基于混杂偏倚消除的谓词统计错误定位方法

王兴亚[1] 姜淑娟[1),2)] 鞠小林[1),3)] 曹鹤玲[1),4)]

[1)](中国矿业大学计算机科学与技术学院 江苏 徐州 221116)

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

[3)](南通大学计算科学与技术学院 江苏 南通 226019)

[4)](河南工业大学信息科学与工程学院 郑州 450001)

摘 要 识别并消除统计错误定位过程中的混杂偏倚效应可以有效提高错误定位结果的精度.该文对谓词可疑度度量过程中的混杂偏倚现象进行了研究,提出一种基于混杂偏倚效应消除的谓词统计错误定位方法.首先,提出一种基于变量类型的错误候选谓词筛选方法来提高错误相关谓词的识别能力;其次,通过分析影响谓词取值和程序执行结果的条件,识别定位过程中的数据依赖和控制依赖混杂偏倚元素;再次,采用一种静态切片与动态约减相结合的谓词信息收集方法,提高谓词信息的收集效率;最后,针对收集的谓词信息、混杂偏倚元素信息和程序执行结果,使用线性回归分析度量谓词的可疑度,降序排列后提供给开发人员开展程序调试.实验表明该文方法可以有效识别错误相关谓词,提高错误定位的精度和效率.

关键词 谓词错误定位;混杂偏倚元素;因果推理;程序依赖关系;软件测试;程序调试

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02671

Predicate-Level Statistical Fault Localization Based on Confounding Bias Mitigation

WANG Xing-Ya[1] JIANG Shu-Juan[1),2)] JU Xiao-Lin[1),3)] CAO He-Ling[1),4)]

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

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

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

[4)](*College of Information Science and Technology, Henan University of Technology, Zhengzhou* 450001)

Abstract Identifying and reducing the confounding effect can improve the effectiveness of statistical fault localization. In this paper, we present a novel approach that accounts for the effects of program dependencies to mitigate the confounding effect during Predicate-based Statistical Fault Localization (PBSFL). First, we present a variable type-based predicate pre-filtering technique to improve the ability of fault-relevant predicate identification. Then we recognize both the control dependence confounding bias and the data dependence confounding bias by analyzing the mutual dependence between the program predicate and the executing result based on the causal inference. To improve the efficiency and accuracy of the PBSFL, we also design a combined predicate collection technique with static slicing and dynamic reducing. With the run-time information and the executing result, for each predicate we estimate its causal effect by a linear regression analysis and treat the failure-causing effect as the suspiciousness. Finally, they are provided to developers for program debugging. Experimental studies demonstrate that the fault-relevant

收稿日期:2015-04-16;在线出版日期:2016-02-25.本课题得到国家自然科学基金(61673384,61502497)、广西可信软件重点实验室研究课题(kx201530)、南京大学计算机软件新技术国家重点实验室开放课题(KFKT2014B19)、江苏省研究生培养创新工程(KYLX_1390)、河南省高等学校重点科研项目(16A520005)、南通市应用研究计划(BK2014055)资助.**王兴亚**,男,1990 年生,博士研究生,主要研究方向为软件分析与测试.E-mail:xingyawang@cumt.edu.cn.**姜淑娟**(通信作者),女,1966 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为编译技术、软件工程等.E-mail:shjjiang@cumt.edu.cn.**鞠小林**,男,1976 年生,博士,副教授,主要研究方向为软件分析与测试.**曹鹤玲**,女,1980 年生,博士,讲师,主要研究方向为软件分析与测试.

predicate can be identified effectively by the proposed variable type-based predicate pre-filtering technique, and the effectiveness as well as the efficiency of the predicate-based statistical fault localization can be significantly improved after mitigating the dependence confounding effect.

Keywords predicate-level fault localization; confounding bias; causal inference; program dependence; software testing; program debugging

1 引 言

当程序执行失败时，开发人员需要进行程序调试来定位并修正错误. 作为程序调试中最为耗时和费力的过程[1]，错误定位受到了研究人员的广泛关注. 统计错误定位方法是一类有效的错误定位方法[2-3]，该方法通过搜集和分析程序在失败执行和成功执行中的状态信息对程序实体(如语句[4-7]、谓词[8-12]、方法[13]等)进行可疑度计算及排名来定位错误.

基于谓词的统计错误定位方法(Predicate-Based Statistical Fault Localization，PBSFL)以谓词作为研究对象，度量谓词与程序执行失败的关联程度. 关联程度越高，谓词的可疑度越高. 然而，高关联度的谓词并不意味着其是程序执行失败的原因[14]. 软件错误定位过程是寻找引发程序失效原因的过程，可以理解为针对程序执行失效这一现象寻找引发该现象产生原因的因果推理过程. 在因果推理过程中，需要在分析处理变量与输出变量间因果关系的同时考虑其它变量对两者的影响，否则推理过程会受到混杂偏倚效应的影响，造成推理结果不精确[15-16]. 因此，在 PBSFL 过程中需要消除混杂偏倚效应带来的负面影响，以提高谓词可疑度度量的准确度.

在错误定位过程中，开发人员按照谓词可疑度由高到低对谓词进行检查. 基于完美检测假设[17]，开发人员发现错误相关谓词时即可认为完成错误定位. 然而，如果谓词序列中的谓词与错误无关，开发人员就无法找到错误相关谓词，此时错误定位失败. 因此，开展合理的错误候选谓词筛选工作十分重要. Liblit 等人[8]在筛选错误候选谓词时对分支语句、赋值语句和返回语句中的数值型变量进行处理. 然而，程序包含多种类型变量，仅筛选与数值型变量相关的谓词，难以完全说明程序的运行时状态，从而导致错误相关谓词识别能力低.

完整的谓词信息可以准确反映程序的运行时状态，但会带来较大的时空开销和冗余. 随机采样方法[10]可以有效提高谓词信息的收集效率，然而该方法收集的谓词信息存在遗漏，降低了 PBSFL 的定位精度[9]. 软件错误是引发程序执行失败的原因，如果可以去除与程序执行失败无关的谓词，则可以减少错误的搜索域，提高谓词信息收集效率. 同时，分析并避免冗余谓词信息的收集同样也可提高谓词信息收集效率.

进一步的，通过一个示例程序说明 PBSFL 过程中存在的问题，并分析问题产生的原因.

示例程序如图 1 所示，该程序中方法 $fun()$ 包含两个参数，分别是布尔型变量 m 和整型变量 n. 语句 s_1 为错误语句，该语句误将参数 m 的值赋予变量 b. 此时，如果 m 的值为 FALSE，b 被赋予一个巧合性正确值，向下传递一个正确的状态；否则，b 被赋予一个错误的值，向下传递一个错误的状态. 如果该错误状态被传递到程序输出语句(语句 s_7、s_8)，程序执行失败. 因此，可以识别谓词$(b=\text{TRUE})_{s_1}$作为错误相关谓词.

```
     fun (bool m, int n) {
s1     b=m;          //correct: b=FALSE;
s2     if (n<4) {
s3       while (n<0) {
s4         n++;
s5         b=TRUE;
s6       }
s7       print(n);
s8       print(b);
s9   } }
```

图 1 示例程序

$$Tarantula(p)=\frac{n_{ft}(p)/n_f}{n_{ft}(p)/n_f+n_{st}(p)/n_s} \tag{1}$$

$$CBI(p)=\frac{2}{\dfrac{1}{\dfrac{n_{ft}(p)}{n_{ft}(p)+n_{st}(p)}}-\dfrac{1}{\dfrac{n_f(p)}{n_f(p)+n_s(p)}}+\dfrac{1}{\dfrac{\log(n_f(p))}{\log(n_f)}}} \tag{2}$$

为了满足语句、分支及谓词覆盖，设计了以下 5 个测试用例：$t_1=(\text{TRUE},3)$，$t_2=(\text{FALSE},6)$，$t_3=(\text{FALSE},-6)$，$t_4=(\text{TRUE},5)$和 $t_5=(\text{TRUE},0)$. 分别收集 5 个测试用例的测试信息，并比较 Tarantula[4]和 CBI[8]方法在测试信息上的错误定位

结果.表 1 给出示例程序的测试信息及错误定位结果,其中列 1 和列 2 分别为程序语句序号及谓词,列 3～7 为程序测试信息(包括谓词信息和执行结果).当谓词取值为真时,记为 1,否则记为 0.当测试结果为成功时,记为 S,否则记为 F.列 8～9 为 Tarantula 和 CBI 在测试信息上的错误定位结果,两者的可疑度计算方法如式(1)和(2)所示.其中,$n_{st}(p)$和$n_{ft}(p)$分别表示谓词 p 在所有成功和所有失败测试用例中取值为真的数目,$n_s(p)$和$n_f(p)$分别表示谓词 p 在所有成功和所有失败测试用例中被评估的数目,n_f和n_s分别表示成功和失败测试用例的数目.

表 1　示例程序测试信息及错误定位结果

行号	谓词	测试信息					错误定位结果	
		t_1	t_2	t_3	t_4	t_5	Tarantula	CBI
s_1	b=TRUE	1	0	0	1	1	4	14
s_1	b=FALSE	0	1	1	0	0	10	14
s_2	if_3_9	1	0	1	0	1	4	14
s_3	while_4_7	0	0	1	0	0	10	14
s_4	$n>0$	0	0	0	0	0	14	14
s_4	$n=0$	0	0	1	0	0	10	14
s_4	$n<0$	0	0	1	0	0	10	14
s_5	b=TRUE	0	0	1	0	0	10	14
s_5	b=FALSE	0	0	0	0	0	14	14
s_7	$n>0$	1	0	0	0	0	1	1
s_7	$n=0$	0	0	1	0	1	5	3
s_7	$n<0$	0	0	1	0	0	14	14
s_8	b=TRUE	1	0	1	0	1	4	2
s_8	b=FALSE	0	0	0	0	0	14	14
S/F	—	F	S	S	S	F	—	—

一般地,谓词可疑度越大表示谓词出错的可能性越大,在定位错误时按照可疑度大小降序检查.可以看到,方法 Tarantula 和 CBI 均将谓词$(n>0)_{s_7}$和$(b=\text{TRUE})_{s_8}$识别为与错误关联度最高的谓词.当参数 m 值为真时,引发 s_1处错误状态.当分支语句 s_2条件满足时,执行真分支,错误状态向下传递;否则,程序执行结束.因此谓词$(n>0)_{s_7}$和$(b=\text{TRUE})_{s_8}$受到其控制依赖的影响.当参数 m 值为真时,引发 s_1处错误状态,由于 s_5可能传入巧合正确值,程序可能执行成功.当程序执行失败时,均是 s_1处错误状态传递到 s_8处,使得$(b=\text{TRUE})_{s_8}$一直为真.因此谓词$(b=\text{TRUE})_{s_8}$的可疑度受到其数据依赖的影响.由此可见,度量谓词与程序执行结果关系时会受到谓词控制依赖和数据依赖的影响,从而使得错误定位过程存在混杂偏倚效应.因此,在错误定位过程中需要考虑和消除混杂偏倚元素带来的负面影响.

