Software Engineering and Applications 软件工程与应用, 2016, 5(1), 1-9
Published Online February 2016 in Hans. http://www.hanspub.org/journal/sea
http://dx.doi.org/10.12677/sea.2016.51001

Hans汉斯

Heuristic Wrong Positioning Method Based on Classification Forecast

Jin Wang, Jie Luo

School of Computer and Engineering, Beihang University, Beijing
Email: wjyz91@163.com

Received: Jan. 27th, 2016; accepted: Feb. 14th, 2016; published: Feb. 17th, 2016

Abstract

This paper mainly forecasts the wrong positioning results for the program of the formal error localization algorithm before using it proceed two wrong positioning or multiple wrong positioning by training learn based on machine learn methods. It leads the user to give the high effective feedback information (expected value or the right value information). Thus, it decreases the wrong positioning number, raises the efficiency of the wrong positioning and realizes the heuristic wrong positioning.

Keywords

Machine Learn Method, Classification Forecast, Heuristic, Wrong Positioning

基于分类预测的启发式错误定位方法

王　瑾，罗　杰

北京航空航天大学计算机学院，北京
Email: wjyz91@163.com

收稿日期：2016年1月27日；录用日期：2016年2月14日；发布日期：2016年2月17日

摘　要

本文利用机器学习的方法通过训练学习，对通过形式化的错误定位算法进行错误定位的程序在二次定位

文章引用: 王瑾, 罗杰. 基于分类预测的启发式错误定位方法[J]. 软件工程与应用, 2016, 5(1): 1-9.
http://dx.doi.org/10.12677/sea.2016.51001

或者多次定位前进行结果预测，引导用户给出高效的反馈信息(期望值或者正确值的确认信息)，从而减少错误定位的次数，提高错误定位的效率，实现启发式的错位定位。

关键词

机器学习方法，分类预测，启发式，错误定位

1. 引言

随着现代社会信息技术的高速发展，软件开发行业已经逐渐成熟。从整个软件开发过程来看，软件的测试、调试和验证一般要占软件开发总消耗的 50%~75%，因此构建一个有机高效的软件测试、调试和验证环境对于保证软件的质量、控制软件开发的成本有着重要的作用[1]。而在程序的调试过程中，90%的工作在于定位和理解错误，所以程序调试的重点在于错误的定位与理解[2]。而现今的错误定位技术多数具有一定的局限性，适用范围小，定位范围不准确等等。比如基于覆盖信息的错误代码定位技术(Coverage-Based Fault Localization, CBFL) [3]，这类方法针对一个程序的错误定位需要大量测试用例来收集分析语句的覆盖信息，偶然性成功测试用例很容易影响该技术的准确率[4]。

为了研究错误定位准确、自动化程度较高且适用范围广的错误定位技术，根据本系统的需求分析，选择基于操作语义的形式化错误定位算法，该算法是由北京航空航天大学软件国家重点实验室软件调试自动化环境研究项目组提出的——它是将程序解析执行后获得程序的执行轨迹栈，根据用户的反馈信息，对于轨迹栈内的每一个路径逆向生成相应的方程组，求解使得方程组无解的第一个语句从而定位出包含错误语句的片段，而通过一次定位出的错误片段可能对用户来讲范围过大，不好准确找到发生错误的语句并修复，这使得用户需求二次定位或者多次定位来缩小范围，以求更好地定位错误。而多次定位中，本文选择相关机器学习方法，对新的测试用例的多个变量进行分类预测，找到预测的定位效果最好的变量来引导用户给出其期望值，从而达到启发式的错误定位。

2. 启发式错误定位

分类预测是通过基于操作语义的形式化错误定位算法进行错误定位并生成分类信息，从而形成训练数据。然后根据机器学习相关算法对训练数据进行筛选、训练及预测。给出的预测信息，将期望值或是确认值按照预测信息进行排序，引导用户给出有效信息来进行错误定位，提高多次定位的效率和效果。

基于操作语义的形式化错误定位算法(BugIso 算法)，输入是路径跟踪中记录的轨迹列表 ρ，其输出包括与用户预期矛盾的极小程序片段 *Seg* 和一个包含 *Seg* 中所有语句的 ρ 的子列表。

