基于函数动态调用关系的多缺陷定位研究

杨隆兴
北京理工大学 软件学院
北京，中国
e-mail: 18701113138@163.com

任鑫磊　余永涛
北京理工大学 软件学院
北京，中国
e-mail: renxinlei@aliyun.com

***摘要*—缺陷定位是国内外计算机行业近年来研究的热点，目前为止已经有很多有突破性的技术得以实现，本文就是在深入了解研究了国内外的缺陷定位方法的基础上，从程序切片分析技术入手，结合了 RankBoost 算法以及模糊 c 均值聚类分析进行多缺陷定位分析，同时对测试的偶然正确性进行了分析，并实现了识别偶然正确性用例的方法，本文主要实现的工作如下：**

（1）针对偶然正确性问题，提出了识别偶然正确性用例的方法。

（2）利用模糊 c 值聚类分析和面向函数调用路径方法进行代码的多缺陷定位。

在删除偶然正确性用例后，我们利用 RankBoost 算法生成的训练模型对函数怀疑度进行排名，相较于经典算法，本文方法对缺陷定位的准确性有所提升。

关键词：软件测试，缺陷定位，程序切片，偶然正确性，Rankboost

I. 引言

随着计算机软件的功能以及结构的日趋复杂和互联网行业的迅速发展，对软件可信度的要求越来越高，可信度要求软件具有高可靠性和高可用性。在软件开发的过程中，不可避免的会出现缺陷，缺陷的存在会导致软件产品在某种程度上不能满足用户的需要，而且周期软件产品发布后，修复软件缺陷会变得十分困难，成本会增加数十甚至百倍，所以在软件开发的过程中，缺陷定位就变得尤为重要。

软件的可靠性与软件中隐含的缺陷数目直接相关，缺陷定位是移除软件缺陷的关键步骤，缺陷定位的及时性和有效性直接影响软件的可用性，有效的缺陷定位能够大幅度提高软件开发的效率。同时有效的缺陷定位能够帮助开发人员尽早的修复软件缺陷，提高软件的可用性以及安全性。

软件缺陷定位是软件调试过程中一项最耗时最困难的工作，但是却是十分重要的一环。如今软件行业发展迅速，软件规模日益越庞大，使得缺陷定位更具困难性和挑战性。缺陷定位即是利用程序信息以及测试信息找出程序中引发程序异常的缺陷语句或者预测可能引发程序异常的缺陷语句的范围，辅助开发人员顺利找到含有缺陷的代码并予以修正。一次成功的软件缺陷定位能够大幅度的提高软件开发效率，同时还能很大程度上提高软件的可靠性。

软件测试中的缺陷定位工作一直是软件测试技术的重点同时也是一大难点，因此也成为了国内外测试行业的研究热点，早在 20 世纪 80 年代 M.Weiser 提出了程序切片技术，利用切片技术缩小定位范围，从而实现有效的错误定位；之后又有 Jones 等人提出来 Tarantula 怀疑度计算方法，一种基于频谱的故障定位方法；Abreue 等人之后又设计出了 Ochiaihe 和 Jaccard 方法，这两种方法也被广泛的应用到软件缺陷定位的研究中；Zhang 等人提出的基于动态切片的错误定位方法，进一步推动了缺陷定位研究的发展；在 2009 年 W.Eric.Wong 将 BP 神经网络用于缺陷定位，将神经网络和软件测试技术相结合，取得了一定的成就。

上面讲述的这些缺陷定位技术都能很有效的提升缺陷定位效率，但是目前大多数的缺陷定位技术都是针对软件中的单缺陷进行定位，而且并没有排除掉有偶然正确性用例的干扰，然而现实中，程序的代码中很可能会出现不止一个错误，这些错误相互影响，对程序的调试和改进都造成了很大的影响，这时单单进行单缺陷定位是不够的，必须要通过对测试结果的分析获取各个缺陷之间的联系，从而定位出软件中的多个缺陷。

同时测试的过程中也不可避免的会出现偶然正确性用例，偶然正确性是指程序执行了缺陷语句，而缺陷语句的错误状态并没有传播到程序的输出结果，使得程序执行成功或没有抛出异常。如果在测试的过程中不排除掉这些偶然正确性用例的干扰，会对缺陷的定位效率产生十分大的影响，甚至导致缺陷定位的失

