软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006262]

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

基于路径分析和信息熵的错误定位方法*

姜淑娟[1,2], 张 旭[1,2], 王荣存[1,2,3], 黄 颖[1,2], 张艳梅[1,2], 薛 猛[1,2]

[1](中国矿业大学 矿山数字化教育部工程研究中心, 江苏 徐州 221116)

[2](中国矿业大学 计算机科学与技术学院, 江苏 徐州 221116)

[3](高安全系统的软件开发与验证技术工业和信息化部重点实验室(南京航空航天大学), 江苏 南京 211106)

通讯作者: 王荣存, E-mail: rcwang@cumt.edu.cn

摘 要: 软件错误定位是一项耗时又费力的工作,因此如何提高软件错误定位的自动化程度一直以来都是软件工程领域研究的热点.现有的基于频谱的错误定位方法很少利用程序的上下文信息,而程序的上下文信息对错误定位至关重要.针对此问题,本文提出了一种基于路径分析和信息熵的错误定位方法 FLPI.该方法在基于频谱信息技术的基础上,通过对所有执行路径中的数据依赖关系进行分析来引入执行上下文信息,同时利用信息熵理论将测试事件信息引入到可疑语句的怀疑度计算公式中,以提高错误定位的精度和效率.为了评价该方法的有效性,基于一组基准程序和开源程序进行实验验证.实验结果表明,本文所提方法 FLPI 能够有效地提高错误定位的精度和效率.

关键词: 错误定位; 上下文信息; 信息熵; 路径分析

中图法分类号: TP311

中文引用格式: 姜淑娟,张旭,王荣存,黄颖,张艳梅,薛猛.基于路径分析和信息熵的错误定位方法.软件学报. http://www.jos.org.cn/1000-9825/6262.htm

英文引用格式:Jiang SJ, Zhang X, Wang RC, Huang Y, Zhang YM, Xue M. A fault localization approach based on path analysis and information entropy. Ruan Jian Xue Bao/Journal of Software, 2021 (in Chinese). http://www.jos.org.cn/1000-9825/6262.htm

A Fault Localization Approach Based on Path Analysis and Information Entropy

JIANG Shu-Juan[1,2], ZHANG Xu[1,2], WANG Rong-Cun[1,2,3], HUANG Ying[1,2], ZHANG Yan-Mei[1,2], XUE Meng[1,2]

[1](Engineering Research Center of Mine Digitalization (China University of Mining and Technology), Ministry of Education, Xuzhou 221116, China)

[2](School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

[3](Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 211106, China)

Abstract: Software fault localization is a time-consuming and laborious work, so determining how to improve the automation of software fault localization has always been a hot topic in the field of software engineering. The existing spectrum-based fault localization (SBFL) methods rarely use the context information of the program, which is very important for fault localization. To solve this problem, the paper proposes a fault localization approach based on path analysis and information entropy (FLPI). Based on the spectrum information technology, this approach introduces the execution context information by analyzing the data dependencies in all execution paths, and introduces the test event information into the suspiciousness formula by using the information entropy theory, so as to maximize the accuracy and efficiency of fault localization. To evaluate the effectiveness of the proposed approach, the experiments were

* 基金项目: 国家自然科学基金(61673384); 江苏省自然科学基金(BK20181353); 高安全系统的软件开发与验证技术工业信息化部重点实验室开放基金(1015-56XCA18164).

Foundation item: National Natural Science Foundation of China (61673384); Natural Science Foundation of Jiangsu Province (BK20181353); Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology (1015-56XCA18164).

收稿时间: 2020-09-14; 修改时间: 2020-10-26; 采用时间: 2020-12-14; jos 在线出版时间:2021-01-22

conducted on a set of benchmark programs and open source programs. Experimental results show that our FLPI approach can effectively improve the accuracy and efficiency of fault localization.

Key words: fault localization; context information; information entropy; path analysis

导致软件运行失效或发生故障的原因称为软件错误[1],研究软件错误产生的机制是定位软件错误的前提,而错误产生的原因主要与其内部因素与外部因素有关.软件测试一直以来都被视为检测和分析软件错误的重要途径[2],软件测试分为静态测试和动态测试.动态分析方法能够准确地捕获程序的执行轨迹、数据流覆盖、逻辑控制依赖等重要信息,动态测试方法只能证明程序中存在错误而不能预测和修复错误,因此软件调试就显得格外重要[3].在软件开发和维护过程中,程序调试是一项十分容易出错的任务,其关键步骤就是错误定位和缺陷修复[4-6].

传统的错误定位技术包括程序日志记录、断言、断点和性能分析.而自动化错误定位技术主要分为基于静态分析的方法和基于动态测试的方法.后者通过测试用例驱动待测程序,将程序执行结果与预期结果进行分析对比完成错误定位过程[7].

基于程序频谱的错误定位技术能够直观的提供程序实体的测试覆盖信息,且计算复杂度低、操作简单、适用范围广,因此一直以来都是研究的热点.Harrold 等人[8]首先提出频谱在错误定位上的应用;在此基础上,Jones 等人[9]提出了 Tarantula 技术,并利用不同深度的颜色可视化地将语句出错的可能性呈现出来;Abreu 等人[10] [11]受聚类分析和分子生物学领域中相似度系数的启发分别提出了 Jaccard 方法和 Ochiai 方法.在此之后,越来越多的基于频谱信息的错误定位技术被相继提出.多数方法仅仅利用二元覆盖信息计算语句的可疑度值,并根据可疑度值大小顺序检查语句.这些方法在错误定位过程中对程序执行过程和轨迹进行了高度简化,忽略了变量之间固有的语义联系和依赖关系等程序执行上下文信息,从而大大影响了定位的精度,使得程序员很难在短时间内根据单个程序实体的可疑度来找出程序中故障的位置.

程序依赖分析表明了程序实体的执行顺序、相互调用和依赖关系,是一种理解和分析软件行为特征的重要手段,因此基于程序依赖信息的错误定位技术更加注重程序实体的内部信息及其之间的相互关系.Baah 等人[12]以概率图形模型为基础,通过对具有统计依赖性信息的程序依赖图进行扩充,提出了基于概率程序依赖图即 PPDG(Probabilistic Program Dependence Graph)的错误定位模型.该模型很好的反映了程序内部行为,很大程度上提高了定位的精度.

针对上述分析两类技术存在的问题,本文提出了一种基于路径分析和信息熵的错误定位方法 FLPI(Fault Localization Based on Path Analysis and Information Entropy).该方法在基于频谱信息技术的基础上,通过对所有执行路径中的数据依赖关系进行分析以获取执行上下文信息,同时利用信息熵理论将测试事件信息引入到可疑度计算公式中,最大程度地提高错误定位的精度和效率.

本文的组织结构如下:第 1 节给出本文方法的技术背景;第 2 节首先描述 FLPI 方法的框架和整体流程,然后具体介绍了该方法,最后给出了一个实例分析;第 3 节使用基准程序集 Siemens Suite、Nano XML、XML-sec 及 Defect4J 对本文所提方法开展实验验证及结果分析;第 4 节介绍了相关工作;最后总结全文并讨论了进一步的研究方向.

1 研究背景

1.1 基于频谱的错误定位技术

基于频谱的错误定位(SBFL)是一种典型的动态分析技术,它主要利用软件测试过程中收集到的两种信息[13,14],即测试执行结果和程序频谱进行分析来识别错误程序实体(如语句、分支、基本块、方法或谓词).SBFL 方法利用程序实体的覆盖信息计算其可疑度值,具有计算复杂度低、操作简单且适用范围广的特点,因此一直是错误定位问题的研究热点,这也是本文研究的重点.

SBFL 方法一般遵循以下步骤:执行测试用例,通过插桩技术获取源程序在给定测试用例集下的覆盖信息

以及执行结果,并根据覆盖信息和执行结果构建测试覆盖矩阵和执行结果向量;统计分析覆盖信息,借助不同的错误定位公式计算程序实体的可疑度,并根据可疑度计算结果由高到低的顺序排列程序实体.程序实体的可疑度越高,其包含错误的可能性就越大,开发人员依次检查程序实体开展错误定位工作.基于频谱的软件错误定位框架如图 1 所示.