使用 Liblit 方法对数值型变量 var 进行分析,并根据其值域筛选得到 3 个与 var 相关的错误候选谓词:$(var>0)$、$(var=0)$和$(var<0)$,未考虑其他类型变量.针对该示例程序,应用 Liblit 方法无法得到$(b=\text{TRUE})_{s_1}$、$(b=\text{FALSE})_{s_1}$、$(b=\text{TRUE})_{s_5}$、$(b=\text{FALSE})_{s_5}$、$(b=\text{TRUE})_{s_8}$、$(b=\text{FALSE})_{s_8}$等谓词的信息.由于错误相关谓词$(b=\text{TRUE})_{s_1}$与非数值型变量 b 相关,此时应用该方法得到的谓词信息无法反映错误的相关状态,导致错误定位失败.因此,如果能在错误候选谓词筛选过程中得到与程序中非数值型变量的谓词信息,即根据变量的类型开展错误候选谓词筛选工作,则可以减少错误相关状态的遗漏,提高错误相关谓词的识别能力.

开发人员需要收集程序谓词信息及程序执行结果来开展统计错误定位.谓词数量越多、谓词被评估的次数越多,需要收集的信息越多、时空消耗越大.针对该示例程序,需要在程序 5 次执行过程中分别评估和记录$(b=\text{TRUE})_{s_1}$等 14 个谓词的取值结果.然而,并非所有的谓词都与程序执行结果相关(如与语句 s_7相关的谓词),并非所有的谓词都需要被多次评估(如测试用例为 t_3时,谓词$(\text{while_4_7})_{s_4}$首次被评估时已取值为真).此时,谓词信息的收集过程和收集结果存在冗余.因此,如果可以去除与程序执行失败无关的谓词,减少谓词的冗余评估,则可以缩小错误相关谓词的搜索域,进而提高谓词信息收集和错误定位的效率.

针对上述问题,我们首先分析 PBSFL 存在混杂偏倚的原因,通过因果推理识别并消除混杂偏倚元素,从而提高 PBSFL 方法的精度.其次,我们根据程序中变量的类型研究错误候选谓词的筛选方法,以期获得更加充分的谓词信息,提高错误相关谓词识别能力.再次,利用程序切片技术缩减谓词搜索域,并设计一种启发式策略动态约减需要监控的谓词,在保证谓词信息完整性的同时,以提高谓词信息收集效率.总之,本文的主要贡献包括:

(1)分析了数据依赖混杂偏倚和控制依赖混杂偏倚对 PBSFL 方法的负面影响,提出了一种基于混杂偏倚消除的谓词层次错误定位方法;

(2)提出一种基于变量类型的错误候选谓词筛选方法(Variable Type based Predicate Pre-filtering, VTPP),提高了错误相关谓词的识别能力;

(3)提出一种静态切片与动态约简相结合的谓词信息收集方法,提高了谓词信息的收集效率;

(4)设计并实现了本文提出的方法,以通用的 Siemens 数据集[18]和 SIR 程序[19]作为实验对象,通过实验验证了本文方法的有效性.

本文第 2 节介绍本文方法使用的基本概念和定义;第 3 节首先描述方法的处理流程,然后对每个处理部分分别进行介绍;第 4 节使用 Siemens 数据集

和 SIR 程序对本文方法进行实验验证;第 5 节介绍错误定位等相关工作;最后总结全文并讨论进一步的研究工作.

2 基本定义

2.1 因果推理

在现实生活中,我们不仅仅需要了解变量间的关联强度,还希望了解变量之间的因果关系.因果推理[15-16]提供了识别变量间因果关系的理论基础.

通过设置随机实验,可以利用个体在不同处理下的潜在结果来计算变量间的因果效应.因果效应 τ 计算方法如下:

$$\tau = E[Y^1] - E[Y^0] \tag{3}$$

其中,Y^1 表示个体在接受处理下的结果,Y^0 表示个体在未接受处理下的结果,$E[Y^1]$ 表示总体中所有个体都接受处理时结果 Y 的分布,$E[Y^0]$ 表示总体中所有个体都未接受处理时结果 Y 的分布.

然而在实际中,随机试验很难被设置,只能通过观测数据来计算因果效应.此时,变量间的关系会受到混杂偏倚元素的影响.Pearl[15]提出的基于因果图的后门准则是一种有效的混杂偏倚元素识别方法:若存在一个变量 Z 相对于 X 和 Y 满足后门准则,则 X 到 Y 的因果效应是可以度量的,Z 是 X 和 Y 间的混杂偏倚元素.

定义 1. 混杂偏倚元素.变量 Z 对 X 和 Y 存在实际或潜在的影响.在度量变量 X 对变量 Y 的因果效应时,如果未考虑变量 Z 对 X 和 Y 的影响,则该度量过程存在混杂偏倚,变量 Z 称为偏倚元素.

变量间的因果关系可以用因果图表示.因果图是一种有向无环图,其中节点代表随机变量,有向边 $X\rightarrow Y$ 代表 X 是 Y 发生的可能或实际原因.因果图嵌入非统计的先验知识和假设,因此可以指导利用观测数据进行因果推理.图中的路径可以包含前向和后向箭头,因此不是有向路径.其中,后门路径表示以后向箭头开始的路径.

定义 2. 阻断[15].因果图 G 中的路径 $path$ 被变量集合 Z 阻断,需要满足下列条件之一:

(1) $path$ 包含一条链 $X\rightarrow M\rightarrow Y$ 或分支 $X\rightarrow M\leftarrow Y$,其中 $M\in Z$;

(2) $path$ 中至少一个碰撞点 $X\rightarrow M\leftarrow Y$ 满足 M 及其后代不属于 Z.

定义 3. 后门准则[15].在因果图 G 中,变量集合 Z 满足关于变量对$\langle X,Y\rangle$的后门准则当且仅当:

(1) Z 中变量不是 X 的后继;

(2) Z 阻断了 X 到 Y 中的每条后门路径,且包含一条指向 X 的路径.

2.2 程序依赖关系

程序依赖关系反映了程序内部语句间的关系,主要包括控制依赖关系和数据依赖关系.如果语句 s_1 决定语句 s_2 是否执行,则称 s_1 与 s_2 存在控制依赖关系,其中 s_2 控制依赖于 s_1;如果语句 s_2 使用了定义在语句 s_1 中的变量,则称 s_1 与 s_2 存在数据依赖关系,其中 s_2 数据依赖于 s_1.对于程序中的一条语句 s,通过程序依赖关系可以找到所有影响 s 的语句集合(后向程序切片)和所有受到 s 影响的语句集合(前向程序切片).本文主要考虑导致程序执行失败的谓词,因此我们主要使用后向程序切片,文中切片指后向程序切片.

3 本文方法

本节首先概述本文方法,对错误定位的流程进行介绍,之后详细描述其中的重要步骤,包括谓词筛选与收集方法、混杂偏倚元素的识别方法以及错误定位模型.

3.1 方法概述

给定测试用例,监控源程序执行,获取程序谓词取值结果和执行结果,并将其作为下一步错误定位分析的基础.本文方法框架如图 2 所示:

第 1 步:静态切片.通过静态切片可以缩减错误范围,降低需要监控的谓词数量.因此,首先对源程序进行静态切片,获得影响输出变量的语句集合,将其作为可疑语句集合(Faulty Statement Candidate,FSC);

第 2 步:错误候选谓词筛选.针对第 1 步获得的可疑语句集合 FSC,依据程序中语句类型及语句中变量的类型进行错误候选谓词筛选,生成可疑谓词集合(Faulty Predicate Candidate,FPC),将其作为本文的研究对象;

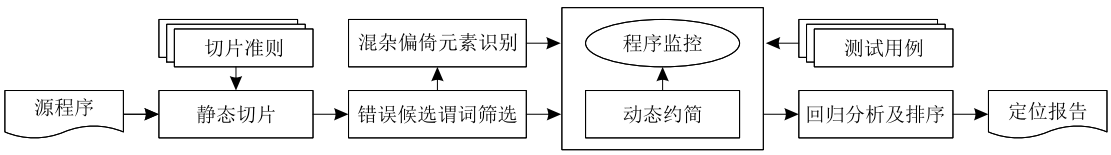


图 2 方法框架

第 3 步：混杂偏倚元素识别. 对第 2 步获得的可疑谓词集合中的每一个元素，通过程序依赖分析构建因果图，并通过因果推理识别导致程序执行失败过程中的混杂偏倚元素；

第 4 步：监控执行与动态约减. 给定测试用例，监控并记录可疑谓词取值结果、可疑谓词偏倚元素的运行时状态和程序执行结果. 在程序运行过程中应用动态约简策略减少冗余的谓词信息；

第 5 步：回归分析及排序. 针对第 4 步收集的谓词取值结果、混杂偏倚元素信息和程序执行结果进行回归分析，度量谓词导致程序执行失败的贡献. 按照贡献度由高到低进行排序，生成谓词序列为开发人员开展错误定位提供帮助.

3.2　谓词筛选与收集

PBSFL 方法以谓词作为研究对象，因此错误候选谓词的筛选和谓词信息的收集效率对 PBSFL 的精度和效率有着重要影响.

3.2.1　基于变量类型的谓词筛选

根据程序中语句类型的不同，基于变量类型的错误候选谓词筛选方法 VTPP 筛选程序中的两类谓词：针对分支语句，VTPP 将其中的逻辑表达式作为原始谓词；针对赋值语句和返回值语句，VTPP 依据语句中的变量筛选得到植入谓词.