败，去除掉偶然正确性用例能够提高代码缺陷定位的准确性以及效率。

本论文运用的程序切片技术主要是 Aspectj，全称是“a seamless aspect-oriented extension to the Java programming language”（一种基于 Java 平台的面向切面编程的语言）。Aspectj 能做干净的模块化横切关注点（也就是说单纯，基本上无侵入），如错误检查和处理，同步，上下文敏感的行为，性能优化，监控和记录，调试支持，多目标的协议。Aspectj 是一种易学易用的基于 java 的面向切面编程的语言，它扩展了 java 语言，定义了 AOP 语法，兼容 java 平台，可以无缝进行扩展。因此所有合法 的 Java 程序也是合法的 aspectJ 程序。AspectJ 编译器生成的是符合 Java 字节编码规范的.class 文件，这使得所有符合规范的 Java 虚拟机都可以解释、执行其所生成的代码。通过选择 Java 为基础语言，AspectJ 继承了 Java 的所有优点并使 Java 程序员能够比较容易上手。

AspectJ 还提供了许多有用的工具。它有一个方面织入器（以编译器的形式）、一个调试器、文档生成工具以及一个独立可用来以可视化的方式观察通知是如何切入系统各部分的方面浏览器。另外，AspectJ 还提供了与流行 IDE 的集成，这使得 AspectJ 成为一个很有用的 AOP 实现，特别是对 Java 开发者而言

为了支持 AOP，AspectJ 在 Java 语言中加入了如下语言构造：

- 连接点：程序执行过程中的点，比如调用某个类中的特定方法的地方
- 切入点：用来指定、组合所需连接点并收集连接点上特定上下文信息的语言结构。
- 通知：在符合特定条件的情况下执行的代码，例如：一个通知可能会执行到一个连接点之前记录一条信息。

切入点和通知一起指定织入规则。而方面是一个类似于 OOP 中类的概念，是把切入点和通知放在一起形成的具有横切功能的单元。切入点制定执行中的点和织入规则的上下文，通知则指定在这些点上所做的选择及动作等。而连接点则看作是一系列的事件，通知就是响应这些事件的选择或者动作。

同时本文对于缺陷的定位还利用了 RankBoost 算法，RankBoost 是当前比较有效的 pairwise 型的排序学习算法之一，是一种结合给定样本的多个弱排序结果（相对排序），给出一个比较精确的最终排序的排序算法。在论文对缺陷定位研究的过程中，利用了模糊 C 均值聚类分析方法(FCM)删除测试过程中的偶然正确性用例，模糊 C 均值算法(FCM)是一种基于划分的聚类算法，它的思想就是使得被划分到同一簇的对象之间相似度最大，而不同簇之间的相似度最小。

论文以 Aspectj 技术为基础，结合了 RankBoost 和函数的动态调用路径的方法，来进行程序中的多缺陷定位分析，同时利用模糊 C 均值聚类分析的方法识别并删除了测试过程中的偶然正确性用例，排除了偶然正确性对缺陷定位的干扰。

II. 错误定位方法框架

本文的错误定位旨在找到源代码中存在错误的函数。方法的整体框架如图 1，我们首先利用错误程序样本对给出的怀疑度量公式进行加权组合，这是一个学习的过程，算法模型选取的是 RankBoost；接着，本文基于模糊 c 均值聚类对测试用例进行分析，以降低偶然正确性因素对结果的干扰；最后，借助训练好的模型对函数进行怀疑度排序，输出排名结果。

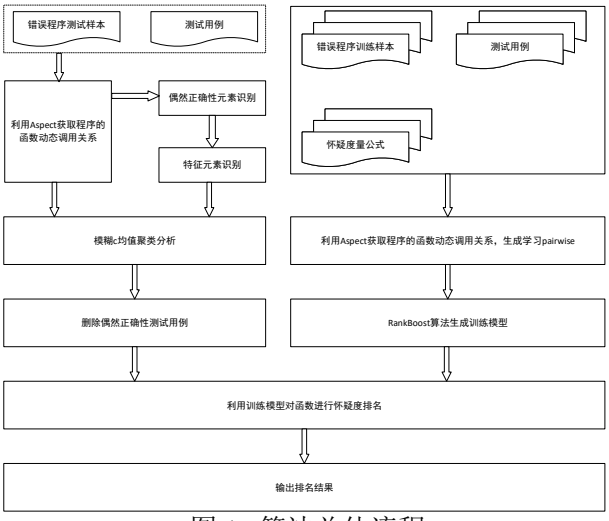

图 1 算法总体流程

本文将用 3 部分来阐述方法的整体框架。第一部分是如何删除偶然正确性测试用例，以减少测试用例的干扰。第二部分是如何利用训练样本和给定的怀疑度量公式，生成训练模型。第三部分则说明框架的整体流程，如何对错误程序进行缺陷定位。

