计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(1):188−193 [doi: 10.15888/j.cnki.csa.006701]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于程序频谱的缺陷定位方法①

蔡　蕊[1], 张　仕[1,2], 余晓菲[1], 蒋建民[1,2]

[1](福建师范大学 数学与信息学院, 福建 350117)
[2](福建师范大学 数字福建环境监测物联网实验室, 福建 350117)
通讯作者: 张　仕, E-mail: shi@fjnu.edu.cn

摘　要: 软件测试是生产可靠软件的重要保障, 对测试所发现缺陷的解决可以分为缺陷定位和缺陷修改两个步骤[1], 其中的缺陷定位是最耗时的. 通常情况下, 测试套件中成功执行的测试用例都占绝大多数, 对基于程序频谱的缺陷定位方法, 应该具备自主调节成功测试用例覆盖比重的能力, 以提高方法的可用性. 即, 随着语句被成功测试用例覆盖的次数增多, 该语句的覆盖次数对怀疑率的贡献度应逐渐减小, 成功测试用例数的有效处理能提高缺陷定位方法的效果. 基于此, 本文提出 *EPStar*(*EP**) 缺陷定位方法, 该方法可以有效调整成功执行用例数的影响, 以避免成功用例数量对缺陷定位效果的过度影响, 从而提高缺陷定位的准确性, 通过实验对比, 说明了 *EP**方法比现有的几种缺陷定位方法具有更高的缺陷定位精度.

关键词: *EP**方法; 缺陷定位; 程序频谱; 测试用例; 定位效果

引用格式: 蔡蕊,张仕,余晓菲,蒋建民.基于程序频谱的缺陷定位方法.计算机系统应用,2019,28(1):188–193. http://www.c-s-a.org.cn/1003-3254/6701.html

Defect Localization Method Based on Program Spectrum

CAI Rui[1], ZHANG Shi[1,2], YU Xiao-Fei[1], JIANG Jian-Min[1,2]

[1](School of Mathematics and Informatics, Fujian Normal University, Fujian 350117, China)
[2](Digital Fujian Environmental Monitoring Network Laboratory, Fujian Normal University, Fujian 350117, China)

Abstract: Software testing plays a vital role in producing reliable software. The debugging of software is divided into two steps: fault localization and fault modification, where the fault localization is the most time-consuming and tedious work. Generally, most of test cases in a test suite performed successfully. In order to improve the availability of the defect location method based on the program spectrum, the method should have the ability to adjust the weight of successful coverage automatically. That is, if the contribution of the number of statements to the suspiciousness is reduced gradually with the increase of the number of successful test cases, the effectiveness of fault localization method can be improved greatly. Based on this idea, this study proposes an *EPStar* (*EP**) defect location method, which can effectively adjust the effect of successful use cases, so as to avoid the excessive influence of the number of successful use cases to the defect location effect. To improve the accuracy of error location, the experimental comparison shows that *EP** method has higher defect location accuracy than several existing defect location methods.

Key words: *EP** method; defect localization; program spectrum; test case; location effect

① 基金项目: 国家自然科学基金 (61772004); 上海可信计算重点实验室开放课题 (07dz22304201401); 福建省自然科学基金 (2018J01777)
Foundation item: National Natural Science Foundation of China (61772004); Open Fund of Shanghai Key Laboratory of Trustworthy Computing (07dz22304201401); Natural Science Foundation of Fujian Province (2018J01777)
收稿时间: 2018-05-22; 修改时间: 2018-06-19; 采用时间: 2018-07-06; csa 在线出版时间: 2018-12-26