算法：BugIso 算法

输入：ρ
输出：$\rho[1,m'],Seg$
步骤：
1 Let m = length(ρ);
2start ← 1, end ← m;
3 k ← (start + end)/2;
4 h ← 0;
5 while k ≠ h do
6 Let ρ[k] *be* $(eq_k, < S_k, \sigma_k, \varepsilon_k >)$;
7 if eq_k has no solution then
8 end ← k;
9 else
10 start ← k;

```
        h ← k;
        k ← (start+end)/2;
    m′ ← end;
    Let ρ[m′]be (eq, < S, σ, ε >).
    while ε ≠ ∅ do
        Let top(ε) be < S, σ >.
        Seg ← {S} ∪ Seg;
        pop(ε);
    i ← m′;
    while i > 1 do
        Let ρ[i] be (eq_i, < S_i, σ_i, ε_i >);
        Seg ← {S_i} ∪ Seg;
        i ← i − 1;
    return (ρ[1, m′], Seg)
```

考虑到样本集的程序结构以及分类类域的交叉比较多，根据样本集主要源于测试用例库中的测试集，而由于测试用例库的分级标准使得用例结构之间具有一定的相似程度，因此选择了 KNN 分类器；由于样本结构的复杂性，因此样本集并不是线性可分的，这需要将其转换为高维线性可分数据集，这里又选择了 SVM 算法。

SVM 算法这里主要借助于 LIBSVM [5]来实现 SVM 算法。而对 kNN [6]算法进行了 C++编码实现，首先对数据进行预处理，建立数据结构存储训练数据和测试元组，并设定参数 k 的值，然后创建一个优先级队列，大小为 k，按照其距离的大小存储近邻的训练数据。首先，从已有的训练数据中随机抽取 k 个数据作为初始化的最近邻样本，然后分别进行这 k 个训练样本到测试样本的距离，然后将该训练样本的编号和计算得出的距离存入优先级队列。对训练数据集进行遍历并计算当前训练样本和测试样本的距离，所得的距离 L 与优先级队列中最大的距离 Lmax 进行比较，如果 L ≥ Lmax，则舍弃该训练样本，继续遍历；如果 L < Lmax，则将优先级队列中 Lmax 所对应的样本删除，然后将当前的训练样本加入优先级队列中，不断重复上述过程，当所有样本遍历完成之后，则对于优先级队列中的 k 个训练样本的所属类别进行统计，选择其中的多数类，将其作为该测试样本的所属类别。当测试数据集测试完成之后计算其误差率，并且继续修改 k 的值，并重新开始训练，最后选取误差率最小的 k 值。

3. 分类预测流程图

分类预测流程图如图 1 所示。在准备工作阶段主要是确定属性特征，在数据集上生成相应的训练样本。训练过程是通过机器学习的相关算法对训练样本集进行训练。预测过程是根据训练的结果对测试样本集进行预测，得到预测的分类结果。

4. 训练数据

由于没有现成的数据集，因此需要构建测试用例和训练数据集，根据第六章所建立起的用例库，通过实现的算法对其定位，分析其定位结果，根据上述的分类的标准，对定位结果进行分类，建立训练数据集。

4.1. 输入空间、输出空间

本系统的输入空间和输出空间，也即输入和输出的可能取值的集合。这里根据分类标准和选择，输出空间主要是分类的类别标识，根据确立的分类标准，这里的标识也就是“1”、“2”、“3”，而输入空间这里假设与特征空间为相同空间，对其不予区分。