III. 删除偶然正确性测试用例

A. *概念定义*

定义 1 偶然正确性元素cc_e. 给定一个程序元素 e，$P(T_F, e)$为失败测试用例T_F执行程序元素 e 的概率，$P(T_p, e)$为成功测试用例T_p执行程序元素 e 的概率。程序元素 e 为偶然正确性元素cc_e,则 e 要出现在所有失败测试中，即$P(T_F, e) = 1$，又要求其出现在成功测试中的概率为$0 < P(T_p, e) < 1$，cc_e定义如下：

$$cc_e = \{e \mid \mathrm{P}(T_F, \mathrm{e}) = 1 \wedge 0 < \mathrm{P}(T_p, \mathrm{e}) < 1\}$$

定义 2 偶然正确性测试用例T_{cc}. 一个测试用例 T 执行了偶然正确性元素cc_e，却没有导致程序执行结果失败，则该测试用例 T 为偶然正确性测试用例T_{cc}。

定义 3 路径矩阵 A 令 m 表示测试用例的数量，n 表示被测程序中函数的数量，则 m×n 阶矩阵 A 为路径矩阵，如下。元素$A_{ij}(1 \le i \le m, 1 \le j \le n)$代表执行第 i 个测试用例时第 j 个函数被覆盖的次数。其中，每一行是一个路径向量。

$$\mathrm{A} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

B. 模糊 c 均值聚类

FCM 算法是一种基于划分的聚类算法，它的思想就是使得被划分到同一簇的对象之间相似度最大，而不同簇之间的相似度最小。模糊 C 均值算法是普通 C 均值算法的改进，普通 C 均值算法对于数据的划分是硬性的，而 FCM 则是一种柔性的模糊划分。Fuzzy c-means 聚类就是通过找到使用的最佳隶属度函数和最佳聚类中心点来求解目标函数 J (U , V)的过程，从而将聚类问题转换成一个带约束的非线性规划问题，通过不断迭代求解得到数据集的最优划分。其目标函数 J (U , V)如下。

$$\begin{aligned} \mathrm{J}(\mathrm{U}, \mathrm{V}) &= \mathrm{J}(\mathrm{U}, v_1, v_2, \ldots, v_c; \mathrm{X}) \\ &= \sum_{i=1}^{l} \sum_{j=1}^{m} u_{ij}^m d_{ij}^2 \end{aligned}$$

其中，U 代表隶属度矩阵，V 表示聚类中心集合，$v_1, v_2, \ldots, v_c$表示 c 个聚类中心点，X 表示聚类处理的数据对象。$d_{ij} = \|v_i - x_j\|$表示x_j与第 i 个聚类中心v_i的距离，这里采用闵式距离，参数为 p，$\mathrm{m} \in [1, \infty)$,随着 m 值的增大，聚类的模糊性增大。本文取 p=2，m=3。隶属度$u_{ij} \in [0,1]$，表示第 j 个数据对象属于第 i 个聚类中心的隶属度。

确认最佳隶属函数，即$\mathrm{J}(\mathrm{U}, \mathrm{V})$相对与$\{u_{ij}\}$的最优化为题，存在极值约束：

$$\sum_{i=1}^{c} u_{ij} = 1$$

由拉格朗日乘子法可以得出，

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}}\right)^{\frac{2}{m-1}}} \quad (1 \le i \le c, 1 \le k \le n)$$

$$v_i = \frac{\sum_{j=1}^{n} u_{ij}^m X_j}{\sum_{j=1}^{n} u_{ij}^m}, (i = 1,2, \ldots, c)$$

其中u_{ij}为隶属度值，v_i为最佳聚类中心点。

这样，Fuzzy c-means 聚类变成一个迭代求最优解的问题。

C. 删除算法

删除偶然正确性的算法如下：

输入：

P,错误程序

δ,测试用例集

输出：

A,删除偶然正确性用例后的路径矩阵

算法：

初始化路径矩阵 A，偶然正确性测试用例集 T

Foreach 测试用例 $c \in \delta$ **do**

根据 P 和 c，由 aspectj 获取路径向量 A_c

If $\exists \mathrm{e} \in \mathrm{A}_c$ **and** $\mathrm{P}(T_F, \mathrm{e}) = 1 \wedge 0 < \mathrm{P}(T_p, \mathrm{e}) < 1$