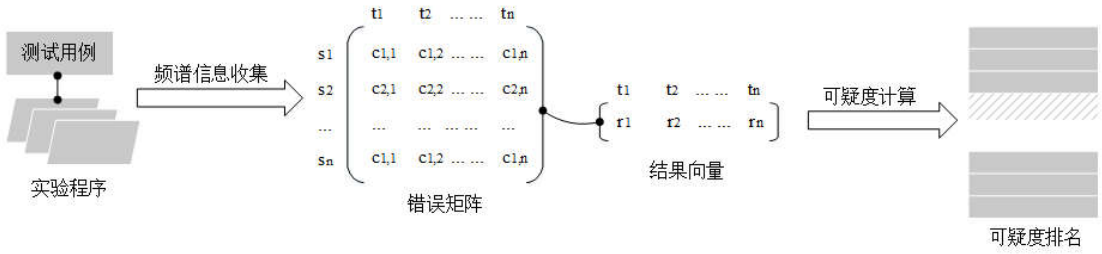


Fig.1 Spectrum-based fault localization framework

图 1 基于频谱的软件错误定位框架

对于任意一个程序实体 S_i,其在测试用例下的执行信息和覆盖特征可以用一个四元组表示,即 $N_S=(N_{CF},N_{UF},N_{CS},N_{US})$. 四个元素分别代表执行失败且覆盖程序实体 S_i 的测试用例次数、执行失败且未覆盖程序实体 S_i 的测试用例次数、执行成功且覆盖程序实体 S_i 的测试用例次数和执行成功且未覆盖程序实体 S_i 的测试用例次数.一般情况下用 N_F 表示所有执行失败的测试用例次数,用 N_C 表示所有执行成功的测试用例次数.表 1 给出了几种典型的错误定位技术可疑度度量公式.

Table 1 The calculation formula of suspicious degree for fault localization

表 1 典型的错误定位怀疑度计算公式

公式名称	公式表达式
Tarantula	$\dfrac{N_{CF}/N_F}{N_{CF}/N_F+N_{CS}/N_S}$
Ochiai	$\dfrac{N_{CF}}{\sqrt{N_F\times(N_{CF}+N_{CS})}}$
Naish1	$\begin{cases}-1 & N_{CS}<N_F\\ N_S-N_{CS} & N_{CS}=N_F\end{cases}$
Dstar	$\dfrac{N_{CF}^{*}}{N_{CS}+N_{UF}}$
Jaccard	$\dfrac{N_{CF}}{N_{CS}+N_{CF}+N_{UF}}$
Wong2	$N_{CF}-N_{CS}$

1.2 程序依赖性分析

程序依赖分析主要分析程序实体的执行顺序、函数之间的相互调用、以及依赖关系,是一种理解和分析软件行为特征的重要手段,在程序分析与调试、软件测试与维护等软件工程领域有着广泛的应用.目前常用的方法主要包括过程内的依赖分析(控制依赖分析和数据依赖分析等)、过程间的依赖分析(程序依赖图分析等)和程序切片等.下面给出程序依赖性分析的相关定义.

定义 1 控制流图 CFG (Control Flow Graph):程序 PG 的控制流图是一对(N,E)组合,其中 N 代表 PG 中所有语句的节点集合,E 代表有向边集合,而边(n_i,n_j)则表示从节点 n_i 到节点 n_j 的控制流,其中条件分支和循环边上的

标识表示取这些边时的条件.

定义 2 控制依赖:在控制流图 G 中,如果节点n_2具有输出边e_1和e_2,则节点n_1控制依赖于n_2,当且仅当满足以下条件:

(1)G 中以e_1开始并以结束节点结束的每条路径都包含n_1

(2)存在一条路径以e_2开始并以结束节点结束,而中间节点不包含n_1

定义 3 数据依赖:在控制流图 G 中,节点n_1数据依赖于节点n_2,当且仅当满足以下条件:

(1)节点n_2定义了变量 v

(2)G 中存在一条从n_2到n_1但未重新定义 v 的路径

(3)节点n_1使用了变量 v

Zeller[15]提出的 Defect- Infection-Failure 模型指出:开发人员在用给定的测试用例执行缺陷程序时,软件缺陷可能会导致其余源代码部分的感染,感染状态的传播最终会导致程序失效.程序实体间的依赖分析不仅有助于定位引起程序失效的根源,而且还充分考虑了感染状态在程序内部的传播机制,为理解程序失效提供了有效的上下文信息,一定程度上可以帮助开发人员理解错误产生的原因.因此在错误定位过程中引入程序依赖性分析可以有效的提高方法的精度和准确性.

下面给出数据依赖路径分析过程中的相关定义.

定义 4 变量轨迹:变量v_x的轨迹为 $Trace(v_x)=\{v_x^0,v_x^1,\ldots,v_x^n\}$,其中$v_x^0,v_x^1,v_x^n$表示程序中变量$v_x$在程序执行过程中的定义与使用.

变量的定义是指其值或内部状态发生改变,通常出现在赋值指令、输入数据指令或过程调用中,而变量的使用是指其值或内部属性被读取,但未发生任何改变的过程.这样一个程序的行为就可以通过一组变量跟踪来表示.

定义 5 变量依赖关系:变量v_x和v_y的依赖关系为 $VDR(v_x,v_y)=\{(v_x^i,v_y^i)|v_x^i \in Trace(v_x), v_y^i \in Trace(v_y), v_x^i \rightarrow v_y^i\}$,其中“→”表示数据依赖关系.

2 本文方法

2.1 总体框架

FLPI 方法以待测程序 PG 和测试用例集T_c作为输入,最终得到错误定位报告.整体框架主要包括三个部分:预处理阶段、错误定位方法和定位结果显示,如图 2 所示.

① 预处理阶段:通过静态分析对待测源程序 PG 构建控制流图, 同时进行程序插桩工作, 编译执行插桩后的程序并运行测试用例集, 获取程序的执行轨迹和覆盖信息.

② 利用控制流图分析得到程序所有执行路径中的不同数据依赖关系, 即以语句节点 l_i 作为数据依赖前驱和依赖后继时的不同 P_l, 结合程序的执行轨迹和数据依赖路径分析结果得到 P_l 的覆盖信息, 根据覆盖信息矩阵计算得到 P_l 在执行过程中四类随机事件的信息熵.

③ 计算所有路径中不同 P_l 的怀疑度值,对于涉及到语句 l_i 的所有 P_l 的怀疑度值进行加权平均,最终得到程序所有语句节点的怀疑度,生成错误定位报告定位到错误语句.

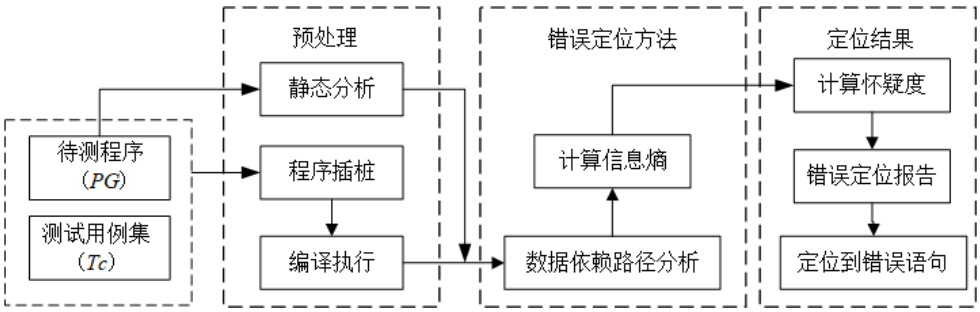


Fig.2 Overall framework of our approach

图 2 本文方法的总体框架

2.2 数据依赖路径分析

程序变量的状态在程序运行过程中沿着不同的执行路径发生改变,变量的不正确定义和使用会造成程序故障.数据依赖路径分析不仅可以得到变量状态变化信息及其传递轨迹,而且可以获取语句之间的相互作用和依赖关系,为错误定位分析提供更加有效的上下文信息.变量依赖关系说明了其值和内部状态发生改变的机理,主要包括变量之间的语义联系和变量的自身依赖.

Aho 等人[16]提出了利用控制依赖图和控制依赖子图进行数据依赖分析的方法.收集控制依赖图的 *DEF[n]*、*REF[n]*、*KILL[n]*、*IN[n]*和 *OUT[n]*集合,其分别代表语句节点 *N* 中产生变量的集合、使用变量的集合、杀死变量的集合、流入语句节点的变量集合和流出语句节点的变量集合.除 *DEF[n]*和 *REF[n]*外,其余集合均为一个语句节点和变量构成的二元组,表示为<l_i,v_x>.本文参考了文献[16]的方法,使用控制流图 CFG 获取上述集合并进行数据依赖分析.

算法 1 描述了对程序所有执行路径进行数据依赖分析的过程.输入控制流图以及各节点的 *IN*、*OUT*、*REF*、*DEF* 集,对 CFG 进行遍历,针对图中的每个节点 *N*,首先判断其变量的使用情况,在使用集不为空时将使用变量 *x* 与 *IN* 集中的变量v_x对比,并建立从节点 *N* 到节点l_i的数据依赖边(行 4—8),最后遍历控制流图,输出节点 *N* 到其父节点的数据依赖边,完成数据依赖分析过程(行 12--14).

