软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1756−1769 [doi: 10.13328/j.cnki.jos.005287]

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

基于条件概率模型的缺陷定位方法*

舒 挺, 黄明献, 丁佐华, 王 磊, 夏劲松

(浙江理工大学 信息学院,浙江 杭州 310018)

通讯作者: 舒挺, E-mail: shuting@zstu.edu.cn

摘 要: 缺陷定位是软件调试的重要阶段,依赖程序频谱信息实现软件缺陷定位,是当前比较行之有效的方法.基于频谱缺陷定位方法应用的前提是,程序频谱和执行结果之间存在的潜在关联.通过经验性分析两者之间的内在关联,借助于统计学的条件概率思想,构建了用以量化分析两者关系强弱的 P 模型,并基于此提出了基于条件概率的缺陷定位方法.以 Siemens 套件中的 7 个程序、Space 程序和 3 个 Unix 工具程序为基准评测对象,与已有的 15 种经典缺陷定位方法进行了对比实验.实证研究结果表明,该方法总体上具有更好的缺陷定位效果.

关键词: 缺陷定位;程序频谱;条件概率;软件调试;测试例

中图法分类号: TP311

中文引用格式: 舒挺,黄明献,丁佐华,王磊,夏劲松.基于条件概率模型的缺陷定位方法.软件学报,2018,29(6):1756−1769. http://www.jos.org.cn/1000-9825/5287.htm

英文引用格式: Shu T, Huang MX, Ding ZH, Wang L, Xia JS. Fault localization method based on conditional probability model. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1756−1769 (in Chinese). http://www.jos.org.cn/1000-9825/5287.htm

Fault Localization Method Based on Conditional Probability Model

SHU Ting, HUANG Ming-Xian, DING Zuo-Hua, WANG Lei, XIA Jin-Song

(School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Fault localization is an essential step of software debugging, and spectrum-based fault localization (SFL) is currently amongst the most effective methods. The fundamental premise underlying SFL is that there exists a potential relationship between program spectra and the corresponding execution results. To formally describe and accurately quantify this relation, this paper introduces the conception of conditional probability to construct a P model by using the statistical analysis of experimental data. In addition, based on the presented P model, a fault localization method is proposed to effectively locate the faulty statement of the program under test. Finally, taking seven programs contained in the Siemens suite, Space program and three real-life Unix utility programs as the benchmark, a detailed experiment is conducted to evaluate the effectiveness and efficiency of the proposed method. Compared with fifteen classic fault localization methods, the experimental results show that the presented approach is more promising.

Key words: fault localization; program spectrum; conditional probability; software debugging; test case

软件规模和复杂程度的与日俱增,给软件开发和调试技术带来了极大的挑战.面对软件开发过程中如影相随的程序缺陷问题,软件测试成为了提升其质量和可靠性的重要技术手段.因此,人们在软件测试方面的投入逐年增加.在软件测试过程中,当发生被测软件的行为与预期不一致(即失效)时,开发人员就需要随即开展软件调试[1].软件调试的首要任务是缺陷定位,它为后续错误代码修复工作提供了基础.

* 基金项目: 国家自然科学基金(61101111, 61572441); 浙江省自然科学基金(LY17F020033)

Foundation item: National Natural Science Foundation of China (61101111, 61572441); Natural Science Foundation of Zhejiang Province (LY17F020033)

收稿时间: 2016-12-13; 修改时间: 2017-01-07, 2017-02-18; 采用时间: 2017-03-20; jos 在线出版时间: 2017-07-12

CNKI 网络优先出版: 2017-07-12 15:33:41, http://kns.cnki.net/kcms/detail/11.2560.TP.20170712.1533.007.html

缺陷定位旨在探测和查找引起软件失效的错误代码,是非常枯燥和耗时的活动.传统的手工设置断点的调试方法,不仅断点位置选择困难,且时间开销巨大.因此,实现缺陷定位的自动化成为了软件学术界和工业界共同追求的目标.近些年来,研究人员从不同的角度尝试提出了一系列辅助自动化缺陷定位的方法,包括基于切片的方法[2−4]、基于程序不变量的方法[5,6]、模型检验方法[7]和基于程序频谱的方法[8−12]等.基于切片的缺陷定位方法主要通过对程序的静态或动态分析,找出与给定变量相关联的语句,从而缩小错误查找范围.基于不变量的方法则通过利用成功和失效测试执行中程序不变量之间的差异信息,辅助定位错误语句.模型检验方法则依赖失效程序行为与期望模型行为的冲突来推导程序错误.相比而言,基于频谱的缺陷定位方法(spectrum- based fault localization,简称 SFL)由于具有不需要考虑程序本身内部结构和执行开销小的特点,成为了一类比较行之有效的重要方法.

程序频谱通常指程序运行时代码覆盖信息的集合,是程序动态行为特征的一种描述形式[13].由于程序运行失效时必定执行到了某条错误语句[14],据此可以得出经验性的推断:当某条语句被更多的失效执行所覆盖,那么它为错误语句的可能性就越高.因此,通过收集程序的频谱和执行结果两方面的信息来推导出程序缺陷位置成为了可能.具体的,SFL 方法主要通过对比分析被测程序在成功执行和失效执行的程序频谱信息,构造相应的可疑度计算公式来估测程序实体(如语句、谓词等)出错的可能性.最终,调试人员将程序实体按照可疑度大小降序逐一排查错误.因此,可疑度值辅助缺陷定位的精确程度成为了衡量 SFL 方法优劣的主要性能指标.

通常,依赖基准测试对象(如 Siemens 套件和 Space 程序等)进行实验对比,是评价现有 SFL 方法的主要途径.作为实验分析方法的补充,Naish 等人[15]首次基于模型和实验结合的评价方法来分类可疑度计算公式,并提出了其实验环境条件下的 2 个最优化公式 Op1 和 Op2.随后,谢晓园[13]和 Tsong[16]等人在特定的假设条件下提出了一套理论框架用以评价可疑度计算公式的优劣.然而,Shin 等人[17]通过理论分析已经证明,不存在绝对最优的可疑度计算公式.针对特定的测试上下文,如错误类型和程序结构等,已有的各种方法各具优势.

程序频谱和执行结果之间存在的潜在关联是可疑度计算公式构造的基础.显然,充分地挖掘和利用这种潜在关联蕴含的缺陷揭示信息,有助于提升可疑度计算公式缺陷定位的效用.本文通过经验性研究程序频谱和执行结果两者之间的内在关联,引入统计学的条件概率思想,构建了用以评估两者关系强弱的量化模型.针对一些经典的 SFL 方法,依赖提出的条件概率 P 模型来分析具体的可疑度计算公式,从而探究其缺陷定位精度和效率差异的本质原因.基于此,提出了一种基于条件概率的缺陷定位新方法.为了验证新方法的有效性,实证研究以 Siemens 套件中的 7 个程序、Space 程序以及 3 个 Unix 工具程序为基准评测对象,与已有的 15 种经典缺陷定位方法进行了对比实验.测试结果表明,新方法在缺陷定位效果上具有一定优势.

1 相关工作

在以往的研究中,人们已经提出了多种基于频谱的软件缺陷定位方法.Jones 等人[9]认为:被更多的失效测试执行覆盖到的语句,是缺陷语句的可能性就越大.基于该前提,他们首次应用程序频谱来构造可疑度计算公式 Tarantula,从而辅助软件缺陷定位.Tarantula 公式中定义了 4 类因子用以描述语句频谱和执行结果之间的关系:a_{11} 表示某条语句在失效测试中覆盖到的次数;a_{10} 表示某条语句在成功测试中覆盖到的次数;a_{00} 表示某条语句在成功测试中未被覆盖到的次数;a_{01} 表示某条语句在失效测试中未被覆盖到的次数.后续提出的频谱方法基本都沿用了这 4 个因子的定义.

