缺陷程序 DS 集合的位置变化以及不同可疑度度量方法对 DS 集合中语句的可疑度评价,说明了多缺陷对 SBFL 方法产生影响的原因.

此外,无论在本文所研究的多缺陷定位领域,还是在数据挖掘领域,缺陷(聚类)数目的确定始终是一项难题,因而也引起研究人员的广泛关注.在多缺陷定位领域,Mardia 等人[7,56]选择 $\sqrt{n_f/2}$作为聚类数目,也有研究人员直接选择一定比例(如 5%)失败测试用例的数目作为聚类数目.可以看到,这些方法的分析结果与失败测试用例的数目密切相关,失败测试用例越多,聚类的数目越多.然而,失败测试用例的数目会受测试用例集规模的影响:相同分布情况下测试用例集的规模越大,失败测试用例的数目越多,从而使得聚类数目具有极大的不稳定性.在数据挖掘领域,Chen 等人[57]和 Mok 等人[58]分别提出了模糊聚类算法和非模糊聚类算法场景下聚类数目的自动确定方法.但与本文方法相同,这些方法同样需要用户提供最大聚类数目,也未提供最大聚类数目选择的指导方法.因此,聚类数目的自动确定在实际应用方面依然存在挑战.软件缺陷预测技术提供了一条可行途径:根据软件历史开发数据以及已发现的缺陷,借助机器学习等方法来预测软件项目中的缺陷数目和类型[46].因此,项目开发人员在缺陷定位前,可通过软件缺陷预测技术估算缺陷的数量,进而选定最大聚类数目.

8　结束语

本文对缺陷间的干扰现象进行了研究,通过构建刻画多缺陷程序运行过程的 E-IP 模型,分析了多缺陷对程序运行过程以及运行结果的干扰,并进一步分析说明了多缺陷对 SBFL 方法缺陷定位效率的影响.在此基础上,本文提出了一种基于 FCM 聚类的多缺陷定位方法 FCMFL,该方法通过 FCM 聚类来分析每个失败测试用例在不同缺陷中的隶属程度,在可疑度计算时根据隶属程度定义不同测试用例的权重,从而降低不相关失败测试用例的干扰影响.实验结果表明:缺陷间的相互干扰会对 SBFL 方法产生影响,降低 SBFL 方法的缺陷定位效率;本文提出的 FCMFL 方法可以降低多缺陷对 SBFL 方法的干扰,提高 SBFL 方法的缺陷定位效率.未来,我们将借助程序结构识别和消除更多的多缺陷定位干扰因素,借助软件缺陷预测选择更加合理的最大聚类数目,而研究不同聚类算法(如层次聚类、密度聚类)以及聚类依赖(如语句覆盖信息、程序执行轨迹)对缺陷划分结果以及缺陷定位效果的影响也是我们下一步工作的重点.

参　考　文　献

[1] Jones J A, Bowring J F, Harrold M J. Debugging in parallel//Proceedings of the 16th International Symposium on Software Testing and Analysis. London, UK, 2007: 16-26

[2] Vessey I. Expertise in debugging computer programs: A process analysis. International Journal of Man-Machine Studies, 1985, 23(5): 459-494

[3] Wang Ke-Chao, Wang Tian-Tian, Su Xiao-Hong, et al. 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, 苏小红等. 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)

[4] Chen Xiang, Ju Xiao-Lin, Wen Wan-Zhi, et al. 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, 文万志等. 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)

[5] 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

[6] Xie X Y, Chen T Y, Kuo F C, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering & Methodology, 2013, 22(4): 1-31

[7] Dickinson W, Leon D, Podgurski A. Finding failures by cluster analysis of execution profiles//Proceedings of the 23rd International Conference on Software Engineering. Toronto, Canada, 2001: 339-348

[8] Liu C, Han J W. Failure proximity: A fault localization-based approach//Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Portland, USA, 2006: 286-295

[9] Liu C, Zhang X Y, Han J W. A systematic study of failure proximity. IEEE Transactions on Software Engineering, 2008, 34(6): 826-843

[10] Zheng A X, Jordan M I, Liblit B, et al. Statistical debugging: Simultaneous identification of multiple bugs//Proceedings of the 23rd International Conference on Machine Learning. Pennsylvania, USA, 2006: 1105-1112