算法 1 数据依赖分析

输入:控制流图 *CFG*、各节点的 *IN*、*OUT*、*REF*、*DEF* 集

输出:数据依赖关系

```
1.  for each node N in CFG do
2.      if REF[n] is not null then
3.          for each variable x in REF[n] do
4.              for each pair <l_i,v_x> in IN[n] do
5.                  if v_x== x then
6.                      establish data-dependent edges from node N to l_i
7.                  end if
8.              end for
9.          end for
10.     end if
11. end for
12. for each node N in CDS do
13.     output the data dependent edges from node N to its parent
14. end for
end
```

在一条执行路径上, 若在语句节点l_i处定义了变量v_x, 而在语句节点l_j处引用了变量v_x, 那么我们使用$P_l(l_i, l_j, v_x)$来表示这种数据依赖关系, 其中l_i为依赖前驱, l_j为依赖后继, v_x为依赖变量. $P_l(l_i, *, v_x)$和$P_l(*, l_i, v_x)$分别代表以l_i为依赖前驱和后继的不同数据依赖关系. 通过对语句节点l_i所有不同执行路径的分析可以得到其有效上下文信息, 利用数据流对变量定义使用的跟踪和变量依赖关系的描述分析程序复杂的行为, 更好的定位程序错误的位置.

2.3 测试事件信息熵与怀疑度计算

SBFL 方法就是针对程序的执行覆盖信息进行一系列操作,进而确定程序语句出错的可能性[17],从本质上讲就是利用怀疑度公式进行概率统计.程序语句在测试用例下的执行信息和覆盖特征可以用一个四元组表示,即(N_{CF},N_{UF},N_{CS},N_{US}),其分别代表不同的随机事件.以往的怀疑度计算仅仅考虑了不同事件发生的次数而忽略了组合事件附带的信息.信息熵是表示随机事件不确定性的数学化度量,因此利用信息熵可以将程序语句存在错误可能性的信息量加权平均,将两者有机结合即可将测试事件信息引入到可疑度计算中.

在错误定位研究过程中,若存在大量执行失败的测试用例,那么对执行成功的测试用例进行研究可获得更多与错误相关的信息;反之若仅有少数测试用例失败,那么其更有利于错误定位.基于此,文献[18]提出了一种基于事件信息量的方法 SIQ.本文在其基础上,利用信息熵加以改进,进一步提高错误定位的精度.

信息熵是表示随机事件不确定性的数学化度量, 是所有随机事件信息量的期望. 对于一组随机事件$X = \{x_1, x_2, \dots, x_n\}$, 假设$x_i$发生的概率为$P(x_i)$, 则其信息熵 H (x)的定义如下:

$$H(x) = E[I(x_i)] = -\sum_{i=1}^{n}(P(x_i) * \log P(x_i))\ H(x) = E[I(x_i)] = -\sum_{i=1}^{n}(P(x_i) * \log P(x_i)) \tag{1}$$

错误定位中的测试事件主要分为四类, 即执行成功的事件 P 和执行失败的事件 F, 覆盖程序实体的测试事件 $C(s)$和未覆盖程序实体的测试事件 $U(s)$, 它们的信息熵表示如下:

$$H(P) = E[I(P)] = -\sum\left(\frac{N_S}{N_S + N_F} * \log\frac{N_S}{N_S + N_F}\right) \tag{2}$$

$$H(F) = E[I(F)] = -\sum\left(\frac{N_F}{N_S + N_F} * \log\frac{N_F}{N_S + N_F}\right) \tag{3}$$

$$H(C(s)) = E[I(C(s))] = -\sum\left(\frac{N_C(s)}{N_C(s) + N_U(s)} * \log\frac{N_C(s)}{N_C(s) + N_U(s)}\right) \tag{4}$$

$$H(U(s)) = E[I(U(s))] = -\sum\left(\frac{N_U(s)}{N_C(s) + N_U(s)} * \log\frac{N_U(s)}{N_C(s) + N_U(s)}\right) \tag{5}$$

为避免程序实体全部被覆盖执行或全部不执行两类随机事件的发生, 我们取未执行一次代替全部执行的事件, 同理用覆盖一次代替全部不执行的事件.

对测试事件进一步分析可知:N_{CF}和N_{US}的数值越高, 说明覆盖程序实体的测试事件更容易导致执行失败而未覆盖程序实体的测试事件更容易使得执行成功, 这两个参数与怀疑度计算成正比. 同理可得到N_{CS}和N_{UF}的数值与怀疑度计算成反比.

利用信息熵$H(P)$, $H(F)$, $H(C(s))$和$H(U(s))$将程序语句存在错误可能性的信息量加权平均, 对计算公式中参数设置权重进行动态调整, 辅助语句可疑度的计算. 由于测试事件执行成功与否和其覆盖信息并不是相互独立的, 两者的信息熵无法相加, 故以乘积的方式引入. 综上所述, 我们得到的怀疑度计算公式如下:

$$susp(s) = \frac{\left(H\left(C(s)\right) * H(F) * N_{CF}(s)\right) + \left(H\left(U(s)\right) * H(P) * N_{US}(s)\right)}{\left(H\left(C(s)\right) * H(P) * N_{CS}(s)\right) + \left(H\left(U(s)\right) * H(F) * N_{UF}(s)\right)} \tag{6}$$

2.4 示例分析

本节引入了一个简单的例子程序,该例子程序包含 12 条语句,错误语句位于s_5处,例子程序及其控制流图如图 3 所示.

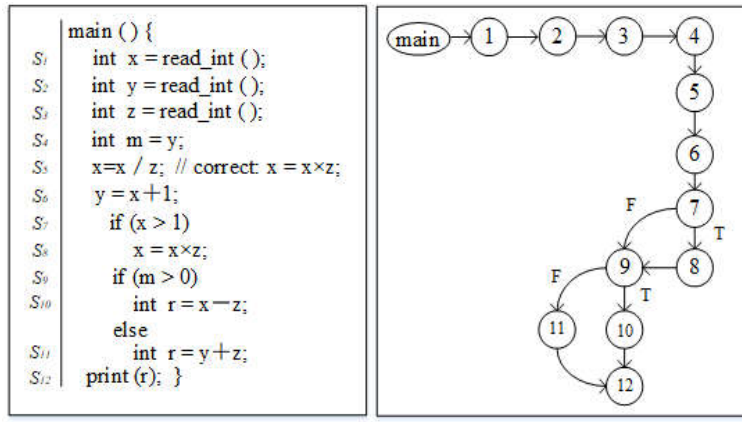


Fig.3 An example program and its CFG

图 3 示例程序及其控制流图

同时设计了如下 8 个测试用例:t_1(6,3,6)、t_2(-4,1,-4)、t_3(4,2,2)、t_4(1,5,1,)、t_5(1,0,-1)、t_6(2,-4,1)、t_7(0,4,-1)和 t_8(0,0,1).程序语句的覆盖信息及使用 Tarantula 和 FLPI 方法对示例程序定位结果如表 2 所示.其中,“语句”表示程序相关的语句;“覆盖信息”表示在各个测试用例执行过程中对各个语句的覆盖情况;“排名”表示用 Tarantula 方法和我们的方法分别计算得到的各个语句出错怀疑度值的排名.

Table 2 The result of the example program

表 2 示例程序结果

语句	覆盖信息								排名	
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	Tar	FLPI
s_1	●	●	●	●	●	●	●	●	3	6 (0.757)
s_2	●	●	●	●	●	●	●	●	3	6 (0.757)
s_3	●	●	●	●	●	●	●	●	3	4 (1.567)
s_4	●	●	●	●	●	●	●	●	3	6 (0.757)
s_5	●	●	●	●	●	●	●	●	3	5 (1.379)
s_6	●	●	●	●	●	●	●	●	3	11 (0.511)
s_7	●	●	●	●	●	●	●	●	3	6 (0.757)
s_8	○	○	●	○	○	●	○	○	2	3 (2.006)
s_9	●	●	●	●	●	●	●	●	3	6 (0.757)
s_{10}	●	●	●	●	○	○	●	○	1	1 (3.614)
s_{11}	○	○	○	○	●	●	○	●	12	12 (0.264)
s_{12}	●	●	●	●	●	●	●	●	3	2 (2.025)
结果	F	F	F	T	T	T	T	T	3-11	5-5

针对上述的示例程序介绍 FLPI 方法的定位过程.首先根据控制流图及各节点的 *IN*、*OUT*、*REF*、*DEF* 集获取程序所有执行路径中的不同数据依赖关系, 例如语句节点 5 的 *IN* 集为(1, *x*)、(2, *y*)、(3, *z*)和(4, *m*),*OUT* 集

为(5, x)、(2, y)、(3, z)和(4, m),*REF* 集为 x 和 z,而 *DEF* 集则为 x.针对语句节点 5,将其 *REF* 集中的使用变量与 *IN* 集中的流入变量进行对比,由此可以建立P_l关系(1,5, x)和(3,5, z),同理可得到所有执行路径中的其他数据依赖关系,分别为(2,4, y)、(3,8, z)、(3,10, z)、(3,11, z)、(4,9, m)、(5,6, x)、(5,7, x)、(5,8, x)、(5,10, x)、(6,11, y)、(8,10, x)、(10,12, r)和(11,12, r).