Abreu 等人[18]随后受到分子生物学基因相似度公式和聚类分析思想的启发,分别提出了 Ochiai 和 Jaccard[8]方法.实证研究表明,Ochiai 和 Jaccard 的缺陷定位效果要优于 Tarantula.Wong 等人[12]认为:程序语句在测试例中的执行轨迹具有模式可循,可以通过分析失效测试例中语句执行轨迹之间的相似信息来辅助缺陷定位.于是,他们扩展了 Kulczynski[19]频谱公式,提出了 Dstar 方法.Dstar 的缺陷定位效果要优于之前提出的 Tarantula,Ochiai 等一些经典的频谱公式.张震宇等人[20]提出了一种基于语句块链 KBC 和降噪规则的缺陷定位新框架.实验结果表明,新框架可以整合到已有的方法提高其缺陷定位效果.丁晖等人[11]则将信息量的相关知识引入到程序频谱中,提出了一种基于信息量的缺陷定位方法 SIQ.该方法应用事件信息量来降低测试数据集差异带来的缺陷定

位影响,因此在稳定性和定位效果两方面更具优势.Dallmeier 等人则认为,成功测试和失效测试中类方法调用序列的差异信息可以帮助缺陷定位.基于此,针对面向对象程序提出了 Ample 方法[21].Le 等人[22]尝试 Tarantula 和信息检索技术的整合,提出了一种复合的频谱缺陷定位方法.在包含 157 个真实错误的 4 个软件系统上的实验数据表明:与经典的 SFL 方法相比,他们的方法更具优势.

近些年,一些学者在经典 SFL 方法中引入了程序切片(program slicing)技术来进一步提升缺陷定位效果.Wong 等人[23]基于执行切片和代码块间数据依赖信息,提出一种缺陷定位方法.文万志等人[24]利用程序失效执行的动态切片及其所包含语句的频谱信息,通过改造 Tarantula 可疑度公式提出了 PSS-SFL 方法.毛晓光等人[4]则提出了一种近似动态后向切片技术来权衡切片规模和精度的冲突,并基于近似切片来构造新的可疑度计算公式.该类方法巧妙地利用切片技术实现了错误排查范围的缩减,从而提高了缺陷定位效果.然而,程序切片带来的额外开销和错误遗漏问题也不容忽视.

综上所述,已有方法有的是引用其他相关领域计算公式,并进行大量实验研究来设计缺陷定位的可疑度计算公式;有的则从被测程序本身入手,通过挖掘程序自身蕴含的一些规律信息,例如事件包含的信息量、执行轨迹模式等来构造出新的频谱方法.这些缺陷定位方法效果差异的部分原因可以归结为:如何利用 $a_{11},a_{10},a_{00},a_{01}$ 这 4 个权重因子蕴含的信息来计算语句的可疑度.根据程序频谱与执行结果之间潜在关联的假设,对这 4 个因子的权重考虑得越准确,那么对应方法的缺陷定位效果也就越好.为了探究和量化描述这种内在关联,本文借助于统计学的条件概率思想,构建了衡量两者关系强弱的 P 模型.基于此,提出了一种基于条件概率的缺陷定位新方法.

2 基于条件概率的缺陷定位方法

为了量化程序频谱与执行结果之间的潜在关系,本文统计分析了实验程序的测试数据.围绕 $a_{11},a_{10},a_{00},a_{01}$ 这 4 个因子,提出了频谱因子的条件概率量化模型.从量化模型的视角,提出了一种基于条件概率模型的缺陷定位新方法.

2.1 频谱参数的条件概率模型

通常,在具体测试例的执行过程中,语句覆盖情况包括执行与不执行两种可能,测试执行结果则分为成功与失效.显然,程序频谱与执行结果之间并非相互独立,而是存在某种潜在的关联.关联关系的明晰和量化有助于设计高效的缺陷定位方法.针对具体的被测程序,直觉上可以感知程序频谱和执行结果之间应该具有某种依赖关系.因此,受到统计学中条件概率思想的启发,本文构建了一种基于条件概率的量化关系模型,取名为 P 模型.为了方便 P 模型的描述,给出如下符号表达:某条语句 s_i 被执行的事件表示为 E,未执行表示为 N,其程序执行失效表示为 F,执行成功表示为 T.根据语句频谱和执行结果的关系组合,提出了如下 5 个具体 P 模型.

定义 1(P_{fe} 模型). 在某条语句执行的条件下,程序运行结果为失效的概率:

$$P_{fe}=P(F\mid E)=\frac{P(EF)}{P(E)}=\frac{a_{11}}{a_{11}+a_{10}},a_{11}+a_{10}\neq 0.$$

当频谱满足 $a_{11}+a_{10}=0$ 时,表示在所有测试执行中,某条语句均未被覆盖到.显然,基于频谱的缺陷定位方法不适用于该情形.

定义 2(P_{te} 模型). 在某条语句执行的条件下,程序运行结果为成功的概率:

$$P_{te}=P(T\mid E)=\frac{P(ET)}{P(E)}=\frac{a_{10}}{a_{11}+a_{10}},a_{11}+a_{10}\neq 0.$$

显然,根据定义 1 和定义 2,我们可以得到 $P_{fe}+P_{te}=1$.

定义 3(P_{ef} 模型). 程序在运行失效的条件下,某条语句被执行的概率:

$$P_{ef}=P(E\mid F)=\frac{P(EF)}{P(F)}=\frac{a_{11}}{a_{01}+a_{11}},a_{01}+a_{11}\neq 0.$$

要借助频谱信息定位缺陷,测试执行必须包含至少一个失效结果.因此,约束条件 $a_{01}+a_{11}\neq 0$ 满足.

定义 4(P_{et} 模型). 程序在运行成功的条件下,某条语句被执行的概率:

$$P_{et}=P(E\mid T)=\frac{P(ET)}{P(T)}=\frac{a_{10}}{a_{10}+a_{00}},a_{10}+a_{00}\neq 0.$$

基于频谱的缺陷定位方法应用中,测试执行必须包含成功和失效两部分.因此,满足 $a_{10}+a_{00}\neq 0$ 的约束条件.

定义 5(P_{tn} 模型). 在某条语句未执行的条件下,程序运行结果为成功的概率:

$$P_{tn}=P(T\mid N)=\frac{P(TN)}{P(N)}=\frac{a_{00}}{a_{00}+a_{01}},a_{00}+a_{01}\neq 0.$$

当 $a_{00}+a_{01}=0$,表达的含义为某条语句在所有的成功执行和失效执行中都被覆盖到.在此情况下,该语句的频谱信息无助于缺陷定位.因此,本文中讨论的语句频谱都满足条件 $a_{00}+a_{01}\neq 0$.

上述给出的 5 个模型,我们将其统称为频谱参数的条件概率 P 模型.P 模型从 5 个不同的视角描述了程序频谱与执行结果的关系.频谱方法的经验性假设指出:在失败用例中被执行次数越多的语句,越有可能是错误语句[9].因此,根据 P 模型的定义,可以直觉猜想模型具有如下特性.

(1) 错误语句的 P_{fe},P_{ef} 以及 P_{tn} 值不小于其他正确语句的相应值;

(2) 错误语句的 P_{et} 和 P_{te} 值不大于其他正确语句的相应权值.

