软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):879−900 [doi: 10.13328/j.cnki.jos.004970]

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

一种基于遗传算法的多缺陷定位方法*

王 赞[1], 樊向宇[1], 邹雨果[1,3], 陈 翔[2]

[1](天津大学 软件学院 软件工程系,天津 300072)

[2](南通大学 计算机科学与技术学院,江苏 南通 226019)

[3](厦门航空有限公司 信息部,福建 厦门 361006)

通信作者: 樊向宇, E-mail: fxy@tju.edu.cn

摘 要: 基于程序频谱的缺陷定位方法可以有效地辅助开发人员定位软件内部缺陷,但大部分已有自动化方法在解决多缺陷定位问题时表现不佳,部分效果尚可的方法因复杂度较高或需要开发人员较多交互而仍需进一步改善.为改善上述问题,提出一种基于遗传算法的多缺陷定位方法 GAMFal,具体来说:首先基于搜索的软件工程思想对多缺陷定位问题进行建模,构建了候选缺陷分布的染色体编码方式,并基于扩展的 Ochiai 系数计算个体的适应度值;随后使用遗传算法在解空间中搜索具有最高适应度值的候选缺陷分布,在终止条件被满足后返回最优解种群;最后根据这个种群对程序实体进行排序.这样开发人员可以依次对程序实体进行检查并最终确定多个缺陷的具体位置.实证研究以 Siemens 套件中的 7 个程序和 Linux 的 3 个程序(gzip、grep 和 sed)作为评测对象,并扩展传统的定位方法评测标准 EXAM 至 EXAMF 和 EXAML,通过与其他经典的缺陷定位方法(Tarantula、Improved Tarantula 及 Ochiai)进行对比,并通过 Friedman 检测和最小显著性差异测试可得,提出的 GAMFal 方法在整体定位效率方面优于传统方法,且需要更少的人工交互.除此之外,GAMFal 的执行时间也在可接受的范围之内.

关键词: 缺陷定位;多缺陷;基于搜索的软件工程;遗传算法;EXAM 评价标准

中图法分类号: TP311

中文引用格式: 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

英文引用格式: 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

Genetic Algorithm Based Multiple Faults Localization Technique

WANG Zan[1], FAN Xiang-Yu[1], ZOU Yu-Guo[1,3], CHEN Xiang[2]

[1](Department of Software Engineering, School of Computer Software, Tianjin University, Tianjin 300072, China)

[2](School of Computer Science and Technology, Nantong University, Nantong 226019, China)

[3](Department of Information, Xiamen Airlines, Xiamen 361006, China)

Abstract: Spectrum-Based fault localization techniques are attractive for their effectiveness, and previous works have demonstrated that they can assist programmers to locate faults automatically. However, most of them can only work better when there is single bug than multiple bugs. Other approaches, although partially successful on multiple faults problem, are complex and need more human intervention. To better address these problems, this paper proposes a new spectrum-based fault localization technique based on genetic algorithm, called GAMFal, which can locate multiple bugs effectively with less human intervention. First, the multiple bugs' localization is converted into a search based model and a candidate expression for multiple bugs' location is encoded as an individual binary string. Then,

* 基金项目: 国家自然科学基金(61202030, 61373012, 61202006)
Foundation item: National Natural Science Foundation of China (61202030, 61373012, 61202006)
收稿时间: 2015-08-31; 修改时间: 2015-10-15; 采用时间: 2015-11-20; jos 在线出版时间: 2016-01-13
CNKI 网络优先出版: 2016-01-14 13:16:12, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1316.009.html

the new approach extends the Ochiai coefficient to calculate the suspiciousness value used by genetic algorithm as a fitness function to search for a best population composed by optimal fault location candidates with highest suspiciousness value, and converts the ranking list of candidates to a checking order of program entities. According to this order, programmers finally examine program entities to locate faults. An empirical study on Siemens suites and three Linux programs (gzip, grep and sed) is conducted to compare GAMFal with other spectrum-based approaches. The Friedman test and Least Significance Difference method are then carried out to investigate the statistical significance of any differences observed in the experiments. The result suggests that the proposed method outperforms other related techniques in some respects and is feasible with respect to running time.

Key words: fault localization; multiple faults; search-based software engineering; genetic algorithm; EXAM

软件缺陷定位(software fault localization)技术是在执行测试用例集后发现有部分测试用例执行失败时,确定缺陷所在具体位置的一种分析方法.在传统的软件开发过程中,通常是由开发人员手工调试、找到缺陷并进行修复.但这种传统的缺陷定位方法成本较高[1].为了提高调试效率,把开发人员从枯燥的手工调试中解放出来,研究人员提出了大量的自动化缺陷定位方法用以辅助开发人员快速准确地定位缺陷.已有的自动化缺陷定位方法可以简单地分为两类:静态定位方法和动态定位方法.其中静态定位方法[2]在程序运行前通过分析代码结构来定位缺陷,而动态定位方法[3−5]则通过分析测试用例的执行轨迹和运行结果来定位缺陷.在动态定位方法中,基于程序频谱的缺陷定位(program spectrum based fault localization,简称 SFL)技术表现出很好的定位效果[5].它们通过计算出每个程序实体(可设置为语句、语句块或函数等)内含有缺陷的可能性,然后生成缺陷分析报告并以此来辅助开发人员进行调试,直至找到真正缺陷位置并完成修复.SFL 是当前软件缺陷定位问题的研究热点,也是本文研究的问题.

程序频谱是执行测试用例时收集到的程序执行信息,包括测试用例的程序实体覆盖信息和执行结果.在大量 SFL 方法中,Tarantula[3]、Jaccard[6]、Ochiai[7]和 O_p[8]等方法取得了较好的效果.它们使用统计学方法计算出程序实体的可疑值,并依次排序.但这类方法在定位单缺陷时的效果要优于定位多缺陷时的效果[4].然而在实际软件开发过程中,被测程序内部含有的缺陷数难以预先获知,并且在绝大多数情况下都多于 1 个.

目前研究人员对多缺陷定位问题进行了初步研究.Jones 等人[4]基于测试用例执行结果将被测程序划分为几个不同部分,随后指派不同的开发人员分别去定位相关部分内的缺陷.这种方法需要多个开发人员来检查代码,因此需要较高的人力成本.Abreu 等人提出 BARINEL 方法[9],这种方法使用贝叶斯模型排序代表多缺陷的候选集.BARINEL 方法在单缺陷定位和多缺陷定位中都具有较好的性能,但是它需要开发人员在检查代码的过程中保持实时交互,以确保可以依此不断修正候选集排序.Steimann 和 Bertschler[10]尝试使用概率分布来估计内部缺陷数.然而,缺陷的概率分布难以预知,并且很难进行估计和验证.MUSE 方法[11]不断地对程序语句执行变异操作,试图得到所有测试用例都能通过的新版本来定位到缺陷.该方法的效果很好,但是需要耗费大量时间,难以在实际中使用.

本文提出一种基于遗传算法的多缺陷定位方法框架 GAMFal.首先以基于搜索的软件工程思想对多缺陷问题进行建模,将多缺陷定位问题转化为一类搜索问题;然后以二进制染色体形式表达潜在的多缺陷候选分布,同时,基于覆盖的频谱信息及“通过/失败”的执行结果拓展 Ochiai 方法以构建适应值函数,并以此为依据采用遗传算法搜索解空间,搜索出的最优解可显性地表示多个缺陷的可能位置.这样,该方法便能在较少人工参与的情况下有效地进行多缺陷定位.本文的主要贡献总结如下:(1) 提出基于遗传算法的多缺陷定位方法 GAMFal.具体来说,将多缺陷定位问题建模为一类搜索问题,拓展传统的 Ochiai 系数,使其能够适应于多缺陷定位问题;随后以此为适应度值函数,基于遗传算法寻找最优候选集,并在此基础上对程序实体进行可疑值排序.(2) 基于现有缺陷定位方法评价标准 $EXAM$ 的两个扩展标准 $EXAM_F$ 和 $EXAM_L$(分别表示找到第 1 个缺陷和找到最后一个缺陷需要检查的程序实体占总程序实体的百分比),实验对比本文方法和其他 3 种方法(Tarantula、Improved Tarantula 和 Ochiai)的性能,证明本文方法在多缺陷定位时具有一定的优势.

本文第 1 节介绍研究背景,并通过一个简单实例,引出研究动机.第 2 节描述我们提出的方法框架 GAMFal.第 3 节为实证研究,描述实验设计、采用的评测数据集、GAMFal 方法在 Siemens 套件的 7 个程序和 Linux 的 3 个程序上的缺陷定位效果及与其他 3 种缺陷定位方法的比较实验,并讨论实验结果以及有效性影响因素.第 4

节总结全文并指出值得关注的下一步研究工作.

1 研究背景和相关工作

1.1 基于频谱的软件缺陷定位

本节首先对 SFL 问题进行描述,随后给出一个具体实例进行说明.

定义 1(测试用例集). $T=\{t_1,t_2,...,t_m\}$ 表示被测程序的配套测试用例集,其中,t_i表示该测试用例集的第 i 个测试用例.

定义 2(待测程序实体集). $E=\{e_1,e_2,...,e_m\}$ 表示被测程序内包含的程序实体集,其中,e_i表示第 i 个程序实体.程序实体粒度可设置为语句、语句块或函数等.

定义 3(覆盖矩阵). $M=(M_{ij})$用来表示 T 和 E 之间的覆盖关系.M 是一个 $m\times n$ 的矩阵,其中第 i 行表示第 i 个测试用例的程序实体覆盖情况,第 j 列表示第 j 个程序实体被不同测试用例覆盖的情况.每一个 M_{ij} 代表第 i 个测试用例对第 j 个程序实体的覆盖情况,当 $M_{ij}=1$ 时,表示测试用例 i 覆盖程序实体 j.当 $M_{ij}=0$ 时,表示测试用例 i 未覆盖程序实体 j.

定义 4(失败测试用例覆盖矩阵). $M_F=(M_{ij})$是覆盖矩阵 M 的一部分,只包含失败测试用例的覆盖信息.

定义 5(通过测试用例覆盖矩阵). $M_F=(M_{ij})$是覆盖矩阵 M 的另一部分,只包含通过测试用例的覆盖信息.

定义 6(执行结果向量). $R=\{r_1,r_2,...,r_m\}$ 代表测试用例集中的测试用例的测试结果,r_i 表示第 i 个测试用例的执行结果,$1\leqslant i\leqslant m$.当 $r_i=0$ 时,表示第 i 个测试用例执行通过.当 $r_i=1$ 时,表示第 i 个测试用例执行失败.

定义 7(程序实体统计量). 程序频谱构造方式仅需简单统计测试用例的程序实体覆盖信息.当程序实体为语句、语句块或函数时,基于各个测试用例的程序频谱和执行结果,可由程序实体统计量表示.程序实体统计量定义为 $a_{p,q}(e)=\left|\{i\mid M_{ij}=p\wedge r_i=q\}p,q\in\{0,1\}\right|$;$a_{1,0}(e)$和 $a_{1,1}(e)$分别表示"覆盖程序实体 e"的通过测试用例数和失败测试用例数,$a_{0,0}(e)$和 $a_{0,1}(e)$分别表示"未覆盖程序实体 e"的通过测试用例数和失败测试用例数.