将 c 加入 T

End

End

由A_c获取路径矩阵 A

随机初始化隶属度矩阵 U，矩阵大小为$|\delta| * 2$

利用 FCM 算法迭代计算，获取隶属度矩阵 U 和最佳聚类中心点v_i

获取错误用例多正确用例少的类 S

Foreach 路径向量 $s \in S$ **do**

If $s \in T$

删除 A 中的 s

End

End

Return A

本方法将测试用例集分成两类，一类错误用例多正确用例少的，一类是正确用例多错误用例少的。由于路径具有相似性，如果满足偶然正确性特征的用例在前一类中，则将其删除。

IV. 怀疑度量公式的组合学习

A. 怀疑度量公式

本文的怀疑度量公式是基于程序谱的怀疑度计算公式，如 tarantula、ochiai 等。公式根据四元组$<n_p, n_f, e_p, e_f>$对每个程序元素 e 计算怀疑度。其中，n_p，n_f表示成功和失败测试用例未覆盖程序元素 e 的

次数，e_p, e_f表示成功和失败测试用例覆盖程序元素 e 的次数。本文用的怀疑度量公式如下，

表 1 怀疑度度量公式

怀疑度量公式	定义
Tarantula	$\frac{\frac{e_f}{e_f+n_f}}{\frac{e_f}{e_f+n_f}+\frac{e_p}{e_p+n_p}}$
Ochiai	$\frac{e_f}{\sqrt{(e_f+e_p)(e_f+n_f)}}$
Jaccard	$\frac{e_f}{e_f+e_p+n_f}$
Naish2	$e_f-\frac{e_p}{e_p+n_p+1}$
GP13	$e_f(1+\frac{1}{2e_p+e_f})$

B. 组合模型

怀疑度量公式的加权组合模型如下，

$$\text{Multric}(e)=\sum_{k\in\kappa} weight_k(e)*susp_k(e)$$

其中,e 表示程序实体，weight 为学习后的权值，susp 为怀疑度量公式。

C. RankBoost 算法训练

首先，我们给定训练对的概念。训练对$<e_+, e_->$表示错误实体e_+和正确实体e_-的有序对，表明实体e_+排名应在e_-之前。

在给定的训练程序集中，每个程序都标明了错误的实体（函数）和正确的实体，由此可以生成训练对。

由所有程序的训练对可以给出权值分布矩阵，定义如下：

定义 4 权值分布矩阵 D 令 n 表示给定程序集所有程序实体的数量，，则 n×n 阶矩阵 D 为权值分布矩阵，如下。如果程序实体i, j能构成训练对$<i, j>$,则元素$D_{ij}>0(1\le i\le n, 1\le j\le n)$,否则$D_{ij}=0$。并且，存在约束$\sum_{i,j} D_{ij}=1$

$$\mathrm{D}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

在训练过程中，我们给定如下代价函数，

$$loss=\sum_{x_0,x_1} D(x_0,x_1)\|h_t(x_1)>h_t(x_0)\|$$

其中，$\|h_t(x_1)>h_t(x_0)\|$表示如果$h_t(x_1)>h_t(x_0)$成立，值为 1，否则值为 0。

RankBoost 训练算法如下，

输入：

D,权值分布矩阵

h_t $\mathrm{t}=1,2,\ldots\mathrm{T}$, 怀疑度量公式，用于排名

输出：

$\mathrm{H}(\mathrm{x})=\sum_{t=1}^{T}\alpha_t h_t(x)$,组合公式

算法：

初始化权值分布矩阵 D

$D_1=D$

For $\mathrm{t}=1,2,\ldots\mathrm{T}$

$loss=\sum_{x_0,x_1} D(x_0,x_1)\|h_t(x_1)>h_t(x_0)\|$

$\alpha_t=\frac{1}{2}\ln(\frac{1+loss}{1-loss})$

$D_{t+1}(x_0,x_1)=\frac{D_t(x_0,x_1)\exp(\alpha_t(h_t(x_0)-h_t(x_1)))}{z_t}$

End

Return $\mathrm{H}(\mathrm{x})=\sum_{t=1}^{T}\alpha_t h_t(x)$

其中， *Zt* 是规范化因子。

V. 缺陷定位流程

缺陷定位的整体流程如下：

1) 由给定的训练程序集得出组合公式 $\mathrm{H}(\mathrm{x})=\sum_{t=1}^{T}\alpha_t h_t(x)$
2) 输入程序和测试用例，得到删除偶然正确性用例后的路径矩阵 A
3) 根据公式对$\mathrm{H}(\mathrm{x})$对 A 中的路径向量进行统计，得出最终的怀疑度排名。
4) 由排名按自上而下的顺序检查程序，定位错误。

VI. 实验设计

此实验采取 Tarantula、Ochiai、改进的 FLAR 共三种定位方法进行对比试验，运行环境：Windows10 64 位操作系统，4 核 CPU3GHz Intel（R），内存 16GB。