为了有个直观的感觉,这里举例说明 P 模型的具体计算和关系量化能力.例子程序见表 1,主要功能为根据 *op* 指令实现两个数的具体操作,并返回执行结果.其中,代码行 s_5 为错误语句,正确版本应为 *result=a+b*.测试用例 T1~T6 具体为:T1=(2,1,sum),T2=(3,4,sum),T3=(5,0,sum),T4=(3,2,average),T5=(1,1,aver- age),T6=(8,3,average).表 1 中,用黑点表示该语句在具体测试例中被执行,否则显示为空白.

Table 1 Spectrum and execution result of example program 1

表 1 示例程序 1 的频谱和执行结果

语句	程序	T1	T2	T3	T4	T5	T6
s_1	*get*(*a*);	•	•	•	•	•	•
s_2	*get*(*b*);	•	•	•	•	•	•
s_3	*get*(*op*);	•	•	•	•	•	•
s_4	if (*op*==“sum”)	•	•	•	•	•	•
s_5	*result=a−b*; //correct *a+b*	•	•	•			
s_6	else if (*op*==“average”)				•	•	•
s_7	*result*=(*a*+*b*)/2;				•	•	•
s_8	print (*result*);	•	•	•	•	•	•
执行结果		F	F	T	T	T	T

根据表 1 的频谱数据,利用 P 模型可以得到一组对应的关系量化值,见表 2.表 2 中的星号表示对应的语句模型值为无意义.分析计算结果可知:错误语句 s_5 的 P_{fe},P_{ef} 以及 P_{tn} 这 3 个模型值大于等于其他正确语句对应的值,而 P_{te} 和 P_{et} 的值则小于其他正确语句对应的模型值.示例数据结果符合 P 模型的两个特性猜想.

Table 2 *P* values of example program 1

表 2 示例程序 1 的 *P* 值计算结果

语句	a_{11}	a_{10}	a_{01}	a_{00}	P_{fe}	P_{ef}	P_{te}	P_{et}	P_{tn}
s_1	2	4	0	0	1/3	1	2/3	1	*
s_2	2	4	0	0	1/3	1	2/3	1	*
s_3	2	4	0	0	1/3	1	2/3	1	*
s_4	2	4	0	0	1/3	1	2/3	1	*
s_5	2	1	0	3	2/3	1	1/3	1/4	1
s_6	0	3	2	1	0	0	1	3/4	1/3
s_7	0	3	2	1	0	0	1	3/4	1/3
s_8	2	4	0	0	1/3	1	2/3	1	*

从语句频谱与执行结果关系的层面,P 模型给出了 5 个不同角度的量化权值.合理利用 P 模型的特性,有助于进一步提升可疑度计算公式的缺陷定位效果.为了经验性验证 P 模型的特性,本文分别针对 Siemens 套件和 Space 程序共 130 个单错误版本,统计分析了 universe 测试集下所有可执行语句的模型值.由于 $P_{fe}+P_{te}=1$,因此我

们只选择 P_{fe} 作为观察对象.围绕着 4 个 P 模型对象,分别统计不同错误版本中,错误语句对应模型值的相对大小排名.图 1 和图 2 分别为 Siemens 套件 102 个错误版本和 Space 程序 28 个错误版本的统计分析结果.图中横坐标表示不同错误版本的编号,纵坐标则为对应版本中错误语句 P 模型值的大小排名比.

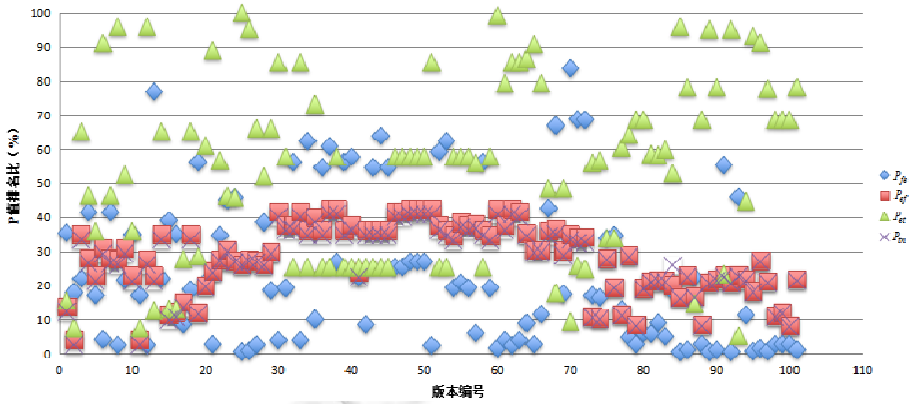


Fig.1 *P*-Value ranks of fault statements in Siemens

图 1 Siemens 套件的错误语句 *P* 值大小排名

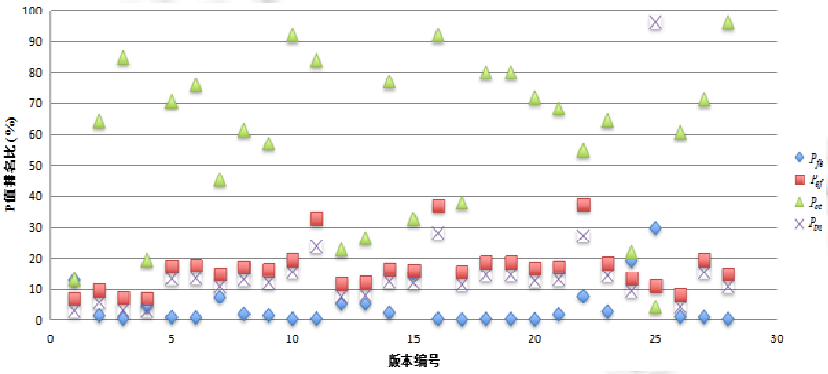


Fig.2 *P*-Value ranks of fault statements in Space

图 2 Space 程序的错误语句 *P* 值大小排名

从图 1 可以看出,除了少数几个错误版本的 P_{fe} 值存在干扰外,大多数错误版本中,错误语句的 P_{fe},P_{ef} 和 P_{tn} 这 3 个模型值大小在所有可执行语句中排名都比较靠前,基本集中在前 40%以内.此外,错误语句的 P_{et} 值,在大多数错误版本的可执行语句中排名处于后 60%.因此,Siemens 套件的统计结果符合 P 模型的特性假设.图 2 为规模更大的 Space 程序的 P 模型值统计结果.图中数据清晰地描述了模型值的分布情况,绝大多数错误版本中错误语句的 P_{fe},P_{ef} 以及 P_{tn} 大小排名靠前,而 P_{et} 排名则比较靠后.与图 1 的统计数据相似,模型 P_{et} 值的分布规律具有不稳定的特征.

由图 1 和图 2 的统计数据可知:缺陷语句的 P_{fe},P_{ef} 和 P_{tn} 值大小在所有可执行语句中排名靠前,排名的平均值分别为前 19.8%,25.5%和 24.4%.相反,缺陷语句的 P_{et} 值大小在所有可执行语句中的排名通常处于后 30%~40%的区间,并存在分布不稳定的情况,平均排名为 55%.统计数据表明:P_{fe} 和 P_{et} 的分布规律存在不稳定的情况, P_{et} 尤为明显.究其原因,巧合性成功[25]问题可能是其中之一,即:一些测试用例虽然执行了错误语句,但是由于没有触发错误发生的条件或者没有影响到程序的输出结果,测试结果仍然为成功,从而影响了 P_{fe} 和 P_{et} 的值.上述实验统计结果进一步验证了提出的 P 模型的特性.基于提出的 P 模型,有助于量化分析程序频谱和执行结果的关联,辅助设计新的可疑度计算公式.