不同程序语言中的分支语句类型和变量类型不同，在谓词筛选过程中需要根据实际情况进行分析. 在此，我们以 Java 语言为例进行说明. 如表 2，Java 语言包含 4 种分支语句：if 语句、while 语句、for 语句、switch 语句. 针对这 4 种分支语句，VTPP 将分支语句中的逻辑表达式(*logical_test*)作为原始谓词，在程序执行过程中，逻辑表达式的取值结果即为谓词的取值结果. Java 语言包含基本类型和引用类型两类变量. 基本类型变量又可以分为数值型、布尔型、字符型 3 种类型. 针对变量 *var*，若 *var* 是数值型变量，VTPP 筛选得到 3 个谓词：(*var*＞0)、(*var*＝0)、(*var*＜0)；若 *var* 是布尔型，VTPP 根据其值取真/假情况，筛选得到 2 个谓词：(*var*＝TRUE)、(*var*＝FALSE)；若 *var* 是字符型，因为在文本处理中，数字字符、字母字符、转义字符通常需要被单独考虑，VTPP 筛选得到 4 个谓词：(*var* is a number)、(*var* is a letter)、(*var* is an escape)、(*var* is an other)；对于引用类型变量(如对象、数组、字符串等)，VTPP 根据其值是否为空筛选得到 2 个谓词：(*var*＝NULL)，(*var*！＝NULL).

表 2　基于变量类型的错误候选谓词筛选

语句类型	变量类型	谓词

if(*logical_test*)
while(*logical_test*)
for(*logical_test*)
switch(*logical_test*)
	—	(*logical_test*)

	numeric	(*var*＞0)，(*var*＝0)，(*var*＜0)
	boolean	(*var*＝TRUE)，(*var*＝FALSE)
assignment return	character	(*var* is a number)， (*var* is a letter)， (*var* is an escape)， (*var* is an other)
	reference	(*var*＝NULL)，(*var*！＝NULL)

3.2.2　谓词信息收集

当程序比较大且执行周期比较长时，记录所有的谓词信息需要占用大量的时间和空间，因此提高谓词信息的收集效率可以提高 PBSFL 的定位效率.

程序错误是引发程序执行失败的原因. 本文第 1 节的示例分析表明，并非所有的谓词都与程序执行失败相关，因此收集所有的谓词信息是没有必要的. 当谓词 *p* 在程序执行过程中至少 1 次取值为真，*p* 在该次执行中取值为真，否则取值为假. 如果在程序执行中，*p* 被多次评估且其在前期取值为真，继续评估 *p* 也是没有必要的. 因此，我们考虑从 2 个方面来提高谓词信息的收集效率：

(1) 仅收集与程序执行失败相关的谓词信息；

(2) 设置谓词评估终止条件，避免谓词被重复评估.

本文 2.2 节分析表明，程序输出结果受到其依赖关系语句的影响. 当程序实际输出与期望输出不一致时，程序执行失败. 此时，错误应该存在于与程序输出具有依赖关系的语句集合中. 通过静态切片方法可以找到所有与程序输出相关的语句集合，进一步确定与切片语句相关的谓词集合，从而缩小谓词信息的收集范围.

谓词与语句、变量密切相关. 在程序运行时，如果谓词取值为真，则可以取消对该谓词的监控；如果与语句、变量相关的谓词均取值为真，则可以取消对该语句、变量的监控. 由此，我们可以在程序运行时动态约简冗余的程序监控.

基于上述分析，我们设计了一个静态切片与动态约减相结合的谓词信息收集算法. 如算法 1 所示，算法以源程序 *prog*、输出变量 *out_var* 作为输入，以程序中的谓词信息 *preds* 作为输出.

算法 1. 谓词信息收集算法.

输入：*prog*：源程序

out_var：输出变量

输出：*preds*：谓词信息

符号：*slice*(*prog*,*criteria*)：程序 *prog* 在切片准则 *criteria* 下的切片结果

cancel(*element*)：取消对程序实体的监控

pred_original(*branch*)：针对分支 *branch* 筛选的谓词

preds_indeed(*var*)：针对变量 *var* 筛选的谓词

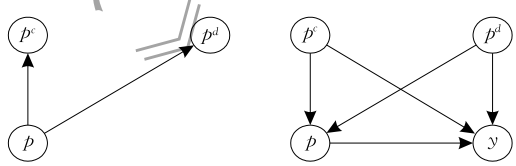
```
1.  criteria←out_var
2.  Faulty Stmt Candidate FSC←slice(prog,criteria)
3.  Faulty Pred Candidate FPC←preds(FSC)
4.  For all pred∈FPC do
5.     pred←FALSE
6.  End for
7.  execute and monitor prog
8.  For each execution instance i do
9.     s←statement(i)
10.    If s∉FSC then continue
11.    If s is a branch then
12.       pred←pred_original(branch)
13.       If pred==TRUE then
14.          (pred←TRUE)&cancel(pred)
15.    Else if s is an assignment or a return then
16.       For all var∈s do
17.          preds←preds(var)
18.          For all pred∈preds do
19.             If pred==TRUE then
20.                (pred←TRUE)&cancel(pred)
21.          End for
22.          If {pred==TRUE|pred∈preds} then
23.             cancel(var)
24.       End for
25.       If {pred==TRUE|pred∈preds(s)} then
26.          cancel(s)
27. End for
28. output predicates information.
```

该算法主要分为 2 步：第 1 步，静态切片(算法行 1～6)．通过静态切片识别与程序输出变量相关的语句，并将其作为错误语句候选集 *FSC*．针对 *FSC* 中的语句依照 3.2.1 节中的错误候选谓词筛选方法筛选谓词，从而生成错误谓词候选集 *FPC*，并将 *FPC* 中谓词的值进行初始化．第 2 步，动态约简(算法行 7～28)．执行源程序并对 *FPC* 中谓词及相关的语句和变量进行监控．对于每一个语句实例 *i*，分析对应的语句 *s* 并根据其类型分别进行处理：如果 *s* 为分支语句 *branch*，如算法行 11～14，生成与 *branch* 相关的原始谓词 *pred*．算法行 13 判断 *pred* 的值是否为真，如果为真则表明 *pred* 在此次执行中取值为真，此时设置 *pred* 值为真，并取消监控 *pred* 及语句 *s*；如果 *s* 为赋值语句 *assignment* 或返回值语句 *return*，如算法行 15～26，此时需要针对 *s* 中所有的谓词分别进行判断．对于 *s* 中的变量 *var*，判断相应的谓词 *pred* 的取值是否为真．如果 *pred* 值为真，则取消监控该谓词．如算法行 22～23，当与 *var* 相关的 *pred* 值均为真时，取消监控该变量．如算法行 25～26，当与 *s* 相关的 *pred* 值均为真时，取消监控该语句．当程序执行结束后，获得该次执行中 *FPC* 中元素的取值结果，从而完成谓词信息的收集．

3.3 混杂偏倚元素识别

在 PBSFL 过程中，谓词的取值不能随意进行设置，因此无法进行随机实验．此时，需要将 PBSFL 过程作为一个观测性研究，在错误定位过程中考虑混杂偏倚元素的影响．

图 3(a)表示谓词 p 及其程序依赖关系．由于因果图是有向无环图，而且因为程序中循环结构的存在，程序中依赖关系可能存在环状结构．对此，本文依据 Baah 等人[20]的方法，只记录谓词 p 的前继控制依赖(下文简称控制依赖)，用 p^c 表示 p 的控制依赖．谓词 p 可能包含多个数据依赖语句，用 p^d 表示 p 的数据依赖集合．



(a)谓词p的程序依赖关系　(b)谓词p的因果关系

图 3 谓词 p 的程序依赖关系及因果关系

在一次程序执行过程中，若一个谓词 p 的值为真，至少需要满足 2 个条件：

(1)谓词 p 所在语句被执行，谓词 p 被评估；

(2)谓词 p 中条件得到满足．

根据 2.2 节分析可知，语句受到其控制依赖语句的支配，因此谓词 p 所在语句是否被执行、谓词 p 是否被评估受到控制依赖语句 p^c 的影响；谓词中变量的值在其数据依赖语句中被定义，因此谓词 p 中条件是否得到满足受到数据依赖语句 p^d 的影响．由此，可以发现 p^c 与 p、p^d 与 p 之间存在因果关系．

在度量程序中元素对程序执行结果的影响时，由于错误的位置未知，程序中的每一个元素都有可能包含错误，都有可能是导致程序执行失败的原因.由此，可以发现 p、p^c、p^d 与程序执行结果 y 之间存在因果关系.

根据上述分析，我们建立与谓词 p 相关的因果图，如图 3(b)所示.可以看到，谓词 p 与程序执行结果 y 之间存在 3 条路径：$path_1$:($p\rightarrow y$)，$path_2$:($p\leftarrow p^c\rightarrow y$)和 $path_3$:($p\leftarrow p^d\rightarrow y$).由于路径 $path_2$ 和 $path_3$ 以后向箭头开始，因此这两个路径是后门路径.由于路径 $path_2$ 包含分支 $p\leftarrow p^c\rightarrow y$，路径 $path_3$ 包含分支 $p\leftarrow p^d\rightarrow y$，根据定义 2 可知，变量集合$\{p^c\}$和$\{p^d\}$分别阻断了路径 $path_2$ 和 $path_3$，变量集合$\{p^c,p^d\}$阻断了谓词 p 与程序执行结果 y 之间所有的后门路径.由于$\{p^c,p^d\}$中元素均非谓词 p 的后继，根据定义 3 可知，$\{p^c,p^d\}$满足 p 与 y 的后门准则.因此，在度量谓词对程序执行失败的贡献度时，存在控制依赖混杂偏倚元素和数据依赖混杂偏倚元素.

3.4　错误定位模型

完成混杂偏倚元素的识别后，我们收集程序中谓词取值信息、谓词混杂偏倚元素信息及程序执行结果，度量谓词的可疑度.错误定位可以理解为针对程序执行失效这一现象寻找引发该现象产生原因的因果推理过程，因此谓词的可疑度可以用谓词取值为真对程序执行失败的因果效应来表示.

存在多种因果效应的度量方法，如逻辑回归模型、非参数与半参数模型，考虑到线性回归模型复杂度比较低且具有较好的诊断能力[21]，我们按照 Baah 等人[20]的研究方法，选择线性回归模型来度量因果效应.线性回归模型如式(4)所示.

$$Y=\alpha+\beta X+\sigma \tag{4}$$

其中，Y 是被解释变量(因变量)，$X=\{X_1,X_2,\cdots,X_n\}$ 是 n 个可以被精确测量并控制的一般变量，表示解释变量(自变量)，α 是回归常数，$\beta=\{\beta_1,\beta_2,\cdots,\beta_n\}$ 是 n 个未知参数，表示回归系数.σ 是随机误差，表示影响 Y 的其它因素.σ 和 X 应当相互独立，即 σ 的取值对 X 不应当有直接的影响.使用最小二乘拟合方程的回归结果，可以计算回归常数和回归系数，得到解释变量对被解释变量的影响程度.