随着软件项目的规模不断变大, 软件的可靠性也越来越难以保障. 精准、有效的软件缺陷定位技术则有利于发现软件中潜在的问题, 从而提高软件的可靠性. 为修改测试中发现的问题, 需要进行程序的调试, 其过程又可以分为缺陷定位和缺陷改正两个步骤[1]. 早期的缺陷定位以人工为主, 通过在程序中设置断点分析程序, 从而逐步找到缺陷发生的源头. 人工定位缺陷的方法不仅难度大, 而且非常耗时, 对于大型、复杂的软件则更是如此. 因此, 研究人员提出多种缺陷定位方法, 根据是否需要执行测试用例, 又可分为静态缺陷定位技术[2−4]和动态缺陷定位方法[5−7]. 其中, 动态缺陷定位方法则需要执行被测程序, 搜集程序执行过程的行为和执行结果来定位缺陷. 在动态缺陷定位方法中, 基于程序频谱的动态缺陷定位 (SFL) 具有很好的定位效果, 是目前软件缺陷定位问题的研究热点.

目前, 已经提出很多基于频谱的缺陷定位方法[8]: 其中, Jone 等人首先提出 *Tarantula*[9]方法, 取得不错的缺陷定位效果; Abreu 等人提出 *Jaccard*[10]方法和 *Ochiai*[11]方法, 其中 *Ochiai* 方法效果相比 *Tarantula* 方法有了一定的提高; Gonzalez 增加了排除正确语句的优化, 提出了 *Zoltar* 方法[12]; Naish 等人使用自组装映射中计算相似度计算相似度的方法来寻找缺陷语句, 引入 $Kulczynski^1$ 方法和 $Kulczynski^2$ 方法[13]; Wong 等人[14]进一步分析了成功测试用例数对缺陷定位的影响. 他们认为, 语句被成功测试用例覆盖的次数越多, 该语句的覆盖次数对可疑值的贡献度越小[15], 因此提出了 $Wong1(s)$、$Wong2(s)$ 和 $Wong3(s)$ 三个公式, 但当成功执行的用例数量较多时, 利用分段函数降低成功用例的影响这些是不够的, 将对缺陷定位效果的提高不会很明显. 随后为了突出失败用例的影响, 又提出 D^*方法[16]. 通过实证研究发现 D^*方法要优于其他缺陷定位方法.

针对 $Wong1(s)$、$Wong2(s)$ 和 $Wong3(s)$ 分段函数不能自主适应用例数量这一缺点, 本文从成功执行用例数量调节出发, 提出 EP^*缺陷定位方法, 该方法具备自主调节成功测试用例覆盖比重的能力, 以提高方法的适用性范围和有效性. 本文把该方法与其他缺陷定位方法进行对比, 实验结果表明该方法的缺陷定位效果优于现有缺陷定位方法.

本文的第 1 节介绍了基于频谱的缺陷定位方法的基本概念, 并通过一个简单的引例说明了本文方法的基本原理和公式; 第 2 节中详细说明了本文实验所使用的测试程序集及评测指标; 第 3 节中则通过对比分析, 详细说明了 EP^*的有效性; 最后 1 节总结全文.

1 基于频谱的缺陷定位方法

1.1 基本概念

Reps 等人[17]首次提出程序谱概念, 后面的研究者们将程序频谱[18−20]用于程序分析. 程序频谱主要是指程序执行过程中产生的关于程序语句的覆盖信息 (被覆盖为 1, 未覆盖为 0), 以及执行是否通过信息.

在利用测试用例集进行程序的测试过程中, 收集程序语句的覆盖情况和测试通过与否等相关信息, 将每一程序实体 s 相关的统计数据用一个四元组 $T(s)=<T_{ep}(s), T_{ef}(s), T_{np}(s), T_{nf}(s)>$来表示, 其中 $T_{ep}(s)$ 和 $T_{ef}(s)$ 分别表示覆盖程序实体 s 的成功测试用例数和失败测试用例数, $T_{np}(s)$ 和 $T_{nf}(s)$ 分别表示未覆盖程序实体 s 的成功测试用例数和失败测试用例数. 测试套件 T 中所有成功测试用例数 $T_p=T_{ep}(s)+T_{np}(s)$, 所有失败测试用例数 $T_f=T_{ef}(s)+T_{nf}(s)$, 所有测试用例数 $T=T_p+T_f$.