2.2 利用P模型评价经典公式

P 模型从相对更高的层次,抽象和量化了程序频谱和执行结果的关联.借助 P 模型的两大特性,可以分析和评价现有的一些经典频谱方法,从崭新的角度分析它们的优劣.因此,我们挑选了 4 个经典的可疑度计算公式,它们共同的特点是:经过适当的公式变换,可以等价描述为 P 模型的组合.具体如下:

- Ochiai

$$susp(s)=\frac{a_{11}}{\sqrt{(a_{11}+a_{01})\times(a_{11}+a_{10})}}=\sqrt{\frac{a_{11}}{a_{11}+a_{10}}}\times\sqrt{\frac{a_{11}}{a_{11}+a_{01}}}=\sqrt{P_{fe}\times P_{ef}}\Leftrightarrow P_{fe}\times P_{ef} \tag{1}$$

- Kulczynskil2

$$susp(s)=\frac{1}{2}\times\left(\frac{a_{11}}{a_{11}+a_{10}}+\frac{a_{11}}{a_{11}+a_{01}}\right)=\frac{1}{2}\times(P_{fe}+P_{ef})\Leftrightarrow P_{fe}+P_{ef} \tag{2}$$

- Tarantula

$$susp(s)=\frac{\frac{a_{11}}{a_{11}+a_{01}}}{\frac{a_{11}}{a_{11}+a_{01}}+\frac{a_{10}}{a_{10}+a_{00}}}=\frac{P_{ef}}{P_{ef}+P_{et}} \tag{3}$$

- Ample

$$susp(s)=\left|\frac{a_{11}}{a_{11}+a_{01}}-\frac{a_{10}}{a_{10}+a_{00}}\right|=|P_{ef}-P_{et}| \tag{4}$$

不难发现:将 Ochiai 公式进行平方运算,结果就等于 P_{fe} 与 P_{ef} 的乘积.而将 Kulczynskil2×2,则等价于 P_{fe} 与 P_{ef} 的和.因为在计算语句可疑度时,上述等价变换不会改变对应语句可疑度值的排序.因此,公式(1)和公式(2)通过等价变换,分别为 $P_{fe}\times P_{ef}$ 和 $P_{fe}+P_{ef}$.根据 P 模型的特性(1)可知:由于错误语句的 P_{fe} 与 P_{ef} 值在所有的可执行语句中排名靠前,因此不管其相加还是相乘,最终计算结果的排名也必然靠前.所以,Ochiai 和 Kulczynskil2 的缺陷定位效果都较为理想.

同理,Tarantula 方法实际上等价于 $\frac{P_{ef}}{P_{ef}+P_{et}}$,而 Ample[21]方法则等价于$|P_{ef}-P_{et}|$.这两个公式的共同点是都引入了错误语句的 P_{et} 模型值.根据 P 模型特性(2),P_{et} 值排名规律相对于其他 P 模型值具有不稳定性.因此,基于公式(3)和公式(4)计算所得的可疑度,在缺陷定位效果上也表现出不稳定性[15].这可以用于解释公式(3)和公式(4)缺陷定位效果普遍不如公式(1)和公式(2)的原因.

2.3 新的可疑度计算公式

基于提出的频谱条件概率 P 模型,合理利用其特性,可以构造出具有更好缺陷定位效果的可疑度计算公式.通过前期大量的观察和实验,从频谱条件概率 P 模型的视角,我们提出了以下两个新公式.

- Cp1

$$f_1(s)=a_{11}\times P_{fe}+a_{11}\times P_{ef}=a_{11}\times\left(\frac{a_{11}}{a_{11}+a_{10}}+\frac{a_{11}}{a_{11}+a_{01}}\right) \tag{5}$$

- Cp2

$$f_2(s)=\frac{P_{fe}+P_{ef}}{1+P_{et}}=\frac{\frac{a_{11}}{a_{11}+a_{10}}+\frac{a_{11}}{a_{11}+a_{01}}}{1+\frac{a_{10}}{a_{10}+a_{00}}} \tag{6}$$

根据 P 模型的特性(1)可知,错误语句的 P_{fe} 和 P_{ef} 值排名通常靠前.因此,公式(5)选择将其作为语句可疑度衡量的主要因子.然而在特定的情况下,正确语句的 P_{fe} 值可能大于错误语句的 P_{fe} 值,从而降低了错误语句的可疑度排名及其缺陷定位效果.例如,表 3 为 Simenes 套件中 tot_info 程序 V19 版本的错误代码片段.s_1 为错误语句,

原因为 MAXLINE 值定义错误.在这个例子中,位于循环体条件分支中的语句(如 s_7 和 s_8),由于其 a_{10} 和 a_{11} 有不同程度的减少,最终导致产生错误语句的 P_{fe} 值大于正确语句的现象.

Table 3 Example program 2

表 3 示例程序 2

	Tot_info 程序代码片段
s_1	while (*fgets*(*line*,*MAXLINE*,*stdin*)!=*NULL*) {//error
s_2	for (*p*=*line*; **p*!='\0' && *isspace*((*int*)**p*); ++*p*)
s_3	if (**p*=='\0')
s_4	continue; /*skip blank line*/
s_5	if (**p*==*COMMENT*)
s_6	{ /*copy comment through*/
s_7	(*void*)*fputs*(*line*,*stdout*);
s_8	continue;
s_9	}
s_{10}	}

表 4 描述了示例程序 2 在 1 052 个测试用例下的执行信息和部分语句的 P 模型值,这里只列出具有代表性的错误语句 s_1 以及正确语句 s_7 和 s_8.根据统计数据可知:如果采用 P_{fe} 与 P_{ef} 相加的方式(即类似 Kulczynskil2 方法)计算语句可疑度,结果正确语句 s_7 和 s_8 的可疑度超过了错误语句 s_1,从而降低了错误定位效果.为了避免该现象,CP1 公式的构造引入了 Wong 的频谱公式构造假设(即,可疑度公式应赋予 a_{11} 更高的权重[12]),在 P_{ef} 与 P_{fe} 和的基础上乘以 a_{11},从而修正可疑度值的有效性.Cp2 公式的构造则基于 P 模型的特性 2.由于 P_{et} 值的统计排名偏后,并具有不稳定.因此,为了改善 Tarantula 与 Ample 方法的效果,Cp2 将 P_{et} 单独放在分母的位置.这样,可疑度计算结果不仅体现了 P_{et} 的特征,同时尽可能地降低了其不稳定性对缺陷定位效果所带来的负面影响.

Table 4 P-Values of partial statements in example program 2

表 4 示例程序 2 部分语句的 P 模型值

语句	a_{11}	a_{10}	a_{01}	a_{00}	P_{fe}	P_{ef}	$P_{fe}+P_{ef}$	$a_{11}\times P_{fe}+a_{11}\times P_{ef}$
s_1	47	1005	0	0	0.0447	1	1.0447	49.100 9
s_7	46	633	1	372	0.0677	0.9787	1.0464	48.134
s_8	46	633	1	372	0.0677	0.9787	1.0464	48.134

3 实验及结果分析

为了验证本文方法的有效性,采用缺陷定位研究中广泛使用的经典基准评测 C 语言程序:Simenes 套件、Space 程序以及 3 个 Unix 工具程序(sed,flex 和 grep)为测试对象[4,7,9−12,15,24],并选择了 15 个主流的可疑度计算公式[13,15,26]进行了对比实验.实验运行环境为 Ubuntu14.04 操作系统、Intel Core i7-4500U CPU 和 8G 内存,其中,Gcc 和 Gcov 工具的版本都为 4.8.4.