A. 实验对象

基于函数动态调用关系的缺陷定位分析，通过运行大量测试集，测试用例的实际输出结果与期望结果相等的为通过用例，不相等的为失败用例，通过分析函数在通过用例集和失败用例集中的调用情况确定函数存在缺陷的可能性。失败用例集中每个测试用例的输出结果都与期望结果不相等，这必然是由于某种原因导致的，缺陷函数的执行是最可能的原因，当然不能排除程序运行中出现的异常（例如，内存不足时被迫中止执行），但是针对大量的测试集，这种小概率事件可以忽略不计。失败测试集中缺陷函数一定是执行了，所以失败测试集中一定包含更多与缺陷函数相关的信息，这个实验主要是针对失败用例集进行分析。

实验采用 AES-RSA 加密程序、DES 加密程序、计算器模拟程序、自创小程序作为代码小规模的程序的程序代表，为了保障实验结论的一般性，选取中等规模的程序两个。实验对象详细信息见表 2，程序来源：GitHub（GitHub 网址：github.com）\SIR（SIR 网址：http: / /sir.unl.edu /portal /index. html）

表 2 实验对象

程序	描述	代码行数	错误版本	函数个数
AES-RSA	加密算法	1035	2	48
DES	加密算法	532	2	22
计算器	模拟程序	1123	2	38
小程序	简单程序	286	2	50
字符串处理	简单程序		5	30
Apollo	简单程序		1	55
迷宫	模拟程序		1	30
DFS-BFS	简单程序		1	12

B. 评测指标

（1）漏报率（False Negatives，FN），主要用于评估某种方法未准确识别偶然正确性测试用例的比例，见式（1），其中其中， true CC NO.表示的是真实的偶然正确测试用例数，CC identifed NO.表示的是通过聚类分析技术判别出来的偶然正确性测试用例数。

$$FN=\frac{|\text{true CC NO.} - \text{CC identifed NO.}|}{|\text{ true CC NO.}|} \quad (1)$$

（2）误报率（Flase Positives, FP）：主要用于评估某种方法将非偶然正确性测试用例判别为偶然正确性测试用例的比率，见公式（2），其中，T_P 表示成功测试用例数

$$FN=\frac{|(\text{Tp-true CC NO.}) - \text{CC identifed NO.}|}{|Tp - \text{true CC NO.}|} \quad (2)$$

（3）平均检索值：查找到缺陷函数平均需要查找的函数个数，平均检索值显示了在所有软件缺陷版本中查找到缺陷函数平均需要查找的次数，从平均值的角度考虑缺陷定位算法的效率。Vi 表示多个缺陷函数的排名，见公式（3）

$$AVG=\frac{\sum_{i=1}^{n} Vi}{n} \quad (3)$$

（4）最大检索值：表示在所有的软件缺陷版本中查找到缺陷函数最多需要查找的函数个数，

$$MAX=MAX（V1,V2,.....Vn） \quad (4)$$

（5）最小检索值：表示在所有的软件缺陷版本中查找到缺陷函数最少需要查找的函数个数。

$$MIN=MIN（V1,V2,.....Vn） \quad (5)$$

VII. 实验结果

A. 实验评估（平均检索值）

对于缺陷定位算法的评估，此评估主要是对平均检索值进行比较，测试程序为前四个程序：

表 1 各错误版本的 AVG

错误版本	Tarantula	Ochiai	改进 FLAR	待测算法
Version1	5.33	2.33	2	2.33
Version2	2.4	4.2	3	1.4
Version3	2	3	4	3
Version4	2.66	3	4.33	3
Version5	2	3	2	2
Version6	2	2	3	3
Version7	4	3	5	7
Version8	3.66	3.33	3	2
ALL_AVG	3.00625	2.9825	3.29125	2.96625

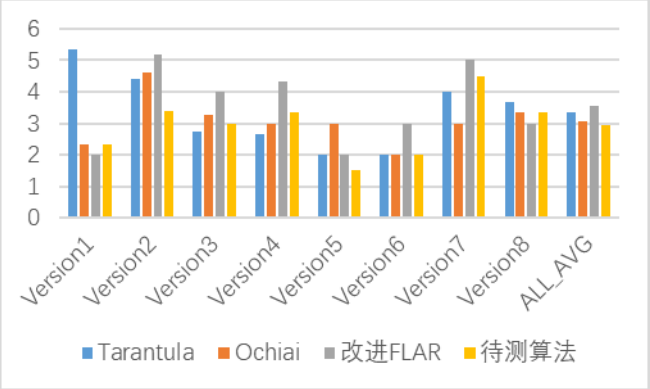

图 2 各错误版本的平均检索值

为了更加直观地比较不同方法的优劣，本章还使用缺陷率检出指标进行评估。实验结果如图 3 所示，横坐标为代码检查率，纵坐标为相应的错误检出率。在相同代码检查率情况下，某方法的错误检出率越高， 则该方法越有效。从图三可知， 我们的算法定位效果明显优于 Tarantula、 Ochiai 和 改进的 FLAR。