在本文中，我们度量谓词取值为真对程序执行失败的贡献度，因此本文的线性回归模型如式(5)所示.

$$Y=\alpha+\tau^{c,d}T+\beta^{c,d}C+\omega^{c,d}D+\sigma \tag{5}$$

其中 Y 表示程序执行结果；T 为自变量，表示谓词 p 的取值结果；C 为二值协变量，表示 p 的控制依赖混杂偏倚；D 是由一组二值协变量组成的向量，表示 p 的数据依赖混杂偏倚向量；α 表示截距；$\tau^{c,d}$、$\beta^{c,d}$、$\omega^{c,d}$分别表示 T、C、D 的相关系数；σ 表示随机误差项，应与 T 相互独立.

对于程序中的每一个谓词 p，使用式(5)计算其可疑度值时，遵循下面的约定：

(1) 如果程序执行失败，输出变量 $Y=1$，否则 $Y=0$；

(2) 如果 p 取值为真，处理变量 $T=1$，否则 $T=0$；

(3) 如果 p 包含控制依赖谓词且该谓词取值为真，协变量 $C=1$.如果控制依赖谓词取值为假，协变量 $C=0$.如果 p 不包含控制依赖谓词，则该模型不包含协变量 C；

(4) 如果 p 包含数据依赖语句且该语句被执行到，协变量 $D=1$.如果数据依赖语句未被执行，协变量 $D=0$.如果 p 不包含数据依赖语句，则该模型不包含协变量 D.

通过上述回归模型，我们可以计算得到每个谓词接受处理的拟合值 $\tau^{c,d}$，将其作为谓词的可疑度值.完成所有谓词的计算后，降序排列提供给开发人员进行程序调试.

当谓词存在控制依赖关系和数据依赖关系时，如果不考虑数据依赖混杂偏倚 D 对程序输出结果 Y 的影响，D 的影响包含在 σ 中，由于 D 与 T 存在因果关系，使得 T 受到 σ 的影响，T 和 σ 不再独立.此时，度量结果 $\tau^{c,d}$ 存在混杂偏倚.同样，如果没有考虑 C 对 Y 的影响，$\tau^{c,d}$的度量结果存在混杂偏倚.因此，当谓词包含控制依赖关系和数据依赖关系时，使用回归分析进行度量需要考虑控制依赖混杂偏倚和数据依赖混杂偏倚元素，不考虑或仅考虑某一个会导致度量结果依旧存在混杂偏倚.

4　实　验

4.1　实验对象

本文选择了 14 个开源程序作为实验对象，程序特征如表 3 所示，列 1～3 分别给出了程序名称、程序功能和可执行代码数.其中，前 6 个程序来自 Santelices 等人[18]翻译的 Java 版本的 Siemens 程序包，后面的 8 个程序来自于 SIR[19].列 4～5 分别给出了程序中的单/多错误版本数以及测试用例数.其中，单错误版本程序由数据集提供，多错误版本程

表 3 实验对象

程序	描述	代码数	单/多错误版本数	测试用例数
JTcas	碰撞检测器	181	41/20	1608
Tot_info	信息测量	283	23/20	1079
Schedule1	优先排队	290	9/20	2650
Schedule2	优先排队	317	10/20	2710
Print_tokens1	词法分析器	478	7/20	4140
Print_tokens2	词法分析器	410	10/20	4140
NanoXML 1	XML 解析器	4351	7/6	235
NanoXML 2	XML 解析器	5671	7/1	235
NanoXML 3	XML 解析器	6838	10/20	237
NanoXML 5	XML 解析器	7160	8/20	237
Siena 1	发布/订阅系统	5848	2/1	567
Siena 2	发布/订阅系统	5849	1/0	567
Siena 5	发布/订阅系统	6098	2/1	567
Siena 7	发布/订阅系统	6034	5/1	567

序由单错误组合而成. 上述实验对象一共提供了 142 个单错误程序版本,考虑到部分单错误程序版本无法获得失败测试用例,且部分单错误程序版本无法确定错误相关谓词,本文在实验过程中去除了上述版本. 每个多错误版本包含 5 个单错误. 当程序的单错误版本少于 5 个时,该程序的所有错误组合生成 1 个多错误版本. 多错误版本程序与单错误版本程序使用相同的测试用例. 由此,本文针对 14 个程序中的 129 个单错误和 170 个多错误版本程序开展错误定位研究. 其中,错误相关谓词的确定策略如下:

(1) 确定错误位置. 若错误在可执行语句上,标记该可执行语句;若错误为可执行语句的缺失,则标记可执行语句的控制依赖语句; 若缺失语句不包含控制依赖语句,则无法确定错误位置,该错误版本被剔除;

(2) 识别错误相关谓词. 对于(1)确定的错误位置,若该位置语句为条件语句,则与该语句对应的初始谓词即为错误相关谓词; 若该位置语句为赋值语句或返回值语句,则与该语句中变量对应的植入谓词即为错误相关谓词. 否则,无法确定错误相关谓词,该错误版本被剔除.

用于实验的错误定位技术有 8 种,分别为 Tarantula[4]、Ochiai[5]、Naish1[6]、Wong1[7]、CBI[8]、SOBER[22]、Wilcoxon[11]、Mann Whitney[12]. 其中,Tarantula 和 Ochiai 是经典的基于语句覆盖的错误定位方法,Naish1 和 Wong1 是 Xie 等人[17]通过理论证明最优的基于语句覆盖的错误定位方法,CBI、SOBER、Wilcoxon 和 Mann Whitney 是经典的基于谓词的错误定位方法.

4.2 实验设计

本文对基于谓词的错误定位方法进行研究,分析了谓词筛选、谓词信息收集、谓词可疑度计算对错误定位结果及效率的影响. 本节设计了 7 个实验对本文方法进行多维评估.

实验 1.

使用 Siemens 程序和 SIR 程序,将 Liblit 谓词筛选方法作为实验对比对象与本文提出基于变量类型的谓词筛选方法 VTPP 进行比较评估.

实验 2.

使用单错误版本的 Siemens 程序和 SIR 程序,将不考虑混杂偏倚元素 τ、仅考虑控制依赖偏倚元素 τ^c、仅考虑数据依赖偏倚元素 τ^d 的方法作为实验对比对象与本文提出的错误定位方法 $\tau^{c,d}$ 进行比较评估.

实验 3.

使用单错误版本的 Siemens 程序和 SIR 程序,将基于语句覆盖的错误定位方法 Tarantula、Ochiai、Naish1、Wong 以及基于谓词的错误定位方法 CBI、SOBER、Wilcoxon、Mann Whitney 作为实验对比对象与本文提出的错误定位方法 $\tau^{c,d}$ 进行比较评估.

实验 4.

使用单错误版本的 SIR 程序,将 τ、τ^c、τ^d 作为实验对比对象与本文提出的错误定位方法 $\tau^{c,d}$ 在切片结果上进行比较评估.

实验 5.

使用单错误版本 SIR 程序,将 Tarantula、Ochiai、Naish1、Wong、CBI、SOBER、Wilcoxon、Mann Whitney 作为实验对比对象与本文提出的错误定位方法 $\tau^{c,d}$ 在切片结果上进行比较评估.

实验 6.

使用多错误版本的 Siemens 程序和 SIR 程序,将 τ、τ^c、τ^d、Tarantula、Ochiai、Naish1、Wong、CBI、SOBER、Wilcoxon、Mann Whitney 作为实验对比对象与本文提出的错误定位方法 $\tau^{c,d}$ 进行比较评估.

实验 7.

使用 Siemens 程序和 SIR 程序,将完整收集谓词信息的方法作为实验对比对象与本文提出的谓词信息收集方法进行比较评估.

实验 1 的目的是分析本文谓词筛选方法用于识别错误相关谓词的可行性与准确度; 实验 2～5 的目的是分析本文错误定位方法用于单错误版本程序上的效果; 实验 6 的目的是分析本文错误定位方法用于多错误版本程序的效果; 实验 2、4、6 的目的是分析不同混杂偏倚元素对错误定位方法的影响; 实

验 7 的目的是分析本文谓词信息收集方法的效率.

4.3 评价指标

本文采用定位代价($Expense$[4])来评估错误定位的准确度，$Expense$ 表示使用选定的错误定位技术定位到错误时需要检查的谓词数 num_{examined} 占所有谓词数 $num_{\text{predicate}}$ 的百分比，使用式(6)计算. $Expense$ 值越低，定位错误时需要检查的谓词数量越少，定位精度越高.

$$Expense=\frac{num_{\text{examined}}}{num_{\text{predicate}}} \tag{6}$$

在单错误版本程序的定位过程中，本文采用绝对差值 AI[9](Absolute Improvement)来比较两种错误定位算法的优劣，AI 使用式(7)计算. 其中，$Expense_{\text{exp}}$ 和 $Expense_{\text{ref}}$ 分别表示实验方法和参照方法的绝对差值. AI 为正时，表示实验方法定位错误时需要检查的谓词数量更少，实验方法的错误定位效果更好.

$$AI=Expense_{\text{ref}}-Expense_{\text{exp}} \tag{7}$$

在多错误版本程序的定位过程中，本文采用检查得分($EXAM$[23])来比较不同错误定位方法的定位效果，$EXAM$ 表示错误检出率 $rate_{\text{fault}}$ 与谓词检查率 $rate_{\text{examined}}$ 的比值，使用式(8)计算. $EXAM$ 值越高，表示相同谓词检测率时，检查出的错误数量越多.

$$EXAM=\frac{rate_{\text{fault}}}{rate_{\text{examined}}} \tag{8}$$

4.4 实验结果分析

我们依据实验步骤进行实验验证，并使用评价指标对实验结果进行分析. 为了更直观地观察实验结果，我们将各技术及本文方法的定位结果用图表方式表示并进行比较和结果分析.

实验 1.