$T_{ep}(s)$ 的数值越大, 说明语句 s 执行且用例成功的次数多, s 是缺陷语句的可能性越小; $T_{ef}(s)$ 的数值越大, 说明语句 s 执行且用例错误的次数多, s 是缺陷语句的可能性越大; $T_{nf}(s)$ 的数值越大, 则说明语句 s 没有被执行到的情况下错误越多, 这间接说明 s 是缺陷语句的可能性越小.

1.2 *EP**的缺陷定位方法

Wong 等人[14]总结的假设 5–7 说明了:可疑值与 $T_{ef}(s)$ 大小成正比, 与 $T_{ep}(s)$ 大小和 $T_{nf}(s)$ 大小成反比, 可以设 *Kulczynski* 系数为 1 来表示 $T_{ef}/(T_{nf}+T_{ep})$. 基于假设 8 对 T_{ef} 设置更高的权重, 可以设置 *Kulczynski* 系数为 2 来表示 $2*T_{ef}/(T_{nf}+T_{ep})$. 但 *Kulczynski* 系数在缺陷定位的准确性上不够. 考虑到这一点, Wong 等人提出采用*系数, 取值范围从 2~50 增量为 0.5. 提出 D^*方法, 通过实验验证该方法在缺陷定位上效果很好.

此外, 在文献[21]中, 其通过预处理的方式去除部分测试用例, 也是希望减少成功执行数量对可疑值的过度影响, 以提高定位精度. 由预处理目的[21]和 Wong 等人 D^*公式启发, 为了有效降低 $T_{ep}(s)$ 对整体数值的影响, 本文提出 EP^*方法, 如公式 (1) 所示. 公式采用*系数, 取值范围与 D^*一样. 在具体的应用中, T_{ef} 数量要比 T_{ep} 小很多, 在降低 T_{ep} 的权重时对缺陷定位的影响会更加明显.本文实验也验证了这个猜想, 说明了 EP^*

方法的缺陷定位要比 D^*方法更优.

$$EP^*(s)=\frac{T_{ef}(s)}{T_{nf}(s)+T_{ep}(s)^{\frac{1}{*}}} \tag{1}$$

1.3 引例

为了更好的理解, 我们通过一个实例[16]来说明 EP^*缺陷定位方法. 在表 1 中, S 表示待测程序的语句集, S={s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8,s_9}, s_i 表示具体的程序语句; P 表示被测程序实体, 其中缺陷语句是 s_3; t_i 表示第 i 个测试用例执行对程序的覆盖向量, 其中黑点代表该语句被执行, 反之未执行; 表 1 中最后一行的 P/F 表示执行成功与否, 0 表示测试用例执行成功, 反之失败; $susp_1$ 表示测试用例基于 *Ochiai* 技术计算的可疑值; $rank_1$ 表示程序中可疑值大于等于该语句的数量 (含该语句本身). 从表 1 中可以看出, 缺陷语句 s_3 的排名与语句 s_4 并列第 3, 表示为确保测试人员找到缺陷语句, 最多需要检查 3 条程序语句.

表 1 示例程序表

S	Program (P)	t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12}	$susp_1$	$rank_1$
s_1	read (*a*, *b*);	· · · · · · · · · · · ·	0.58	6
s_2	if (*a*<10 && *b*<10)	· · · · · · · · · · · ·	0.58	6
s_3	result=*a*-*b*; // Correct : *a*+*b*	· · · · · · · · · · · ·	0.6	3
s_4	if (result >0)	· · · · · · · · · · · ·	0.6	3
s_5	print (“positive”);	· · · · · ·	0.2	8
s_6	else if (result==0)	· · · · · ·	0.67	1
s_7	print (“zero”);	· · ·	0.35	7
s_8	else print (“negative”);	· · · ·	0.58	6
s_9	else print (“invalid input”);		0	9
P/F	(pass=0 / faild=1)	0 0 0 0 0 0 0 0 1 1 1 1	Ochiai	