4.2. 分类标准

分类预测首先需要确立分类的标准。考虑到多次定位的目的是缩小定位范围，因此分类预测的目的

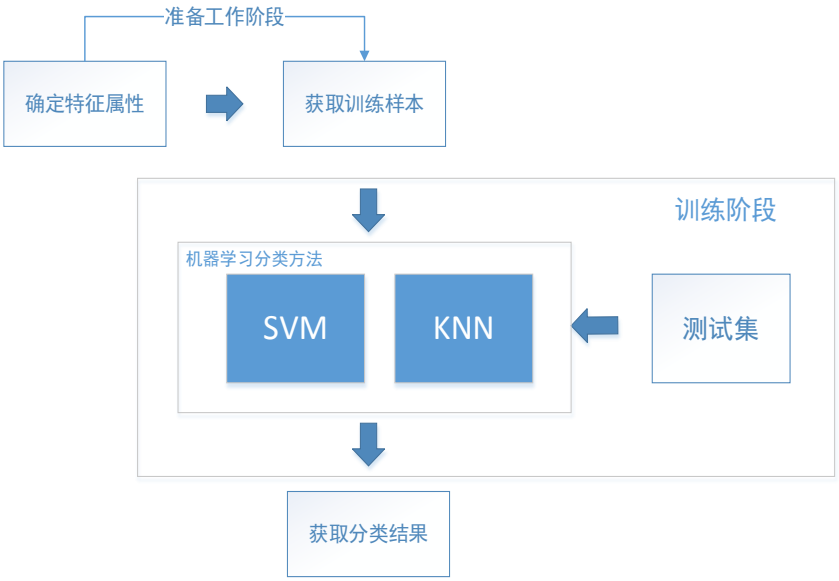

Figure 1. Flowchart of classification and prediction
图 1. 分类预测流程图

是找到使得二次定位或者多次定位出的结果定位范围较小的变量。

本文所采用的是根据定位范围缩小的百分比来进行分类，也即对于从交互所获取的期望值或者确认值来说，定位范围比首次定位的范围缩小的百分比作为分类的标准。这里的定位范围是以定位结果覆盖的语句数来进行计算。

也即程序的首次定位出的语句数为，而对于某条路径上的一个变量，根据它的期望值或者正确信息确认，进行的再次定位出的语句片段的语句数为，那么范围缩减率 $p=\frac{n_1-n_2}{n_1}\times 100\%$，根据 p 的大小平均分为十类，也即 10%以下为第一类，标签为 1；10%~20%为第二类，标签为 2,；20%~30%以上为第三类，标签为 3；以此类推。

这里分类的含义即变量所属类别标签越小，代表在首次定位的基础上，该变量获取期望值或者对该变量值进行确认之后进行的二次定位效果越好，也即定位出的包含错误语句片段范围越小，易于找到出错的语句。

4.3. 特征空间

这里本系统的特征空间是高维空间，每一维度都对应一个特征。由于输入空间和特征空间相同，因此本系统的学习模型是建立在特征空间基础上的。

特征空间中存在的特征向量是学习模型的重点。在其中特征选择是一个重要问题。对于一个训练集，每个记录包含两个部分，一是特征空间的取值，二是该记录的分类标签。一般情况下，机器学习选择特征的方式有两种，一是在原有特征基础上创造新特征，二是从原有特征中筛选出无关或者冗余特征，将其去除后保留一个特征子集。这里我们采取了第一种方法。

首先，对于构建的测试用例，能够表现用例结构的特征有以下几点：条件语句数、循环语句数和赋值语句数。经过一个测试集的测试，两种算法的预测结果均表现一般，因此我们在原有特征基础上创造出新的特征，如条件和循环语句嵌套层数，独立的赋值语句数，条件环境中的赋值语句数，循环环境中的赋值语句数，条件循环环境中的赋值语句数等，构成了以下几个特征集。

第一特征集：条件语句数、循环语句数、赋值语句数。

第二特征集：条件语句数、循环语句数、赋值语句数、条件和循环语句嵌套层数。

第三特征集：条件语句数、循环语句数、独立的赋值语句数、条件环境中的赋值语句数、循环环境中的赋值语句数。

第四特征集：条件语句数、循环语句数、独立的赋值语句数、条件循环环境中的赋值语句数。

通过一个测试集的测试，结果如表 1。

因此，选择第四特征集，对测试用例利用已实现的算法进行定位，并且计算各个特征的数值以及分类结果，建立有效的训练数据。

4.4. 训练数据

根据测试用例库的分级效果，将测试用例库分为训练集、测试集和交叉验证集。样本集占测试用例库的 60%，测试集和交叉验证集分别占 20%。这里根据测试用例库中的用例分类，分别选取不同的测试用例作为训练数据的源，形成三个不同的样本集。