表 4 给出了 Liblit 方法及本文方法 VTPP 在 Siemens 程序和 SIR 程序上的错误候选谓词筛选结果，其中列 2～5 分别给出 Liblit 方法及 VTPP 方法在每个程序上生成的谓词数目、初始谓词数目和植入谓词数目. 可以看到，Liblit 方法与 VTPP 方法均将程序分支语句中的逻辑表达式作为初始谓词，两种方法筛选的初始谓词数目相同. 在植入谓词筛选过程中，Liblit 方法只对数值型变量进行谓词筛选，而 VTPP 方法需要对各种类型变量进行谓词筛选，因此 VTPP 方法筛选的植入谓词数目和总谓词数目更多.

表 4　错误候选谓词筛选结果

程序	Liblit		VTPP		切片后 VTPP	
	谓词数	原始/植入谓词数	谓词数	原始/植入谓词数	谓词数	原始/植入谓词数
JTcas	299	86/213	439	86/353	—	—
Tot_info	1031	146/885	1397	146/1251	—	—
Schedule1	484	112/372	1018	112/906	—	—
Schedule2	549	128/421	1029	128/901	—	—
Print_tokens1	614	155/459	1395	155/1240	—	—
Print_tokens2	557	212/345	1187	212/975	—	—
NanoXML 1	1716	354/1362	9675	354/9321	1289	58/1231
NanoXML 2	1927	364/1563	10648	364/10284	1273	51/1222
NanoXML 3	2200	436/1764	12075	436/9839	1495	79/1416
NanoXML 5	2325	463/1862	10192	463/9729	1667	82/1585
Siena 1	4286	1633/2653	12496	1633/10863	1466	205/1261
Siena 2	4290	1634/2656	12502	1634/10868	1474	206/1268
Siena 5	4339	1626/2713	12693	1626/11067	1473	205/1268
Siena 7	4248	1616/2632	12562	1616/10946	1372	188/1184

应用 Liblit 方法的谓词结果对 129 个单错误程序版本的错误相关谓词进行识别研究，可以发现程序 JTcas 中的 17 个单错误版本、Schedule 中的 2 个单错误版本、Print_tokens1 中的 4 个单错误版本、NanoXML 中的 5 个单错误版本、Siena 程序中的 2 个单错误版本中的错误相关谓词无法被识别. 而本文方法筛选的谓词可以识别所有单错误版本程序中的错误相关谓词.

表 4 中列 6 和列 7 给出了本文方法在 SIR 程序切片结果上生成的谓词数目、初始谓词数目和植入谓词数目，可以发现切片后 8 个版本中谓词数分别下降了 86.68%、88.04%、87.62%、83.64%、88.27%、88.21%、88.40%、89.08%，平均下降了 87.60%. 应用切片后的谓词结果对 SIR 程序中的 36 个单错误程序版本的错误相关谓词进行研究，可以发现其可以识别所有的错误相关谓词.

实验结果表明，应用本文方法筛选得到的谓词结果可以有效地识别程序中的错误相关谓词，应用切片方法可以在不影响错误相关谓词检测能力的同时有效地减少筛选的谓词数目.

实验 2.

图 4 表示本文方法 $\tau^{c,d}$ 与不考虑混杂偏倚元素 τ、仅考虑控制依赖混杂偏倚元素 τ^{c}、仅考虑数据依赖混杂在偏倚元素 τ^{d} 等 3 个方法在定位单错误 Siemens 程序和 SIR 程序时的比较结果. 其中，x 轴表示各个错误版本，y 轴表示 $\tau^{c,d}$ 与其它方法的定位结果比较，用 AI 评价指标度量，其中 $\tau^{c,d}$ 作为实验方法，τ、τ^{c}、τ^{d} 作为参照方法.

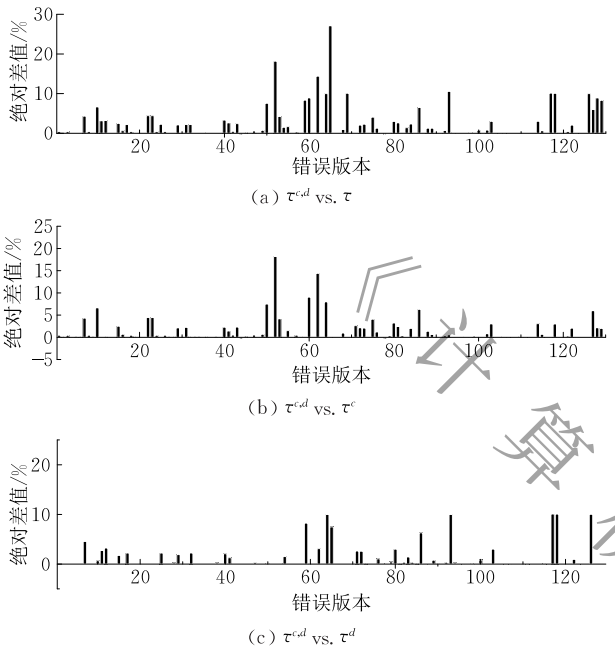


图 4 不同偏倚元素对错误定位结果的影响

图 4(a)表明，在 129 个单错误定位实验中，本文方法 $\tau^{c,d}$ 在 94.57%的情况下不低于 τ 方法，平均绝对差值为 9.85%－7.72%＝2.13%. 图 4(b)表明，$\tau^{c,d}$ 在 94.57%的情况下不低于 τ^{c} 方法，平均绝对差值为 8.97%－7.72%＝1.24%. 图 4(c)表明，$\tau^{c,d}$ 在 92.25%的情况下不低于 τ^{d} 方法，平均绝对差值为 8.66%－7.72%＝0.94%. 通过以上数据得知，本文方法比不考虑偏倚元素或仅考虑单个依赖性偏倚元素的方法在单错误程序上的定位代价更小. 当谓词与程序执行结果无关时，消除谓词与执行结果间的偏倚所带来的影响是无法判断的. 考虑偏倚后，缺陷相关谓词排名提高的速度可能没有无关谓词提升的速度高，因此存在错误相关谓词排名下降的现象.

为进一步分析本文方法的有效性，对本文方法和 τ、τ^{c}、τ^{d} 方法的定位代价开展 Wilcoxon 配对检验，给出的备择假设为：本文方法比其它方法的定位代价更小. Wilcoxon 检验所得 p 值分别为 0.03、0.18、0.30. 因此，我们至少在置信度水平 70%下接受备择假设.

实验结果表明：在单错误程序的错误定位过程中，同时考虑程序中的数据依赖偏倚和控制依赖偏倚比不考虑依赖偏倚元素或仅考虑单个依赖偏倚元素可以获得更小的错误定位代价.

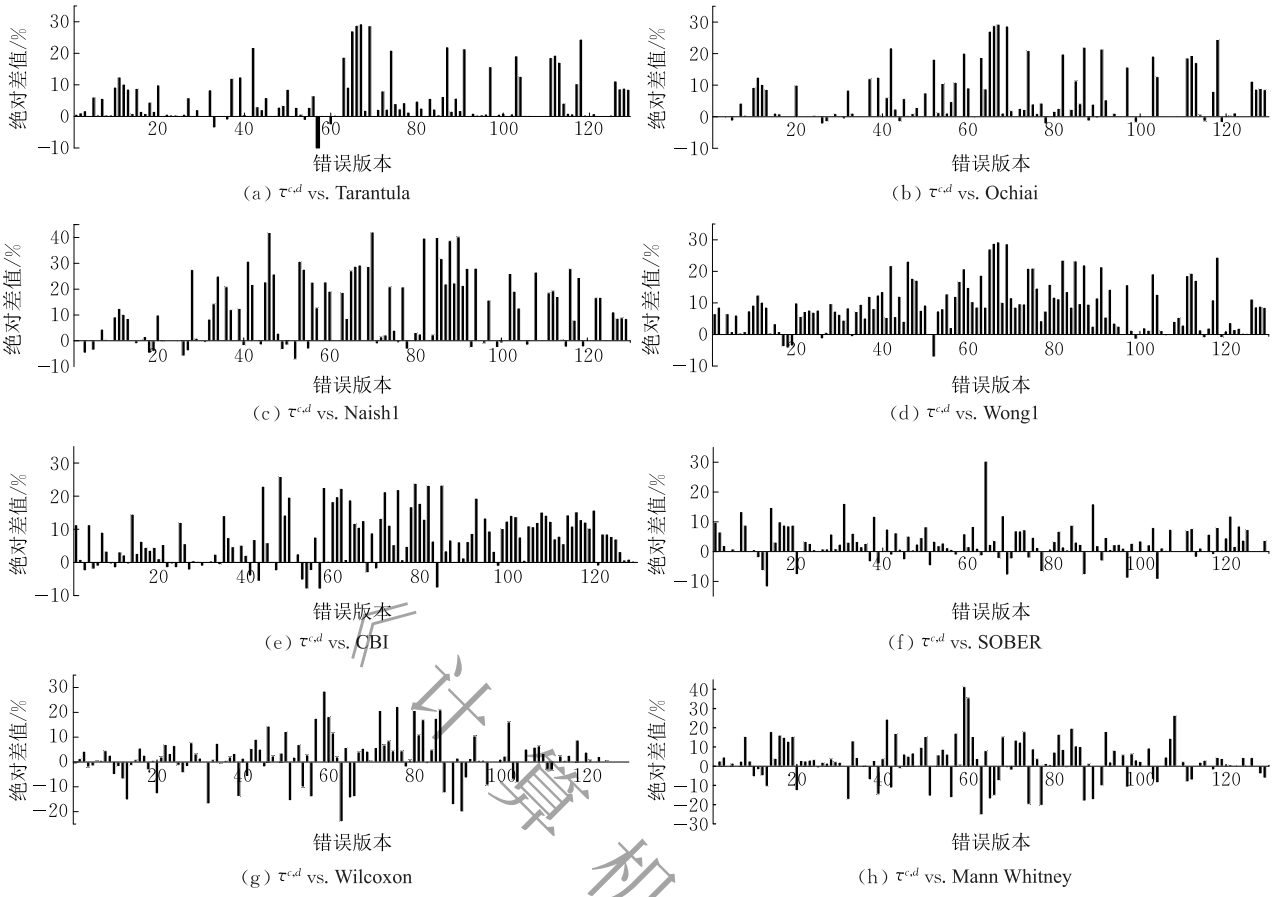
实验 3.

图 5 表示本文方法与 Tarantula、Ochiai、Naish1、Wong1、CBI、SOBER、Wilcoxon、Mann Whitney 等 8 个方法在定位单错误 Siemens 程序和 SIR 程序时的比较结果. 图 5 的结构与图 4 相同，限于篇幅限制不再赘述. 本文方法 $\tau^{c,d}$ 作为实验方法，其它方法作为参照方法.