表 2 对比了 D^*方法和 EP^*方法中的 *Kulczynski* 系数取为 1、2、3 时的示例程序排名情况. 结合表 1, 可以发现, 当 *Kulczynski* 系数取 1 时, 缺陷语句排名 $rank_2$=4; *Ochiai* 方法的缺陷语句排名 $rank_1$=3, 则 $Kulczynski^1$ 方法的定位效果不如 *Ochiai* 方法. 当 *Kulczynski* 系数取 2 时, D^*方法的缺陷语句排名 $rank_3$=3; EP^*方法的缺陷语句排名 $rank_5$=2, 提升两名, 则 EP^*方法的定位效果比 D^*方法和 *Ochiai* 方法好. 当 *Kulczynski* 系数取 3 时, D^*方法的缺陷语句排名 $rank_4$=2, 提升一名; EP^*方法的缺陷语句排名 $rank_6$=2 保持不变. 从该例中可以看出, EP^*方法在取相同系数时, 能够更快逼近最优值. 同时也可以发现, 由于 s_3 和 s_4 具有相同的覆盖情况, 任何可疑度计算都无法区分二者, 即当 *Kulczynski* 系数取 2 和 3 时, $rank_5$=2、 $rank_6$=2 已经达到最佳缺陷定位效果.

在表 2 中, 语句 s_1、s_3 和 s_6 的执行通过语句数分别是 T_{ep}=8、T_{ep}=7 和 T_{ep}=2.在 $Kulczynski^1$ 方法中不同的语句的覆盖次数对怀疑率的贡献度是一样的, 但语句 s_1 和 s_3 的 T_{ep} 值要比 s_6 大很多, 所以 s_1 和 s_3 的排名都在 s_6 后面, 而 s_1 和 s_3 的排名很接近. 在 EP^2 方法中, 不同的语句的覆盖次数对怀疑率的贡献度不同, 其中 T_{ep} 值越大, 贡献度越低. 这样就减弱了执行通过语句数 T_{ep} 对语句 s_1、s_3 与 s_6 的影响. 但 s_3 中的 T_{ef} 比 s_1 和 s_6 大, 所以 s_3 的排名都在 s_1 和 s_6 前面. 因此 EP^*方法能够减弱执行通过语句数 T_{ep} 对可疑值的过度影响.

表 2 可疑值计算对比表

Program (*P*)				Kulczynski(D^1)		D^2		D^3		EP^2		EP^3	
S	T_{ef}	T_{nf}	T_{ep}	$sups_2$	$rank_2$	$sups_3$	$rank_3$	$sups_4$	$rank_4$	$sups_5$	$rank_5$	$sups_6$	$rank_6$
s_1	4	0	8	0.50	6	2.00	5	8.00	5	1.41	4	2.00	4
s_2	4	0	8	0.50	6	2.00	5	8.00	5	1.41	4	2.00	4
s_3	4	0	7	0.57	4	2.29	3	9.14	2	1.51	2	2.09	2
s_4	4	0	7	0.57	4	2.29	3	9.14	2	1.51	2	2.09	2
s_5	1	3	5	0.13	8	0.13	8	0.13	8	0.19	8	0.21	8
s_6	3	1	2	1.00	1	3.00	1	9.00	3	1.24	5	1.33	5
s_7	1	3	1	0.25	7	0.25	7	0.25	7	0.25	7	0.25	7
s_8	2	2	1	0.67	2	1.33	6	2.67	6	0.67	6	0.67	6
s_9	0	4	1	0.00	9	0.00	9	0.00	9	0.00	9	0.00	9

基于上述分析, 简要说明了本文提出的 EP^*方法在缺陷定位中的应用, 在该实例中, EP^*方法显著减弱了执行通过语句数 T_{ep} 对可疑值的过度影响, 使其缺陷定位效果比 *Ochiai* 和 D^*方法好. 当系数取 2 时, EP^*方法中的缺陷语句排名已经达到 D^3 方法的缺陷语句排名, 从中可以看出, EP^*方法比 D^*方法收敛更快. 在下一节

中将通过实验对比分析 EP^*缺陷定位方法与其他缺陷定位方法的实验结果.