3.1 实验对象和配置

Simenes 套件最初由西门子公司开发,已经成为软件测试领域实证研究中使用最为普遍的评测程序之一.它共包括 7 个小规模程序,每个程序都提供了一个正确版本以及若干个错误版本.其中,每个错误版本都包含了单个手工植入的错误.相对 Simenes 套件而言,Space 程序和 3 个 Unix 工具程序则分别属于中等规模和大规模程序[27],而且它们除了部分植入错误以外,还包含了软件实际开发中产生的真实错误.因此,基于它们的实证研究结果更具代表性.Space 程序是欧洲航天局开发的针对 ADL 语言的解释器,具有 38 个包含真实错误的错误版本和 13 525 个测试用例.sed、flex 和 grep 都是 Unix 系统中的实用工具程序,这为评测结果的有效性提供了信任基础.具体的,它们各自包含了一系列演化中的程序版本,每个演化版本又对应包含了几十个单错误版本.本文所有评测对象均可以从 SIR[28]库获取,具体程序的概要信息见表 5(程序名括号内为名称缩写).

Table 5 Programs under test

表 5 实验程序

程序	版本数	代码行数	测试例数	描述
print_tokens(pt)	7	563	4 130	Lexical analyzer
print_tokens2(pt2)	9	406	4 115	Lexical analyzer
replace(re)	30	563	5 542	Pattern replacement
tcas(tc)	40	173	1 608	Altitude separation
schedule(sc)	5	410	2 650	Priority scheduler
schedule2(sc2)	9	307	2 710	Priority scheduler
tot_info(ti)	23	406	1 052	Information measure
space	28	9 127	13 525	ADL interpreter
sed (1.18-4.1.5)	22	6 671~11 990	359	Stream editor
flex(2.5.1-2.5.4)	53	12 421~14 244	524	Lexical analyzer
grep(2.2-2.4.2)	16	12 658~13 372	469	Search for patterns in files

在SIR库中,每个被测程序都分别提供了一些具有代表性的测试集.本次实验选用了最普遍的universe测试用例集.在错误版本的选择上,由于SFL方法需要依赖失效测试执行的信息,因此,一些不存在失效测试用例的版本被排除,例如schedule2程序的v9和Space程序的v1,v2,v32和v34等.此外,一些导致程序执行异常终止的版本,例如print_tokens2的v10,tcas的v38,schedule的v1,v5,v6和v9,replace的v27,v32以及Space的v26,v30,v35,v36和v38等也被剔除.在这些因为缺陷而导致程序异常终止的版本中,有一些虽然只有部分用例导致程序运行异常终止,但是由于实验采取的是universe测试用例,为了确保实验的严谨性和一致性,故也将这些版本排除.最后,实验中共采用了Simenes套件的123个版本、Space程序的28个版本以及3个Unix工具程序的91个版本,合计共242个错误版本.

实验频谱统计以程序语句为基本单位.在实验程序中,一些版本的错误为预定义数值错误、代码缺失或者变量类型定义错误,这三者无法直接标识出错误所在的具体行号.因此,在错误行标识过程中,我们对于预定义数值错误以及变量类型定义错误这两种类型,将错误代码标识到具体变量的使用行.而对于代码缺失所引起的错误,则将错误代码行数随机标识到该缺失代码所在代码块的某一行.

3.2 实验设计

为了验证所提出的Cp1与Cp2这两个可疑度计算公式的缺陷定位效果,实验选择了当前主流的15个频谱可疑度计算公式作为评测基准,具体信息见表6.实验选择的15个公式涵盖了文献[13,26]中理论分析得出的两组最优公式ER1和ER5以及基于遗传算法生成的最优公式GP02,GP03和GP19,具有一定的代表性.关于两个最优公式等价组ER1和ER5,我们分别挑选了Op2(Naish2)和Wong1公式作为实验代表.在所有实验的频谱可疑度计算公式中,只有Dstar方法需要设定具体的参数,本次实验设置star参数为3[12].

实验从EXAM指标[29]和缺陷定位代价两个角度进行对比分析.EXAM指标刻画了在可疑语句集中定位错误所需的代码检查代价,即,找到错误所需排查的代码量占程序总代码量的百分比.根据EXAM指标可知:在相同的代码排查百分比下,找到的缺陷版本数占总版本数的百分比越高,则该方法缺陷定位效果越好.缺陷定位代价则是定位被测程序所有错误版本中错误语句需要检查的语句总数占总代码的百分比.缺陷定位代价越小,说明该缺陷定位方法的定位效率越高.

关于实验中不同方法定位缺陷效果差异的显著性,应用非参数检验中的曼-惠特尼(Mann-Whitney) U检验方法,针对EXAM指标进行假设检验验证.我们设定零假设H0为:本文提出的方法与实验对比方法之间在缺陷定位效果方面没有显著差异;对立假设H1为:在缺陷定位效果方面,本文提出的方法与实验对比方法之间存在显著差异.实验中,显著性水平α设定为0.05.若检验结果p值小于α,则拒绝零假设H0,即可以认为本文提出的方法在缺陷定位效果方面要显著优于对比方法;反之,则差异不显著.

Table 6 Formulas of 15 spectrum-based fault localization methods

表 6 主流的 15 个可疑度计算公式

频谱方法	公式
Tarantula	$susp(s)=\frac{\frac{a_{11}}{a_{11}+a_{01}}}{\frac{a_{11}}{a_{11}+a_{01}}+\frac{a_{10}}{a_{10}+a_{00}}}$
Jaccard	$susp(s)=\frac{a_{11}}{a_{11}+a_{01}+a_{10}}$
Wong1	$susp(s)=a_{11}$
Wong2	$susp(s)=a_{11}-a_{10}$
Wong3	$susp(s)=\begin{cases}a_{11}-a_{10}, & \text{if } a_{10}\leqslant 2\\ 2+0.1(a_{10}-2), & \text{if } 2<a_{10}<10\\ 2.8+0.001(a_{10}-10), & \text{if } a_{10}>10\end{cases}$
Ochiai	$susp(s)=\frac{a_{11}}{\sqrt{(a_{11}+a_{01})\times(a_{11}+a_{10})}}$
Ochiai2	$susp(s)=\frac{a_{11}a_{10}}{\sqrt{(a_{11}+a_{10})\times(a_{00}+a_{10})\times(a_{11}+a_{01})\times(a_{01}+a_{00})}}$
Dstar	$susp(s)=\frac{a_{11}^{star}}{a_{01}+a_{10}}$
Kulczynski1	$susp(s)=\frac{a_{11}}{a_{01}+a_{10}}$
Kulczynski2	$susp(s)=\frac{1}{2}\times\left(\frac{a_{11}}{a_{11}+a_{10}}+\frac{a_{11}}{a_{11}+a_{01}}\right)$
M2	$susp(s)=\frac{a_{11}}{a_{11}+a_{00}+2(a_{01}+a_{10})}$
Op2	$susp(s)=a_{11}-\frac{a_{10}}{a_{10}+a_{00}+1}$
GP02	$susp(s)=2(a_{11}+\sqrt{a_{00}})+\sqrt{a_{10}}$
GP03	$susp(s)=\sqrt{\lvert a_{11}^2-\sqrt{a_{10}}\rvert}$
GP19	$susp(s)=a_{11}\sqrt{\lvert a_{10}-a_{11}+a_{01}-a_{00}\rvert}$

注:P 和 F 分别表示成功和失效测试例的数量;$star$ 为自定义参数,取值范围为任意正整数

3.3 实验结果

基于 EXAM 指标的缺陷定位效果比较结果如图 3 所示,横坐标表示检查代码的百分比(代码排查比),纵坐标表示所能发现的缺陷版本数占总版本数的百分比(错误检出率).从图 3 统计数据可知:初期小于 5%的代码排查比下,Cp1 的错误检出率最高,但与 Op2 和 M2 比较接近;之后,随着代码排查比的不断增加,Cp1 的错误检出率逐渐超过其他方法.当代码排查比处于 1%~19%之间,Cp1 的缺陷定位效果处于领先地位;当代码排查比超过 19%时,Kulczynski2,Op2 和 M2 的错误检出率开始逐渐接近 Cp1,它们都稍稍领先于其他方法;当代码排查比超过 31%时,Cp1,Op2,M2 和 Kulczynsik2 的错误检出率均比较接近,都领先于其他各类方法;当代码排查比处于 42%~60%区间时,则 Cp2 与 Wong1 的优势较为明显.最后,随着代码排查比的不断增加,各种缺陷定位方法的错误检出率逐渐持平.