图 5(a)表明：在 129 个单错误定位实验中，$\tau^{c,d}$ 在 95.35%的情况下不低于 Tarantula 方法，平均绝对差值为 12.61%－7.72%＝4.89%；图 5(b)表明，$\tau^{c,d}$ 在 82.95%的情况下不低于 Ochiai 方法，平均绝对差值为 12.79%－7.72%＝5.07%；图 5(c)表明，$\tau^{c,d}$ 在 74.42%的情况下不低于 Naish1 方法，平均绝对差值为 17.32%－7.72%＝9.60%；图 5(d)表明，$\tau^{c,d}$ 在 90.70%的情况下不低于 Wong1 方法，平均绝对差值为 16.46%－7.72%＝8.74%；图 5(e)表明，$\tau^{c,d}$ 在 81.40%的情况下不低于 CBI 方法，平均绝对差值为 14.65%－7.72%＝6.93%；图 5(f)表明，$\tau^{c,d}$ 在 82.17%的情况下不低于 SOBER 方法，平均绝对差值为 10.49%－7.72%＝2.77%；图 5(g)表明，$\tau^{c,d}$ 在 70.54%的情况下不低于 Wilcoxon 方法，平均绝对差值为 9.29%－7.72%＝1.57%；图 5(h)表明，$\tau^{c,d}$ 在 71.31%的情况下不低于 Mann Whitney 方法，平均绝对差值为 10.36%－7.72%＝2.64%. 通过以上数据得知，本文方法比其它方法的定位代价更小.

对本文方法和其它方法的定位代价开展 Wilcoxon 配对检验来进一步分析本文方法的有效性，给出的备择假设为：本文方法比其它方法的定位代价更小. Wilcoxon 检验所得 p 值除 Wilcoxon 方法(0.10)之外均小于 0.05，这表明至少可以在 90%的情况下接受备择假设，在 95%的情况下接受其余的检验假设. 因此，可以表明本文方法 $\tau^{c,d}$ 可以显著降低定位代价.

实验结果表明：在单错误程序的错误定位过程中，本文方法较其它方法可以获得更小的错误定位代价.



(a) $\tau^{c,d}$ vs. Tarantula　(b) $\tau^{c,d}$ vs. Ochiai

(c) $\tau^{c,d}$ vs. Naish1　(d) $\tau^{c,d}$ vs. Wong1

(e) $\tau^{c,d}$ vs. CBI　(f) $\tau^{c,d}$ vs. SOBER

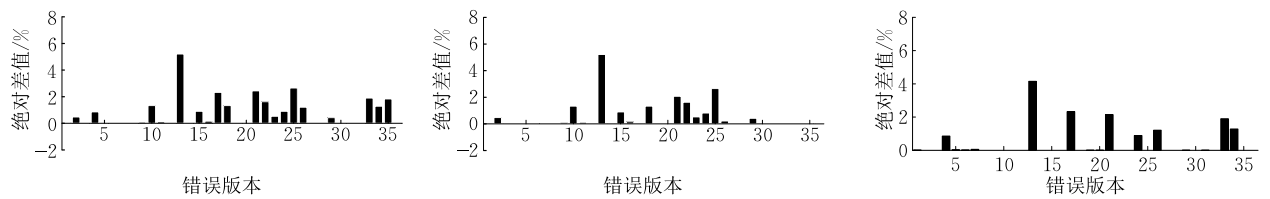
(g) $\tau^{c,d}$ vs. Wilcoxon　(h) $\tau^{c,d}$ vs. Mann Whitney

图 5　本文方法与其它方法的对比

实验 4.

图 6 表示本文方法与不考虑混杂偏倚元素 τ、仅考虑控制依赖混杂偏倚元素 τ^c、仅考虑数据依混杂在偏倚元素 τ^d 等 3 个方法在定位切片后单错误 SIR 程序时的比较结果. 其中本文方法 $\tau^{c,d}$ 作为实验方法，τ、τ^c、τ^d 作为参照方法.

图 6(a)表明，在 36 个切片后单错误定位实验中，本文方法 $\tau^{c,d}$ 在 97.22% 的情况下不低于 τ 方法，平均绝对差值为 1.48%－0.71%＝0.77%. 图 6(b)表明，$\tau^{c,d}$ 在 97.22% 的情况下不低于 τ^c 方法，平均绝对差值为 1.21%－0.71%＝0.50%. 图 6(c)表明，$\tau^{c,d}$ 在 100% 的情况下不低于 τ^d 方法，平均绝对差值为 1.12%－0.71%＝0.41%. 通过以上数据得知，本文方法比不考虑偏倚元素或仅考虑单个依赖性偏倚元素的方法在切片后单错误程序上的定位代价更小.



(a) $\tau^{c,d}$ vs. τ　(b) $\tau^{c,d}$ vs. τ^c　(c) $\tau^{c,d}$ vs. τ^d

图 6　不同偏倚元素在切片上对错误定位结果的影响

对本文方法和其它方法的定位代价开展 Wilcoxon 配对检验来进一步分析本文方法的有效性，给出的备择假设为：本文方法比其它方法的定位代价更小. Wilcoxon 检验所得 p 值分别为 0.02、0.12、0.17. 因此，我们至少在置信度水平 83% 下接受备择假设.

实验结果表明：在切片后单错误程序的错误定位过程中，同时考虑程序中的数据依赖偏倚和控制依赖偏倚比不考虑依赖偏倚元素或仅考虑单个依赖偏倚元素可以付出更小的错误定位代价.

实验 5.

图 7 表示本文方法与 Tarantula、Ochiai、Naish1、Wong1、CBI、SOBER、Wilcoxon、Mann Whitney 等

8 个方法在定位切片后单错误 SIR 程序时的比较结果.本文方法 $\tau^{c,d}$ 作为实验方法,其它方法作为参照方法.

图 7(a)表明,在 36 个切片后单错误定位实验中,本文方法 $\tau^{c,d}$ 在 100%的情况下不低于 Tarantula 方法,平均绝对差值为 12.61%－7.72%＝4.89%;图 7(b)表明,$\tau^{c,d}$ 在 86.11%的情况下不低于 Ochiai 方法,平均绝对差值为 12.79%－7.72%＝5.07%;图 7(c)表明,$\tau^{c,d}$ 在 86.11%的情况下不低于 Naish1 方法,平均绝对差值为 17.32%－7.72%＝9.60%;图 7(d)表明,$\tau^{c,d}$ 在 86.11%的情况下不低于 Wong1 方法,平均绝对差值为 16.46%－7.72%＝8.74%;图 7(e)表明,$\tau^{c,d}$ 在 97.22%的情况下不低于 CBI 方法,平均绝对差值为 14.65%－7.72%＝6.93%;图 7(f)表明,$\tau^{c,d}$ 在 83.33%的情况下不低于 SOBER 方法,平均绝对差值为 10.49%－7.72%＝2.77%;图 7(g)表明,$\tau^{c,d}$ 在 86.11%的情况下不低于 Wilcoxon 方法,平均绝对差值为 9.29%－7.72%＝1.57%;图 7(h)表明,$\tau^{c,d}$ 在 88.89%的情况下不低于 Mann Whitney 方法,平均绝对差值为 10.36%－7.72%＝2.64%.通过以上数据得知,本文方法比其它方法的定位代价更小.

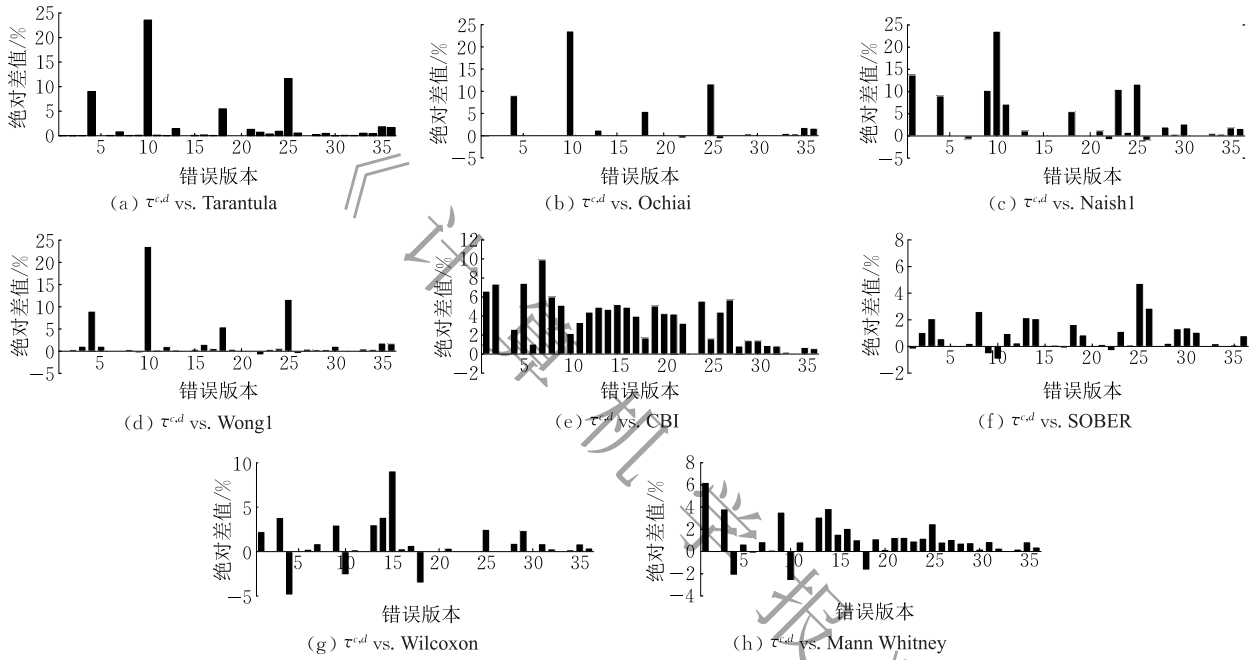


图 7 本文方法与其它方法在切片上的对比

对本文方法和其它方法的定位代价开展 Wilcoxon 配对检验来进一步分析本文方法的有效性,给出的备择假设为:本文方法比其它方法的定位代价更小.Wilcoxon 检验所得 p 值除 Ochiai 方法(0.09)和 Wilcoxon 方法(0.05)之外均小于 0.05,这表明至少可以在 91%的情况下接受备择假设,在 95%的情况下接受其余的检验假设.因此,可以表明本文方法 $\tau^{c,d}$ 可以显著降低定位代价.