2 实验

本节首先介绍目标程序信息, 再通过实验说明 EP^* 方法的缺陷定位效果.

2.1 测试程序集

实验使用了 SIR 中的 Siemens suite 测试套件和 space 程序[22], Siemens suite 包含 7 个程序, 它是由实现不同功能的 C 程序组成, 每组程序通过人工注入的方式植入缺陷. space 程序测试包含的多个版本都是实际的缺陷, 其详细信息如表 3 所示. 表 3 中各列分别表示程序名、每个程序缺陷版本数、程序代码行数、测试用例个数、程序功能的简要描述. 该测试集涉及各种缺陷类型, 以模拟实际可能存在的缺陷. 本实验用到了 Siemens suite 测试套件中的 121 个版本和 space 程序的 20 个版本. 其中, 去除某些缺陷版本的原因有:“Core dumped”错误无法生成覆盖文件、宏定义错误和头文件缺陷等.

表 3 实验程序信息

Name	Num	Line	Quantity	Describe
print_tokens	7	472	4130	词法分析程序
print_tokens2	10	399	4115	词法分析程序
replace	32	512	5542	正则表达式替换
schedule	9	292	2650	优先级调度
schedule2	10	301	2710	优先级调度
tcas	41	141	1608	海拔高度分离
totinfo	23	440	1052	信息度量
space	38	9562	13525	解释器

2.2 缺陷定位评测指标

基于语句排名的评测采用了 *Score* 评测指标. 按照可疑值 *Suspiciousness* 的取值从高到低进行排序, 值越高代表该语句为缺陷语句的可能性越大, 反之, 则代表该语句是缺陷语句的可能性越小. 假设排查程序 P=$\{s_1,s_2,\cdots,s_n\}$, 其中 s_f 为缺陷语句, n 为程序 P 的语句数. 衡量缺陷定位方法优劣的指标是看在该方法中, 按可疑率排序缺陷语句被检查到的位序 ($Rank$). 当存在 $m(m>0)$ 个语句与缺陷语句 s_f 的可疑值取值相等时, 目前有两种方法计算 *Rank* 值. 假设可疑值高于 s_f 的程序语句个数为 t, 则方法一: 考虑最坏情况, 即设缺陷语句的 $Rank=m+t$, 本实验采用该方法计算 *Rank* 值; 方法二: 考虑平均情况, 即设缺陷语句的 $Rank=t+m/2$. 基于缺陷语句的 $Rank$ 值, 常用的评测指标为 $Score$ 如公式 (2) 所示, 其表示不用检查的语句占所有语句的百分比. *Score* 值越高, 则说明程序员在缺陷定位时不用检查的语句越多, 需要检查的代码行数越少, 也就说明了缺陷定位的效果越好, 反之, 则说明缺陷定位方法效果差.

$$Score = (1 - Rank(s_f)/n) * 100\% \quad (2)$$

3 实验分析

为了能够清晰对比本文所提出的缺陷定位方法 EP^*对缺陷定位的影响, 本节通过实验对比 *Tarantula* 方法、*Jaccard* 方法、*Ochiai* 方法和 D^*方法的 *Score* 评价指标来说明该方法的有效性, 实验设置系数 (*) 分别为 3、5 和 10. 表 4 是基于西门子套件的每个分数范围运行百分比, 而表 5 是基于 space 程序的每个分数范围运行百分比.