Tarantula 可疑度系数[3]S_T 和 Ochiai 可疑度系数[7]S_O 的计算公式依次定义如下:

$$S_T(e)=\frac{\dfrac{a_{11}(e)}{a_{11}(e)+a_{01}(e)}}{\dfrac{a_{11}(e)}{a_{11}(e)+a_{01}(e)}+\dfrac{a_{10}(e)}{a_{10}(e)+a_{00}(e)}} \tag{1}$$

$$S_o(e)=\frac{a_{11}(e)}{\sqrt{\left(a_{11}(e)+a_{01}(e)\right)\times\left(a_{11}(e)+a_{10}(e)\right)}} \tag{2}$$

Jones 等人[3]还提出引入了置信度的改进 Tarantula 方法,用来辅助 Tarantula 方法排序,其定义如下:

$$S_T^*(e)=S_T(e)\times\max\left\{\frac{a_{11}(e)}{a_{11}(e)+a_{01}(e)},\frac{a_{10}(e)}{a_{10}(e)+a_{00}(e)}\right\} \tag{3}$$

下面通过图 1 所示的函数 *findSecond* 来说明这类方法的工作方式,并对比 GAMFal 方法与已有方法的差异,讨论 GAMFal 方法的可行性.*findSecond* 函数的输入是 4 个整数,输出是这些整数中的第二大整数.在图 1 给出的版本中,第 14 行和第 17 行程序都有错误,即在第 14 行中代码 *temp=b* 应为 *temp*=min,在第 17 行中代码 *temp=b* 应为 *temp=c*.

目前已有的可疑度值计算公式一般基于陈翔等人总结的 8 个假设[12],该文是对虞凯等人工作的进一步完善[13],其中一个假设是程序实体的怀疑度值与该程序实体被失败测试用例覆盖的次数成正比.如果程序中只有一个缺陷,那么这个假设是可以被接受的,但是如果程序中含有两个或两个以上的缺陷,则存在如下问题:(1) 被失败测试用例覆盖最多的一般不是这两个缺陷所在的代码行,而是这些失败测试用例共同覆盖的那部分程序实体,这样得到的怀疑度值排序结果显然是不够准确的;(2) 单缺陷定位方法会在很大程度上忽略缺陷间的干扰以及相互作用情况[14].

```
函数 findSecond.
1.    int findSecond(int a, int b, int c, int d) {
2.        int max, min, temp;
3.        if (a<=b) {
4.            max=b;
5.            min=a;
6.        } else {
7.            max=a;
8.            min=b;
9.        }
10.       if (c>max) {
11.           temp=max;
12.           max=c;
13.       } else if (c<min) {
14.           temp=b;   /*缺陷.应为 temp=min*/
15.           min=c;
16.       } else {
17.           temp=b;   /*缺陷.应为 temp=c*/
18.       }
19.       if (d<max && d>temp) {
20.           return d;
21.       } else if (d>max) {
22.           return max;
23.       } else {
24.           return temp;
25.       }
26.   }
```

Fig.1 Function *findSecond* with multiple faults

图 1 含有多个缺陷的函数 *findSecond*

根据图 1 的被测程序及表 1 中测试用例的覆盖情况和执行结果可以看出,现有的 3 种方法中只有 Tarantula 可以快速找到第 1 个缺陷.在 3 种方法计算出的可疑值中,排名第 2 的程序实体都是失败测试用例共同经过的第 24 行,但第 24 行并没有缺陷.同时,没有缺陷的第 4 行、第 5 行也因被较多的失败测试用例覆盖而排名较为靠前.而运用本文提出的 GAMFal 方法搜索出的 3 个最优解分别为⟨13⟩、⟨14,17⟩组合以及⟨15,17⟩组合,根据这 3 个最优解可首先定位到第 17 行包含的缺陷,然后至多在第 4 次就可以定位到第 14 行包含的缺陷;如果采取一次定位多缺陷方法,则至多在第 3 次就可以同时得到两个缺陷的位置,这将有效提高多缺陷定位的准确性,也在一定程度上减少了人工参与.基于上述观察我们认为,GAMFal 方法在保证单缺陷定位效果的基础上,能够有效地解决多缺陷定位问题,因此更加符合实际开发中对缺陷定位的需求.

Table 1 Coverage matrix and sorted result of function *findsecond*

表 1 函数 *findSecond* 的缺陷覆盖矩阵及排序结果

可执行代码行	*t*1	*t*2	*t*3	*t*4	*t*5	*t*6	*t*7	*t*8	*t*9	…	Tarantula	Ochai	Tarantula*	GAMFal
3	1	1	1	1	1	1	1	1	1	…	9	6	3	9
4	1	1	1	1	1	1	0	0	1	…	3	4	4	14
5	1	1	1	1	1	1	0	0	1	…	4	5	5	10
7	0	0	0	0	0	0	1	1	0	…	13	12	12	11
8	0	0	0	0	0	0	1	1	0	…	14	13	13	16
10	1	1	1	1	1	1	1	1	1	…	10	7	6	8
11	1	1	0	1	0	0	1	1	0	…	15	15	15	13
12	1	1	0	1	0	0	1	1	0	…	16	16	16	15
13	0	0	1	0	1	1	0	0	1	…	5	1	1	3
14	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	…	**6**	**10**	**10**	**1**
15	0	0	0	0	0	0	0	0	1	…	7	11	11	4
17	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	…	**1**	**3**	**8**	**2**
19	1	1	1	1	1	1	1	1	1	…	11	8	7	7
20	0	1	0	0	0	0	0	1	0	…	12	14	14	6
21	1	0	1	1	1	1	1	0	1	…	8	9	9	17
22	1	0	1	0	0	0	1	0	1	…	17	17	17	12
24	0	0	0	1	1	1	0	0	0	…	2	2	2	5
执行结果	0	0	0	0	1	1	0	0	0	…				

1.2 基于搜索的软件工程和遗传算法

Harman 和 Jones 在 2001 年首次提出基于搜索的软件工程(search based software engineering,简称 SBSE)概念.该领域尝试采用元启发式搜索技术,以合理的计算开销求解一类可转化为组合优化问题的软件工程问题[15].2012 年 Harman 等人总结了该领域 10 多年来的发展并提出新的趋势、技术和应用[16].李征等人也在“2013~2014 年度中国计算机科学年度报告”中对“基于搜索的软件工程”进行了全面的综述[17].随着计算能力的增强,越来越多的启发式搜索方法被应用到软件工程中来,为软件工程问题的求解提供了一个新的思路.经过 10 几年的发展,基于搜索的软件工程研究取得了显著的成绩,已成为软件工程研究领域内的一个活跃方向,且在软件测试领域有很好的应用,经统计,有超过 50%的论文关注的是软件测试与挑错方向[17].

基于搜索的软件工程广泛应用于软件测试领域,成功地解决了功能测试[18]、执行时间测试[19]、集成测试[20]、压力测试[21]、变异测试[22]、组合测试[23]、回归测试[24]等各种测试的问题.其中,测试数据的生成应用最为广泛,McMinn[25]和 Ali[26]等人曾对基于搜索的测试数据生成技术进行了总结.

基于搜索的软件工程在软件缺陷定位和缺陷自动修复领域也有很多应用.研究者根据缺陷自动修复中的量变和质变现象,利用面向自动修复的缺陷定位,提高了定位的准确度[27,28].研究者还通过构造面向缺陷定位的轻量级近似动态后向切片算法,提出程序逻辑与统计相结合的软件缺陷定位方法 SSFL,提高了定位准确度[29,30].

软件缺陷定位研究是当前软件工程研究领域中的一个活跃分支,本文将多缺陷定位问题建模为组合优化问题,并尝试用遗传算法这一主流的元启发式搜索技术进行求解.接下来我们对遗传算法的相关知识进行简介.

遗传算法是一种模拟自然选择进化过程(即适者生存)的搜索算法[31],其使用选择算子、交叉算子以及变异算子等对原始种群进行进化和筛选,经过数轮迭代得到可能的最优解.遗传算法最初由美国密歇根大学 Holland 教授于 1975 年提出[32],解决了优化模型中的两个重要问题:优化的方向和新的更优解生成方法.下面依次给出遗传算法的相关定义[31].

定义 8(染色体编码). 染色体编码是把一个问题的可行解从其解空间转换到遗传算法所能处理的搜索空间的转换方法.在遗传算法中,Holland 建议采取二进制串的编码方式.

定义 9(适应度值函数). 适应度值用于评价个体的优劣程度,一般来说,适应度值越大,则个体越好;反之,适应度越小,则个体越差.针对具体问题可以选择与问题相适宜的适应度值函数,作为个体评价的标准.

定义 10(遗传操作算子). 遗传算法的操作算子包括选择、交叉和变异 3 种基本形式,构成了遗传算法高效搜索能力的基础,是模拟自然选择和遗传过程中发生的繁殖、杂交和突变现象的主要载体[31].选择操作体现适者生存的原理,通过适应度值选择优质个体而抛弃劣质个体;交叉操作通过交换个体之间的遗传信息从而产生新的个体;变异算子通过对部分基因位信息的突变而得到尚未被开发的遗传信息,以防止个体在形成最优解过程中过早收敛.

遗传算法的流程大致描述为:首先初始化一个指定规模的种群,种群中每个染色体随机生成;然后,算法将根据每个个体的适应度值采取选择、交叉、变异等算子对种群进行演化迭代,直至满足特定终止条件(终止条件通常为达到指定迭代次数或最优适应度值变化低于某个阈值),并返回迄今为止已找到的最优解.算法的关键是染色体的编码方式、适应度值函数的设定以及选择、交叉、变异等遗传操作算子的设定.

1.3 缺陷定位其他研究成果

SFL 方法是一种动态的缺陷定位技术,它利用大量测试用例的执行结果和程序的覆盖信息,通过统计学的方法得到代码可疑值的排序.研究者们提出很多不同的统计学公式[3−7]来计算代码的可疑值排序,这些公式在不同的评测程序上各有优劣,研究者们在此基础上不断进行优化研究.

有研究人员利用谓词将程序划分成块,通过统计谓词在成功测试用例和失败测试用例中的执行情况来进行缺陷定位.Liblit 等人[33]提出 CBI 方法来寻找与程序缺陷最相关的谓词,通过谓词在成功测试用例和失败测试用例执行过程中的取值信息计算谓词的可疑值.随后,Liu 等人[34]提出了 SOBER 方法,该方法通过搜集谓词的执

行次数,比较谓词执行结果在成功测试用例和失败测试用例上的分布情况计算谓词的可疑值.并且实验结果表明,SOBER 方法要优于 Liblit 方法.郑征等人提出一种基于谓词执行序列的缺陷定位方法[35],通过搜集更多的谓词执行信息能够进一步优化缺陷定位的效果.随后,他们又对此方法进行补充,提出一种自适应的缺陷定位方法[36],该方法能够动态地选择每个谓词需要搜集的信息强度.

近年来,不断有新的方法被引入该研究领域,尤其是机器学习和基于搜索的方法[37].Yoo 提出一种通过遗传算法自动生成 SBFL 公式的方法,并可得到与人工设计公式相近的定位效果[38].谢晓园等人使用基于搜索的方法从 30 个不同怀疑率计算公式中,发现了 4 个最优的公式[39].玄跻峰等人使用机器学习的方法将多个排序指标结合使用,根据不同的程序选择最优公式[40].

研究者还从测试用例维护和程序分析角度进一步提高缺陷定位效果,充分考虑测试用例及程序本身的特性对缺陷定位结果的影响,并以此优化缺陷定位效果.郝丹等人从测试用例出发,通过消除相似测试用例[41]和有害测试用例[42]来优化测试用例集,从而优化缺陷定位的结果.贺韬等人使用变异分析来降低偶然因素的影响,并以此提高缺陷定位的效果[43].Masri 通过分析程序中的信息流来辅助缺陷定位,在分析过程中考虑程序间的依赖关系[44].张震宇等人加入布尔表达式求值中的短路现象处理来提高缺陷定位效果[45,46].此外,概率图模型[47−49]和程序切片技术[50,51]也被引入到程序缺陷定位技术.