样本集 1 的源代码主要是从测试用例库中每个三级类别里选择该类别其中五分之三数目的测试用例的源文件，然后生成相应的训练数据，形成样本集 1。

样本集 2 的源代码主要是从测试用例库中随机选择不同的几个三级类别，每个类别随机选取该类别三分之一到三分之二数目的测试用例，形成样本集 2。

样本集 3 的源代码是从测试用例库中随机选择一个三级类别，选择该类别的测试用例，然后生成相应的训练数据，形成样本集 3。

由类似上述生成样本集的方法生成相应的测试集 1、2 和交叉验证集。

这里系统对于这些测试用例生成训练数据是通过 traindata 函数来实现的，主要是对于一个测试用例而言，其中各个路径对应的环境状态信息表中的变量或者由于源代码程序中的错误导致产生用户期望值，或者变量取值是正确的，我们通过对测试用例中的这些变量手动输入用户的反馈信息，也即变量的期望值或者正确信息的确认，从而进行二次定位，将其定位结果范围与程序首次定位的结果相比较，将其语句片段计算出语句数，也即定位范围大小，根据二次定位相比首次定位出的结果缩减的语句数所占首次定位语句数的百分比，也就是根据分类标准来进行类别信息的生成，而特征向量的生成主要是通过对于所有这些变量，根据其路径语序，倒数计算其经过的条件语句数、循环语句数、独立的赋值语句数、条件环境中的赋值语句数和循环环境中的赋值语句数，从而生成训练数据。

5. 结果分析

5.1. SVM 参数选择

在实现算法的过程中，最优 SVM 算法参数选择需要大量实验对比、大范围的搜寻或者利用软件包提供的交互检验功能进行寻优。本文主要是通过样本集和测试集测试并进行参数选择，做了大量实验。

其中包含交叉验证实验，主要是确定效率最大的参数 c 和 g。这里用到了 LibSvm 里的 k 折交叉验证，是将训练样本平均分成 k 份，每次拿出 k-1 份作为训练数据，剩下的一份作为测试数据，这样重复做 k 次，获得 k 次的平均交叉验证准确率作为结果，这里 k 取 5。经过实验得到了效率最大的参数 c 为 16，g 为 0.130479。

经过实验结果的对比分析，参数选择如表 2。

对已建立的测试用例库，包含多种错误类型的测试集，通过 SVM 的学习和预测，准确率达到 85%左右。

5.2. KNN 参数选择

通过样本集进行测试，并对 k 选取不同的取值，分别计算其分类的准确率，也即准确 = 1 − 误差率。

Table 1. Analysis of feature set
表 1. 特征集分析

特征集	准确率 Accuracy	分析
1	75.75%(+/−8.23%)	原始特征，表现一般
2	81.79%(+/−6.09%)	增加了条件和循环语句嵌套层数这个特征，准确率提升不大，说明该特征不宜采纳
3	90.56%(+/−5.23%)	准确率高，增加的特征有效
4	90.91%(+/−4.44%)	与第三特征集准确率相差不大，但是特征数目少，处理开销小，模型较好

Table 2. Table of model parameter
表 2. 模型参数表

参数	值	意义
svm_type	c_svc	训练采用的模型为 C-SVC
kernel_type	rbf	训练采用的核函数类型为 RBF 核
cost	16	C-SVC 的损失函数参数
gamma	0.130479	gamma 是 RBF 的系数
nr_class	10	分类时的类别数，此处为三分类问题

通过图 2 的实验结果，找到了实验效果相对较好的参数 k = 8。而由于 K 最近邻方法的特性，在进行分类时主要依据周围相邻的少量样本点，并不是依靠判别类域的方法来确定所属类别。考虑到由于本系统的样本集主要是通过系统的测试用例集进行特征向量生成而产生的，尤其样本集 3 其中涉及到的很多测试用例所包含的数据类型以及程序的结构特征都会具有一定程度的相似性，分类类域也有一定程度的交叉，因此 KNN 方法比较适合这类样本及测试集。