表 4 西门子套件中的每个分数范围运行百分比

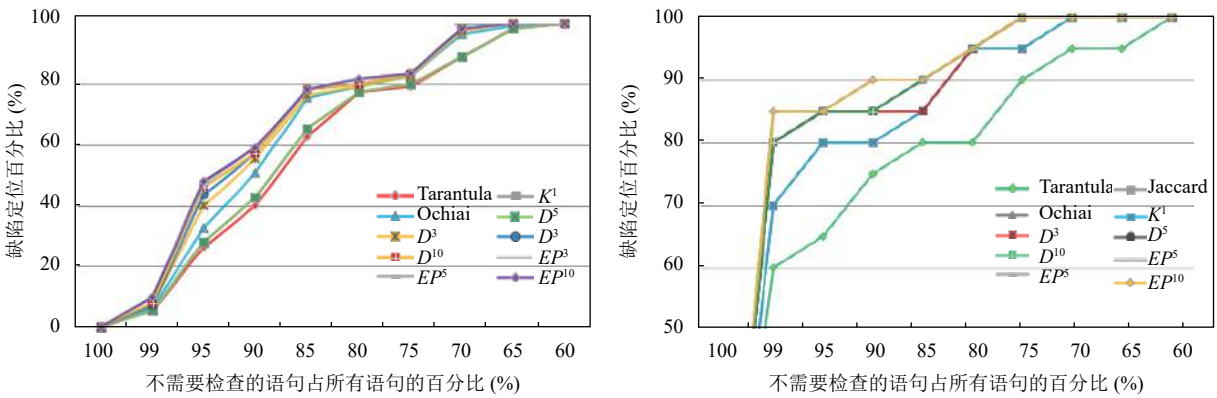
分数 (%)	Tarantula	Jaccard	Ochiai	Kulczynski[1]	D^3	D^5	D^{10}	EP^3	EP^5	EP^{10}
99–100	5.45	5.45	6.36	5.45	7.27	7.27	8.18	10	10	10
95–99	20.91	22.73	26.36	22.73	32.73	36.36	38.18	37.27	38.18	38.18
90–95	13.64	14.55	18.18	14.55	15.45	13.64	10.91	11.82	10.91	10.91
85–90	22.73	22.73	24.55	22.73	20.91	20.91	20.91	19.09	19.09	19.09
80–85	14.55	11.82	3.64	11.82	2.73	1.82	1.82	3.64	3.64	3.64
75–80	1.82	2.73	3.64	2.73	3.64	3.64	3.64	1.82	1.82	1.82
70–75	10	9.09	13.64	9.09	14.55	14.55	14.55	14.55	14.55	14.55
65–70	9.09	9.09	2.73	9.09	2.73	1.82	1.82	1.82	1.82	1.82
0–65	1.82	1.82	0.91	1.82	0	0	0	0	0	0

从表 4 中可以看出, D^3、D^5、D^{10}、EP^3、EP^5 和 EP^{10} 的所有缺陷语句均能在不用检查的语句占所有语句的 65% 内被发现, 但 *Tarantula*、*Jaccard*、*Ochiai* 和 *Kulczynski*[1] 在检查 35% 代码后还有少部分缺陷语句是无法排查到. 查看 *Score* 在 99–100% 区间内, 即, 检查代码数量在 1% 内便可确定缺陷代码位置

的情况. 其中 EP^3、EP^5 和 EP^{10} 在检查代码数量的 1% 内可排查缺陷占 10%, 分别比 *Kulczynski*[1]、*Jaccard*、*Tarantula*、D^{10}、D^5、D^3、*Ochiai* 的准确率提升了 4.55、4.55、4.55、1.82、2.73、2.73、3.64 个百分点. 图 1(a) 对应于表 4 数据, 表示基于西门子套件的 *Score* 值的缺陷定位效果对比图. 从图中可以发现 EP^3、EP^5 和 EP^{10} 曲线均在其它方法的曲线上面, 然后是 D^{10}、D^5 和 D^3 曲线, 最下面的曲线是 *Tarantula*. 由此可知: 在缺陷定位效果上, 基于西门子套件中 EP^*方法只需检查更少量的代码行就可以找出缺陷位置, 即, EP^3、EP^5 和 EP^{10} 比其它方法好, 其中 *Tarantula* 方法最差.

