基于谓词执行信息分析的自适应缺陷定位算法

郝 鹏[1] 郑 征[1] 张震宇[2] 高乙超[1] 宫 成[1] 薛云志[3]

[1](北京航空航天大学自动化科学与电气工程学院 北京 100191)

[2](中国科学院软件研究所 北京 100190)

[3](中国科学院软件研究所基础软件测评实验室 北京 100190)

摘 要 查找程序中缺陷代码所在的位置是一项值得深入开展的研究,同时也是实际软件调试过程中所面临的一个难题,这一过程往往需要耗费大量的时间和人力资源.研究软件缺陷定位的一类重要方法是基于谓词的统计学缺陷定位方法(简称PBSFL).PBSFL通过比较程序运行成功与失败时谓词的执行信息差异来获得谓词与缺陷的关联程度.然而实验研究发现,固定算法中信息利用的强度会造成信息利用不足或过分利用现象的发生,导致现有PBSFL方法对某些缺陷定位不够准确.针对这一问题,文中设计了一种基于谓词执行信息分析的自适应缺陷定位算法,该算法通过分析测试用例运行时谓词的执行情况来动态地为每个谓词选择合适的信息利用强度.实验结果表明,该方法在Siemens和space两个程序包上表现出很好的定位效果以及定位稳定性.

关键词 统计学缺陷定位;谓词执行信息;自适应;软件测试;程序分析

中图法分类号 TP311 **DOI号** 10.3724/SP.J.1016.2013.00500

Self-Adaptive Fault Localization Algorithm Based on Predicate Execution Information Analysis

HAO Peng[1] ZHENG Zheng[1] ZHANG Zhen-Yu[2] GAO Yi-Chao[1] GONG Cheng[1] XUE Yun-Zhi[3]

[1](*School of Automation Science and Electrical Engineering, Beihang University, Beijing* 100191)

[2](*Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[3](*Laboratory for Fundamental Software Testing and Evaluation, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract Finding the location of a fault in code is an important research and practical problem, which often requires much time and manual effort. Predicate-based statistical fault localization (PBSFL) is a promising method, which obtains the correlative relationship between predicates and faults by comparing the predicate execution information in both correct and incorrect runs. However, experiment results show that existing PBSFL methods fail to locate some faults because they use predicate execution information in a fixed intensity, which may cause insufficient or excessive usage. To solve the problem, we propose a new method, called self-adaptive fault localization algorithm based on predicate execution information analysis, which dynamically select the intensity of information utilization for each predicate through the analysis of test cases run. Experimental results demonstrate that our approach performs well in both accuracy and stability for localizing faults in subject programs of the Siemens and space suites.

Keywords statistical fault localization; predicate execution information; self-adaptive; software testing; program analysis

---

收稿日期:2011-10-21;最终修改稿收到日期:2013-11-20.本课题得到国家自然科学基金(60904066,61003027)、国家科技重大专项经费(2012ZX01039-004)资助.**郝 鹏**,男,1986年生,硕士研究生,主要研究方向为软件缺陷定位、软件测试.E-mail:haopeng24@126.com.**郑 征**(通信作者),男,1980年生,博士,副教授,主要研究方向为智能决策、软件测试.E-mail:zhengz@buaa.edu.cn.**张震宇**,男,1979年生,博士,副研究员,主要研究方向为软件测试、软件调试等.**高乙超**,男,1988年生,硕士研究生,主要研究方向为软件缺陷定位、软件测试.**宫 成**,男,1987年生,硕士研究生,主要研究方向为软件缺陷定位、软件测试.**薛云志**,男,1979年生,博士,副研究员,主要研究方向为自动化软件测试、编译优化.

1 引　言

现代软件和软件系统中仍会不可避免地存在缺陷，而软件调试则是一个查找软件缺陷的有效手段. 为了定位并且修复软件缺陷，一种常用的做法是调试失效的程序. 由于手工调试存在耗时、易错等缺点，因此研究人员一直致力于开发自动化或半自动化的软件缺陷定位技术. 近几十年里，软件缺陷定位领域研究活跃，一批统计学缺陷定位(Statistical Fault Localization，SFL)方法被相继提出[1-14]. SFL方法通过收集大量成功与失败测试用例运行时程序元素(例如语句[1-4]、谓词[5-8]、分支[9]或信息流[10])的动态信息，然后利用这些动态信息与程序失效之间的关联性为每个程序元素计算可疑度值. 从直观上理解，如果关联性越强，该程序元素的可疑度值就越大. 最后，程序开发人员按照可疑度值由大到小的顺序依次检查程序元素的排位表，直到最终找到存在缺陷的程序元素.