接下来结合程序语句覆盖信息和所有执行路径中的数据依赖关系对示例程序进行路径分析,得到P_l的覆盖信息并计算其相应的信息熵等值,所得结果如表 3 所示.

Table 3 The result of path analysis

表 3 路径分析结果

P_l信息	覆盖信息								计算结果				
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	N_{CF}	N_{US}	N_{CS}	N_{UF}	Sus
(1,5, x)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(2,4, y)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(3,5, z)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(3,8, z)	○	○	●	○	○	●	○	○	1	4	1	2	1.459
(3,10, z)	●	●	●	●	○	○	●	○	3	3	2	0	3.786
(3,11, z)	○	○	○	○	●	●	○	●	0	2	3	3	0.264
(4,9, m)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(5,6, x)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(5,7, x)	●	●	●	●	●	●	●	●	3	0	5	0	0.757
(5,8, x)	○	○	●	○	○	●	○	○	1	4	1	2	1.459
(5,10, x)	●	●	●	●	○	○	●	○	3	3	2	0	3.786
(6,11, y)	○	○	○	○	●	●	○	●	0	2	3	3	0.264
(8,10, x)	○	○	●	○	○	○	○	○	1	5	0	2	3.099
(10,12, r)	●	●	●	●	○	○	●	○	3	3	2	0	3.786
(11,12, r)	○	○	○	○	●	●	○	●	0	2	3	3	0.264
结果	F	F	F	T	T	T	T	T					

与s_5语句相关的P_l为(1,5, x)、(3,5, z)、(5,6, x)(5,7, x)、(5,8, x)和(5,10, x),以(1,5, x)为例,其信息熵等值计算如下:

$$H(P)=-\sum\left(\frac{N_S}{N_S+N_F}*\log\frac{N_S}{N_S+N_F}\right)=-\frac{5}{5+3}*\log\frac{5}{5+3}=0.42 \quad H(F)=0.53$$

$$H(C(s))=-\sum\left(\frac{N_C(s)}{N_C(s)+N_U(s)}*\log\frac{N_C(s)}{N_C(s)+N_U(s)}\right)=-\frac{7}{7+1}*\log\frac{7}{7+1}=0.17 \quad H(U(s))=0.38$$

$$susp(s)=\frac{\left(H(C(s))*H(F)*N_{CF}(s)\right)+\left(H(U(s))*H(P)*N_{US}(s)\right)}{\left(H(C(s))*H(P)*N_{CS}(s)\right)+\left(H(U(s))*H(F)*N_{UF}(s)\right)}=\frac{0.17*0.53*3+0.38*0.42*0}{0.17*0.42*5+0.38*0.53*0}=0.757$$

同理可得到其他与s_5语句相关的P_l的怀疑度值分别为 0.757、0.757、0.757、1.459 和 3.786,加权平均得到$susp(s_5)$=1.379.同理可得到其余语句的怀疑度值,详见表 1 最后一列.表 1 最后一行显示了两种方法在最好和最坏情况下需要检查的语句数,分别为 3-11 和 5-5.由此示例可知,FLPI 方法是可行的.综上,FLPI 方法可以减少语句检查数量,提高错误定位的精度.

3 实验

针对本文提出的基于路径分析和信息熵的错误定位方法进行实验验证评估其有效性.实验运行环境为

Windows10和Linux系统,2.60GHz Intel(R) i5四核处理器,4GB物理内存.

3.1 实验对象

本文选取了 17 个程序作为实验对象,程序特征如表 4 所示.前 14 个程序来自 SIR 库[19],后 3 个开源项目 JFreeChar、Joda-Time 和 Mockito 来自 Defect4J 库[20].

Table 4 Experimental subject

表 4 实验对象

程序	简要描述	错误版本数	代码行数	测试用例数
print_tokens	词法分析	7	565	4130
print_tokens2	词法分析	10	510	4115
replace	模式替换	32	563	5542
schedule	优先级调度	9	412	2650
schedule2	优先级调度	10	307	2710
tcas	高度分离	41	173	1608
totinfo	信息度量	23	406	1052
NanoXML v_1	XML 解析器	7	4351	237
NanoXML v_2	XML 解析器	7	5671	237
NanoXML v_3	XML 解析器	10	6838	237
NanoXML v_5	XML 解析器	8	7160	237
XML-sec v_1	XML 加密程序	8	21613	92
XML-sec v_2	XML 加密程序	6	22318	94
XML-sec v_3	XML 加密程序	7	19895	84
JFreeChart	图表绘制类库	26	96300	2205
Joda-Time	日期时间类库	27	28400	4130
Mockito	单元测试框架	38	23000	1366

本文的实验最终选取了基准程序中的 249 个错误版本.剩余程序被排除的主要原因如下:部分错误版本程序不存在失败的测试用例,如 replace 的版本 32 和 schedule2 的版本 9;部分错误版本程序无法收集到覆盖信息,如 print_tokens 的版本 4、schedule 的版本 5 和 print_tokens2 的版本 10 等;部分错误版本程序没有输出错误,如 Time 的版本 2 和版本 19、Mockito 的版本 6 和版本 10 等.

3.2 评价标准

为了评估所提错误定位方法的有效性,本文采用以下四个指标与现有方法进行实验比较.

- 累积检查语句数.即对于不同错误版本的基准程序定位到错误时累积检查的总语句数,其值越低,说明方法更有效.
- *Expense*(错误定位代价)[21].*Expense* 从相对指标的角度描述了错误定位方法的精度,该指标表示在检测到错误时需要检查的程序语句的百分比,其值越低,则定位精度越高,公式具体如下:

$$Expense = \frac{Number\ of\ statements\ examined}{Total\ number\ of\ statements\ in\ the\ program} \times 100\% \qquad (6)$$

- *EXAM*(检查得分)[22].*EXAM* 得分可以直观的反映不同错误定位方法的效率,该指标表示在相同程序语句检查范围内定位到程序错误的数量,其值越高,则定位效果越好,公式具体如下:

$$EXAM = \frac{Number\ of\ fault\ examined}{Number\ of\ statements\ examined} \times 100\% \qquad (7)$$

- 时间开销.定位错误需要的时间代价,它反映了不同错误定位方法的性能及时间效率.

3.3 实验设计

为了验证本文提出的基于路径分析和信息熵的上下文错误定位方法,设计对比实验评估其有效性.用于对比实验的错误定位方法主要有三组共 7 种类型,其中第一组为经典的错误定位技术即 Tarantula、Ochiai 和 Naish1[1];第二组为较优的 Dstar[1]、Naish2(公式 8)和 Russel &Rao(公式 9)其中 Dstar 公式中的*通常取 2[5],后两个是经理论证明的最优公式[23];第三组为基于信息量的错误定位方法 SIQ[18].

$$Naish2 = N_{CF} - \frac{N_{CS}}{N_{CS} + N_{US} + 1} \tag{8}$$

$$Russel\ \&Rao = \frac{N_{CF}}{N_{CF} + N_{UF} + N_{CS} + N_{US}} \tag{9}$$

同时实验设计了如下两个问题:

(1)本文提出的 FLPI 方法能否提高错误定位的准确度？若能,具体提高了多少？

(2)本文提出的 FLPI 方法能否提高错误定位的效率？若能,提高了多少？

3.4 实验结果与分析

在 17 个开源程序上开展对比实验,错误定位方法在每个程序不同版本上定位到错误时累积检查语句数的对比结果如表 5 所示,而不同方法定位代价的对比结果如表 6 所示.