多缺陷定位问题是缺陷定位技术实际应用必须要解决的问题.Jones 等人通过分析提出缺陷定位效果与缺陷数量成反比关系,缺陷数量越多,缺陷定位方法得到的效果越差[3].而 DiGiuseppe 等人通过大量实验发现,缺陷数量对缺陷定位效果的影响可以忽略不计,他们同时也对缺陷间的干扰现象进行了深入研究,并将失败测试用例聚类,使得同一类中的测试用例只与一个缺陷相关[14].Abreu 等人提出一种基于贝叶斯的多缺陷定位方法,它能预测程序实体中包含缺陷的概率[52].文万志等人提出一种基于切片技术的多缺陷定位方法,该方法通过程序切片降低了不同缺陷之间的相互影响[53].

2 GAMFal 方法框架

我们提出的 GAMFal 方法包括两个阶段:基于遗传算法进行最优多缺陷候选分布种群的搜索阶段和根据最优种群进行多缺陷定位阶段.其中遗传算法搜索最优种群阶段的任务是对问题进行建模并通过遗传算法搜索出最优种群,这一阶段需要重点研究染色体编码方式和适应度值函数的设定、初始种群的生成方式、遗传算子的设计及执行终止条件的定义等,最终生成最优解种群;多缺陷定位阶段的任务是将第 1 阶段所得最优种群中的候选解映射到程序实体的真实位置,从而辅助开发人员完成多缺陷的定位.

2.1 基本思想和整体流程

传统的 SFL 算法主要是基于每个程序实体被通过或失败测试用例数的覆盖情况来计算出其含有缺陷的可疑值.GAMFal 算法的思想则是先假设某些程序实体有错,然后根据测试用例集的运行结果来评价这些假设并通过遗传算法进行迭代进化,最终得到最优种群,并以此为依据进行多缺陷定位.GAMFal 方法首先构建一个长度为程序实体(代码行、代码块)总数的二进制向量,该向量表示可能的缺陷位置分布,其中,1 表示该位置所对应的程序实体有错,0 表示该位置所对应的程序实体无错;这种情况下所有可能的解有 2^n 个,我们可以依据所有测试用例对每一个解进行评价,缺陷定位的目标就是去寻找最优解,以得到最可能的缺陷位置.这便将传统的软件缺陷定位问题转化为一类搜索问题,可用常见的搜索方法来求解,但在求解之前需要解决如下 3 个问题.

(1) 如何评价一个候选解和最优解(真实多缺陷位置)的接近程度,即适应度值函数的设置问题.

(2) 在给定适应度值函数的情况下如何搜索以得到最优解.

(3) 如何根据最优解再去映射返回得到缺陷程序实体最可能的位置.

对于第 1 个问题,本文基于程序频谱信息和运行结果对 Ochiai 方法进行拓展以适应多缺陷情况,同时考虑到表 1 例子中存在的问题,对由通过测试用例经过的可能缺陷位置进行惩罚,我们提出了 Multi-Ochiai 系数,并以此作为候选缺陷分布的评价函数.对于第 2 个问题,由于整体解空间的大小为 2^n,当程序实体数 n 较大时,解空间规模将较为庞大,借助穷举法找到最优解的方案并不可行,因此我们需要采用更好的搜索算法来求解.遗传算

法是一种模拟生物进化的启发式搜索算法,它能得到一系列的最优候选解,而其他的启发式搜索算法,如爬山算法、模拟退火算法等容易收敛于一个局部最优解,因为本问题中可能存在多个最优解,所以我们选择遗传算法进行搜索.对于第 3 个问题,考虑到遗传算法是一种基于种群迭代的算法,在算法达到终止条件时将会得到一个最优种群,考虑到算法的稳定性,本文将针对得到的最优解种群确定多缺陷最终可能的位置.

GAMFal 算法的核心是 Multi-Ochiai 可疑度系数计算公式和遗传操作算子的选择,图 2 给出了整个算法的流程图,算法共分为两个阶段,第 1 阶段首先使用贪心算法对多缺陷分布的种群进行初始化;然后执行选择、交叉和变异算子,生成新的个体并添加到种群中,同时以 Multi-Ochiai 可疑度系数作为适应度值对个体进行评价并进化得到新种群;如果终止条件被满足,则将得到最终的最优多缺陷分布种群.然后进入第 2 阶段,依据最优种群中的多缺陷分布得到对应的程序实体的可疑度排序,算法结束.

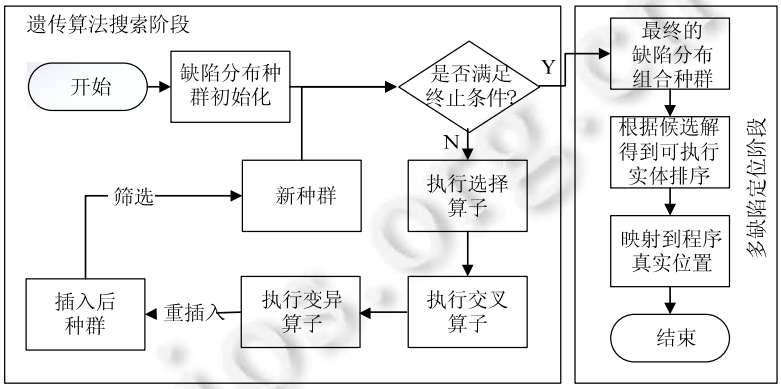

Fig.2 Flow chart of GAMFal algorithm

图 2 GAMFal 算法的流程图

2.2 遗传算法搜索最优种群阶段

本节主要介绍 GAMFal 方法的第 1 阶段计算过程,首先介绍对候选缺陷分布的编码方式,然后给出具体的适应度值函数,最后描述遗传操作算子并介绍搜索的整体流程.具体如下.

2.2.1 染色体编码方式

GAMFal 算法将候选缺陷分布[54]表示为一个二进制向量:

$$C=\{c_1,c_2,c_3,...,c_n\} \tag{4}$$

其中,n 为被测程序中可执行的程序实体数量,c_j 表示被测程序中的第 j 个程序实体是否有错,如果 c_j=1,则在该候选缺陷分布中,第 j 个程序实体被认为是存在缺陷的;如果 c_j=0,则在该候选缺陷分布中,第 j 个程序实体被认为没有缺陷.例如一个候选缺陷分布 C={0,0,1,0,0,0,0,0,1,0},表示被测程序共有 9 条可执行的程序实体,其中第 3 个和第 8 个程序实体被认为是有缺陷的,其余的则被认为是没有缺陷的.

2.2.2 适应度值函数

Tarantula 和 Ochiai 等可疑度系数计算公式通过计算测试用例的频谱信息和执行结果得到可疑度系数,进而得出每一个程序实体的可疑度值.然后根据程序实体的可疑度值从大到小进行排序,可以得到可疑度排名列表,开发人员便可以根据该列表依次检查程序,从而发现缺陷在程序中的具体位置.

Tarantula 及 Ochiai 等可疑度系数计算公式所度量的是每一条语句的可疑度值,此类方法在单缺陷的情况下表现出了很好的效果.但在实际的软件调试过程中,被测程序往往含有多个缺陷.Tarantula 和 Ochiai 所得出的可疑度排名列表在存在多缺陷的情况下能够作为参考,但是因为并没有根据多缺陷情况进行优化,因此会存在潜在的问题[3].文献[3]提出的 Multi-Ochiai 可疑度系数计算公式便是针对这一问题在 Ochiai 可疑度系数计算公式上所作的改进.

当程序中存在多个缺陷时,与单缺陷程序的假设[12]会有较大的差别,因此本文的方法基于如下假设.

假设 1. 一个缺陷分布的可疑度与该分布能解释的失败用例数成正比.

假设 2. 一个缺陷分布的可疑度与该分布能解释的通过用例数成反比.

假设 3. 在设定的可疑度公式中,假设 1 所占的比重要更大一些.

Multi-Ochiai 可疑度系数计算公式度量候选缺陷分布的可疑度值.对于一个候选缺陷分布 C,其可疑度值(即该候选缺陷分布为真实缺陷分布的概率)可由如下公式计算出来.

$$MultiOchiai(C)=\frac{\phi(C)}{\sqrt{|T_F\times(\phi(C)+P(C))|}} \tag{5}$$

其中,$\phi(C)$为候选缺陷分布 C 解释失败测试用例的能力,其定义为

$$\phi(C)=\sum_{M_i\in M_F,1\leqslant i\leqslant|T_F|}\lambda\left(\sum_{1\leqslant j\leqslant n}C_j\times M_{ij}\right) \tag{6}$$

公式(6)中的函数$\lambda(x)$被称为示性函数,定义为

$$\lambda(x)=\begin{cases}1, & \text{if}(x>0)\\ 0, & \text{if}(x=0)\end{cases} \tag{7}$$

公式(5)中的 $P(C)$定义为

$$P(C)=\sum_{M_i\in M_P,1\leqslant i\leqslant|T_P|}\left(\sum_{1\leqslant j\leqslant n}C_j\times M_{ij}\right) \tag{8}$$

公式(6)和公式(8)中的 M_{ij} 为定义 3 中覆盖矩阵的元素,$|T_F|$为失败测试用例的数量.与假设 1 对应,公式(5)和公式(6)中的$\phi(C)$表达了候选缺陷分布 C"解释"失败测试用例的能力;即如果一个失败测试用例运行了一个或者多个在 C 中被假定为有错的语句,则该失败测试用例能够被 C 解释.反之,如果该失败测试用例运行到的语句在 C 中都被认为是无错的,则该失败测试用例不能被 C 解释.如公式(6)所示, $\phi(C)$的值为候选缺陷分布 C 能够解释的失败测试用例的数量; $\phi(C)$的值加 1,当且仅当 C 能够解释一个失败的测试用例.

公式(8)中的$|T_P|$是通过测试用例的数量.$P(C)$是一个惩罚函数,与假设 2 相对应,如果一个候选缺陷分布 C 所假设为有错的程序实体被通过的测试用例执行到,则 $P(C)$的值加 1.需要说明的是,如果 C 能解释一个失败的测试用例,则$\phi(C)$的值加 1;而 $P(C)$中并未引入示性函数,则只要 C 中有错的语句被通过的测试用例执行一次,则 $P(C)$的值为 1.这样做的目的是让候选缺陷分布C中被认为有错的语句数量保持在一个比较低的水平,这与实际调试时程序的缺陷数量一致.如果一个候选缺陷分布 C 中被认为有错的语句数量较多,则其 $P(C)$值会更大,从而降低了该候选缺陷分布的可疑度值.反之,如果不进行惩罚,则可能出现极端情况,即包含缺陷实体数量最多的候选分布的 Multi-Ochiai 可疑度值最高,这样的结论将推出所有程序实体都是有缺陷的,显然无法有效地辅助开发人员定位缺陷.因此,GAMFal 方法使用惩罚函数以降低那些过多估计缺陷数量候选分布的适应度值,这样既可以进一步优化 GAMFal 算法的执行结果,又能在一定程度上提高算法的执行效率.在 GAMFal 方法中,对于一个给定的缺陷分布,其 Multi-Ochiai 可疑度值越高,则表示该分布解释失败测试用例的能力越强,解释正确测试用例的能力越弱.

2.2.3 遗传操作算子及最优种群搜索过程

如上节所述,Multi-Ochiai 可疑度系数计算公式可计算出每一个候选缺陷分布的可疑度值.本节所述方法就是对候选缺陷分布进行搜索,找出其中可疑度值较大的种群,最终得到程序实体的可疑度排名列表.由于候选解的空间较大且无法用解析的方法求解该问题,本文将采用遗传算法来对该问题进行求解.

遗传算法由可行候选解集的一个初始种群开始,种群由经过基因编码的一定数目个体组成,而每个个体是染色体带有一定特征的实体.通常采用简化的编码方法,即二进制编码对原始问题中的表现型进行编码,使之成为基因型的个体.初始种群产生后,按照适者生存和优胜劣汰的原理,逐代地演化以产生新的个体,并保留更好的个体进入下一代.在每一代中,根据个体的适应度值来选择个体,进行交叉和变异,从而产生新的个体.这一过程迭代进行,直到达到终止条件,终止条件一般为预设的迭代次数,或者是种群中的个体的适应度达到一个预设的值等等.最终代种群中的最优个体经过解码,作为问题的近似最优解.