[11] Podgurski A, Leon D, Francis P, et al. Automated support for classifying software failure reports//Proceedings of the 25th International Conference on Software Engineering. Portland, USA, 2003: 465-475

[12] Dean B C, Pressly W B, Malloy B A, et al. A linear programming approach for automated localization of multiple faults//Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 640-644

[13] Gao R Z, Wong W E. MSeer—An advanced technique for locating multiple bugs in parallel. Journal, 10.1109/TSE.2017.2776912

[14] Wang Zan, Fan Xiang-Yu, Zou Yu-Guo, et al. GAMFal: A genetic algorithm based multiple faults localization technique. Journal of Software, 2016, 27(4): 1-23(in Chinese)
(王赞, 樊向宇, 邹雨果等. 一种基于遗传算法的多缺陷定位方法. 软件学报, 2016, 27(4): 1-23)

[15] Debroy V, Wong W E. Insights on fault interference for programs with multiple bugs//Procedings of the 20th International Conference on Software Reliability Engineering. Mysuru, India, 2009: 165-174

[16] Santelices R, Jones J A, Yu Y B, et al. Lightweight fault-localization using multiple coverage types//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 56-66

[17] 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

[18] Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization//Proceedings of the 24th International Conference on Software Engineering. Orlando, USA, 2002: 467-477

[19] Abreu R, Zoeteweij P, van Gemund A J C. An evaluation of similarity coefficients for software fault localization//Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing. California, USA, 2006: 39-46

[20] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering & Methodology, 2011, 20(3): 563-574

[21] Wong W E, Qi Y, Zhao L, et al. Effective fault localization using code coverage//Proceedings of the 31st Annual International Computer Software and Applications Conference. Beijing, China, 2007: 449-456

[22] Hao Peng, Zheng Zheng, Zhang Zhen-Yu, et al. Self-adaptive fault localization algorithm based on predicate execution information analysis. Chinese Journal of Computers, 2014, 37(3): 500-511(in Chinese)
(郝鹏, 郑征, 张震宇等. 基于谓词执行信息分析的自适应缺陷定位算法. 计算机学报, 2014, 37(3): 500-511)

[23] Wang Xing-Ya, Jiang Shu-Juan, Ju Xiao-Lin, et al. Effective fault localization technique using parameter-value replacement. Journal on Communications, 2015, 36(4): 137-145(in Chinese)
(王兴亚, 姜淑娟, 鞠小林等. 基于参数-值替换的错误定位方法. 通信学报, 2015, 36(4): 137-145)

[24] Weiser M. Program slicing. IEEE Transactions on Software Engineering, 1984, 10(4): 352-357

[25] Korel B. Dynamic program slicing. Information Processing Letters, 1988, 29(88): 155-163

[26] Han J W, Kamber M, Pei J. Data Mining: Concepts and Techniques. 3rd Edition. San Francisco, USA: Morgan Kaufmann, 2012

[27] Hartigan J A, Wong M A. A *K*-means clustering algorithm. Applied Statistics, 1979, 28(1): 100-108

[28] Bezdek J C, Ehrlich R, Full W. FCM: The fuzzy c-means clustering algorithm. Computers & Geosciences, 1984, 10(2&3): 191-203

[29] Xie X L, Beni G. A validity measure for fuzzy clustering. IEEE Transactions on Pattern Analysis & Machine Intelligence, 1991, 13(8): 841-847

[30] Voas J M. PIE: A dynamic failure-based technique. IEEE Transactions on Software Engineering, 1992, 18(8): 717-727

[31] Ammann P, Offutt J. Introduction to Software Testing. 2nd Edition. Cambridge, UK: Cambridge University Press, 2016

[32] Wes M, Abou A R. Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Transactions on Software Engineering & Methodology, 2014, 23(1): 1-28

[33] Cao He-Ling, Jiang Shu-Juan, Wang Xing-Ya, et al. Identifying coincidental correctness for effective fault localization. Acta Electronica Sinica, 2016, 44(12): 3026-3031(in Chinese)
(曹鹤玲, 姜淑娟, 王兴亚等. 面向有效错误定位的偶然正确性识别方法. 电子学报, 2016, 44(12): 3026-3031)