Table 5 Comparison of cumulative check statements between FLPI and other methods

表 5 FLPI 与其他方法累积检查语句数的比较

程序	Tarantula	Ochiai	Naish1	Dstar	Naish2	R&R	SIQ	FLPI
print_tokens	769	739	905	744	840	846	723	687
print_tokens2	271	213	284	219	177	202	197	165
replace	1448	1417	1460	1503	1611	1684	1411	1348
schedule	390	396	634	446	486	596	377	346
schedule2	1182	1133	1017	1097	1034	950	1010	939
tcas	2709	2682	2527	2518	2527	2511	2414	2335
totinfo	922	799	622	730	643	859	615	594
NanoXML v_1	1853	1772	1884	1826	1892	2068	1720	1694
NanoXML v_2	1592	1501	1703	1491	1736	1569	1501	1489
NanoXML v_3	2396	2177	1752	1815	1807	1793	1748	1690
NanoXML v_5	2847	2693	2516	2573	2684	2904	2510	2486
XML-sec v_1	4369	4184	4010	4022	4107	4236	3852	3692
XML-sec v_2	3072	2979	2968	2896	3061	3308	2809	2611
XML-sec v_3	5384	5306	5247	5390	5413	5610	4767	4494
JFreeChart	45632	44826	44192	43771	44667	45075	41037	37833
Joda-Time	29156	28365	28053	28126	27892	28904	26195	24231
Mockito	13934	13700	12973	13084	12905	14098	11965	10586
累积检查语句数	117926	114882	112647	112251	113482	117113	104851	97220

注:将 Russel &Rao 简称为 R&R

使用累积检查语句数指标分析可知:在 17 个程序上,FLPI 方法累积检查语句总数均少于其他对比方法,其

1 Tarantula、Ochiai、Nsish1、Dstar 公式参见表 1

占 Tarantula、Ochiai、Naish1、Naish2、Dstar、Russel &Rao 和 SIQ 方法累积检查语句总数比例分别为 82.44 %、84.63%、86.31%、86.61%、85.67%、83.01%和 92.72%.针对三组对比对象,累积检查语句数平均减少了 14.09%.

使用错误定位代价 *Expense* 指标分析可知:针对第一组对比方法 Tarantula、Ochiai 和 Naish1,FLPI 方法在 17 个基准程序上错误定位代价分别平均降低了 1.87%、1.43%和 1.73%,其中在 schedule2 和 tcas 两个程序上最为明显,平均降低了 6.21%和 4.75%;与 Dstar、Naish2 和 Russel &Rao 相比,FLPI 方法错误定位代价分别降低了 1.29%、1.47%和 1.61%,其中在 schedule 和 print_tokens 两个子程序上最为明显,平均降低了 6.31%和 4.36%;与方法 SIQ 相比,FLPI 错误定位代价最多降低了 2.57%.

Table 6 *Expense* comparison between FLPI and other methods

表 6 FLPI 与其他方法错误定位代价的比较

程序	Tarantula	Ochiai	Naish1	Dstar	Naish2	R&R	SIQ	FLPI
print_tokens	27.22%	26.16%	32.04%	26.34%	29.73%	29.95%	25.59%	24.32%
print_tokens2	5.90%	4.64%	6.19%	4.77%	3.86%	4.40%	4.29%	3.59%
replace	9.19%	8.99%	9.26%	9.53%	10.22%	10.68%	8.95%	8.55%
schedule	18.93%	19.22%	25.92%	21.65%	23.59%	24.08%	18.30%	16.80%
schedule2	42.78%	41.01%	36.81%	39.70%	37.42%	34.38%	36.55%	33.98%
tcas	42.32%	41.90%	39.48%	39.34%	39.48%	39.23%	37.71%	36.48%
totinfo	11.95%	10.36%	8.06%	9.46%	8.34%	11.14%	7.97%	7.70%
NanoXML v_1	6.08%	5.82%	6.19%	5.99%	6.21%	6.79%	5.65%	5.56%
NanoXML v_2	4.01%	3.78%	4.29%	3.76%	4.37%	3.95%	3.78%	3.75%
NanoXML v_3	3.50%	3.18%	2.56%	2.65%	2.64%	2.62%	2.56%	2.47%
NanoXML v_5	4.97%	4.70%	4.39%	4.49%	4.69%	5.07%	4.38%	4.34%
XML—sec v_1	2.53%	2.42%	2.32%	2.33%	2.38%	2.45%	2.23%	2.14%
XML—sec v_2	2.21%	2.14%	2.13%	2.08%	2.20%	2.38%	2.02%	1.87%
XML—sec v_3	3.87%	3.81%	3.77%	3.87%	3.89%	4.03%	3.42%	3.23%
JFreeChart	2.22%	2.18%	2.15%	2.13%	2.17%	2.19%	2.00%	1.84%
Joda-Time	3.81%	3.70%	3.66%	3.67%	3.64%	3.77%	3.42%	3.16%
Mockito	1.59%	1.57%	1.48%	1.51%	1.47%	1.61%	1.37%	1.21%
平均定位代价	11.36%	10.92%	11.22%	10.78%	10.96%	11.10%	10.01%	9.47%

为了更加直观的对比 FLPI 和其他方法的定位效果,使用 *EXAM* 指标评估不同方法的效率,首先分析 FLPI 和其他方法在 Siemens Suite 程序集上的对比结果,所得结果如图 4 所示.其中横坐标表示代码检查百分比,纵坐标表示在代码检查范围内,定位到程序错误的百分比.

观察图 4 可知,在 Siemens Suite 程序集上,本文提出的 FLPI 方法在检查相同比例代码时可以定位到更多的错误,而且至多只需要检查 70%多的代码语句就可以定位到全部的错误.针对第一组和第二组对比对象 Tarantula、Ochiai 和 Naish2 等,*EXAM* 分数有了很大的提高,在代码检查率为 10%时平均提高了 15%,而当检查量超过 40%时,提高比例尤为明显;对比 SIQ 方法,FLPI 也有一定的提高.

各取三组对比对象中的一种方法,使用 *EXAM* 指标在 Siemens Suite 中单个程序上开展实验研究,所得结果如图 5 所示.

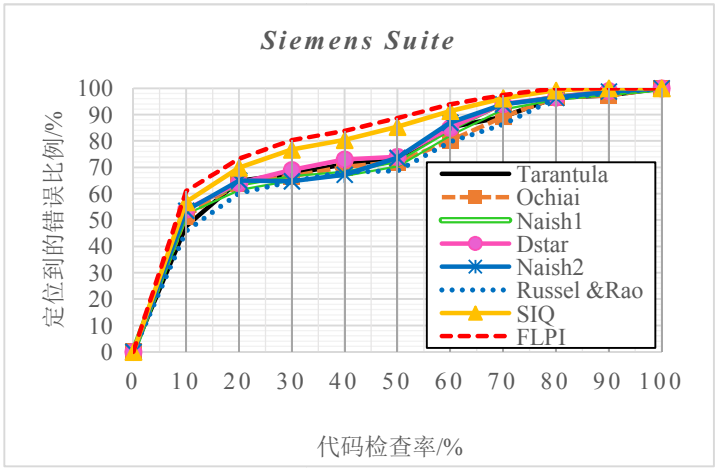
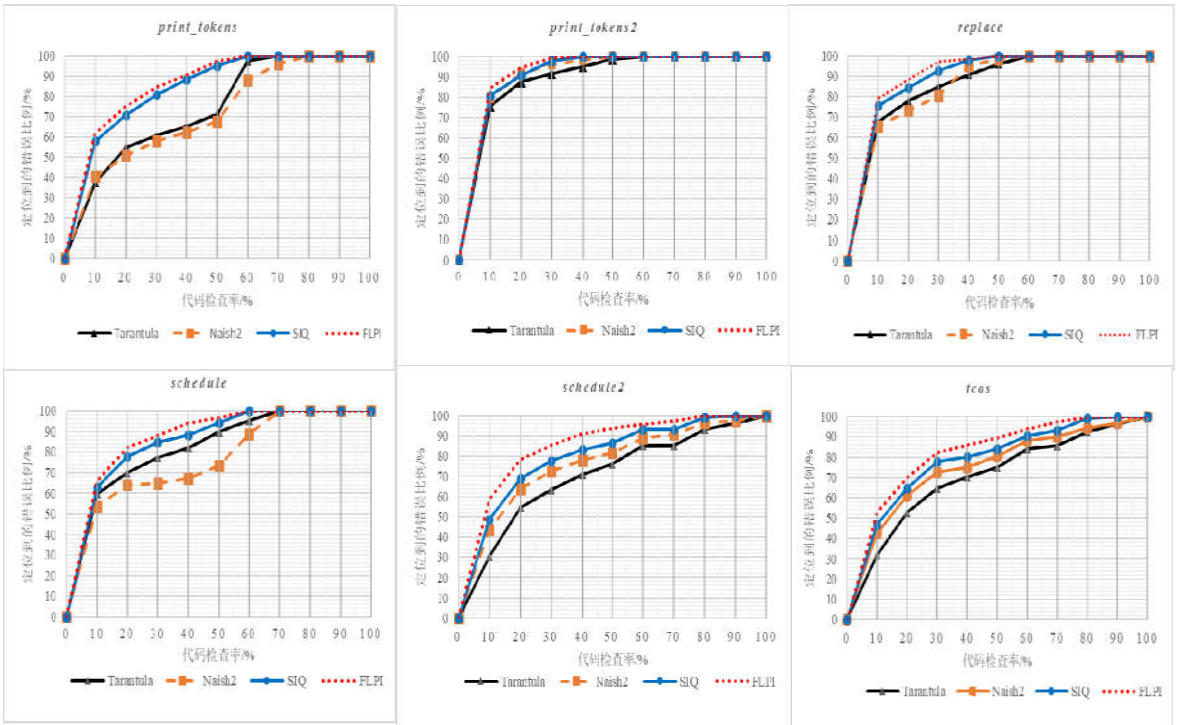