表 1 各错误版本的 AVG

	10%	20%	30%	40%
Tarantula	33%	75%	100%	100%
Ochiai	34%	70%	100%	100%
改进 FLAR	40%	82%	96.10%	100%
待测算法	39%	85%	100%	100%

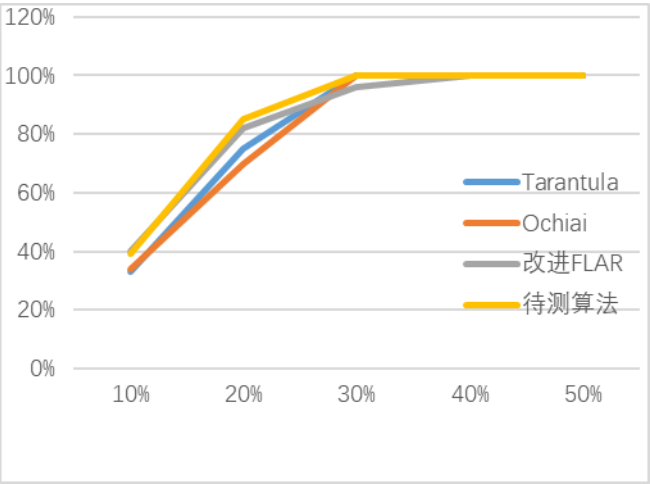

图 2 各错误版本的平均检索值

B. 实验评估（极限检索值）

对于缺陷定位算法的评估，此评估主要是对最大检索值和最小检索值进行比较，测试程序为全部 8 个程序选取一个错误版本，共 8 个测试版本。

通过实验数据可以发现，最小检索值的算法定位效果在 Tarantula、 Ochiai、 改进的 FLAR 和待测算法之间差别不大，对实验的参考价值不高，所以此评估只对最大检索值进行分析。通过对 8 个错误版本的分析，得到的实验数据如下：

表 1 各错误版本的最大检索值

错误版本	Tarantula	Ochiai	改进 FLAR	待测算法
Version1	10	3	3	4
Version2	4	4	6	5
Version3	3	4	2	2
Version4	6	5	4	5
Version5	4	5	7	5
Version6	12	15	13	13
Version7	4	5	5	5
Version8	4	4	4	4
MAX_AVG	5.875	5.625	5.5	5.375

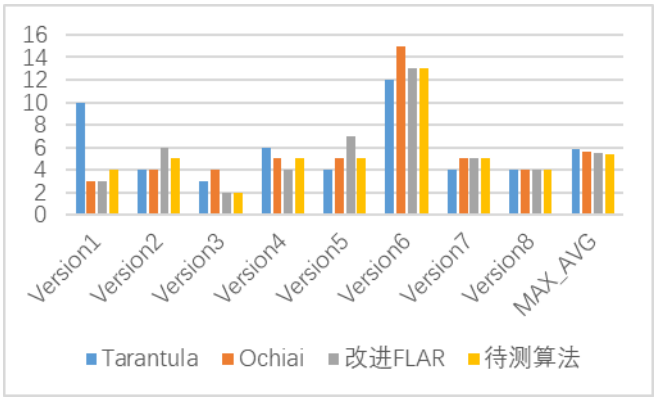

图 2 各错误版本的最大检索图

通过图表可以看出，待测算法的最大检索值的平均值在四类算法中表现最好，无论是小规模程序还是大规模程序，都表现出了良好的适应性。

C. 实验小结

本文主要是通过分析函数动态调用关系对函数缺陷进行定位。程序中每个函数实现一个独立的业务逻辑功能，程序中的缺陷更多的存在于逻辑处理中，以函数为单位进行缺陷分析，符合程序设计的思想也更方便进行缺陷查找。主要研究函数动态调用关系的两个方面，1.动态调用次数与软件缺陷的关联性 2.函数动态调用序列匹配与软件缺陷的关联性。3.多种错误定位算法的融合。