在 GAMFal 算法中,使用长度为 n 的二进制向量 C 来表示一个候选缺陷分布,n 为被测程序的可执行语句数量.二进制向量C即可作为遗传算法中的一个个体.而 Multi-Ochiai 可疑度系数计算公式即作为遗传算法中的

适应度函数,即 Multi-Ochiai 值可作为在遗传算法中度量候选缺陷分布 C 的标准.GAMFal 方法中使用遗传算法搜索高可疑度值候选缺陷分布,表 2 是 GAMFal 算法的伪代码.

Table 2 Pseudo code of GAMFal algorithm

表 2 GAMFal 算法伪代码

算法. GAMFal.
输入: 覆盖矩阵 M,结果向量 R;
输出: 近似最优解种群 P.

```
1.  pop←AG(M,R,N_p)
2.  CandidatePool←pop
3.  i←0
4.  Repeat
5.      i←i+1
6.      C_selected←select(pop,s_mo(pop),GGAP)
7.      C_crossovered←crossover(C_selected,P_c)
8.      C_mutated←∅
9.      for all C∈C_crossovered
10.         if sum (C)≤Threshold_m
11.             mutate_1(C,P_m1)
12.         else
13.             mutate_2(C,P_m2)
14.         end if
15.         C_mutated←C_mutated∪C
16.     end for
17.     reinsert(pop,s_mo(pop),C_mutated,s_mo(C_mutated))
18.     CandidatePool←CandidatePool∪C_mutated
19. until i=N_gen
20. P←screen(CandidatePool)
21. return P
```

算法的详细过程如下.

(1) 初始化种群(如图 3 所示)

$$\begin{array}{cccccccc} unit1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ unit2 & 0 & 1 & 0 & \cdots & 0 & 0 \\ unit3 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ unit(n-1) & 0 & 0 & 0 & \cdots & 1 & 0 \\ unit(n) & 0 & 0 & 0 & \cdots & 0 & 1 \end{array}$$

Fig.3 The initial population generated by AG

图 3 AG 过程初始化生成的 n 个个体

在该方法中,使用一个贪心算法过程 Additional-Greedy(AG)[55]来生成初始候选解种群,该算法能在满足所有个体符合条件的情况下,尽可能地保证初始种群的多样性,算法的流程图如图 4 所示.AG 过程的输入包括覆盖矩阵 M、测试用例结果向量 R 以及遗传算法种群中包含的个体数量 N_p.AG 过程首先生成 n 个个体,每一个个体(即候选缺陷分布)均只有一条语句被认为有错,即长度为 n 的二进制串中只有一个位置为 1,其他为 0,且个体之间 1 的位置互异,如图 3 所示.计算每一个个体的$\phi(C)$值,如果$\phi(C)=|T_F|$,则该候选缺陷分布 C 能够解释所有的失败测试用例,是本问题的一个可行解.如果$\phi(C)\neq|T_F|$,则需要对该个体进行修正,即在该候选缺陷分布中加入语句 e_x(即把缺陷分布组合中 e_x 对应的位置置 1),使得该候选缺陷分布能够解释所有的失败的测试用例.e_x 需要能够解释最多的原候选缺陷分布不能解释的失败测试用例;也就是说,在 e_x 被加入到候选缺陷分布 C 后,$\phi(C)$增加的幅度最大;e_x 加入后,C 如果能够解释所有的失败测试用例,则该过程结束,若不能,则继续加入语句,直到该个体能够解释所有失败测试用例为止.至此,AG 过程获得了一个有 n 个可行个体的种群.如果 $n<N_p$,则复制数个可行个体直至将初始种群中个体数量补充至 N_p;如果 $n>N_p$,则选择其中 Multi-Ochiai 可疑度值最大,即适应度值最高的 N_p 个个体作为初始种群.AG 过程生成的 N_p 个个体的初始种群具有相对较高的适应度值,且具有较好的多样性,因为每个个体都是由具有不同缺陷位置的候选缺陷分布扩展而来.高质量的初始种群有利于遗传算法最终得到较好的结果.

初始化得到的种群将依次使用选择、交叉和变异算子来产生新的个体.新个体再重插入到种群中,开始新一轮的选择、交叉和变异.

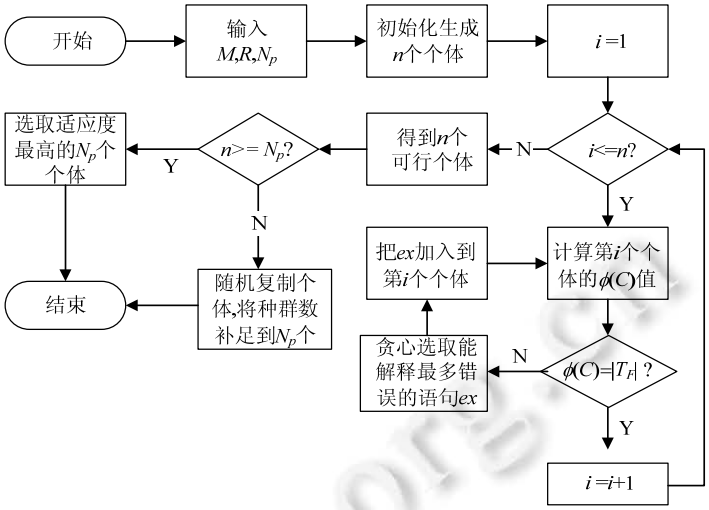

Fig.4 Flow chart of AG process

图 4 AG 过程的流程图

(2) 选择算子

选择算子用来确定重组或交叉的个体,以及被选个体将产生多少个子代个体.选择算子是一个基于适应度值的操作,适应度较高的个体被选择的可能性更大.本方法中的适应度函数为 Multi-Ochiai 可疑度系数.参数 $GGAP$($0<GGAP\leqslant 1$)用以确定种群中被选择的个体所占的比例.选择算子有很多种,本方法所使用的是轮盘赌选择算子.轮盘赌选择算子类似于博彩游戏中的轮盘赌,个体适应度按比例转化为选中的概率,将轮盘分为 N_p 个扇区,进行选择的次数是 $GGAP\times N_p$ 次,所以需要产生 $GGAP\times N_p$ 个(0,1]之间的随机数,也相当于转动 $GGAP\times N_p$ 次转盘.随机数所落区间所代表的个体被选中,在轮盘上直观地显示为转动停止时被指针指向的扇区所代表的个体被选中.以 N_{nc} 表示当前次选择时种群内还未被选中的个体数量,C_l 表示某一未被选中的个体,则其中个体 C 在该次选择中被选中的概率可由下式计算:

$$P(C)=\frac{MultiOchiai(C)}{\sum_{l=1}^{N_{nc}} MultiOchiai(C)} \tag{9}$$

(3) 交叉算子

交叉算子在两个已有染色体的基础上进行操作,产生新的染色体.常见的交叉算子包括单点交叉、多点交叉、洗牌交叉等等.参数 P_C 指定两个染色体上相同位置的基因进行交叉的概率.在本方法中,使用的交叉算子是缩小代理(reduced surrogate)的洗牌交叉算子.在洗牌交叉算子中,进行交叉的染色体的每一对相同位置的基因都有 P_C 的概率进行交换;而缩小代理是指交叉后的染色体需与父辈染色体不同.例如,在选择算子中有两个个体被选择,作为父辈个体进行交叉.

$$C_{old1}=\langle 0,1,0,1,0,0,0,1,0,0\rangle$$
$$C_{old2}=\langle 1,0,1,0,0,0,0,0,1,1\rangle$$

经过交叉算子运算后,得到的两个子代的个体为

$$C_{new1}=\langle 1,1,1,0,0,0,0,0,1,0\rangle$$
$$C_{new2}=\langle 0,0,0,1,0,0,0,1,0,1\rangle$$

可见在交叉前后,两个个体共有 5 个位置的基因得到交换,从而产生了两个不同于父辈个体的新个体.需要说明的是,有些如本例中的第 5 个基因位置,也有可能得到了交叉,但因为在父辈个体中,该位置均为 0,因此并没有在子代中产生新的基因型.

(4) 变异算子

在交叉算子运算中,父代的个体经过交叉得到新的基因型以组成新的子代个体.但如果在某个基因位置上父代的基因型均相同,则子代就不可能产生新的基因型.因此,算法将引入变异算子以确保种群的多样性.参数 P_m 用以指定由交叉算子产生的子代个体的每一个基因位变异的概率,即从 0 变异为 1 或从 1 变异为 0 的概率,$0<P_m<1$.P_m 的取值通常很小,因为变异算子不能大幅度改变群体的基因型.通常在缺陷定位问题中,假设程序中的缺陷数量是很少的,因此需要限制变异算子以免其无限制地增加候选缺陷分布中的"1".本方法引入参数 $Threshold_m$ 来保证候选分布中的缺陷数量维持在比较低的水平,通常 $Threshold_m$ 是一个比较小的整数.同时,本方法使用了两个参数 P_{m1} 和 P_{m2} 来指定变异概率,$0<P_{m1}<P_{m2}<1$.在进行变异之前,计算由交叉算子获得的个体中"1"的数量.如果该数量小于或等于 $Threshold_m$,则该个体中的每一个 0 以 P_{m1} 概率变异为 1;如果该数量大于 $Threshold_m$,则该个体中的每一个 1 以 P_{m2} 的概率变异为 0.

(5) 重插入

由选择算子、交叉算子及变异算子运算后得到的新个体需要被重新插入原种群中,形成下一代种群.此时需要计算原种群中的个体以及新生成个体的适应度值,选择其中适应度值最高的 N_p 个个体组成新种群.因此,原种群中适应度值较差的个体将被新生成的适应度值较高的个体替换.每一代所生成的最优缺陷分布均被保留至候选缺陷分布种群中,以作为得到程序实体排名列表的依据.

在本方法中,参数 N_{gen} 为最大迭代次数,即终止条件.迭代第(2)~第(5)步,即循环进行选择算子、交叉算子、变异算子及重插入过程.直到循环次数达到 N_{gen}.当过程停止时,得到的最优候选缺陷分布种群将作为第 2 阶段多缺陷定位的依据.

由遗传算法获得的候选缺陷分布种群中的个体有可能并不能完全解释所有的失败测试用例,因此需要一个筛选的过程.筛选过程即计算种群中每一个个体的$\phi(C)$值,保留$\phi(C)$值等于$|T_F|$的个体,抛弃其他个体.此时的候选缺陷分布种群中的个体均能够完全解释所有失败测试用例.然后将种群中的个体按其适应度值由大到小进行排序,得到候选缺陷分布的可疑度值列表.

2.3 确定多缺陷位置阶段

本阶段将最优候选缺陷分布种群转化为语句的可疑度排名列表.在可疑度值较高的候选缺陷分布中被认为有错的语句的可疑度应该更高.如果在同一候选缺陷分布中有数条语句,则需要参考种群内其他个体包含的缺陷位置,如果完全相同,则按随机的顺序进行排列.例如,以下是一个已排序的候选缺陷分布种群:

$\langle 0,0,0,1,0,1,0,0,0,0\rangle$

$\langle 0,1,0,1,0,0,0,0,0,0\rangle$

$\langle 0,1,0,0,0,1,0,0,0,0\rangle$

$\langle 1,0,1,0,0,0,1,1,0,1\rangle$

$\langle 1,0,0,0,1,0,0,1,1,0\rangle$

按从上至下的顺序,其可疑度(适应度)依次降低.因此,在第 1 个个体(候选缺陷分布)中被认为有错的语句的可疑度最高,即第 4 条和第 6 条语句可疑度最高,这两条语句之间的排序可参考最优种群内其他个体包含的缺陷位置进行排序,如果完全相同,则随机进行排序.因此,由这个候选缺陷分布种群可得语句的可疑度排名列表为$\langle e_4,e_6,e_2,e_1,e_8,e_3,e_7,e_{10},e_5,e_9\rangle$.由于对处于同一候选缺陷分布中的语句采用了随机排序的策略,因此由同一候选缺陷分布种群得出的语句可疑度排名列表有可能不同.