Fig.4 EXAM comparison between FLPI and other methods on Siemens Programs

图 4 在 Siemens 的程序集上 FLPI 与其他方法检查得分 *EXAM* 比较

观察图 5 可知,在单个程序上不同错误定位方法也有一定的差异,对于 print_tokens2 和 replace,所有方法的定位效果都比较好,FLPI 在检查相同规模的代码量时可以定位到更多的错误,在 print_tokens、schedule 和 schedule2 程序上,FLPI 方法定位效果提高较多,而所有方法在 tcas 程序上定位效果相比其他程序略差一些,主要是因为 tacs 程序错误版本较多而可执行语句较少.



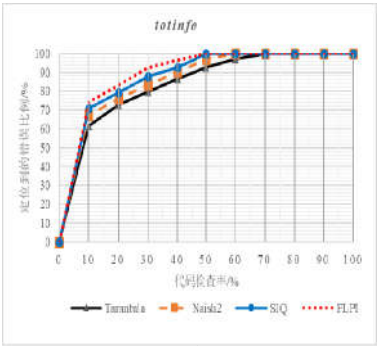


Fig 5 EXAM comparison between FLPI and other methods on each Siemens Program

图 5 在 Siemens 的单个程序上 FLPI 与其他方法检查得分 *EXAM* 比较

同样使用 *EXAM* 指标评估 FLPI 和对比方法在 NanoXML 程序集和 XML-sec 程序集和 Defect4J 上的定位效果,所得结果如图 6 所示.

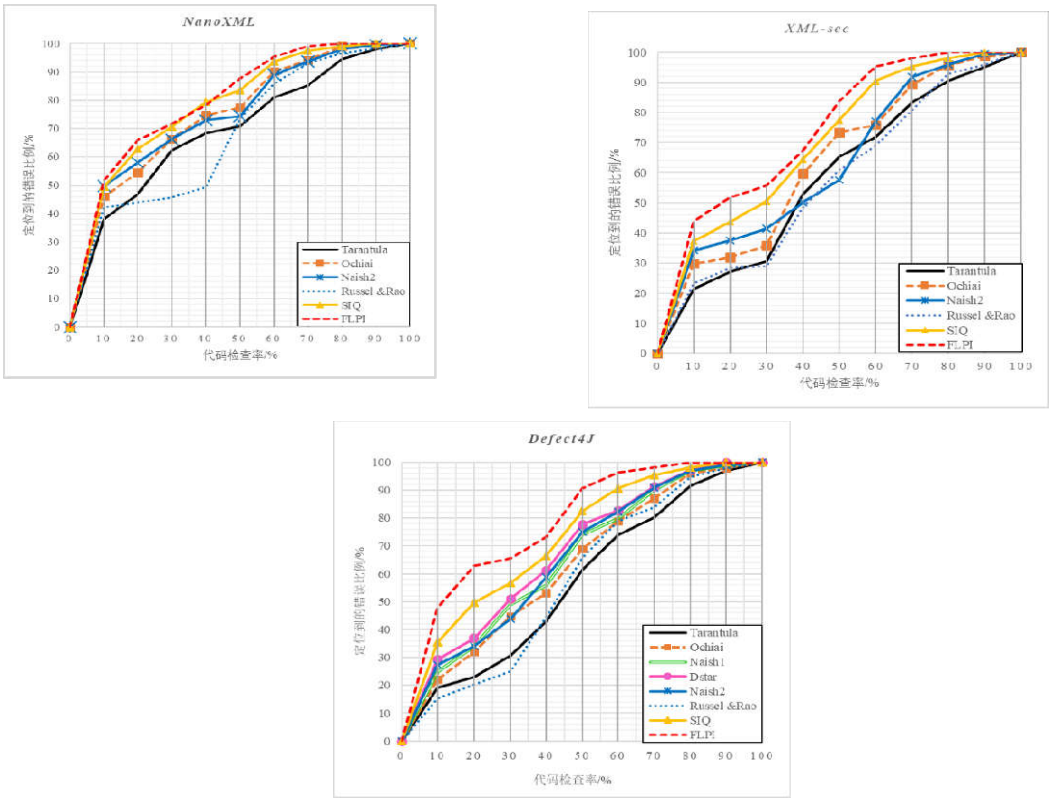


Fig. 6 EXAM comparison between FLPI and other methods on NanoXML,XML-sec, and Defect4J

图 6 NanoXML、XML-sec 和 Defect4J 上 FLPI 与其他方法检查得分 *EXAM* 比较

观察图 6 可知,在 NanoXML 程序集上,本文提出的 FLPI 方法在检查相同比例代码时可以定位到更多的错误,而且最多只需要检查 80%的代码语句就可以定位到全部的错误.各取三组对比对象中的一种方法即 Tarantula、Naish2 和 SIQ 进行分析,而在代码检查比例为 10%、20%、30%、40%和 50%时,FLCP 方法定位到的错误比 Tarantula 多了 14%、19%、9%、10%和 17%,相比较 Naish2 多了 2%、8%、5%、5%和 13%,而相比

较 SIQ 分别多了 2%、3%、1%、-1%和 4%.同理可得在 XML-sec 程序集上,FLPI 方法至多只需要检查 70%的代码语句就可以定位到全部的错误,同样与 Tarantula、Naish2 和 SIQ 对比分析得:在代码检查比例为 10%、20%、30%、40%和 50%时,FLPI 方法定位到的错误比 Tarantula 多了 23%、25%、25%、14%和 18%,相比较 Naish2 多了 10%、14%、14%、17%和 26%,而相比较 SIQ 分别多了 6%、8%、5%、3%和 6%.而在 Defect4J 的三个开源项目上,FLPI 方法错误定位效果有了明显的提升,在检查可执行语句为 10%和 20%时最为明显,与所有对比方法比较分别提高了 28%和 37%、26%和 31%、23%和 29%、18%和 26%、20%和 29%、32%和 41%、12%和 13%.

为了进一步验证 FLPI 方法的有效性,使用 EXAM 指标将其与 Context-F[24]方法进行对比,所得结果如图 7 所示.在代码检查比例为 10%、20%、30%、40%和 50%时,FLPI 方法比 Russel &Rao(Context)提高了 29%、14%、6%、1%和 3%,且优于四种方法中最优的 Dstar(Context).

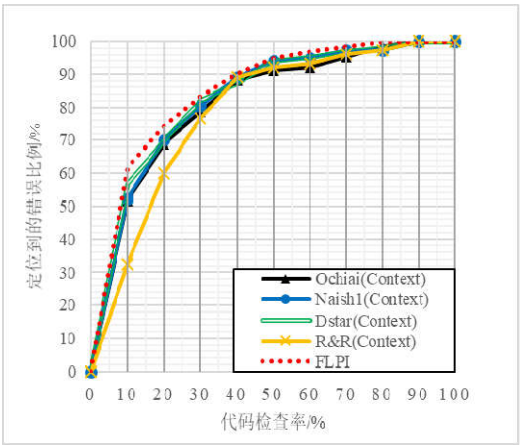


Fig. 7 EXAM comparison between FLPI and Context-FL

图 7 FLPI 与 Context-FL 检查得分 EXAM 比较

表 7 给出了 FLPI 方法与对比方法在 17 个基准程序上定位到每个错误版本的平均时间开销情况,可以看出,随着程序规模的扩大,由于对程序执行路径进行数据依赖分析的原因,FLPI 的时间成本会有所增加.

基于上述对实验结果的分析, 回答 3.3 节提出的两个研究问题.

(1) 使用 *Expense* 指标对 FLPI 方法和其他方法的对比实验结果表明:在不同的实验程序上,FLPI 对比 Tarantula、Ochiai、Naish1、Dstar、Naish2、Russel &Rao 和 SIQ 的错误定位代价最多降低了 8.80%、7.03%、9.12%、7.03%、6.79%、7.28%和 2.57%,而平均错误定位代价分别降低了 1.90%、1.45%、1.75%、1.31%、1.49%、1.63%和 0.55%.因此 FLPI 方法在一定程度上提高了错误定位的精度.