[34] Wong W E, Debroy V, Golden R, et al. Effective software fault localization using an RBF neural network. IEEE Transactions on Reliability, 2012, 61(1): 149-169

[35] Xu B W, Qian J, Zhang X F, et al. A brief survey of program slicing. ACM Sigsoft Software Engineering Notes, 2005, 30(2): 1-36

[36] Halkidi M, Batistakis Y, Vazirgiannis M. On clustering validation techniques. Journal of Intelligent Information Systems, 2001, 17(2): 107-145

[37] Wong W E, Debroy V, Gao R Z, et al. The DStar method for effective software fault localization. IEEE Transactions on Reliability, 2014, 63(1): 290-308

[38] Wang Xing-Ya, Jiang Shu-Juan, Gao Peng-Fei, et al. Cost-effective testing based fault localization with distance based test-suite reduction. Science China (Information Sciences), 2017, 60(9): 092112:1-092112:15

[39] Ju X L, Jiang S J, Chen X, et al. HSFAL: Effective fault localization using hybrid spectrum of full slices and execution slices. Journal of Systems and Software, 2014, 90(2): 3-17

[40] Tan P N, Steinbach M, Vipin K. Introduction to Data Mining. Boston, USA: Addison Wesley, 2005

[41] Le T, Thung F, Lo D. Theory and practice, do they match? A case with spectrum-based fault localization//Proceedings of the 21st IEEE International Conference on Software Maintenance. Eindhoven, the Netherlands, 2013: 380-383

[42] Pal N R, Bezdek J C. On cluster validity for the fuzzy c-means model. IEEE Transactions on Fuzzy Systems, 2002, 3(3): 370-379

[43] Cao He-Ling, Jiang Shu-Juan, Ju Xiao-Lin, et al. Fault localization based on dynamic slicing and association analysis. Chinese Journal of Computers, 2015, 38(11): 2188-2202(in Chinese)
(曹鹤玲，姜淑娟，鞠小林等. 基于动态切片和关联分析的错误定位方法. 计算机学报，2015，38(11)：2188-2202)

[44] Wuensch K. L. Straightforward statistics for the behavioral sciences. Journal of the American Statistical Association, 1996, 91(436): 1750

[45] Ali S, Andrews J H, Dhandapani T, et al. Evaluating the accuracy of fault localization techniques//Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 76-87

[46] Chen Xiang, Gu Qing, Liu Wang-Shu, et al. Survey of static software defect prediction. Journal of Software, 2016, 27(1): 1-25(in Chinese)
(陈翔，顾庆，刘望舒等. 静态软件缺陷预测方法研究. 软件学报，2016，27(1)：1-25)

[47] Wang Xing-Ya, Jiang Shu-Juan, Ju Xiao-Lin, et al. Dynamic program slicing based on forward computation. Computer Science, 2014, 41(1): 250-253(in Chinese)
(王兴亚，姜淑娟，鞠小林等. 一种基于前向计算的动态程序切片方法. 计算机科学，2014，41(1)：250-253)

[48] Pytlik B, Renieris M, Krishnamurthi S, et al. Automated fault localization using potential invariants//Proceedings of the 5th International Workshop on Automated and Algorithmic Debugging. Ghent, Belgium, 2003: 1-5

[49] Zhang X Y, Gupta N, Gupta R. Pruning dynamic slices with confidence. ACM SIGPLAN Notices, 2006, 41(6): 169-180

[50] Mao X G, Lei Y, Dai Z Y, et al. Slice-based statistical fault localization. Journal of Systems and Software, 2014, 89(2): 51-62

[51] Xu Gao-Yang, Li Bi-Xin, Sun Xiao-Bing, et al. Software fault localization based on hierarchical slicing. Journal of Southeast University(Natural Science Edition), 2010, 40(4): 692-698(in Chinese)
(许高阳，李必信，孙小兵等. 一种基于层次切片的软件错误定位方法. 东南大学学报(自然科学版)，2010，40(4)：692-698)

[52] Gong D D, Su X H, Wang T T, et al. State dependency probabilistic model for fault localization. Information & Software Technology, 2014, 57(1): 430-445