3 实证研究

本节我们将借助真实程序以评估我们所提出的 GAMFal 方法在多缺陷定位问题中的有效性.首先,我们确定了实验设计并针对本次实证研究设计了 3 个需要回答的研究问题;其次,我们列出了实证研究所用到的真实数据集,包括程序、缺陷、测试用例集及运行结果;在描述实验结果之前,我们拓展定义了多缺陷定位方法的评价标准 $EXAM_F$ 和 $EXAM_L$,然后对实证研究结果进行总结,借助显著性检验与已有的经典 SFL 方法进行比较,并

验证 GAMFal 的有效性.

3.1 实验设计和研究问题

为了评价本文提出的算法框架,我们采用 Matlab 实现了文中的算法框架.在执行算法框架之前,我们采用两种方式进行数据采集:(1) 所有版本的 Siemens 程序是在配置为 3.10GHz Intel Core i5-2400 CPU、4GB 物理内存,安装了 SunOs 5.10 操作系统的 Dell 计算机上执行;(2) Linux 的 3 个程序是在配置为 3.00GHz Intel(R) Xeon(R) E5-2623 v3 CPU、32GB 物理内存,安装了 CentOs 7.0 操作系统的服务器上运行.GAMFal 和其他缺陷定位方法是在配置为 3.40GHz Intel Core i7-3770 CPU、8GB 物理内存、Microsoft Windows 7 64-bits 操作系统的 Dell 计算机上执行.Gcov 工具用来收集可执行语句的覆盖信息,它能得到每一条可执行语句在测试用例执行过程中的执行信息.随后使用 Python 脚本从 gcov 执行结果中生成程序的覆盖矩阵.

实证研究的主要目标包括两个方面,首先基于评测标准 $EXAM_F$ 和 $EXAM_L$,本文提出的 GAMFal 方法是否有效改善现有 SFL 方法在多缺陷定位问题的性能?另一方面是考虑方法的效率问题,对于真实的缺陷定位问题,GAMFal 在执行时间方面是否是可接受的?基于上述两个目标,我们设计实验以解决如下 3 个研究问题.

RQ1. 在给定的评价标准 $EXAM_F$ 和 $EXAM_L$ 下,GAMFal 算法在多(单)缺陷定位问题的效果是否优于现有的 SFL 方法?

RQ2. GAMFal 算法是否可以需要较少的人工参与?

RQ3. GAMFal 算法在算法效率上是否可行?

3.2 评测数据集

我们的实证研究同时使用了小规模程序和大规模程序,这些程序都可以从 SIR 库[56]中下载.其中,(1) 小规模程序来自西门子套件中的 7 个程序,包括 print_tokens、print_tokens2、replace、schedule、schedule2、tcas、tot_info 等.这些程序最少有 174 行,最多有 539 行,其中一半以上是可执行语句.每个程序都有一个正确版本和多个错误版本,其中每个错误版本内仅含有一个缺陷.这些程序配套的测试用例集中,最少的有 1 052 个测试用例,最多的有 5 542 个测试用例.(2) 大规模程序来自 Linux 程序,分别是 gzip、grep 和 sed.这 3 个程序分别有 6 576 行、12 635 行和 7 125 行,其中可执行语句大约占 1/4.但这些程序配套的测试用例集规模较小,最少的有 213 个,最多的有 470 个.

Siemens 程序的错误版本中有些缺陷所在行不是可执行的,这超出了 SFL 方法的适用范围,很难得到准确的结果.因此实验中的有些缺陷需要重新植入.我们在 Siemens 的 7 个程序中植入单个缺陷进行单缺陷的评估实验,在 print_tokens、print_tokens2、replace 和 tot_info 这 4 个程序中植入多个缺陷来完成多缺陷的评估实验.选用这 4 个程序植入多个缺陷是因为它们包含更多的可执行代码行数,以便植入更多的缺陷组合.3 个 Linux 源程序中已有的植入缺陷都是在可执行代码行,所以我们在其单缺陷版本的基础上将缺陷进行不同组合,生成一批包含两个缺陷和 3 个缺陷的版本.实证研究中评测程序的具体信息见表 3.

Table 3 The subject programs used in empirical studies

表 3 实证研究中考虑的评测程序

被测程序	单缺陷版本数(多缺陷版本数)	所有代码行	可执行代码行	测试用例数
print_tokens	52(32)	539	203	4 130
print_tokens2	54(34)	489	201	4 115
replace	66(45)	507	273	5 542
schedule	13	397	166	2 650
schedule2	15	299	146	2 710
tcas	18	174	73	1 608
tot_info	79(48)	398	138	1 052
gzip	45(29)	6 576	1 744	213
grep	24(15)	12 635	3 197	470
sed	6(3)	7 125	2 027	360

实验中的参数设置见表 4,其中 N_{gen} 表示遗传算法最大迭代次数,N_p 表示初始种群及每次迭代待选种群的个

体数,$GGAP$ 表示每次迭代被选择个体占待选个体的比例,P_c 表示两个染色体上相同位置的基因进行交叉的概率,P_{m1} 和 P_{m2} 表示个体中染色体变异的概率,$Threshold_m$ 表示个体中的缺陷数量上限.

Table 4 Parameters setting

表 4 实验中参数设定

N_{gen}	N_p	$GGAP$	P_c	P_{m1}	P_{m2}	$Threshold_m$
300	500	0.4	0.7	0.001	0.2	4

3.3 评测指标

在单缺陷定位中,一般借助 $EXAM$ 指标[57]对方法的有效性进行评估,该指标返回的是检测到缺陷语句前需要检查的语句占所有语句的百分比.对于指定的被测程序,$EXAM$ 值越小,表示该方法的缺陷定位效果越好(需要说明的是,有相同怀疑度的实体在检查时的排序是随机的).而在多缺陷定位中,针对单缺陷的 $EXAM$ 指标并不完全适用.因此我们拓展 $EXAM$ 的定义使之适用于多缺陷问题定位方法的评价.

定义 11($EXAM_F$ 指标). 该指标返回的是检测到第 1 个缺陷前需要检查的语句占所有语句的百分比.

定义 12($EXAM_L$ 指标). 该指标返回的是检测到最后 1 个缺陷前需要检查的语句占所有语句的百分比.

在多缺陷定位时,开发人员通常有两种检查代码寻找缺陷的策略.第 1 种是找到立刻解决,开发人员根据可疑度排序找到第 1 个缺陷的位置后便停止搜索并修复缺陷.之后重新运行算法得到新的可疑度排序,直到程序中没有缺陷为止.第 2 种是一次性找出所有的缺陷并修改,开发人员预先估计程序中可能的缺陷数量,找到最后一个缺陷,最后再进行修复.值得注意的是,在多缺陷查找过程中,应设定一个查找阈值,当查找百分比超过阈值时就停止查找,以免因为缺陷数量估计错误而检查很多无用代码.

$EXAM_F$ 指标在一次性只找一个缺陷的策略中更有效,因为开发人员只关心第 1 个缺陷的位置.这种情况下,缺陷定位的方法需要尽可能地减弱一个缺陷与其他缺陷的联系.也就是说,这种方法能在只有一个缺陷的程序中得到较好的可疑度排序结果,而在多个缺陷的程序中则表现不太好.相反地,$EXAM_L$ 指标在多缺陷定位中显得更为重要,它保证了所有的缺陷程序实体的可疑度值排序较为靠前.本文将同时使用 $EXAM_F$ 指标和 $EXAM_L$ 指标来评价 GAMFal 方法和其他方法在多缺陷定位程序中的效果.

3.4 结果分析

3.4.1 单缺陷版本实证评估

虽然本文主要是研究多缺陷定位问题,但为检验新方法的适用性,我们首先检验 GAMFal 在单缺陷版本的表现.本节中我们在单缺陷版本中对比 GAMFal 和其他几种 SFL 方法的定位效果,其他缺陷定位方法包括:Tarantula(Ta)、Improved Tarantula(IT)和 Ochiai(Oc).多缺陷版本的比较实验将在下一节中展示.实验中采用的参数取值见表 4.

如表 3 所示,我们共选择了 138 个 Siemens 单缺陷版本和 28 个 Linux 单缺陷版本,每种缺陷定位方法都执行 30 次取平均值来代表它的定位效果,记录下单缺陷版本的平均 $EXAM$ 值.

图 5 借助箱形图展示了 4 种方法在单缺陷程序上的执行结果,其中,图 5(a)所示为 GAMFal 在 Siemens 套件部分程序的结果,图 5(b)所示为 GAMFal 在 Linux 上部分程序的结果.如图 5(a)所示,GAMFal 与 Improved Tarantula 和 Ochiai 有比较接近的定位效果,均优于 Tarantula 方法.图 5(b)显示大规模 Linux 程序上 GAMFal 在平均定位效果和低百分比定位结果占的比例更高.根据图 5 显示的结果可以得出,GAMFal 在单缺陷程序上的定位效果不逊于另外 3 种方法,在大规模程序上的表现甚至要更好.

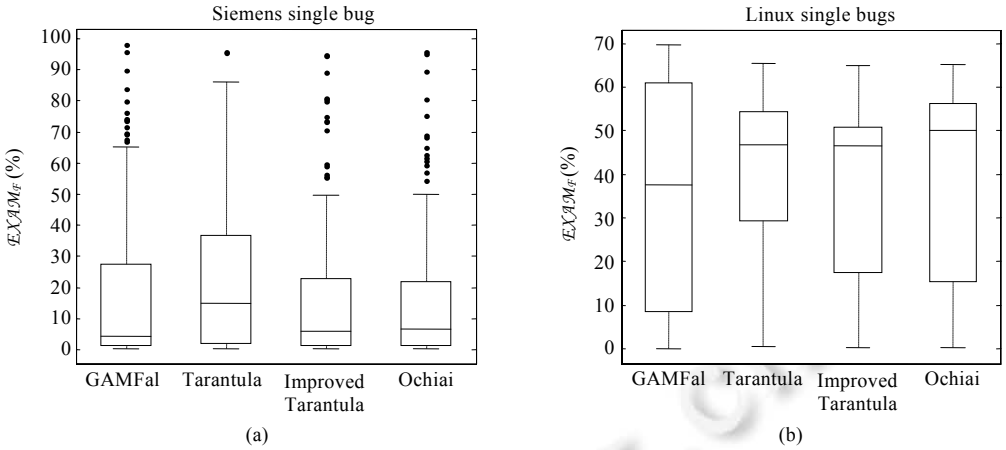

Fig.5 Result of single bug localization

图 5 单缺陷定位结果比较

3.4.2 多缺陷版本实证评估

表 3 中其他 4 个 Siemens 程序(print_tokens,print_tokens2,replace 和 tot_info)中的 159 个版本和 47 个 Linux 版本是用来评估多缺陷定位问题的.与单缺陷评估实验一样,每种缺陷定位方法都被执行 30 次取平均值来展示它的定位效果,所有缺陷版本的 $EXAM_F$ 和 $EXAM_L$ 都被记录下来.4 种方法的效果比较如图 6~图 11 所示.图 6~图 10 显示,对于 Siemens 程序,在 $EXAM_F$ 指标下,4 种方法的结果好坏不一,但总体结果非常相近;而在 $EXAM_L$ 指标下,GAMFal 方法的结果明显好于其他 3 种方法.图 11 显示,对于 Linux 的大规模程序,GAMFal 方法在两个指标 $EXAM_F$ 和 $EXAM_L$ 下的结果都要明显好于其他 3 种方法.