1.动态调用次数与软件缺陷的关联性。那些在失败的测试用例中出现的次数越多，在通过的测试用例

中出现的次数越少的函数，存在缺陷的可能性越大，这符合我们对缺陷的直观理解。若一个测试用例表现为失败的测试用例，则必然是由于缺陷函数被触发引起的，当然存在软件运行异常导致的缺陷，但是针对大量的测试用例，这种小概率事件可以忽略不计。每个失败的测试用例的函数调用集中必包含至少一个缺陷函数。若程序只有一个缺陷函数， 则其必属于所有失败测试用例的函数调用集的交集。若程序有多个缺陷函数，则缺陷函数程序的全部缺陷函数必属于所有失败测试用例的函数调用集的并集。函数风险向量通过一定的算法对函数调用次数进行运算，预示着函数存在缺陷可能性的概率。通过实验分析发现，通过结合函数缺陷引理、使用函数风险向量，定位缺陷函数的效率很高。

2. 函数动态调用序列匹配与软件缺陷的关联性。函数缺陷很多时候会体现在函数调用路径异常上，如果对于函数每一步特定的输入都能够确定对应的输出或将要调用的函数，将能够准确的定位到大部分的缺陷函数的具体位置。一部分测试用例之所以为失败测试用例，是由于缺陷函数被触发引起的，因此失败的测试用例集包含更多与缺陷函数相关的信息。对失败测试用例集的函数动态调用序列进行分析匹配，确定函数调用序列首次暴露出异常调用与缺陷函数的关联性。通过实验分析发现，缺陷函数位于函数调用序列异常之前被调用的函数中。函数中包含复杂的处理逻辑，大部分的缺陷函数会引起函数调用路径出现异常，还有很少一部分不会引起函数调用路径异常。对于会引起函数调用路径异常的缺陷函数，缺陷函数基本上位于首次暴露出异常的上一步调用中。获取错误版本的函数异常调用起始点，很多时候可以快速定位到软件中的缺陷位置，缺陷函数大部分位于异常调用起始点的前一步调用中，这个想法可以应用到软件版本升级中的回归测试，不同版本间路径调用差异很小，对于版本升级中的路径调用修改可以进行保存，该方法可以用于回归测试。

3. 多种错误定位算法的融合。通过怀疑度公式的组合学习，调整不同公式的使用权重，克服单一公式的不足，使新生成的组合公式更加适合寻找错误，从而提升错误定位的准确性，提高效率。在这里，我们借鉴了 Adaboost 的思想，弱分类器通过组合提升可以生成强分类器。错误定位中，各种经典公式可以认为是一般错误定位器，通过组合学习将其强化成强错误定位器，以提升效果。

通过平均检索值以及最大检索值的实验数据可以看出，待测算法在小程序和中等程序规模的 JAVA 程序的错误定位中，都优于其他三类错误定位算法，同时，相对于其它三类算法。待测算法的稳定性也相对较高，不会出现太大的偏差。

VIII. 相关工作

早在 20 世纪 80 年代 M.Weiser 提出了程序切片技术，利用切片技术缩小定位范围，从而实现有效的错误定位；之后又有 Jones 等人 提出了基于程序谱信息的怀疑度排名计算公式，Tarantula，它是基于覆盖的缺陷定位方法。Abreuet 等人引入分子生物学领域的 Ochiai 相似系数，对 Tarantula 进行改进并提出 Ochiai 和 Jaccard 怀疑度指标公式。随后，Jeffrerey [6] 提出基于检查缺陷测试用例覆盖情况的值替换方法。Naish 等人 [8] 在 C 程序上通过实验比较了不同排名计算公式的效果，从理论上证明 Naish1 和 Naish2 是两种最佳的怀疑排名计算公式。

Xie [8]总结当时已知的怀疑计算公式并从理论上对其进行等价划分。Loet [9]等人就多种存在的排名指标进行综合研究并证明实践中没有最好的单一排名方式。Steimann [10]等人研究了基于频谱的错误定位方式的不足。该实验分析了 10 个 java 的开源项目，并以函数方法为粒度对缺陷定位结果进行分析。

与此同时，有不少研究工作通过利用数据对错误定位方法进行训练，Feng 和 R. Gupta 提出利用贝叶斯网络建立源代码到测试结果的依赖关系模型。该模型建立了一种缺陷流图，该图通过随机选择训练集来修正模型中的依赖关系。接着，Yoo[11] 提出用遗传算法来组织源代码的频谱关系，自动性的产生排名指标的方法。在该方法中，30 个排名指标中高效者有 6 种，其中效果最为优异的是 GP13。