实验结果表明:在切片后单错误程序的错误定位过程中,本文方法较其它方法而言可以付出更小的错误定位代价.

实验 6.

图 8 表示 Tarantula、Ochiai、Naish1、Wong、CBI、SOBER、Wilcoxon、Mann Whitney、τ、τ^c、τ^d 以及本文方法 $\tau^{c,d}$ 等 12 个方法在定位多错误 Siemens 程序和 SIR 程序时的结果.在图 8 中,x 轴表示需要检查的谓词数占谓词数的百分比,y 轴表示发现的错误数占错误数的百分比,用 $EXAM$ 指标度量.当谓词检测率相同时,$EXAM$ 值越大,表示检出的错误数量越多,错误定位效果越好.

如图 8 所示,本文方法 $\tau^{c,d}$ 一直具有良好的定

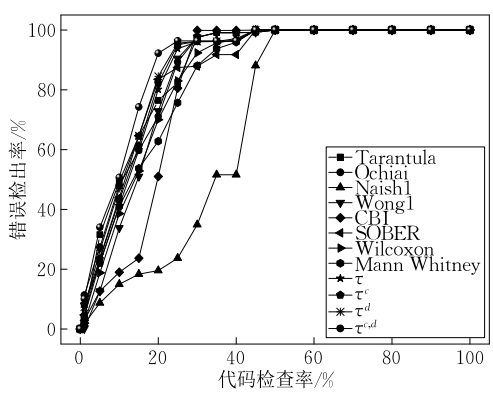


图 8 多种方法在多错误程序版本上的错误定位效果[0,100%]

位效果，当谓词检查率低于 25%时，$\tau^{c,d}$方法总是比其它方法具有更高的 *EXAM* 值. 此时，$\tau^{c,d}$方法已经可以检测出 96.32%的错误. 当谓词检查率在区间[25%，45%]时，$\tau^{c,d}$的 *EXAM* 值略低于 CBI 方法和 Wong1 方法，依然具有较好的定位效果.

表 5　本文方法与其它方法在不同程序上的时间代价对比

程序	定位方法	监控执行/s	可疑度计算/s	静态切片/s	动态约简执行/s	其它/s	总计/s	$\frac{\#\tau}{\#\mathrm{SFL}}$
JTcas	SFL	554.76	13.63	0	0	2.69	571.08	1.12
	$\tau^{c,d}$	619.08	16.86	0	0	6.52	642.46	0.80
	τ^{x}	0	16.86	0	435.77	6.52	459.15	
Tot_info	SFL	11396.01	45.33	0	0	3.02	11444.36	1.03
	$\tau^{c,d}$	11772.71	53.13	0	0	7.44	11833.28	0.74
	τ^{x}	0	53.13	0	8455.71	7.44	8516.28	
Schedule1	SFL	4058.47	61.83	0	0	2.98	4123.28	1.18
	$\tau^{c,d}$	4763.37	83.01	0	0	6.62	4853.00	0.88
	τ^{x}	0	83.01	0	3554.98	6.62	3644.61	
Schedule2	SFL	10352.20	66.12	0	0	3.00	10421.32	1.31
	$\tau^{c,d}$	13582.52	88.16	0	0	6.67	13677.35	0.29
	τ^{x}	0	88.16	0	2883.44	6.67	2978.27	
Print_tokens1	SFL	4512.19	90.01	0	0	3.59	4605.79	1.22
	$\tau^{c,d}$	5501.23	98.16	0	0	8.65	5608.04	0.89
	τ^{x}	0	98.16	0	4005.45	8.65	4112.26	
Print_tokens2	SFL	19406.66	74.98	0	0	3.71	19485.35	1.05
	$\tau^{c,d}$	20461.54	82.47	0	0	7.62	20551.63	0.81
	τ^{x}	0	82.47	0	15736.97	7.62	15827.06	
NanoXML 1	SFL	1489.43	236.79	0	0	5.39	1731.61	1.21
	$\tau^{c,d}$	1814.51	265.41	0	0	12.59	2092.51	0.34
	τ^{x}	0	104.45	155.21	316.62	12.59	588.87	
Siena	SFL	2833.02	239.89	0	0	5.58	3078.49	1.03
	$\tau^{c,d}$	2875.82	280.44	0	0	15.38	3171.64	0.35
	τ^{x}	0	74.75	43.34	950.292	15.38	1083.76	

在错误定位过程中，当定位代价高于 20%时，错误定位结果并不能显著提高程序调试效率，此次错误定位过程意义较小[24]. 因此，我们比较 12 种方法在谓词检查率区间[0，20%]的错误定位效果，比较结果如图 9 所示.

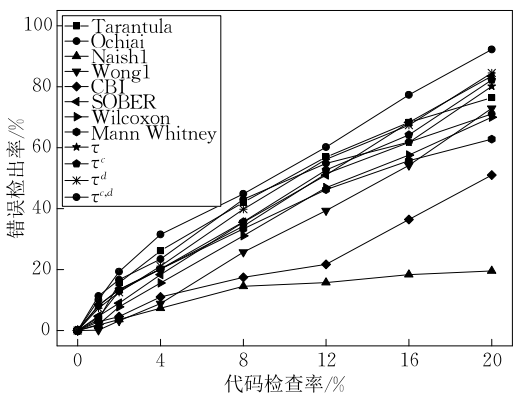


图 9　多种方法在多错误程序版本上的错误定位效果[0，20%]

如图 9 所示，本文方法 $\tau^{c,d}$在谓词检查率区间[0，20%]段一直具有良好的定位效果，并始终领先于其它 11 种方法. 当谓词检查率为 20%时，$\tau^{c,d}$方法可以检测出 92.19%的错误，其它方法可以检测出 19.60%～84.53%的错误，由此表明 $\tau^{c,d}$在多数情况中具有较合理的错误定位代价.

实验结果表明：在多错误程序的错误定位过程中，本文方法较其它方法可以获得更好的错误定位效果.

实验 7.

表 5 给出了统计错误定位方法 SFL、本文方法 $\tau^{c,d}$以及对本文方法进行优化后的方法 τ^{x}在定位 Siemens 程序和 SIR 程序时的时间代价. 不失一般性，选择 Tarantula 方法作为 SFL 方法的代表用于比较. 其中列 3 表示 SFL 和 $\tau^{c,d}$方法在运行所有测试用例耗费的时间；列 4 表示 SFL、$\tau^{c,d}$和 τ^{x}方法计算谓词可疑度耗费的时间；同 SFL 方法相比，$\tau^{c,d}$方法需要额外使用混杂偏倚元素的信息，运行测试用例和计算谓词可疑度时间较多. 列 5 和列 6 给出了优化策略下 τ^{x}耗费的时间. 其中列 5 给出了静态切片需要耗费的时间，考虑到 Siemens 程序规模较小，本文仅对规模较大的 SIR 程序开展静态切片优化；

列 6 给出了 τ^x 方法在运行所有测试用例耗费的时间;列 7 给出了筛选谓词及识别谓词偏倚元素需要的时间,由于 SFL 方法无需识别偏倚元素,其耗费时间较少;最后 1 列给出本文方法与 SFL 方法整体耗费时间的比值.

本文借助 Java Debugger Interface(JDI)的中断机制来收集程序的运行时信息,测试信息的收集过程需要消耗大量的时间.计算谓词可疑度时,需要分析每个谓词在不同执行中的状态,可疑度计算过程需要消耗较多的时间.因此,τ^x 采用静态切片来约简谓词数目,采用动态约简来减少收集冗余信息.列 9 数据表明,同 SFL 方法相比,本文方法在未采用优化策略前时间耗费较高,最差情况下需要耗费 1.31 倍(Schedule2)的时间.采用动态约简策略后,测试用例的运行时间有了较大的改进.对于 SIR 程序,采用静态切片减少了谓词的数目,因此谓词可疑度计算时间也有了一定的改进.总体上,采用优化策略后,本文方法的时间代价得以改善,最差情况下需要耗费 0.89 倍的时间(Print_tokens 1),最好情况下需要耗费 0.29 倍的时间(Schedule2).

实验结果表明:与完整收集谓词信息的方法相比,本文提出的谓词信息收集方法具有更低的时间代价.

4.5 实验结论

通过 4.4 节中对实验结果的观察与分析,我们得到 4 点初步结论.

结论 1.

通过实验 1 的分析,我们观察到,与 Liblit 方法相比,本文提出的缺陷候选谓词筛选方法 VTPP 可以识别更多的错误相关谓词.因此得出结论:本文谓词筛选方法 VTPP 可以更有效地识别错误相关谓词,保证错误定位正常开展.

结论 2.

结合实验 2、实验 4 和实验 6 的分析,我们观察到,与不考虑或仅考虑单个依赖性混杂偏倚元素的方法相比,本文提出的错误定位方法 $\tau^{c,d}$ 无论在单错误程序实验中,还是在多错误程序实验中均表现出较小的定位代价.因此得出结论:谓词层次错误定位过程中受到了混杂偏倚元素的负面影响,通过回归分析方法消除混杂偏倚效应可以提高错误定位的精度.

结论 3.

结合实验 3、实验 5 和实验 6 的分析,我们观察到,与经典的基于语句覆盖的错误定位方法和基于谓词的错误定位方法相比,本文提出的错误定位方法 $\tau^{c,d}$ 无论在单错误程序实验中,还是在多错误程序实验中均表现出较小的定位代价.因此得出结论:本文错误定位方法具有更好的错误定位效果.

结论 4.

通过实验 7 的分析,我们观察到,采用静态切片和动态约简策略后,谓词信息的规模和收集时间代价均有较大幅度的降低.因此得出结论:本文的谓词信息收集方法可以更有效地收集谓词信息,提高错误定位的效率.

4.6 有效性分析

本文从内部效度、外部效度、构造效度等 3 个方面来分析本文方法的有效性.

影响本文方法内部效度的因素主要包括 2 个方面.首先,错误定位对比方法的实现准确度对实验结果有影响.本文选用 4 个基于语句的覆盖的错误定位方法[4-7]和 4 个基于谓词的错误定位方法[8,11-12,22]作为实验对比对象,这些方法按照文献给出的描述进行复现,因此实验对比结果是准确的.其次,程序依赖关系及切片结果的计算准确度对实验结果有影响.为此,本文在方法实现时借鉴 Zhang 等人[25]提出的程序切片算法,并引入别名分析方法来提高切片精度[26].