(2) 使用 *EXAM* 指标对 FLPI 方法和其他方法的对比实验结果表明:在 Siemens Suite 程序集上,FLPI 对比 Tarantula、Ochiai、Naish1、Naish2、Russel &Rao 和 SIQ 定位效率都有提高,其中在 print_tokens、schedule 和 schedule2 程序上提高较多.总体来看 FLPI 方法在检查相同比例代码时可以定位到更多的错误,而且至多只需要检查 70%多的代码语句就可以定位到程序的全部错误.因此 FLPI 方法在一定程度上提高了错误定位的效率.而在 NanoXML 程序集上,FLPI 方法最多只需要检查 80%的代码语句就可以定位到全部的错误.而在代码检查比例为 10%、20%、30%、40%和 50%时,FLCP 方法定位到的错误比 Tarantula 多了 14%、19%、9%、10%和 17%,相比较 Naish2 多了 2%、8%、5%、5%和 13%,而相比较 SIQ 分别多了 2%、3%、1%、-1%和 4%.同理可得在 XML-sec 上,FLPI 方法至多只需要检查 70%的代码语句就可以定位到全部的错误,对比上述三种方法,FLPI 方法定位到的错误比 Tarantula 多了 23%、25%、25%、14%和 18%,相比较 Naish2 多了 10%、14%、14%、17%和 26%,而相比较 SIQ 分别多了 6%、8%、5%、3%和 6%.而在 Defect4J 的三个开源项目上,FLPI 方法错误定位效果有了很大的提升,在检查可执行语句为 10%和 20%时最为明显,与所有对比方法比较分别提高了 28%和 37%、26%和 31%、23%和 29%、18%和 26%、20%和 29%、32%和 41%、12%和 13%.

Table 7 Comparison of time cost among FLPI and other techniques

表 7 FLPI 与其他方法时间开销对比

程序	定位到每个错误版本的平均时间开销/ms							
	Tarantula	Ochiai	Naish1	Dstar	Naish2	R&R	SIQ	FLPI
print_tokens	237	234	235	246	230	223	261	848
print_tokens2	216	220	214	223	218	211	239	495
replace	243	244	240	242	237	249	277	891
schedule	175	169	153	161	164	180	202	520
schedule2	188	182	174	165	171	216	223	579
tcas	132	130	126	134	131	123	158	501
totinfo	164	145	150	139	152	186	199	723
NanoXML v_1	1713	1467	1747	1633	1802	1925	2246	11908
NanoXML v_2	1792	1873	1613	1595	1607	1842	2567	18995
NanoXML v_3	3154	3046	2873	2809	2978	3464	3906	34378
NanoXML v_5	3407	2998	3120	2925	3176	3445	4725	44240
XML—sec v_1	4905	4810	4142	4096	4127	4730	5749	51992
XML—sec v_2	5302	5189	5403	5213	5537	6755	7286	78380
XML—sec v_3	3710	3767	2913	3896	3401	3532	5124	45953
JFreeChart	6148	6019	6268	6047	6423	7836	7927	90887
Joda-Time	6497	6463	7443	6339	7085	7193	8067	86604
Mockito	5247	4931	3804	5091	4352	4600	6721	67686

4 有效性分析

本节从内部有效性、外部有效性、结构有效性等方面来分析威胁本文所提方法有效性的因素.

对 FLPI 方法内部有效性的影响主要在于两个方面. 第一是三组对比实验在全部基准程序集上数据的准确性, 本文按照相关文献复现实验过程, 保证了对比实验数据的真实性; 第二是数据依赖路径分析及信息熵计算结果对定位精度的影响, 本文借鉴了 Aho 等人[16]提出的数据依赖分析方法, 同时严格按照信息论中信息熵的定义优化了怀疑度计算公式.

对 FLPI 方法外部有效性的影响主要在于实验基准程序的代表性. 本文选取了在错误定位研究中广泛使用的 Siemens Suite、NanoXML 程序集、XML-sec 程序集、Defect4J 中的 3 个开源项目 JFreeChar、Joda-Time 和 Mockito 作为研究对象. 实验对象既有小、中规模的程序，又有大规模的程序. 实验程序中的错误既有人工植入的错误, 又有真实的错误.

对 FLPI 方法构造有效性的影响主要在于本文用来评估错误定位方法有效性的评价标准，本文采用 3 个软件错误定位研究中常用的评价指标即累积检查语句数、错误定位代价 Expense、检查得分 EXAM 和时间开销分别评价 FLPI 方法和对比对象所有基准程序集上的定位效果, 因此可以有效评价不同方法的错误定位效果. 分别评价某个错误定位方法在某个缺陷上和所有缺陷上的错误定位效果，因此可以有效评价不同方法的错误定位效果.

5 相关工作

目前国内外研究人员针对软件错误定位这一课题已经进行了大量的研究,提出了很多基于测试的自动化软件错误定位技术[6,7],主要包括以下几种类别:基于程序切片的技术、基于程序频谱的技术、基于机器学习与

数据挖掘的技术、基于谓词统计的技术以及基于程序状态变更的技术.本文主要重点研究基于程序频谱的错误定位技术[7].

程序频谱描述了程序动态执行的行为信息,例如程序内部条件分支的执行信息或循环路径的覆盖信息,开发人员可以使用这些信息追踪记录程序的行为[25,26].Collofello 和 Coussis[27]的早期研究表明这种频谱可以用于软件错误定位.当执行失败时,这些信息可以用来识别导致失败的可疑代码.代码覆盖率或可执行语句命中谱(Executable Statement Hit Spectrum)表示在成功执行和失败执行过程中测试程序的哪些部分被覆盖.通过对比就可以识别出哪些程序组件与错误相关,从而缩小对导致执行失败的故障组件的搜索范围.早期的研究[28]仅使用失败的测试用例进行基于频谱的错误定位,后来一些学者同时研究成功和失败的测试用例并度量统计两者之间的差异,最终获得了更好的结果.Renieris 和 Reiss[29]提出了一种基于 ESHS 的技术即最近邻查询技术,它将失败的测试用例与成功测试用例中和其最相似的那个进行比较,也就是语句的执行模式越接近所有测试用例的失败模式,那么该语句出错的可能性就越大.

丁晖等人[18]提出基于信息量的错误定位方法,该方法根据测试信息中不同事件的类型及其发生的概率,结合语句的执行信息, 动态计算和调整错误定位的结果.与他们的方法不同,我们除了增加了信息熵来调整可疑语句的怀疑度计算公式外,还增加了数据流的分析,提高错误定位的精度.

Tarantula[9]是一种经典的基于程序频谱的错误定位技术,该方法使用覆盖率和执行结果(成功或失败)来度量每个语句的可疑度.在此基础上,其他研究学者也相继提出了一些方法,例如 Abreu 等人[10][11]受聚类分析和分子生物学领域中相似度系数的启发分别提出了 Jaccard 方法和 Ochiai 方法,这些方法极大地提高了错误定位的有效性.不仅如此,目前基于相似系数的技术已经用于不同的研究,研究人员还开发了很多实用的程序插件和工具使得程序实体的可疑度计算变得更加便捷.Naish 等人[30]提出了两种技术 O 和 O^P,其中技术 O 是为寻找单一 bug 的程序设计的,而 O^P 更适合于多个错误的程序.Wong 等人还提出了 H3b 和 H3c[31]、交叉表[32]和 DStar 技术[33].Souza 等人[34]将两种基于路线图的技术与新的过滤策略相结合实现了语句块级别和方法级别的可疑度度量,极大地优化了错误定位技术.王赞等人[35,36]将遗传算法和基于搜索的软件工程思想有机结合,寻找最高适应度值的错误实体,在此基础上,他们还利用模拟退火算法和遗传算法,构建了一个称为 FSMFL 的框架来解决多错误定位问题.宗芳芳等人[37]提出了二次定位策略,粗分细找,准确地确定故障的具体位置. 在先前的研究中,我们提出了利用失败执行的上下文信息的缺陷定位方法,进一步提高基于频谱的错误定位的精度[38].与以上的方法不同,在基于频谱信息的基础上,本文所提方法增加了路径分析的相关信息,并在可疑语句怀疑度计算公式中增加了信息熵的相关信息.