以上方法多是利用单一怀疑度量公式进行多错误定位，其缺点是识别率和精度不高，进一步提升的空间有限。

在多怀疑度量公式的组合应用方面，Debroy 和 Wong 组合了 3 种怀疑度量公式（Tarantula、Ochiai 和 Wong3），根据它们排名结果的相同部分进行缺陷定位。同时，Wang 等人提出了基于搜索的缺陷定位方法。该方法模拟退火和遗传算法，为排名指标分配数值权重。本文则通过 rankboost 算法对多种怀疑度量公式进行加权组合，其中的 rankboost 具有收敛速度快的效果，而且可以对多个弱分类器进行组合，具有较大的使用自由度。

针对偶然正确性问题，Wang 等人假定调试人员可以提前知道程序缺陷类型，通过将错误触发前后控制流和数据流与特定模式匹配，提纯代码覆盖信息，从而移出偶然正确性测试用例，降低其对缺陷定位效

率的不良影响。Gong[12] 等人提出了一种可交互的缺陷定位框架，其利用简单的用户反馈来提升定位的准确性。Masri 等人采用多元可视化技术以识别偶然正确性用例，该方法借助 MDS 算法将每个程序执行轨迹变成二维空间的一个点，然后根据点间距离来度量程序执行轨迹的相似性。Xuan 和 Monperrus 两人给出简化测试用例的方法，该法将测试用例进行分类以增强缺陷定位用例的代表性，从而避免偶然正确性用例的影响。

此外，文献 [13]用 k-means 聚类来识别偶然正确性测试用例，移除之后提升了错误定位方法的效率。但 k-means 聚类属于硬分类方法，每个程序执行轨迹只属于所有类别中的某一类，非此即彼。导致所有程序执行轨迹信息在计算中对聚类中心贡献度相同，不利于找出最有聚类。为此，我们使用模糊 c 均值聚类来识别偶然正确性测试用例，并通过阈值设定决定是否删除识别的用例，这样既避免了硬聚类的影响，又可以防止删除过多测试用例带来的坏处，从而提升了识别的准确率。

参考文献

[1] Xuan, Jifeng, and M. Monperrus. "Learning to Combine Multiple Ranking Metrics for Fault Localization." IEEE International Conference on Software Maintenance and Evolution IEEE, 2014:191-200.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Wei Z, Han B. Multiple-Bug Oriented Fault Localization: A Parameter-Based Combination Approach[C]. Proceedings of the 7th IEEE International Conference on Software Security and Reliability-Companion, Gaithersburg, MD: IEEE, 2013: 125-130.

[3] Masri W, Assi R A, Zaraket F, et al. Enhancing fault localization via multivariate visualization[C]. Proceedings of the 5th IEEE International Conference on Software Testing,Verification and Validation, Montreal, QC: IEEE, 2012: 737-741.

[4] W. Eric Wong，Ruizhi Gao，Yihao Li，Rui Abreu，Franz Wotawa，“A Survey on Software Fault Localization”，Software Engineering IEEE Transactions on，vol。42，pp。707-740，2016，ISSN 0098-5589

[5] 赵芳, 牟永敏, and 张志华. "基于神经网络的面向函数调用路径的错误定位." 计算机仿真 33.6(2016):391-395

[6] T. Peng and K. Wang, “Program Verification by Reachability Searching on Dynamic Call Tree”, in Proceedings of the International Conference on Advanced Data Mining and Applications, China, November 2014

[7] T. Peng and K. Zhang, “Verification of the Integration and Instantiation of Security Patterns”, accepted by International Journal of Software Engineering and Knowledge Engineering, 2014

[8] S. Nessa, M. Abedin, W. Eric Wong, L. Khan, and Y. Qi, “Fault Localization Using N-gram Analysis,” in Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications, pp. 548-559, Richardson, Texas, USA, April 2009 (Lecture Notes In Computer Science, Volume 5258)

[9] IEEE 610.12-1990, Glossary of Software Engineering Terminology[S]

[10] L. C. Ascari, L. Y. Araki, A. R. T. Pozo, and S. R. Vergilio, “Exploring Machine Learning Techniques for Fault Localization”, in Proceedings of the 10th Latin American Test Workshop, pp. 1-6, Buzios, Brazil, March 2009.

[11] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In Proceedings of the 24th international conference on Software engineering, pages 467–477. ACM, 2002.

[12] L. Naish, H. J. Lee, and K. Ramamohanarao. A model for spectrabased software diagnosis. ACM Transactions on software engineering and methodology (TOSEM), 20(3):11, 2011.

[13] F. Steimann, M. Frenkel, and R. Abreu. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In Proceedings of the 2013 International Symposium on Software Testing and Analysis, pages 314–324. ACM, 2013.