5.3. 分析

在同样的样本集和测试集上，对 SVM 和 KNN 这两种方法进行对比分析(图 3)。

由于测试用例库的分类标准使得测试用例具有较强的交叉性和相似性，因此 kNN 在同类的测试用例中比 SVM 表现良好，分类的准确度可以达到 90%以上，效果显著，但是这种方法计算量较大，学习预测的开销较大。在给定一个新的测试用例的情况下，通过首次定位的结果以及特征向量的计算，对比 SVM 和 KNN 预测的分类结果，选择分类类别为 3 的变量(如果没有 3 则选择 2，以此类推)。然后将信息提示给用户，引导用户尽可能地给出这些变量的期望值，从而使得用户给出最为有效的反馈信息，提高错误定位的效率。

5.4. 对比实验

考虑到基于覆盖信息的错误代码定位 CBFL 技术是现今错误错位研究的热点之一，本文从中选取了三种知名的技术 Tarantula [7]、Naish [8]、Muffler [9]技术，并选取 Siemens 程序集中的程序来进行 Tarantula 系统、Naish 系统、Muffler 系统和本系统的实验对比。

CBFL 技术主要是通过收集程序的不同测试用例的覆盖信息，计算每条语句的可疑度，给出疑似度由高至低的语句排序列表，从而帮助调试人员进行错误定位。而本系统主要是执行测试用例并记录执行轨迹信息，根据期望值逆向生成方程组并找到使得方程组无解的第一个语句，从而给出包含错误语句的程序片段，再根据多次交互定位缩小定位范围，通过反馈信息的引导策略减少交互定位次数，提高定位效率。