综合来看,Cp1 方法在代码排查比小于 42%时,与实验的其他方法相比,在错误检出率方面具有优势.因为它充分利用了 P_{fe} 与 P_{ef} 的属性特征,准确量化了程序频谱与执行结果之间的潜在关联,实验结果符合预期.相对而言,当代码排查比较大时(即大于 42%),Cp2,Wong1 和 GP19 方法的缺陷定位效果最为显著.同时,在代码排查比小于 20%时,Cp2 的缺陷定位效果要明显优于 Wong1 和 GP19.

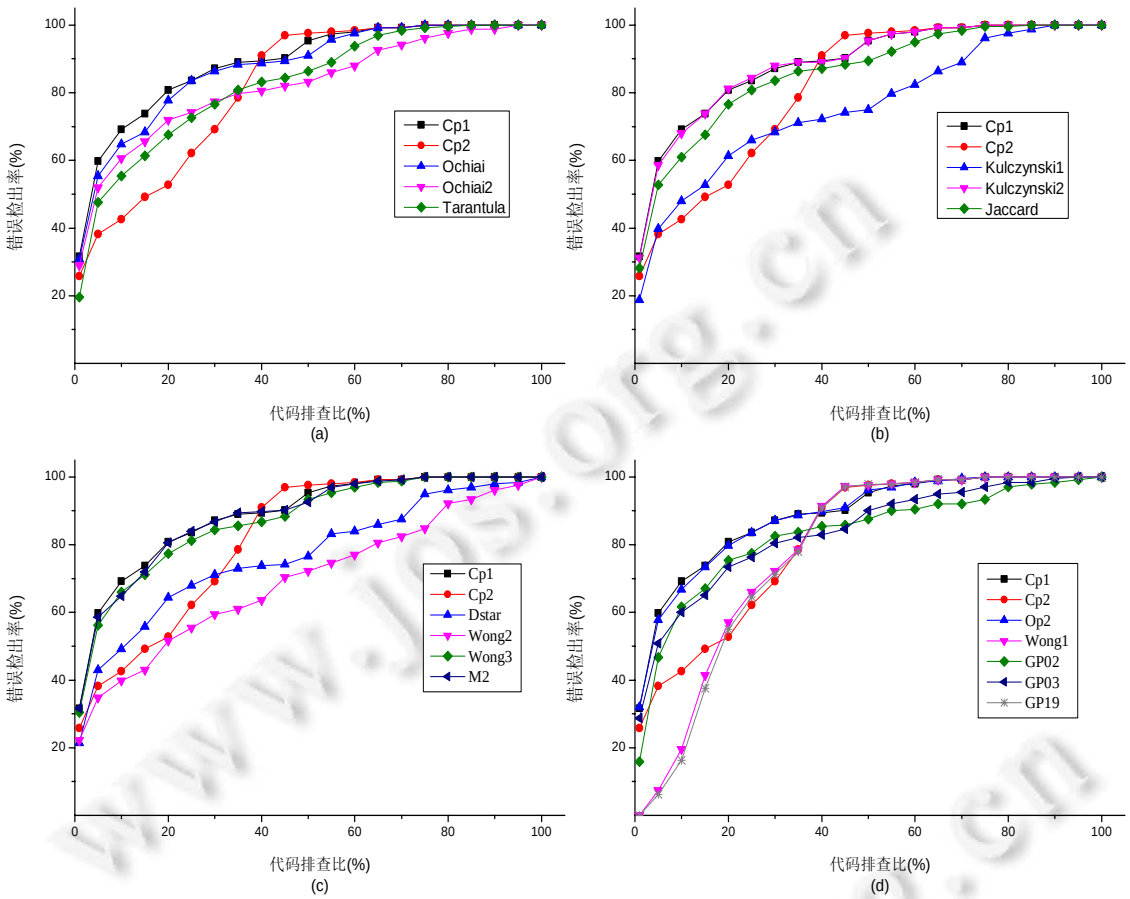


Fig.3 Fault-Localization effectiveness comparison based on EXAM

图 3 基于 EXAM 的缺陷定位效果比较

EXAM 指标假设检验的实验结果见表 7,表中的统计数据表明:在大多数情况下,本文提出的方法与实验方法相比,在缺陷定位效果方面具有显著性的差异.总体上,Cp1 比 Cp2 的显著性效果更为明显.但针对其中的部分程序(print_tokens,schedule 和 schedule2),Cp1 和 Cp2 的表现并不理想.具体的,Cp1 除了与 Wong3 以及最优公式 Op2 相比较整体显著性差异不明显以外,较之其他方法均有比较显著的缺陷定位效果改善.值得注意的是:相对于部分具体的验证场景(如 Op2 方法应用于 space 和 flex 程序以及 Wong3 应用于 tcas,replace,flex 和 grep 程序),对立假设 H1 仍然成立,即,Cp1 方法具有更加显著的缺陷定位效果.此外,Cp2 相对于 Ochiai, Kulczynsik2,Wong1,Wong3,M2,Op2,GP02 和 GP03 等方法,其效果改善并不显著,但相比于其余实验公式则表现出了显著性的差异.

表 8 为各种方法缺陷定位代价的具体统计数据.缺陷定位代价是找到被测程序所有错误版本的错误需要排查的语句总数与所有版本代码总行数的比值,这个比值越小,说明对应方法的缺陷定位效率越高.表中 all 所对应竖列的数据表示定位所有实验程序全部错误需要排查的语句总数与所有代码总行数的比值.统计数据表明,Cp1 公式在总体缺陷定位代价方面表现最佳.与实验的各种方法相比,总体缺陷定位代价平均降幅达 30.52%.其中,Cp1 比 wong2 方法降低最多为 63.12%,经典方法 Tarantula,Jaccard 和 Ochiai 分别降低了 38.31%、21.12%和 11.84%.Cp1 公式的平均代价与最优公式 Op2 以及 Kulczynsik2 非常接近,分别降低了 1.58%和 0.88%.然而通过分析可以发现:针对 Space 这样中型规模的基准测试程序,CP1 的缺陷定位代价比 Op2 显著降低了 42.57%;同时,Cp2 的平均代价方面也有较好的表现,要普遍优于实验的对比方法.CP2 仅次于 Ochiai,

Kulczynsik2,Wong3,M2 和 Op2 方法,平均增加幅度为 12.7%.但是在 Schedule、Space 和 sed 等程序的所有错误版本缺陷定位中,Cp2 的平均代价则要比 Cp1 低 40.26%.总体而言,实验数据表明基于条件概率模型的 Cp1 和 Cp2 方法可以有效提高缺陷定位的效率.