表 5　space 程序中的每个分数范围运行百分比

分数 (%)	Tarantula	Jaccard	Ochiai	Kulczynski[1]	D^3	D^5	D^{10}	EP^3	EP^5	EP^{10}
99–100	60	70	80	70	80	80	80	85	85	85
98–99	5	10	5	10	5	5	5	0	0	0
97–98	10	0	0	0	0	0	0	5	5	5
96–97	5	5	0	5	0	5	5	0	0	0
95–96	0	10	10	10	10	5	5	5	5	5
90–95	10	0	5	0	5	5	5	5	5	5
80–90	5	5	0	5	0	0	0	0	0	0
70–80	0	0	0	0	0	0	0	0	0	0
60–70	5	0	0	0	0	0	0	0	0	0



(a) 西门子套件　　(b) space 程序

图 1　基于 *Score* 值的缺陷定位效果对比图

从表 5 中可以看出, D^3、D^5、D^{10}、EP^3、EP^5、EP^{10} 和 *Ochiai* 的所有缺陷语句均能在不用检查的语句占所有语句的 90% 内被发现, 但 *Tarantula*、*Jaccard* 和 *Kulczynski*[1] 在检查 10% 代码后还有部分缺陷语句是无法排查到.其中 EP^3、EP^5 和 EP^{10} 在检查代码数量的 1% 内可排查缺陷占 85%, 比 D^{10}、D^5、D^3 和 *Ochiai* 提升了 5 个百分点; 比 *Kulczynski*[1] 和 *Jaccard* 准确率提升了 15 个百分点; 比 *Tarantula* 准确率提升了 25 个百分点. 图 1(b) 对应于表 5 数据, 表示基于 space 程序的 *Score* 值的缺陷定位效果对比图. 从图中可以发现 EP^3、EP^5 和 EP^{10} 曲线均在其它方法的曲线上面, 然后是 D^{10}、D^5 和 D^3 曲线, 最下面的曲线是 *Tarantula*. 由此可知: 在缺陷定位效果上, 对 space 程序, EP^*方法只需检查更少量的代码行就可以找出缺陷位置, 即, EP^3、EP^5 和 EP^{10} 比其它方法好, 其中 *Tarantula* 方法最差.

从收敛性上分析表 4 数据, 分数范围运行百分比为 99% 时 EP^3、EP^5 和 EP^{10} 已经收敛达到最佳定位效果 *Score* 值为 10, 但 D^3、D^5 和 D^{10} 的 *Score* 值分别为 7.27、7.27 和 8.18, 显然比 EP^*低. 再查看 *Score* 值在 90~100% 之间的情况 (即发现缺陷需检查代码在 10% 范围内), EP^3、EP^5 和 EP^{10} 值均达到 59.09, 而 D^3、D^5 和 D^{10} 的值分别为 55.45、57.27 和 57.27. 从中可以明显看出, 随着系数增大, EP^*比.D^*更快接近最佳定位效果. 所以 EP^*方法的收敛性比 D^*方法好.

通过上述实验及分析可以发现, 在目标测试程序中, EP^*方法表现出很好的缺陷定位效果, 比现有的缺陷定位方法都要好很多.

4　结果与展望

通过实验发现, 基于 EP^*的缺陷定位方法有利于提高缺陷定位效果, 实验结果表明, EP^*方法的缺陷定位效果比现有的方法好, 而且能够有效调整成功执行用例数, 以避免成功用例数量对缺陷定位效果的影响.