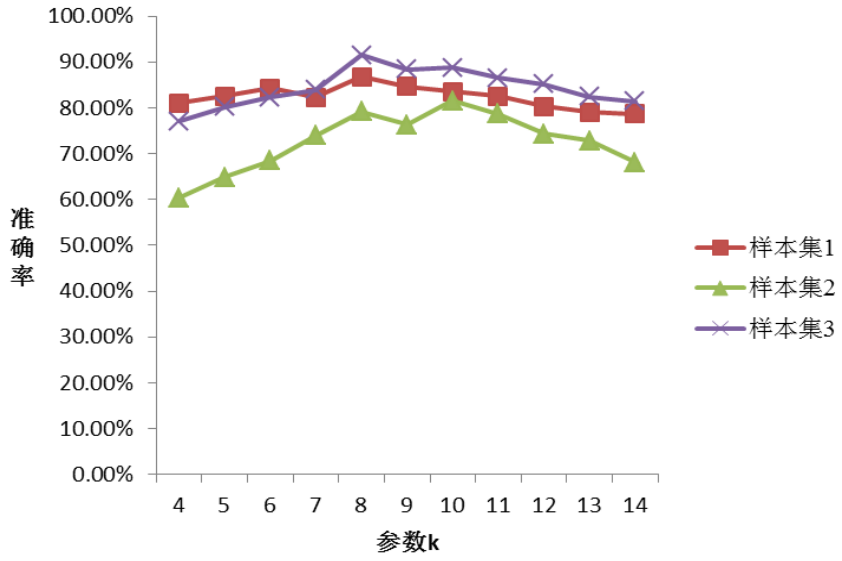

Figure 2. Selection of parameter k
图 2. KNN 参数 k 选择

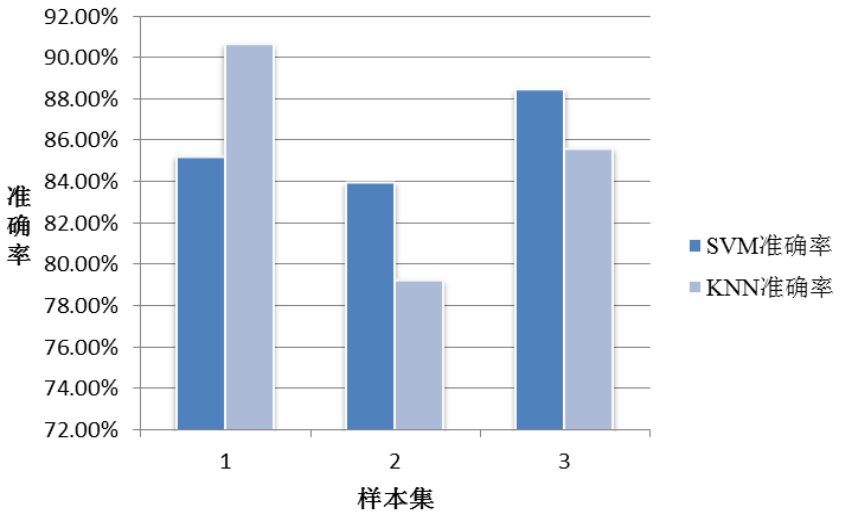

Figure 3. A comparative analysis of the accuracy of the two algorithms
图 3. 两种算法准确率对比分析

综合考虑这些技术的特性，这里考虑系统提供给调试人员的需要检查的语句所占源代码程序的百分比，也即系统错误定位后用户需要检查的语句集 S_{loc} 的数目 n_{loc} 占所有语句 S 数目 n_{total} 的比例 $p=(n_{loc}/n_{total})\times 100\%$。

通过在不同的测试用例(图 4)上的实验表明，本系统平均表现良好，首次定位的效果相较于 Tarantula [8]和 Naish [9]来说较好，平均需要检查 19%的代码。三种 CBFL 技术中 Muffler 的有效性最好，平均需要检查 15%。本系统并且由上述实验可以看出，本系统通过反馈信息的引导策略，实现了有效的多次交互定位，这是对系统错误定位算法实现的优化，使得能够更加有效地缩减定位范围，使得需要调试人员检查的代码比例在 13%左右，效果显著。

本系统采用的基于操作语义的形式化错误定位算法，相较于 CBFL 技术而言，对于一个测试程序，CBFL 在计算语句的可疑度时对于不同的测试用例的输入依赖性较强，容易受到偶然性成功测试用例的