Table 7 P-Values of fault-localization effectiveness of different methods

表 7 各种方法缺陷定位效果的检验 P 值

		ti	pt	pt2	tc	sc	sc2	re	space	sed	flex	grep
Ochiai	Cp1	<0.001	0.2	0.016	<0.001	0.8	0.024	<0.001	<0.001	<0.001	0.001	0.025
	Cp2	0.001	0.2	0.056	<0.001	0.8	0.016	<0.001	0.107	0.444	0.003	0.364
Ochai2	Cp1	<0.001	0.2	0.024	<0.001	0.4	0.8	<0.001	<0.001	<0.001	0.001	0.003
	Cp2	<0.001	0.2	0.024	<0.001	0.4	0.056	<0.001	0.02	<0.001	0.001	0.009
Kulczynsik1	Cp1	0.001	0.2	0.024	<0.001	0.4	0.001	<0.001	<0.001	<0.001	0.002	0.021
	Cp2	0.001	0.2	0.024	<0.001	0.4	0.222	<0.001	0.009	<0.001	<0.001	0.042
Kulczynsik2	Cp1	<0.001	0.6	0.024	0.003	0.8	0.222	<0.001	<0.001	<0.001	0.003	0.067
	Cp2	0.898	0.6	0.8	0.75	0.4	0.222	0.333	0.075	0.528	0.091	0.625
Tarantula	Cp1	0.001	0.2	0.024	<0.001	0.4	0.9	<0.001	<0.001	<0.001	0.118	0.039
	Cp2	0.001	0.2	0.024	<0.001	0.4	0.222	<0.001	0.007	<0.001	0.066	0.001
Jaccard	Cp1	<0.001	0.2	0.024	<0.001	0.4	0.667	<0.001	<0.001	<0.001	0.001	0.042
	Cp2	<0.001	0.2	0.024	<0.001	0.4	0.056	<0.001	0.033	0.209	<0.001	0.364
Wong1	Cp1	<0.001	<0.001	<0.001	<0.001	0.008	0.016	0.001	0.011	<0.001	<0.001	0.001
	Cp2	0.159	<0.001	0.01	<0.001	0.159	0.286	<0.001	0.143	<0.001	0.091	0.071
Wong2	Cp1	<0.001	0.4	0.024	0.03	0.05	<0.001	0.005	<0.001	<0.001	<0.001	0.005
	Cp2	<0.001	<0.001	0.4	0.04	0.06	<0.001	0.005	<0.001	<0.001	<0.001	0.02
Wong3	Cp1	0.261	0.9	0.222	<0.001	0.9	0.667	0.001	0.071	<0.001	0.182	0.001
	Cp2	0.348	0.9	0.4	0.002	0.4	0.9	0.005	0.041	0.075	0.091	0.02
M2	Cp1	0.001	0.4	0.016	<0.001	0.8	0.286	<0.001	<0.001	<0.001	0.009	0.15
	Cp2	0.9	0.8	0.8	0.8	0.9	0.9	0.8	0.051	0.9	0.091	0.9
Dstar	Cp1	0.006	0.2	0.056	<0.001	0.8	0.286	<0.001	<0.001	<0.001	0.001	0.038
	Cp2	0.035	0.4	0.056	0.003	0.8	0.9	0.018	0.005	<0.001	<0.001	0.011
Op2	Cp1	0.087	0.9	0.444	0.9	0.9	0.667	0.333	0.005	0.006	0.182	0.125
	Cp2	0.087	0.9	0.8	0.9	0.4	0.9	0.8	0.009	0.038	0.091	0.375
GP02	Cp1	<0.001	0.4	0.056	0.001	0.6	0.024	0.003	<0.001	0.007	<0.001	0.002
	Cp2	0.832	1	0.222	0.01	0.9	0.024	0.007	0.041	0.003	0.001	0.442
GP03	Cp1	0.001	0.4	0.444	<0.001	0.9	0.111	0.028	<0.001	0.003	<0.001	<0.001
	Cp2	0.919	0.9	0.683	<0.001	0.4	0.024	0.003	<0.001	0.001	0.001	0.055
GP19	Cp1	<0.001	<0.001	<0.001	<0.001	0.4	0.016	0.001	<0.001	0.002	<0.001	0.001
	Cp2	0.159	0.003	0.222	<0.001	0.009	0.063	<0.001	0.009	0.009	<0.001	0.071

Table 8 Performance comparison of different methods under subject programs (%)

表 8 各种方法缺陷定位代价比较(%)

程序	ti	pt	pt2	tc	sc	sc2	re	space	sed	flex	grep	all
Cp1	9.83	3.65	3.54	27.11	16.32	28.89	6.56	2.9	2.86	9.03	13.35	11.24
Cp2	16.81	4.57	10.66	28.61	5.81	30.84	7.05	2.38	1.76	14.01	17.60	13.48
Ochiai	18.24	8.83	12.09	29.34	4.39	33.71	8.04	1.97	1.88	9.41	13.39	12.75
Ochai2	43.04	16.07	20.55	31.11	5.85	42.66	9.49	5.07	1.98	14.54	26.07	18.35
Kulczynsik1	26.56	17.91	20.80	29.73	5.85	39.67	10.95	4.31	17.07	39.77	34.15	24.80
Kulczynsik2	10.36	3.65	4.27	27.19	16.32	28.98	6.80	3.09	1.87	9.29	13.28	11.34
Tarantula	26.28	17.91	20.82	29.65	5.85	37.49	10.55	4.31	2.86	22.45	21.91	18.22
Jaccard	23.98	15.61	20.38	29.58	5.45	37.23	10.19	2.75	2.06	9.62	13.17	14.25
Wong1	21.13	20.68	27.97	38.61	43.03	35.87	18.86	16.62	10.05	15.20	16.85	21.25
Wong2	59.37	28.25	30.07	47.12	12.35	49.27	35.78	20.61	12.37	18.61	23.80	30.48
Wong3	9.28	3.65	3.68	27.77	16.32	26.71	9.02	3.84	2.99	9.45	20.27	12.64
M2	14.16	4.37	7.31	28.23	5.02	30.84	6.82	2.96	2.91	9.01	12.82	11.80
Dstar	22.46	4.32	29.26	30.63	7.62	30.67	7.08	7.99	17.93	39.25	33.94	23.85
Op2	10.24	3.65	3.45	27.00	16.32	26.63	5.89	5.05	2.99	8.98	12.78	11.42
GP02	25.33	11.85	7.69	31.88	3.33	45.52	10.23	7.68	3.93	12.89	21.50	16.87
GP03	16.50	4.12	3.45	31.21	16.32	37.93	8.52	5.71	6.41	15.03	23.81	15.98
GP19	21.13	20.68	27.97	38.61	43.03	35.87	18.86	17.04	10.05	15.20	16.84	21.93

通过上述对比实验,可以得到如下结论.

(1) Cp1 和 Cp2 方法在 EXAM 指标方面各具优势.在代码排查比少于 42%时,Cp1 具有较好的定位效果;

而当代码排查比处于 42%~60%区间时,Cp2 的错误检出率则占据明显优势;

(2) 本文提出的新方法在缺陷定位效果显著性方面,虽然针对部分实验程序和实验方法显著性并不明显,但整体而言,缺陷定位效果普遍有所改善;

(3) 在平均定位代价方面,Cp1 方法总体最优,但与最优公式 Op2 以及 Kulczynsik2 方法非常接近.当针对不同规模和类型的具体被测程序,则本文方法和最优公式各具优势.

总体来说,基于条件概率 P 模型构建的可疑度计算公式 Cp1 和 Cp2,由于充分考虑了程序频谱和执行结果的内在关联,计算出的可疑语句排查序列更为有效,与实验的 15 种经典方法比较具有较好的缺陷定位效果.

