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以及贡献如下：**

（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。

综合利用了 RankBoost 算法，同时删除偶然正确性用例，之后利用 RankBoost 算法生成的训练模型对还是函数怀疑度进行排名，提高了缺陷定位的准确性。

关键词：软件测试，缺陷定位，程序切片，偶然正确性

I. 引言

随着计算机软件的日趋复杂和网络的迅速发展，软件可信的要求越来越高, 可信要求软件具有高可靠性和高可用性。软件中隐含的缺陷数目与可靠性直接相关, 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。

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，如今软件规模日益越庞大，使得缺陷定位更具挑战性。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。

软件测试中的缺陷定位工作一直是软件测试技术的重点和难点，是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，Zhang 等人提出了基于动态切片的错误定位方法，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。以上这些缺陷定位技术都能很有效的提升缺陷定位效率，但目前大多数的缺陷定位技术都是针对单缺陷定位， 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，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，去除掉偶然正确性用例能够提高代码缺陷定位的效率。

本论文运用的程序切片技术主要是 Aspectj，Aspectj 是一种易学易用的基于 java 的面向切面编程的语言，兼容 java 平台，可以无缝进行扩展。

论文以 Aspectj 技术为基础，利用模糊 c 均值聚类分析的方法删除了测试过程中的偶然正确性用例，同时结合了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。

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。

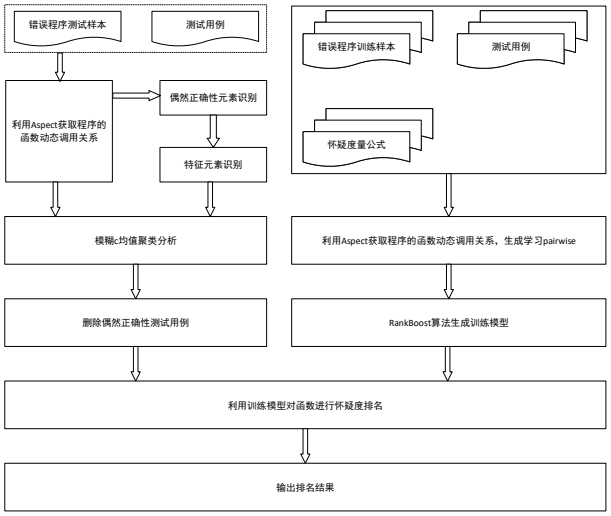


图 1　算法总体流程

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，$\mathrm{P}(T_F,\mathrm{e})$为失败测试用例$T_F$执行程序元素 e 的概率，$\mathrm{P}(T_p,\mathrm{e})$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$\mathrm{P}(T_F,\mathrm{e})=1$，又要求其出现在成功测试中的概率为$0<\mathrm{P}(T_p,\mathrm{e})<1$，$cc_e$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，

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，

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

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$\sum_{i,j} D_{ij} = 1$

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

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 t = 1,2,...T, 怀疑度量公式，用于排名

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

算法：
初始化权值分布矩阵 D

$D_1 = D$
For t = 1,2,...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 $H(x)=\sum_{t=1}^{T} \alpha_t h_t(x)$

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 $H(x)=\sum_{t=1}^{T}\alpha_t h_t(x)$
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$H(x)$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、FLAR、改进的 FLAR 共四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

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	测试用例
AES-RSA	加密算法	1035	2	100
DES	加密算法	532	2	100
计算器	模拟程序	1123	2	100
小程序	简单程序	286	2	50

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。

$$FN=\frac{|\text{true CC NO.} - \text{CC identifed NO.}|}{|\text{true CC NO.}|}$$

（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

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

（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）

$$AVG=\frac{\sum_{i=1}^{n}Vi}{n}$$

（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，

$$MAX=MAX(V1,V2,.....Vn)$$

（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。

$$MIN=MIN(V1,V2,.....Vn)$$

VII. 实验结果

A. 实验评估

对于缺陷定位算法的评估，主要是对平均检索值进行比较：

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

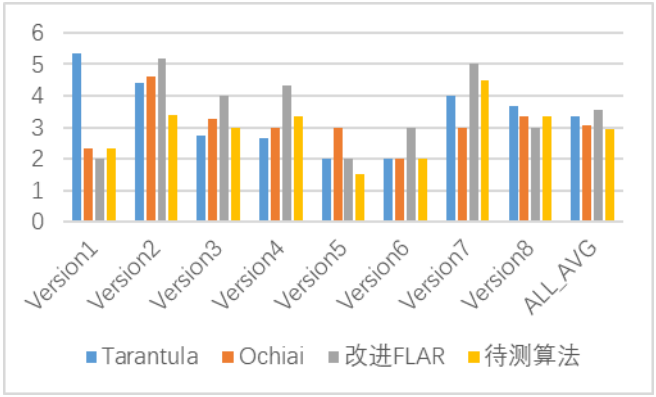
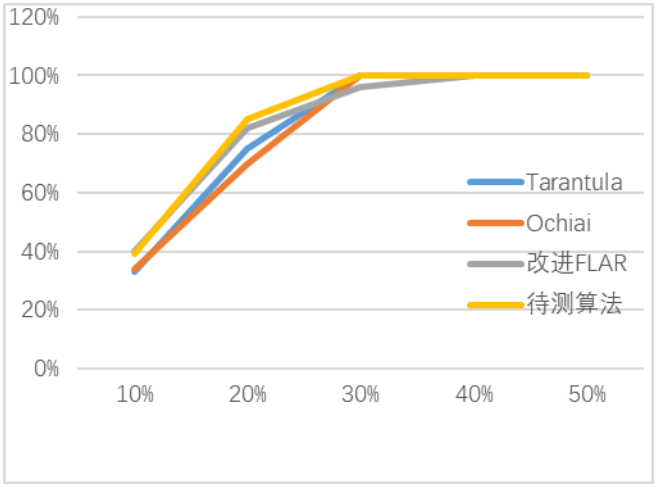


图 2 各错误版本的平均检索值

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。



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