6 结束语

针对现有的错误定位方法在定位错误时单独考虑程序实体怀疑度,忽视程序语句之间上下文信息的问题,本文提出了一种基于路径分析和信息熵的上下文错误定位方法.该方法首先进行程序插桩工作,执行插桩后的程序并运行测试用例得到程序的执行覆盖信息,同时通过静态分析对待测源程序构建图形结构,并利用其进行数据依赖分析;接下来根据程序的执行轨迹和数据依赖分析结果对程序的执行路径进行分析,同时利用信息熵理论重新设计怀疑度计算公式,并结合所有执行路径中不同数据依赖关系的覆盖信息分析得到可疑语句的错误定位报告;最后按照可疑度值大小顺序检查语句,定位并修复错误.实验部分本文使用 17 个基准程序对 FLPI 方法进行了验证,对比实验结果表明 FLPI 方法能够有效的提高错误定位的准确度和效率.

虽然我们在不同规模的基准程序集上验证本文方法的有效性,但未来需要在更大规模、更多语言和真实程序上开展研究.同时软件错误定位领域还有许多亟需解决的问题,例如偶然正确性问题、方法级别的错误定位与多错误定位研究等,因此如何将本文方法及程序上下文信息有效的结合并解决这些问题还有待进一步的探索.此外,本文所提方法在提高定位精度的同时也增加了一定的时间开销,因此如何在提高错误定位精度的同时优化时间效率也是一个非常值得深入研究的问题.

References:

[1] Collofello JS, Woodfield SN. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software, 1989, 9(3): 191-195.

[2] Pai GJ, Dugan JB. Empirical analysis of software fault content and fault proneness using bayesian methods. IEEE Transactions on Software Engineering, 2007, 33(10): 675-686.

[3] Zeller A, Hildebrandt R. Simplifying and isolating failure inducing input. IEEE Transactions on Software Engineering, 2002, 28 (2): 183-200.

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

[5] Pearson S, Campos J, Just R, Fraser G, Abreu R, Ernst MD, Pang D, Keller B. Evaluating and improving fault localization. In: Proc. of the IEEE/ACM 39th International Conference on Software Engineering. 2017. 609-620.

[6] Wang KC, Wang TT, Su XH, Ma PJ. Key scientific issues and state-art of automatic software fault localization. Chinese Journal of Computers, 2015, 38(11):2262-2278.

[7] Wong WE, Gao R, Li Y, Abreu R, Wotawa F. A survey on software fault localization. IEEE Transactions on Software Engineering, 2016, 42(8): 707-740.

[8] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the 24th International Conference on Software Engineering. 2002. 467-477.

[9] Jones JA, Harrold MJ, Stasko J. Visualization for fault localization. In: Proc. of the 23rd International Conference on Software engineering. 2001. 71-75.

[10] Abreu R, Zoeteweij P, Golsteijn R, Gemund A. A practical evaluation of spectrum-based fault localization. Journal of Systems and Software, 2009, 82(11): 1780-1792.

[11] Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: Problem determination in large, dynamic internet services. In: Proc. of the 32th International Conference on Dependable Systems and Networks. 2002. 595-604.

[12] Baah GK, Podgurski A, Harrold MJ. The probabilistic program dependence graph and its application to fault diagnosis. IEEE Transactions on Software Engineering, 2010, 36(4): 528-545.

[13] Feyzi F, Parsa S. FPA-FL: Incorporating static fault-proneness analysis into statistical fault localization. Journal of Systems and Software, 2018, 136: 39-58.

[14] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. .Ruan Jian Xue Bao/Journal of Software, 2015, 26(2):390−412 (in Chinese). http://www.jos.org.cn/1000-9825/4708.htm

[15] Zeller A. Why programs fail: a guide to systematic debugging [M]. Holland: Elsevier, 2009.

[16] Aho AV, Lam MS, Sethi R, Ullman JD. Compilers: principles, techniques, & tools [M]. UK: Pearson, 2007.

[17] Troya J, Segura S, Parejo JA, Ruiz-cortes A. Spectrum-based fault localization in model transformations. ACM Transactions on Software Engineering and Methodology, 2018, 27: 3-13.

[18] Ding H, Chen L, Qian J, Xu L, Xu BW. Fault localization method using information quantity. Ruan Jian Xue Bao/Journal of Software, 2013, 24(7):1484−1494 (in Chinese). http://www.jos.org.cn/1000-9825/4294.htm

[19] 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.

[20] Just R, Jalali D, Ernst MD. Defect4J: A database of existing faults to enable controlled testing studies for java programs. In: Proc. of the 23rd International Symposium on Software Testing and Analysis. 2013. 437-440.

[21] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization techniques. In: Proc. of International Conference on Automation Software Engineering. 2005. 273-282.

[22] Wong WE, Qi Y. BP neural network-based effective fault localization. International Journal of Software Engineering and Knowledge Engineering, 2009, 19(4):573-597.

[23] Xie X, Chen TY, Kuo FC, Xu B. 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.

[24] Zhang Z, Tan QP, Mao XG, Lei Y, Chang X, Xue JX. Effective fault localization approach based on enhanced contexts. Ruan Jian Xue Bao/Journal of Software, 2019,30(2):266-281 (in Chinese). http://www.jos.org.cn/1000-9825/5677.htm

[25] Li ZJ, Yan LF, Liu YZ, Zhang ZY, Jiang B, MURE: Making Use of MUtations to refine spectrum-based fault localization. In: Proc. of IEEE International Conference on Software Quality, Reliability and Security Companion. 2018. 56-63.

[26] Kochhar PS, Xia X, Lo D, Li S. Practitioners' expectations on automated fault localization. In: Proc. of the 25th International Symposium on Software Testing and Analysis. 2016. 165-176.

[27] Collofello JS, Cousins L. Towards automatic software fault localization through decision-to-decision path analysis. In: Proc. of the National Computer Conference. 1987. 539-544.

[28] Korel B. PELAS – program error-locating assistant system. IEEE Transactions on Software Engineering, 1988, 14 (9): 1253-1260.

[29] Renieris M, Reiss SP. Fault localization with nearest neighbor queries. In: Proc. the 18th IEEE International Conference on Automated Software Engineering. 2003. 30-39.

[30] Naish L, Lee HJ, Ramamohanarao. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, 2011, 20 (3): 1-32.

[31] Wong WE, Debroy V, Choi B. A family of code coverage based heuristics for effective fault localization. Journal of Systems and Software, 2010, 83 (2): 188-208.

[32] Wong WE, Debroy V, Xu D. Towards better fault localization: A crosstab-based statistical approach. IEEE Transactions on Systems man and Cybernetics Part C-Applications and Reviews, 2012, 42 (3): 378-396.

[33] Wong WE, Debroy V, Gao R, Li Y. The DStar method for effective software fault localization. IEEE Transactions on Reliability, 2014, 63 (1): 290-308.

[34] Souza HA, Mutti D, Chaim ML, Kon F. Contextualizing spectrum-based fault localization. Information and Software Technology, 2018, (94): 245-261.

[35] Wang Z, Fan XY, Zou YG, Chen X. Genetic algorithm based multiple faults localization technique. Ruan Jian Xue Bao/Journal of Software, 2016, 27(4):879−900 (in Chinese). http://www.jos.org.cn/1000-9825/4970.htm

[36] Zheng Y, Wang Z, Fan X, Chen X, Yang Z. Localizing multiple software faults based on evolution algorithm. Journal of Systems and Software, 2018, 139: 107-123.

[37] Zong FF, Huang HY, Ding ZH. Software fault location based on double-times-locating strategy. Ruan Jian Xue Bao/ Journal of Software, 2016, 27(8):1993-2007.

[38] Wang RC, Jiang SJ, Zhang K, Yu Q. Improving the accuracy of spectrum-based fault localization using multiple rules. IEICE Transactions on Information and System, 2020, E103-D(6): 1328-1338.

附中文参考文献:

[6] 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] 陈翔,鞠小林,文万志,顾庆. 基于程序频谱的动态缺陷错误定位方法. 软件学报, 2015.26(2): 390-412. http:// www.jos.org.cn/1000-9825/4708.htm

[18] 丁晖,陈林,钱巨,许蕾,徐宝文. 一种基于信息量的缺陷定位方法. 软件学报, 2013.24(7): 1484-1494. http://www.jos.org.cn/1000-9825/4294.htm

[24] 张卓,谭庆平,毛晓光,雷晏,常曦,薛建新.增强上下文的错误定位技术.软件学报,2019,30(2):266−281. http:// www.jos.org.cn/1000-9825/5677.htm

[35] 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. http://www. jos.org.cn/1000-9825/4970.htm

[37] 宗芳芳, 黄鸿云, 丁佐华. 基于二次定位策略的软件故障定位. 软件学报, 2016, 27(08): 1993-2007. http://www.jos.org.cn/1000-9825/4858.htm