[53] DiGiuseppe N, Jones J A. Fault density, fault types, and spectra-based fault localization. Empirical Software Engineering, 2014, 19(1): 1-40

[54] Digiuseppe N, Jones J A. On the influence of multiple faults on coverage-based fault localization//Proceedings of the 20th International Symposium on Software Testing and Analysis. Toronto, Canada, 2011: 210-220

[55] Zhang Y W, Lo E, Kao B. Evaluation metric for multiple-bug localization with simple and complex predicates//Proceedings of the 19th Asia-Pacific Software Engineering Conference. Hong Kong, China, 2012: 288-293

[56] Huang Y Q, Wu J H, Feng Y, et al. An empirical study on clustering for isolating bugs in fault localization//Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering Workshops. Pasadena, USA, 2013: 138-143

[57] Chen Hai-Peng, Shen Xuan-Jing, Long Jian-Wu, et al. Fuzzy clustering algorithm for automatic identification of clusters. Acta Electronica Sinica, 2017, 45(3): 687-694(in Chinese)
(陈海鹏，申铉京，龙建武等. 自动确定聚类个数的模糊聚类算法. 电子学报，2017，45(3)：687-694)

[58] Mok P Y, Huang H Q, Kwok Y L, et al. A robust adaptive clustering analysis method for automatic identification of clusters. Pattern Recognition, 2012, 45(8): 3017-3033

WANG Xing-Ya, Ph. D. His research interests include program debugging and fault localization.

JIANG Shu-Juan, Ph. D., professor, Ph. D. supervisor. Her research interests include compilation techniques and software engineering.

GAO Peng-Fei, M. S. His research interests include software analysis and testing.

LU Kai, M. S. His research interests focus on fault localization.

BO Li-Li, Ph. D. Her research interests include concurrent program analysis and verification.

JU Xiao-Lin, Ph. D., associate professor. His research interests include software analysis and testing.

ZHANG Yan-Mei, Ph. D., associate professor. Her research interests include software analysis and testing.

Background

Fault localization is a time-consuming but essential step to ensure the quality of software when its execution fails. Kinds of fault localization techniques have been proposed to reduce the burden on developers. Among them, Spectrum-based Fault Localization (SBFL) provides a representative light-weighted technology. It relies on coverage information and executing results to identify the possible scope of faults, thereby reducing the cost of fault localization for developers. Suspiciousness measurement is one critical step in SBFL. Regarding the program with only one fault, intuitively, the more the failed executions covered, the more suspicious the statement is. However, regarding the multi-fault program, fault interference may change the coverage information and executing results, and thus, the assumption, as mentioned earlier, is no longer valid. Therefore, in a multi-fault program, it is hard to analyze the distribution of faults, and accurately measuring the suspiciousness of a statement in the multi-fault program becomes a critical challenge.

Many researchers have studied the problem of multi-fault localization, along with a set of localizing approaches and prototype have been proposed and implemented. However, few of them investigate the fault interference as well as its impact on the effectiveness of SBFL. In this paper, we addressed both problems. First, we propose a novel model (Execution-Infection Propagation, E-IP) to describe the execution condition and the infection & propagation condition for each fault in a multi-fault program, and use it to determine the interference among multiple faults. After that, we provide a theoretical analysis to investigate the impact of fault interference on SBFL. Our analysis indicates that the failed E-IP process $\langle \neg E, \neg IP \rangle$ is the main reason to reduce the effectiveness of SBFL. We also propose FCMFL (Fuzzy C-Means Clustering based Multi-Fault Localization) to mitigate the negative influence of fault interference on SBFL. Our empirical study on 12 Java programs (i. e., 6 Siemens programs and 6 SIR programs) verifies that fault interference influences SBFL and decreases the effectiveness of SBFL, and FCMFL can effectively mitigate the interference, and significantly improve the effectiveness of SBFL.

This work was supported by awards from the National Natural Science Foundation of China (Nos. 61673384, 61832009, 61772260, 61502497, 61562015), the Jiangsu Planned Projects for Postdoctoral Research Funds (No. 2018K028C), the Innovation Project for State Key Laboratory for Novel Software Technology (No. ZZKT2018B02), and the Guangxi Key Laboratory of Trusted Software (Nos. kx201609, kx201532).