参考文献

1 Liu H, Wang WH, Zhang DF. A methodology for mapping and partitioning arbitrary n-dimensional nested loops into 2-dimensional VLSI arrays. Journal of Computer Science and Technology, 1993, 8(3): 221–232. [doi: 10.1007/BF02939529]

2 Wen WZ, Li BX, Sun XB, *et al*. Technique of software fault localization based on hierarchical slicing spectrum. Journal of Software, 2013, 24(5): 977–992. [doi: 10.3724/SP.J.1001.2013.04342]

3 Mao XG, Lei Y, Dai ZY, *et al*. Slice-based statistical fault localization. Journal of Systems and Software, 2014, 89: 51–62. [doi: 10.1016/j.jss.2013.08.031]

4 Wong WE, Gao RZ, Li YH, *et al*. A survey on software fault localization. IEEE Transactions on Software Engineering, 2016, 42(8): 707–740. [doi: 10.1109/TSE.2016.2521368]

5 Renieres M, Reiss SP. Fault localization with nearest neighbor queries. Proceedings of the 18th IEEE International Conference on Automated Software Engineering. Montreal, Quebec, Canada. 2003. 30–39.

6 Liu C, Yan XF, Fei L, *et al*. SOBER: Statistical model-based bug localization. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 286–295. [doi: 10.1145/1095430]

7 Abreu R, Zoeteweij P, van Gemund AJC. On the accuracy of spectrum-based fault localization. Testing: Academic and Industrial Conference Practice and Research Techniques. Windsor, UK. 2007. 89–98.

8 Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011, 34(8): 1411–1422. [doi: 10.3724/SP.J.1016.2011.01411]

9 Jones JA, Harrold MJ. Empirical evaluation of the Tarantula automatic fault-localization technique. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. Long Beach, CA, USA. 2005. 273–282.

10 Baudry B, Fleurey F, Le Traon Y. Improving test suites for efficient fault localization. Proceedings of the 28th International Conference on Software Engineering. Shanghai, China. 2006. 82–91.

11 Wen W, Li B, Sun X, *et al*. Program slicing spectrum-based software fault localization. In Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering. Miami, FL, UAS. 2011. 213–218.

12 Gonzalez A. Automatic error detection techniques based on dynamic invariants[Master's Thesis]. Delft City: Delft University of Technology, 2007.

13 Lourenço F, Lobo V, Bação F. Binary-Based similarity measures for categorical data and their application in self-organizing maps. In Proceedings of the JOCLAD, 2004-XI Jornadas de Classificacao e Anlise de Dados. Lisbon. 2004. 1–18.

14 Wong WE, Debroy V, Gao RZ, *et al*. The DStar method for effective software fault localization. IEEE Transactions on Reliability, 2014, 63(1): 290–308. [doi: 10.1109/TR.2013.2285319]

15 Wong WE, Qi Y, Zhao L, *et al*. Effective fault localization using code coverage. In: Proceedings of the 31sh Annual International Computer Software and Applications Conference. Beijing, China. 2007. 449–456.

16 Wong WE, Debroy V, Li YH, *et al*. Software fault localization using DStar (D*). Proceedings of the 6th IEEE International Conference on Software Security and Reliability. Gaithersburg, MD, USA. 2012. 21–30.

17 Reps T, Ball T, Das M, *et al*. The use of program profiling for software maintenance with applications to the year 2000 problem. Proceedings of the 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Zurich, Switzerland. 1997. 432–449.

18 Steimann F, Frenkel M, Abreu R. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. Proceedings of 2013 International Symposium on Software Testing and Analysis. Lugano, Switzerland. 2013. 314–324.

19 Xie XY, Chen TY, Kuo FC, *et al*. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 31.

20 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]

21 Yu YB, Jones J, Harrold MJ. An empirical study of the effects of test-suite reduction on fault localization. Proceedings of the ACM/IEEE 30th International Conference on Software Engineering. Leipzig, Germany. 2008. 201–210.

22 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]