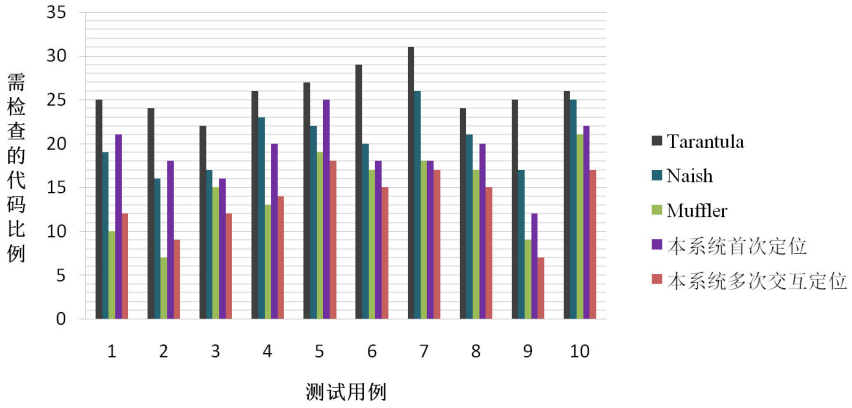

Figure 4. Effectiveness analysis of CBFL and this system
图 4. CBFL 和本系统有效性对比

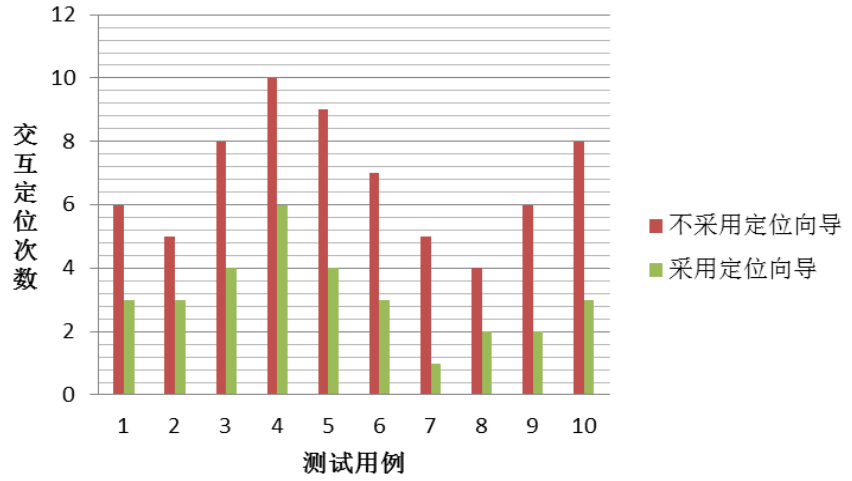

Figure 5. Correlation analysis of interaction times and fault location guide
图 5. 交互次数与定位向导的相关性分析

影响，给出的根据可疑度排序的语句序列可能由于偶然性测试用例的影响而产生误报。而本系统则不依赖于多种测试用例的收集，而是依赖于程序结构特征。基于操作语义的形式化错误定位算法通过理论证明保证了定位出的是包含错误语句的极小程序片段，这使得系统错误定位的准确率得到保证。

系统通过引导策略实现了定位向导功能，这里通过统计系统应用过程中是否采用定位向导对交互定位的次数来分析定位向导的效果。考虑交互定位会不断缩小定位范围，也即缩小需要用户检查的代码百分比，这里统计交互定位直到需要检查的代码小于 7%所需要的交互次数，分别对系统不采用定位向导和采用定位向导进行实验。

由图 5 可以看出，采用定位向导之后有效降低了所需的交互定位的次数。这说明定位向导可以正确的引导用户给出有效的期望值，使得用户不必再依次查看定位片段的各个语句中的变量值，可以根据向导给出的提示来进行反馈，进一步降低了对于人工的依赖程序，减轻调试人员的工作量，提高了错误定位的效率。

参考文献 (References)

[1] Silva, P., Moreno, A.M. and Peters, L. (2015) Software Project Management: Learning from Our Mistakes. *IEEE Software*, **32**, 40-43. http://dx.doi.org/10.1109/MS.2015.71

[2] Perscheid, M. and Hirschfeld, R. (2014) Follow the Path: Debugging Tools for Test-driven Fault Navigation. Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE). *IEEE Conference on Software Evolution Week.* Victoria, City of Gardens, British Columbia, Canada, 2014, 446-449.

[3] Liang, X., Mao, L.Q. and Huang, M. (2014) Research on Improved the Tarantula spectrum Fault Localization Algorithm. 2*nd International Conference on Information Technology and Electronic Commerce* (*ICITEC*), Dalian, China, 2014, 631-654.

[4] Digiuseppe, N. and Jones, J.A. (2011) On the Influence of Multiple Faults on Coverage-Based Fault Localization. *Proceedings of the* 2011 *International Symposium on Software Testing and Analysis*, Toronto, Canada, 2011, 210-220. http://dx.doi.org/10.1145/2001420.2001446

[5] Mathur, A. and Foody, G.M. (2008) Multiclass and Binary SVM Classification: Implications for Training and Classification Users. *IEEE Geoscience & Remote Sensing Letters*, **5**, 241-245. http://dx.doi.org/10.1109/LGRS.2008.915597

[6] Adeniyi, D.A., Wei, Z. and Yongquan, Y. (2014) Automated Web Usage Data Mining and Recommendation System Using K-Nearest Neighbor (KNN) Classification Method. *Applied Computing & Informatics*, **2**, 36-38.

[7] Digiuseppe, N. and Jones, J.A. (2015) Fault Density, Fault Types, and Spectra-Based Fault Localization. *Empirical Software Engineering*, **20**, 928-967. http://dx.doi.org/10.1007/s10664-014-9304-1

[8] Xu, Q., Pei, Y. and Wang, L. (2013) An Evaluation Framework of Coverage-Based Fault Localization for Object-Oriented Programs. Trustworthy Computing and Services, Springer Berlin Heidelberg, Berlin, 591-597.

[9] Wong, W.E., Debroy, V. and Choi, B. (2010) A Family of Code Coverage-Based Heuristics for Effective Fault Localization. *Journal of Systems & Software*, **83**, 188-208. http://dx.doi.org/10.1016/j.jss.2009.09.037