为了保证实验结论的可靠性,下面对实验结果进行方差检验.检验的假设是 4 种方法之间的平均定位效果没有显著差异,设置显著性水平为 0.05.我们分析多缺陷实验中的 $EXAM_F$ 和 $EXAM_L$ 指标,如图 5 所示,它们的分布是非正态的,所以我们使用一种非参数的统计检验方法 Friedman 检验[58]来检验假设的可靠性.

表 5 和表 6 分别显示了 $EXAM_F$ 和 $EXAM_L$ 的 Friedman 检验结果.F 值能根据组内平方和及组间平方和计算得到,F 值越大,则拒绝原假设的可信度越高.p 值是 F 值对应的显著性水平,表 5 和表 6 中的结果 p 值都小于 0.05,所以原假设应该被拒绝.也就是说,4 种方法的结果存在显著差异.

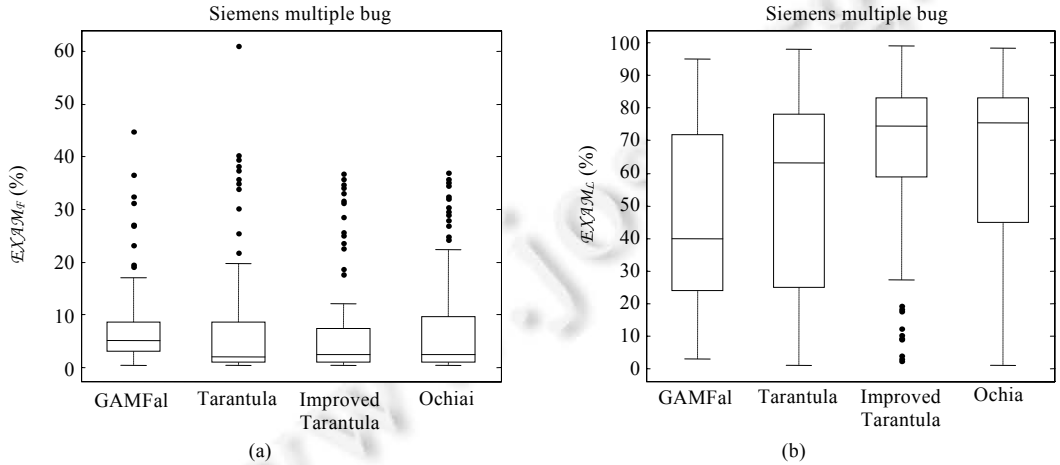

Fig.6 Multiple bugs localization result for Siemens

图 6 Siemens 程序多缺陷定位结果比较

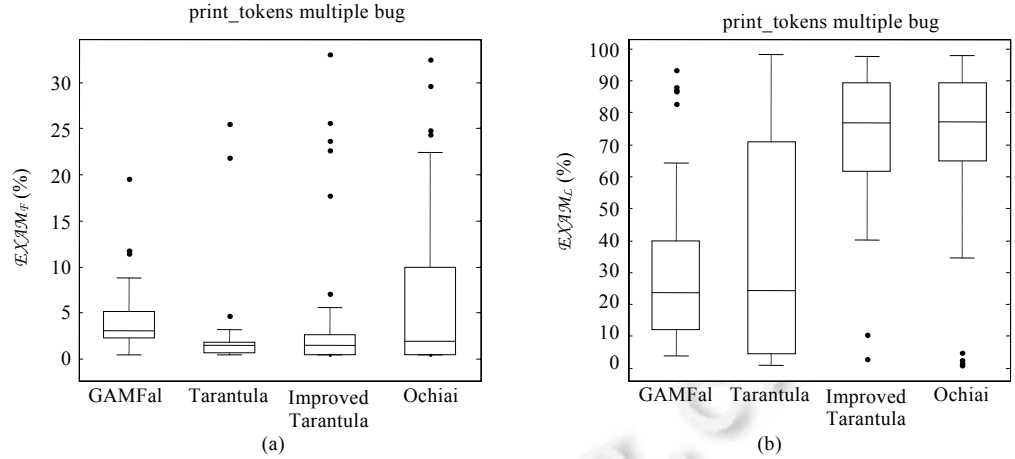

Fig.7 Multiple bugs localization result for print_tokens

图 7 print_tokens 程序多缺陷定位结果比较

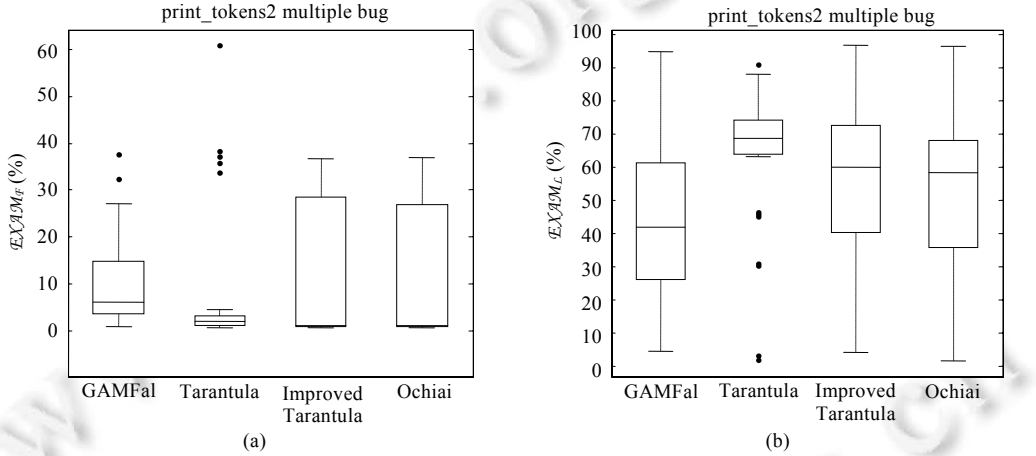

Fig.8 Multiple bugs localization result for print_tokens2

图 8 print_tokens2 程序多缺陷定位结果比较

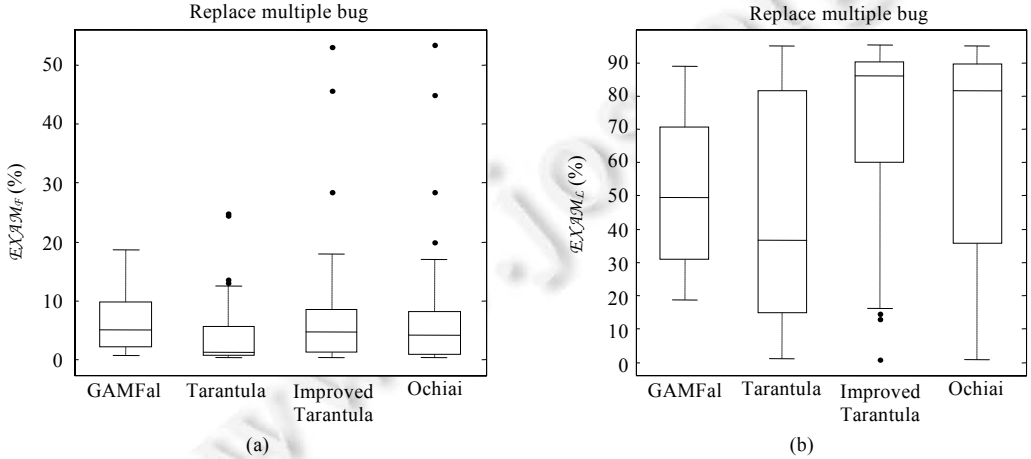

Fig.9 Multiple bugs localization result for replace

图 9 Replace 程序多缺陷定位结果比较

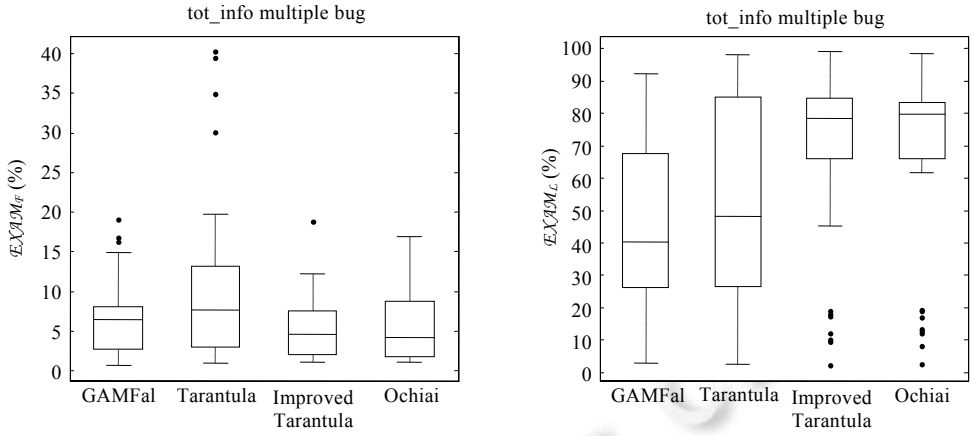

Fig.10 Multiple bugs localization result for tot_info

图 10 tot_info 程序多缺陷定位结果比较

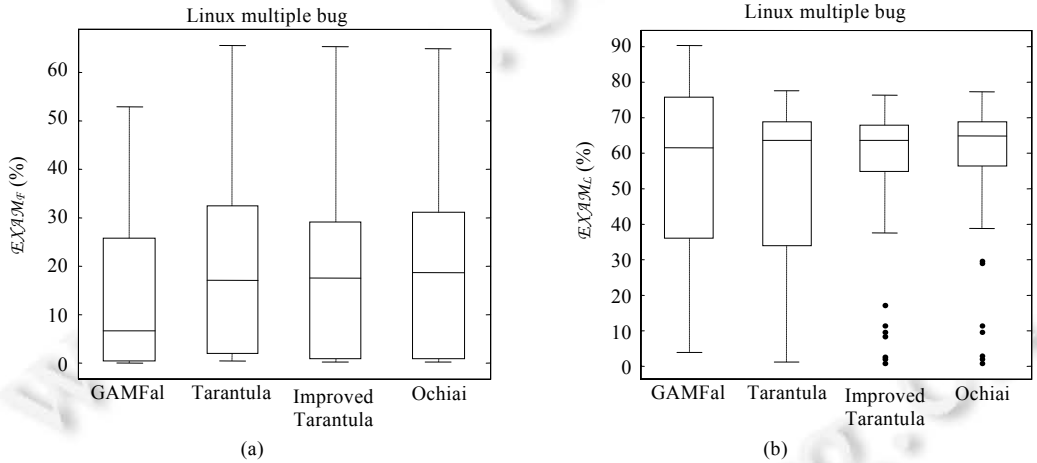

Fig.11 Multiple bugs localization result for Linux programs

图 11 Linux 程序多缺陷定位结果比较

Friedman 检验只能发现 4 种方法的结果有显著差异,但却不能发现具体的差异性.因此有必要使用最小显著差异法(least-significant difference,简称 LSD)比较任意两种方法之间的差异.表 7 和表 8 显示了使用 LSD 方法[59]分别对 $EXAM_F$ 和 $EXAM_L$ 进行检验的结果.如果置信度下限和置信度上限之间没有包含 0,则可以认为两种方法之间的差异是显著的.在表 7 中,4 种方法之间的差异是不显著的,只有 Ochiai 方法略优于其他 3 种方法.表 8 显示,GAMFal 要显著优于其他 3 种方法.Tarantula 也明显要好于 Improved Tarantula 和 Ochiai 两种方法,而 Improved Tarantula 和 Ochiai 两种方法没有明显差异.