4 结束语

程序频谱和执行结果之间的相关性是基于频谱的缺陷定位方法应用的基本前提.探究和量化它们的潜在关联,有助于设计高效的可疑度计算公式,提高缺陷定位方法的效率.本文通过经验性分析两者之间的内在关联,借助于统计学的条件概率思想,构建了用以量化分析两者关系强弱评估的 P 模型.基于 P 模型的特性,设计了两个新的可疑度计算公式:Cp1 和 Cp2.基于此,提出了一种基于条件概率的缺陷定位新方法.为了验证提出方法的有效性,以 Siemens 套件中的 7 个程序、Space 程序和 3 个 Unix 工具程序为基准评测对象,从 EXAM 指标和平均代价两个角度,与已有的 15 个经典缺陷定位方法进行了实验对比分析.实验结果表明,新方法在总体缺陷定位效果方面具有优势.

虽然对比实验基于不同规模的基准测试程序验证了新方法的有效性,但针对实际软件开发中更多的错误场景,新方法的定位效果则需要进一步的实验验证和观察.此外,Cp1 和 Cp2 两个新可疑度计算公式,在错误检出率和排查比等方面各具优势.如何合理地融合各自的特点,设计更高效的缺陷定位方法,将是我们下一步的研究重点.

References:

[1] Hailpern B, Santhanam P. Software debugging, testing, and verification. IBM Systems Journal, 2002,41(1):4−12. [doi: 10.1147/sj.411.0004]

[2] Zhang YJ, Santelices R. Prioritized static slicing and its application to fault localization. Journal of Systems and Software, 2016, 114:38−53. [doi: 10.1016/j.jss.2015.10.052]

[3] Wen WZ, Li BX, Sun XB, Liu CC. Technique of software fault localization based on hierarchical slicing spectrum. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):977−992 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[4] Mao XG, Lei Y, Dai ZY, Qi YH, Wang CS. Slice-Based statistical fault localization. Journal of Systems and Software, 2014,89: 51−62. [doi: 10.1016/j.jss.2013.08.031]

[5] Abreu R, González A, Zoeteweij P, van Gemund AJ. Automatic software fault localization using generic program invariants. In: Proc. of the 2008 ACM Symp. on Applied Computing. 2008. 712−717. [doi: 10.1145/1363686.1363855]

[6] Mesbah A, van Deursen A, Roest D. Invariant-Based automatic testing of modern Web applications. IEEE Trans. on Software Engineering, 2012,38(1):35−53. [doi: 10.1109/TSE.2011.28]

[7] Gong DD, Su XH, Wang TT, Ma PJ, Yu W. State dependency probabilistic model for fault localization. Information and Software Technology, 2015,57:430−445. [doi: 10.1016/j.infsof.2014.05.022]

[8] Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: Problem determination in large, dynamic internet services. In: Proc. of the Int'l Conf. on Dependable Systems and Networks (DSN 2002). 2002. 595−604. [doi: 10.1109/DSN.2002.1029005]

[9] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2005. 273−282. [doi: 10.1145/1101908.1101949]

[10] Abreu R, Zoeteweij P. An evaluation of similarity coefficients for software fault localization. In: Proc. of the 2006 12th Pacific Rim Int'l Symp. on Dependable Computing (PRDC 2006). 2006. 39−46. [doi: 10.1109/PRDC.2006.18]

[11] 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 with English abstract). http://www.jos.org.cn/1000-9825/4294.htm [doi: 10.3724/SP.J.1001.2013.04294]

[12] Wong WE, Debroy V, Gao R, Li Y. The dstar method for effective software fault localization. IEEE Trans. on Reliability, 2014, 63(1):290−308. [doi: 10.1109/TR.2013.2285319]

[13] Xie XY, Chen TY, Kuo FC, Xu BW. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. on Software Engineering and Methodology (TOSEM), 2013,22(4):31. [doi: 10.1145/2522920.2522924]

[14] Steimann F, Frenkel M, Abreu R. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In: Proc. of the 2013 Int'l Symp. on Software Testing and Analysis. 2013. 314−324. [doi: 10.1145/2483760.2483767]

[15] Naish L, Lee HJ, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Trans. on Software Engineering and Methodology (TOSEM), 2011,20(3):11:1−11:32. [doi: 10.1145/2000791.2000795]

[16] Chen TY, Xie XY, Kuo FC, Xu BW. A revisit of a theoretical analysis on spectrum-based fault localization. In: Proc. of the 2015 IEEE 39th Annual Computer Software and Applications Conf. (COMPSAC). 2015. 17−22. [doi: 10.1109/COMPSAC.2015.196]

[17] Yoo S, Xie XY, Kuo FC, Chen TY, Harman M. No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist. Technical Report, London: University College London, 2014. 1−19.

[18] Abreu R, Zoeteweij P, Van Gemund AJ. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing: Academic and Industrial Conf. on Practice and Research Techniques. 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[19] Willett P. Similarity-Based approaches to virtual screening. Biochemical Society Transactions, 2003,31(3):603−606. [doi: 10.1042/bst0310603]

[20] Xu J, Zhang ZY, Chan WK, Tse T, Li SP. A general noise-reduction framework for fault localization of Java programs. Information and Software Technology, 2013,55(5):880−896. [doi: 10.1016/j.infsof.2012.08.006]

[21] Dallmeier V, Lindig C, Zeller A. Lightweight defect localization for Java. In: Proc. of the European Conf. on Object-Oriented Programming. 2005. 528−550. [doi: 10.1007/11531142_23]

[22] Le TDB, Oentaryo RJ, Lo D. Information retrieval and spectrum based bug localization: Better together. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015. 579−590. [doi: 10.1145/2786805.2786880]

[23] Wong WE, Qi Y. Effective program debugging based on execution slices and inter-block data dependency. Journal of Systems and Software, 2006,79(7):891−903. [doi: 10.1016/j.jss.2005.06.045]

[24] Wen WZ, Li BX, Sun XB, Li J. Program slicing spectrum-based software fault localization. In: Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering. 2011. 213−218. [doi: 10.1109/ICSE.2012.6227049]

[25] Masri W, Abou-Assi R, El-Ghali M, Al-Fatairi N. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. In: Proc. of the 2nd Int'l Workshop on Defects in Large Software Systems: Held in Conjunction with the ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2009). 2009. 1−5. [doi: 10.1145/1555860.1555862]

[26] Xie XY, Kuo FC, Chen TY, Yoo S, Harman M. Provably optimal and human-competitive results in sbse for spectrum based fault localisation. In: Proc. of the 5th Symp. on Search-Based Software Engineering (SSBSE 2013). 2013. 224−238. [doi: 10.1007/978-3-642-39742-4_17]

[27] 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 with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[28] 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. [doi: 10.1007/s10664-005-3861-2]

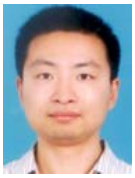
[29] Wong WE, Debroy V, Golden R, Xu X, Thuraisingham B. Effective software fault localization using an RBF neural network. IEEE Trans. on Reliability, 2012,61(1):149−169. [doi: 10.1109/TR.2011.2172031]

附中文参考文献:

[3] 文万志,李必信,孙小兵,刘翠翠.一种基于层次切片谱的软件错误定位技术.软件学报,2013,24(5):977−992. http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[11] 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 [doi: 10.3724/SP.J.1001.2013.04294]

[27] 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]



舒挺(1979－),男,浙江宁海人,博士,副教授,CCF 专业会员,主要研究领域为软件建模,分析与测试.

黄明献(1989－),男,硕士,主要研究领域为软件测试.

丁佐华(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试与可靠性,软件建模与分析,软件自适应控制系统,智能计算及应用.

王磊(1991－),男,硕士生,主要研究领域为软件测试,缺陷定位.

夏劲松(1967－),男,讲师,主要研究领域为自适应软件建模与验证,协同工程,数据分析.