影响本文方法外部效度的因素主要包括 2 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 8 个中等规模程序作为实验对象[18-19],缺乏大规模实验对象,因此不足以保证本文结论的推广性.为此,本文选择了在错误定位研究领域被广泛使用的实验对象[2-3],这在一定程度上弥补了不足.其次,人工注入错误对本文结论的可推广性同样有影响.本文研究的单错误程序和多错误程序中的错误均为人工注入错误.尽管这些错误并非原发错误,这些错误均为研究人员精心设计[19],与原发错误对错误定位方法的影响一致[27].

影响本文方法构造效度的因素主要在于本文错误定位方法的复杂度.本文错误定位方法分析了数据依赖偏倚元素和控制依赖混杂偏倚元素对谓词统计错误定位过程中的混杂偏倚影响,并利用回归分析方法消除该影响以提高错误定位的精度.与其它方法相比,本文方法由此引入的程序依赖性分析势必会增加方法的复杂度,降低错误定位的效率.为此,本文提出了静态切片与动态约简相结合的谓词信息收集方法,实验表明该方法有效地提高了错误定位效率.

5 相关工作

基于谓词的错误定位方法是一类有效的错误定

位方法. Liblit 等人[8,10]提出CBI方法，该方法通过平衡谓词的特异性(specificity)和灵敏性(sensitivity)来度量谓词出错的可疑度. Liu等人[22,24]提出 SOBER 定位方法，该方法对谓词在成功执行和失败执行中取值模式进行建模，然后基于统计学中假设检验的原理，量化每个谓词的错误相关性. Zhang 等人[11-12]提出了基于非参数假设检验的谓词错误定位方法，并应用 Wilcoxon 检验和 Mann Whitney 检验度量谓词的可疑度. 实验研究表明上述方法可以取得较好的错误定位效果，然而，上述方法在分析过程中仅考虑了谓词与程序执行结果间的关系，忽略了其它因素对两者共同造成的影响，因此度量过程存在混杂偏倚. Gore 等人[9]分析了控制依赖混杂偏倚元素对谓词和程序执行结果造成的混杂偏倚影响. 本文在度量谓词可疑度时，进一步考虑了谓词数据依赖和控制依赖对度量过程造成的混杂偏倚效应，并通过回归分析消除该偏倚影响，由此获得了无偏的度量结果. 实验结果表明本文方法错误定位精度更高.

谓词信息反映了程序的运行时状态，筛选合理的缺陷候选谓词可以帮助程序调试人员快速定位到程序中的错误位置. Liblit 等人[10]针对赋值语句和返回值语句中的数值型变量 var 筛选得到 3 个错误谓词：($var>0$)、($var=0$)、($var<0$). Gore 等人[28]提出 ESP 方法，对 Liblit 方法中的谓词进行修改：($var=\mu_{var}$)、($var>\mu_{var}+\sigma_{var}$)、($var<\mu_{var}+\sigma_{var}$)，其中 μ_{var} 和 σ_{var} 分别表示变量 var 在程序执行过程中的平均值和标准差. 上述方法筛选得到的谓词在一定程度反映了程序的状态，但由于未考虑其他类型变量，导致部分错误相关谓词无法识别. 本文提出了基于变量类型的缺陷候选谓词筛选方法，提高了错误相关谓词的识别能力.

本文实验研究表明，谓词信息收集是谓词层次错误定位过程中耗时最多的步骤，完整收集程序运行时的谓词信息占用大量的时间. Liblit 等人[10]通过随机采样方法 Sample 来收集谓词信息，实验表明该方法可以有效降低谓词信息收集的时间. 然而，Sample 方法不能收集完整的谓词信息，难以真实地反映谓词的取值结果，对 PBSFL 的定位精度有较大的影响[9]. 本文提出了静态切片和动态约简 2 个策略来去除错误无关谓词和冗余谓词信息，以保证不遗漏错误相关谓词信息的同时，提高谓词信息的收集效率.

6　结束语

本文提出了一种基于混杂偏倚消除的谓词层次错误定位方法. 首先，分析了谓词筛选对错误定位的影响，提出了一种基于变量类型的缺陷候选谓词筛选方法以提高错误谓词的识别能力；其次，提出了静态切片和动态约简两个优化策略来提高谓词信息的收集效率；最后，分析了谓词错误定位过程中存在混杂偏倚的原因，并应用回归分析消除了混杂偏倚影响. 实验表明，基于变量类型的缺陷候选谓词筛选策略可以有效提供错误的相关谓词信息；静态切片和动态约简策略可以提高错误定位的效率；识别并消除错误定位过程的数据依赖混杂偏倚元素和控制依赖混杂偏倚元素可以有效提高错误定位的精度. 借助程序结构识别和消除更多的混杂偏倚元素的影响是我们下一步分析的重点.

致　谢　在此，我们对审稿人提出的有益建议表示感谢，对本文提出建议的同行表示感谢！

参　考　文　献

[1] Weiser M. Programmers use slices when debugging. Communications of the ACM, 1982, 25(7): 446-452

[2] Yu Kai, Lin Meng-Xiang. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011, 34(8): 1411-1422(in Chinese)
(虞凯，林梦香. 自动化软件错误定位技术研究进展. 计算机学报，2011，34(8)：1411-1422)

[3] 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)

[4] 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

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

[6] 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

[7] 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

[8] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation. ACM SIGPLAN Notices, 2005, 40(6): 15-26

[9] Gore R, Reynolds Jr P F. Reducing confounding bias in predicate-level statistical debugging metrics//Proceedings of the 34th International Conference on Software Engineering. Zurich, Switzerland, 2012: 463-473

[10] Liblit B, Aiken A, Zheng A X, et al. Bug isolation via remote program sampling. ACM SIGPLAN Notices, 2003, 38(5): 141-154

[11] Zhang Z, Chan W, Tse T, et al. Non-parametric statistical fault localization. Journal of Systems and Software, 2011, 84(6): 885-905

[12] Hu P, Zhang Z, Chan W K, et al. Fault localization with non-parametric program behavior model//Proceedings of the 8th International Conference on Quality Software. Oxford, UK, 2008: 385-395

[13] Shu G, Sun B, Podgurski A, et al. MFL: Method-level fault localization with causal inference//Proceedings of the 6th International Conference on Software Testing, Verification and Validation. Luxembourg, 2013: 124-133

[14] Holland P W. Statistics and causal inference. Journal of the American Statistical Association, 1986, 81(396): 945-960

[15] Pearl J, Causality: Models, Reasoning and Inference. Cambridge, UK: Cambridge University Press, 2000

[16] Morgan S L, Winship C, Counterfactuals and Causal Inference: Methods and Principles for Social Research. Cambridge, UK: Cambridge University Press, 2007

[17] Xie X, 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 and Methodology, 2013, 22(4): 1-31

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

[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] Baah G K, Podgurski A, Harrold M J. Causal inference for statistical fault localization//Proceedings of the 19th International Symposium on Software Testing and Analysis. Trento, Italy, 2010: 73-84

[21] Zhou Y, Xu B, Leung H, et al. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Transactions on Software Engineering and Methodology, 2014, 23(1): 1-51

[22] Liu C, Yan X, Fei L, et al. SOBER: Statistical model-based bug localization. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 286-295

[23] Chung Man T, Chan W K, Yu Y T. Extending the theoretical fault localization effectiveness hierarchy with empirical results at different code abstraction levels//Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference. Vasteras, Sweden, 2014: 161-170

[24] Liu C, Fei L, Yan X, et al. Statistical debugging: A hypothesis testing-based approach. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848

[25] Zhang X, He H, Gupta N, et al. Experimental evaluation of using dynamic slices for fault location//Proceedings of the 6th International Symposium on Automated Analysis-Driven Debugging. Monterey, USA, 2005: 33-42

[26] Jiang Shu-Juan, Wang Xing-Ya, Zhang Yan-Mei, et al. Fault localization approach for null pointer exception. Journal on Communications, 2015, 36(1): 18-29 (in Chinese)
(姜淑娟, 王兴亚, 张艳梅等. 空指针异常的自动故障定位方法. 通信学报, 2015, 36(1): 18-29)

[27] 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

[28] Gore R, Reynolds Jr P F, Kamensky D. Statistical debugging with elastic predicates//Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 492-495

WANG Xing-Ya, born in 1990, Ph. D. candidate. His research interests include software analysis and testing.

JIANG Shu-Juan, born in 1966, professor, Ph. D. supervisor. Her research interests include compilation techniques, software engineering.

JU Xiao-Lin, born in 1976, Ph. D., associate professor. His research interests include software analysis and testing.

CAO He-Ling, born in 1980, Ph. D., lecturer. Her research interests include software analysis and testing.

Background

Fault localization has been a hot spot issue in the program software quality assurance (SQA) field during the past decade. Especially, statistical fault localization (SFL) has attracted the attention of an increasing number of researchers. Because there is still a lack of knowledge about the association between the fault and the execution result, it is a big challenge to extract "good" locating results that work well for various programs.

Suspiciousness computation is one of the critical steps in the SFL. Many SFL approaches treat the association strength between the program element and the execution result as the suspiciousness. They are effective in some cases. However, a program element which has high association to failed execution does not necessarily mean the root cause of the failure, because those metrics process do not consider the factors or confounding biases that influence both the predicate and the execution result.

To combat the aforementioned limitation, this paper presents a novel approach that accounts for the effects of program dependencies to mitigate the confounding effect during Predicate-based Statistical Fault Localization. First, we present a variable type-based predicate pre-filtering technique to improve the ability of fault-relevant predicate identification. Then we recognize the control and data dependence confounding bias by analyzing the mutual dependence between the predicate and the execution result. To improve the efficiency and accuracy, we also design a combined predicate collection technique with static slicing and dynamic reducing. Experimental studies demonstrate that the fault-relevant predicate can be identified effectively by the proposed predicate pre-filtering technique, and the effectiveness as well as the efficiency of fault localization can be significantly improved after mitigating the dependence confounding effect.

This work was supported by awards from National Natural Science Foundation of China (Nos. 61673384, 61502497), the Guangxi Key Laboratory of Trusted Software (No. kx201530), the State Key Laboratory for Novel Software Technology at Nanjing University (No. KFKT2014B19), the Scientific Research Innovation Project for Graduate Students of Jiangsu Province (No. KYLX_1390), the Henan Province Major Research Program of Colleges and Universities (No. 16A520005) and the Nantong Application Research Plan under Grant (No. BK2014055).