Table 5 Friedman test for $EXAM_F$

表 5 $EXAM_F$ 的 Friedman 检验

	平方和	自由度	均方	*F* 值	*p* 值
组间误差	645 660	3	215 220	1 371	0.000
组内误差	129 870	824	157		
总和	775 530	827			

Table 6 Friedman test for $EXAM_L$

表 6 $EXAM_L$的 Friedman 检验

	平方和	自由度	均方	F 值	p 值
组间误差	610 730	3	203 577	7.11	0.000 1
组内误差	23 570 000	824	28 604		
总和	24 180 730	827			

Table 7 LSD test for $EXAM_F$

表 7 $EXAM_F$的 LSD 检验结果

方法 x	方法 y	置信下限	平均误差(y–x)	置信上限
GAMFal	Tarantula	–0.158 8	0.164 3	0.487 3
GAMFal	Improved Tarantula	–0.059 8	0.263 3	0.586 4
GAMFal	Ochiai	0.070 6	0.393 7	0.716 8
Tarantula	Improved Tarantula	–0.224 0	0.099 0	0.422 1
Tarantula	Ochiai	–0.093 6	0.229 5	0.552 5
Improved Tarantula	Ochiai	–0.192 6	0.130 4	0.453 5

Table 8 LSD test for $EXAM_L$

表 8 $EXAM_L$的 LSD 检验结果

方法 x	方法 y	置信下限	平均误差(y–x)	置信上限
GAMFal	Tarantula	–0.658 4	–0.333 3	–0.008 3
GAMFal	Improved Tarantula	–1.076 3	–0.751 2	–0.426 1
GAMFal	Ochiai	–1.037 6	–0.712 6	–0.387 5
Tarantula	Improved Tarantula	–0.742 9	–0.417 9	–0.092 8
Tarantula	Ochiai	–0.704 3	–0.379 2	–0.054 2
Improved Tarantula	Ochiai	–0.286 4	0.038 6	0.363 7

3.4.3 实验结果总结

本节将根据上述描述的实验结果讨论第 2.1 节提出的 3 个研究问题,具体如下.

(1) **RQ1**:在给定的评价标准 $EXAM_F$和 $EXAM_L$下,GAMFal 算法在多(单)缺陷定位问题的效果是否优于现有的 SFL 方法?

本节将分单缺陷和多缺陷、$EXAM_F$和 $EXAM_L$标准以及大规模程序(Linux)和小规模的程序(Siemens 套件)对此问题分别进行讨论.

根据图 5~图 11 的实验结果可知,对于单缺陷问题,在小规模 Siemens 程序集上,GAMFal 方法的定位效率不劣于其他 3 种方法;而在大规模的 Linux 程序集上要明显好于其他 3 种方法.对于多缺陷问题,在小规模 Siemens 程序集上,GAMFal 方法在 $EXAM_F$指标上与其他 3 种方法的结果近似,而在 $EXAM_L$指标上明显优于其他 3 种方法.这就说明,GAMFal 方法应用于 Siemens 程序集上时能在保证不降低找到第 1 个缺陷效率的情况下,能更快地找到最后一个缺陷,并使得包含缺陷语句的怀疑值排名整体提高,从而帮助开发人员更快地锁定缺陷.在大规模 Linux 程序上的实验结果说明,GAMFal 方法在 $EXAM_F$指标和 $EXAM_L$指标下都明显优于其他 3 种方法.这说明,在程序中存在多个缺陷时,本文提出的方法能更快地定位到第 1 个缺陷的位置,同时使得程序中包含缺陷的语句排名整体提高.

通过上一节的显著性检验可以看出,本文提出的方法在平均性能上要显著优于其他 3 种方法,特别是在多缺陷定位中.新方法在保证单缺陷定位准确性的前提下,主要针对多个缺陷的覆盖路径上有重复非缺陷程序实体的情况,因此在某些程序(如小规模的 Siemens 程序)上没有表现出明显的优化效果.而在实际的开发过程中,一方面,程序中的缺陷数无法确定,且往往超过一个.另一方面,实际程序中的代码复用量很高,这使得不同缺陷的执行路径上的重复率也很高,使用单缺陷定位方法很容易将这些复用模块的可疑值计算为最高,影响缺陷定位效果.本文提出的方法能够很好地应对这两个问题,这使得本文的方法更容易应用到实际开发中去.

(2) **RQ2**:GAMFal 算法是否可以需要较少的人工参与?

传统的缺陷定位方法大多是一次检查一个缺陷,而 GAMFal 方法在计算过程中以多缺陷分布个体为检查

基础,且根据实验结果可知,在算法终止时的最优种群中,包含多个缺陷的个体数量占总种群的95%以上,这将使得开发人员可一次检测多个缺陷,从而减少开发人员在多缺陷定位的参与,以提高多缺陷定位的效率.同时遗传算法是一种基于搜索的方法,需要人工参与的情况也较少.总体来看,GAMFal 方法将可以减少人工的参与.

(3) **RQ3**:GAMFal 算法在算法效率上是否可行?

表 9 所示为 GAMFal 算法在 7 个多缺陷程序上的执行效率.从表 9 可以看出,GAMFal 算法的执行时间与程序的可执行代码数、测试用例数及遗传算法的种群数、迭代次数等因素相关.在相同的遗传算法参数设定情况下,程序的可执行代码行数和测试用例数越多,GAMFal 算法执行时间越长.表中,gzip 和 sed 程序的可执行代码行与测试用例数的乘积小于 print_tokens 和 print_tokens2,实际执行时间却更长,这说明,可执行代码行数对算法执行效率的影响更大.另外,GAMFal 算法在包含 3 197 个程序实体和 470 个测试用例的 grep 程序上的执行时间约为 25s,这说明,GAMFal 算法在大程序上的执行效率不会大幅度下降,由于遗传算法是一种基于搜索的算法,算法执行时间相对传统运算求解的方法要长;但在缺陷定位过程中,秒级时间消耗对于开发人员排查缺陷是可以接受的,符合实际应用的效率要求.

Table 9 Effectiveness of GAMFal

表 9 GAMFal 算法的执行效率

被测程序	可执行代码行数	测试用例数	可执行代码行×测试用例数	GAMFal 算法执行时间(s)
tot_info	138	1 052	145 176	7.25
print_tokens2	201	4 115	827 115	10.88
print_tokens	203	4 130	838 390	11.12
gzip	1 744	213	371 472	11.79
sed	2 027	360	729 760	14.93
replace	273	5 542	1 512 966	18.85
grep	3 197	470	1 502 590	24.92

3.5 有效性影响因素分析

本节主要讨论可能影响实证研究有效性的一些影响因素.

首先考虑内部有效性,内部有效性主要涉及到可能影响到实验结果正确性的内部因素,主要包括 3 个方面.(1) 在遗传算法的计算过程中,更好的参数取值将有助于获得更好的结果,但如何确定更好的参数取值组合仍是遗传算法优化研究中的一个重要问题.(2) 本文仅使用遗传算法进行求解,可考虑其他的元启发式搜索算法进行最优种群求解.(3) 本文在实现算法时进行了反复的检验,以避免出现实现中的错误.在执行多缺陷定位求解之前,我们首先实现了 Tarantula、Ochiai 等传统方法在单缺陷定位的实验,得到的结果与之前研究结果相比,基本保持一致.

外部有效性主要涉及到实证研究得到的结论是否具有一般性,主要包括两个方面:(1) 遗传算法是一种随机优化算法,也是具有较高鲁棒性的一种算法[31].(2) 本文用来做实证研究的程序数据集 Siemens 套件是软件测试领域的标准数据集,而 Linux 下的 3 个程序也在之前的研究中被使用过,具有一定的代表性.在实证研究的过程中,程序均是运行 30 次取平均值,以保证实证研究结论的可信度.

结论有效性主要涉及到使用的评测指标是否合理.$EXAM$ 指标是软件缺陷定位问题的主要评测指标,计算了开发人员在使用该缺陷定位方法时需要检测多少比例的代码才能定位到真正的缺陷,体现了该缺陷定位方法辅助定位的效率;本文重点考虑了该评测指标在多缺陷情形下的定位问题,拓展了 $EXAM$ 指标至 $EXAM_F$ 和 $EXAM_L$ 指标,不仅考虑了第 1 个缺陷的定位效率,还考虑了最后一个缺陷的定位效率,从而可以全面评价方法在多缺陷问题上的定位效果.最后我们还通过 Friedman 检验和 LSD 方法对各定位方法在 $EXAM_F$ 和 $EXAM_L$ 指标上进行显著性检验,以验证结论的有效性.

4 总结和展望

本文提出了一种基于遗传算法和 Multi-Ochiai 可疑度系数的多缺陷定位方法 GAMFal,其中 Multi-Ochiai 可疑度系数是对 Ochiai 的改进,作为候选解优劣的评价标准.算法首先将多缺陷定位问题建模成一类搜索问题,

然后再在此基础上使用遗传算法搜索状态空间中的最优解,最后根据遗传算法的结果对程序实体排序以辅助开发人员查找和修改缺陷.本文的实验结果表明,GAMFal 方法在单缺陷程序中有与其他方法类似的定位效果,而在多缺陷程序中的定位效果要明显好于其他方法.同时,该方法能够有效处理开发中缺陷数量不确定和通用模块误判的情况,能够更好地满足实际开发中对缺陷定位的需求.

我们认为该方法仍有一些后续工作值得扩展,具体来说:(1) 我们将尝试将该方法应用到其他大规模程序中,分析本文结论是否具有一般性.(2) 在算法的第 1 阶段将进一步关注算法的执行效率,并尝试不同算法以期更高效、准确地搜索到最优种群.(3) 在算法的第 2 阶段将重点研究种群内个体排序对定位的影响,尝试采用不同方法研究定位问题.

References:

[1] Vessey I. Expertise in debugging computer programs: A process analysis. Int'l Journal of Man-Machine Studies, 1985,23(5): 459−494. [doi: 10.1016/S0020-7373(85)80054-7]

[2] Mayer W, Stumptner M. Evaluating models for model-based debugging. In: Proc. of the Int'l Conf. on Automated Software Engineering. L'Aquila: Springer-Verlag, 2008. 128−137. [doi: 10.1109/ASE.2008.23]

[3] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the Int'l Conf. on Software Engineering. Orlando: ACM Press, 2002. 467−477. [doi: 10.1145/581339.581397]

[4] Jones JA, Bowring JF, Harrold MJ. Debugging in parallel. In: Proc. of the Int'l Symp. on Software Testing and Analysis. London: ACM Press, 2007. 16−26. [doi: 10.1145/1273463.1273468]

[5] Rui A, Zoeteweij P, Gemund AJC. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing Academic and Industrial Conf. Practice and Research Techniques. Cumberland Lodge: Springer-Verlag, 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[6] Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pin-Point: Problem determination in large, dynamic internet services. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. Washington: IEEE Press, 2002. 595−604. [doi: 10.1109/DSN.2002.1029005]

[7] Ochiai A. Zoogeographic studies on the soleoid fishes found in Japan and its neighboring regions. Nihon-Suisan-Gakkai-Shi, 1957, 22(9):526−530. [doi: 10.2331/suisan.22.526]

[8] Naish L, Hua JL, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Trans. on Software Engineering and Methodology, 2011,20(3):563−574. [doi: 10.1145/2000791.2000795]

[9] Abreu R, Zoeteweij P, Gemund AJCV. Spectrum-Based multiple fault localization. In: Proc. of the Int'l Conf. on Automated Software Engineering. Auckland: Springer-Verlag, 2009. 88−99. [doi: 10.1109/ASE.2009.25]

[10] Steimann F, Bertschler M. A simple coverage-based locator for multiple faults. In: Proc. of the Int'l Conf. on Software Testing Verification and Validation. Denver: IEEE Press, 2009. 366−375. [doi: 10.1109/ICST.2009.24]

[11] Moon S, Kim Y, Kim M, Yoo S. Ask the mutants: Mutating faulty programs for fault localization. In: Proc. of the Int'l Conf. on Software Testing, Verification and Validation. Abano Terme: IEEE Press, 2014. 153−162. [doi: 10.1109/ICST.2014.28]

[12] 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]

[13] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011,34(8):1411−1423 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01411]

[14] DiGiuseppe N, Jones JA. On the influence of multiple faults on coverage-based fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. Toronto: ACM Press, 2011. 210−220. [doi: 10.1145/2001420.2001446]

[15] Harman M, Jones BF. Search-Based software engineering. Information and Software Technology, 2001,43(14):833−839. [doi: 10.1016/S0950-5849(01)00189-6]

[16] Harman M, Mansouri SA, Zhang Y. Search-Based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012,45(1):1−61 [doi: 10.1145/2379776.2379787]

[17] Li Z, Gong DW, Nie CH, Jiang H. The progress and development tendency of the research on search-based software engineering. 2013-2014 Annual Report of Computer Science and Technology, Beijing: China Machine Press, 2014. 139−187 (in Chinese with English abstract).

[18] Buhler O, Wegener J. Evolutionary functional testing. Computers & Operations Research, 2008,35(10):3144−3160. [doi: 10.1016/j.cor.2007.01.015]

[19] Wegener J, Sthamer H, Jones BF, Eyres DE. Testing real-time systems using genetic algorithms. Software Quality Journal, 1997, 6(2):127−135. [doi: 10.1023/A:1018551716639]

[20] Briand LC, Feng J, Labiche Y. Using genetic algorithms and coupling measures to devise optimal integration test orders. In: Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering. 2002. 43−50. [doi: 10.1145/568760.568769]

[21] Briand LC, Labiche Y, Shousha M. Stress testing real-time systems with genetic algorithms. In: Proc. of the Genetic & Evolutionary Computation Conf. Washington: ACM Press, 2005. 1021−1028. [doi: 10.1145/1068009.1068183]

[22] Masud M, Nayak A, Zaman M, Bansal N. Strategy for mutation testing using genetic algorithms. In: Proc. of the Canadian Conf. on Electrical and Computer Engineering. Madrid: IEEE Press, 2005. 1049−1052. [doi: 10.1109/CCECE.2005.1557156]

[23] Ghazi SA, Ahmed MA. Pair-Wise test coverage using genetic algorithms. Evolution Computation, 2003,2:1420−1424. [doi: 10.1109/CEC.2003.1299837]

[24] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33(4):225−237. [doi: 10.1109/TSE.2007.38]

[25] McMinn P. Search-Based software test data generation: A survey. Software Testing Verification & Reliability, 2004,14(2):105−156. [doi: 10.1002/stvr.294]

[26] Shaukat A, Briand LC, Hemmati H, Panesar-Walawege RK. A systematic review of the application and empirical investigation of search-based test case generation. IEEE Trans. on Software Engineering, 2011,36(6):742−762. [doi: 10.1109/TSE.2009.52]

[27] Qi YH, Mao XG, Lei Y, Dai ZY, Wang CS. The strength of random search on automated program repair. In: Proc. of the Int'l Conf. on Software Engineering. Hyderabad: ACM Press, 2014. 254−265. [doi: 10.1145/2568225.2568254]

[28] Qi YH, Mao XG, Lei Y, Wang CS. Using automated program repair for evaluating the effectiveness of fault localization techniques. In: Proc. of the Int'l Symp. on Software Testing and Analysis. Lugano: ACM Press, 2013. 191−201. [doi: 10.1145/2483760.2483785]

[29] Mao XG, Lei Y, Dai ZY, Qi YH, Wang CS. Slice-Based statistical fault localization. Journal of Systems & Software, 2014,89(2): 51−62. [doi: 10.1016/j.jss.2013.08.031]

[30] Lei Y, Mao XG, Dai ZY, Wang CS. Effective statistical fault localization using program slices. In: Proc. of the IEEE Computer Software & Applications Conf. Lzmir: IEEE Press, 2012,7204(4):1−10. [doi: 10.1109/COMPSAC.2012.9]

[31] Li MQ, Kou JS, Lin D. Genetic Algorithm Basic Theory and Application. Beijing: Science Press, 2002 (in Chinese).

[32] Holland JH. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, 1975. [doi: 10.1086/418447]

[33] Liblit B, Naik M, Zheng AX, Aiken A, Jordan MI. Scalable statistical bug isolation. In: Proc. of the Conf. on Programming Language Design and Implementation. Chicago: ACM Press, 2005. 15−26. [doi: 10.1145/1065010.1065014]

[34] Liu C, Yan X, Fei L, Han JW, Midkiff SP. SOBER: Statistical model-based bug localization. In: Proc. of the European Software Engineering Conf. on Held Jointly with Int'l Symp. on Foundations of Software Engineering. Lisbon: ACM Press, 2005. 286−295. [doi: 10.1145/1081706.1081753]

[35] Li W, Zheng Z, Hao P, Gao YC, Rao PF, Gong C. Predicate execution-sequence based fault localization algorithm. Chinese Journal of Computers, 2013,36(12):2406−2419 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02406]

[36] Hao P, Zheng Z, Zhang ZY, Gao YC, Gong C, Xue YZ. Self-Adaptive fault localization algorithm based on predicate executioninformation analysis. Chinese Journal of Computers, 2014,37(3):500−511 (in Chinese with English abstract).

[37] Wong WE, Debroy V. A survey on software fault localization [Ph.D. Thesis]. Department of Computer Science, UT Dallas, 2009.

[38] Yoo S. Evolving human competitive spectra-based fault localisation techniques. In: Search Based Software Engineering. Berlin, Heidelberg: Springer-Verlag, 2012. 244−258. [doi: 10.1007/978-3-642-33119-0_18]

[39] Xie XY, Kuo FC, Chen TY, Yoo S, Harman M. Provably optimal and human-competitive results in SBSE for spectrum based fault localization. In: Proc. of the Int'l Conf. on Search Based Software Engineering. Saint Petersburg: Springer-Verlag, 2013. 224−238. [doi: 10.1007/978-3-642-39742-4_17]

[40] Xuan JF, Monperrus M. Learning to combine multiple ranking metrics for fault localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE Press, 2014. 191−200. [doi: 10.1109/ICSME.2014.41]

[41] Hao D, Zhang L, Pan Y, Mei H, Sun JS. On similarity-awareness in testing-based fault localization. Automated Software Engineering, 2008,15(2):207−249. [doi: 10.1007/s10515-008-0025-9]

[42] Hao D, Zhang L, Zhong H, Mei H, Sun JS. Eliminating harmful redundancy for testing-based fault localization using test suite reduction: An experimental study. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. Amsterdam: IEEE Press, 2005. 683−686. [doi: 10.1109/ICSM.2005.43]

[43] He T, Wang XM, Zhou XC, Li WJ, Zhang ZY, Cheung SC. A software fault localization technique based on program mutations. Chinese Journal of Computers, 2013,36(11):2236−2244 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02236]

[44] Masri W. Fault localization based on information flow coverage. Software Testing, Verification and Reliability, 2010,20(2): 121−147. [doi: 10.1002/stvr.409]

[45] Zhang ZY, Jiang B, Chan WK, Tse TH. Debugging through evaluation sequences: A controlled experimental study. In: Proc. of the Int'l Computer Software and Applications Conf. Turku: IEEE Press, 2008. 128−135. [doi: 10.1109/COMPSAC.2008.207]

[46] Zhang ZY, Jiang B, Chan WK, Tse TH, Wang X. Fault localization through evaluation sequences. Journal of Systems and Software, 2010,83(2):174−187. [doi: 10.1016/j.jss.2009.09.041]

[47] Baah GK, Podgurski A, Harrold MJ. The probabilistic program dependence graph and its application to fault diagnosis. In: Proc of the Int'l Symp. on Software Testing and Analysis. Seattle: ACM Press, 2008. 189−200. [doi: 10.1145/1390630.1390654]

[48] Baah GK, Podgurski A, Harrold MJ. Causal inference for statistical fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. Trento: ACM Press, 2010. 73−84. [doi: 10.1145/1831708.1831717]

[49] Baah GK, Podgurski A, Harrold MJ. Mitigating the confounding effects of program dependences for effective fault localization. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Szeged: ACM Press, 2011. 146−156. [doi: 10.1145/2025113.2025136]

[50] Agrawal H, Horgan JR. Dynamic program slicing. In: Proc. of the Conf. on Programming Language Design and Implementation. White Plains: ACM Press, 1990. 246−256. [doi: 10.1145/93542.93576]

[51] Agrawal H, Horgan JR, London S, Wong WE. Fault localization using execution slices and dataflow tests. In: Proc. of the Int'l Symp. on Software Reliability Engineering. Toulouse: IEEE Press, 1995. 143−151. [doi: 10.1109/ISSRE.1995.497652]

[52] Abreu R, Zoeteweij P, Gemund AJCV. Spectrum-Based multiple fault localization. In: Proc. of the Int'l Conf. on Automated Software Engineering. Auckland: Springer-Verlag, 2009. 88−99. [doi: 10.1109/ASE.2009.25]

[53] Wen WZ, Li BX, Sun XB, Qi SS. A technique of multiple fault localization based on conditional execution slicing spectrum. Journal of Computer Research and Development, 2013,50(5):1030−1043 (in Chinese with English abstract).

[54] Groce A. Error explanation with distance metrics. In: Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Vienna: Springer-Verlag, 2004. 108−122. [doi: 10.1007/s10009-005-0202-0]

[55] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33(4):225−237. [doi: 10.1109/TSE.2007.38]

[56] 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]

[57] Wong WE, Wei T, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation. Lillehammer: IEEE Press, 2008. 42−51. [doi: 10.1109/ICST.2008.65]

[58] Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 1937,32(200):675−701. [doi: 10.2307/2279372]

[59] Wilcoxon F. Individual comparisons by ranking methods. Biometrics, 1945,1(6):80−83. [doi: 10.2307/3001968]

附中文参考文献:

[12] 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]

[13] 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. [doi: 10.3724/SP.J.1016.2011.01411]

[17] 李征,巩敦卫,聂长海,江贺.基于搜索的软件工程研究进展与趋势.2013-2014 年度中国计算机科学技术年度报告.北京:机械工业出版社,2014.139−187.

[31] 李敏强,寇纪淞,林丹.遗传算法的基本理论与应用.北京:科学出版社,2002.

[35] 李伟,郑征,郝鹏,高乙超,饶培峰,宫成.基于谓词执行序列的软件缺陷定位算法.计算机学报,2013,36(12):2406−2419. [doi: 10.3724/SP.J.1016. 2013.02406]

[36] 郝鹏,郑征,张震宇,高乙超,宫成,薛云志.基于谓词执行信息分析的自适应缺陷定位算法.计算机学报,2014,37(3):500−511.

[43] 贺韬,王欣明,周晓聪,李文军,张震宇,张成志.一种基于程序变异的软件错误定位技术.计算机学报,2013,36(11):2236−2244. [doi: 10.3724/SP.J.1016.2013.02236]

[53] 文万志,李必信,孙小兵,齐珊珊.基于条件执行切片谱的多错误定位.计算机研究与发展,2013,50(5):1030−1043.

王赞(1979－),男,江苏泗洪人,博士,副教授,CCF 会员,主要研究领域为软件工程,软件测试,机器学习.

邹雨果(1989－),男,工程师,主要研究领域为软件开发,软件测试.

樊向宇(1992－),男,硕士生,CCF 学生会员,主要研究领域为软件测试,机器学习,软件质量.

陈翔(1980－),男,博士,副教授,CCF 会员,主要研究领域为软件缺陷预测,软件缺陷定位,回归测试,组合测试.