尽管SFL方法可以取得很好的定位效果，但是该类方法的有效性仍会受一些因素的影响. 例如，在之前的研究中，我们曾指出，基于谓词的统计学缺陷定位(Predicate-Based Statistical Fault Localization，PBSFL)方法的定位效果会受谓词执行信息量多少的影响[8]. 对谓词执行信息利用不足或过分利用都会削弱缺陷相关谓词与程序失效之间的关联性，造成PBSFL方法无法有效区分缺陷谓词与非缺陷谓词，从而影响定位精度. 谓词执行信息量是指程序运行时谓词实际可以提供信息的数量，谓词提供的信息包括谓词分别判断为“真”或“假”的次数以及判断的顺序. 由于程序执行剖面不同，导致不同谓词在每次程序运行时提供信息的数量也存在着差异. 从直观上理解，谓词执行的次数越多，可以提供的信息数量也就越多. 谓词执行信息利用强度是指对谓词执行信息提取多与少的衡量. 如果提取的信息量超过谓词实际可以提供的信息数量，则会造成过分利用谓词执行信息的现象；相反，则会导致信息利用不足. 现有PBSFL方法对程序中所有插桩谓词都采用统一且固定的信息利用强度，无法做到根据程序实际运行情况充分利用谓词的执行信息，这也促使我们对如何有效利用谓词执行信息来提高PBSFL方法的定位精度以及适应性进行深入研究.

在本文中，我们提出一种新的谓词执行信息利用方法，该方法根据测试用例运行时每个谓词的执行情况，动态地选择适合于每个谓词的信息利用强度. 因为不同测试用例运行时程序的执行谱不同，导致程序中谓词的执行情况也不尽相同，最直观表现为不同谓词在相同测试用例运行时取值为“真”或“假”的顺序以及次数不同. 如果对所有谓词均采用相同的信息利用强度，则会不可避免地对某些谓词存在信息利用不足或过分利用的情况. 只有充分利用每个谓词的执行信息，才可能达到提高算法定位精度的目的，因此需要根据每个谓词执行信息量的变化有针对性地选择信息利用强度，即让算法的信息利用强度适应于谓词执行信息量的变化.

为了验证这一想法，我们提出一种基于谓词执行信息分析的自适应缺陷定位算法(Self-adaptive Fault Localization Algorithm based on Predicate execution information Analysis，以下简称Safla). 与现有PBSFL方法的流程类似，Safla也是先收集缺陷程序运行时谓词的执行信息，然后进行信息的提取和转换，再根据缺陷关联度公式计算每个谓词的可疑度，最后给出谓词可疑度的排序. 与现有方法固定信息利用强度的做法不同的是：Safla采取分析失败测试用例运行时谓词的执行情况，动态地选择适合于每个谓词的信息利用强度，确保谓词执行信息的利用更加充分有效.

为了验证Safla的定位效果，我们选取CBI[5]、SOBER[6]、Mann-Whitney、Wilcoxon、F-test、t-test[7]以及N bit-Pesla[8]算法进行对比，并在Siemens和space程序包上进行实验. 实验结果表明，Safla在Siemens和space两个程序包上表现出很好地定位效果以及定位稳定性.

本文贡献总结如下：

(1) 提出自适应选择信息利用强度的思想，并设计了一种具体的选择方法.

(2) 基于上述思想，提出了一种新的基于谓词的统计学缺陷定位方法.

本文第2节简单介绍基于谓词的统计学缺陷定位方法的研究现状；第3节给出研究动机；第4节对Safla的基本思想和详细步骤进行介绍；第5节给出Safla分别在Siemens和space程序包中进行实验的结果，并与多个典型算法进行比较分析，得出结论；第6节对实验结果的有效性进行讨论；第7节是对本文的总结和展望.

2 相关研究

近十年里，关于使用统计学方法进行缺陷定位

已有大量的研究.统计学缺陷定位方法通常要求具备3个要素:大量测试用例、相应测试用例的执行剖面以及测试用例的运行结果(成功或失败).其中,执行剖面包含测试用例经过每条可执行语句、分支或插桩谓词时的覆盖信息.由于本文所讨论的方法属于基于谓词的统计学缺陷定位方法范畴,因此本节主要对现有的几种基于谓词的统计学缺陷定位方法进行简单的介绍.

CBI[5]算法提出者认为,程序运行失败与程序中缺陷谓词被判断为"真"有关.因此,该算法建立的统计学模型是比较程序运行成功与失败时相应谓词被判断为"真"的概率.若概率数值差异越大,则认为该谓词越可疑.

在此之后提出的SOBER[6]算法弥补了CBI模型上的一些缺陷,如CBI模型没有考虑分布情况,而只是单纯地将谓词的两个不同期望值相减.若遇到两种分布具有相同的期望而方差不同的情况时,CBI算法将无法发挥很好的定位作用.SOBER模型通过比较谓词evaluation bias(程序一次运行过程中谓词被判断为"真"的概率)在成功与失败分布中的差异来寻找缺陷相关谓词.实验证明,考虑分布情况的SOBER算法的定位效果要优于CBI算法.

但后来大量实验表明,SOBER模型中谓词在成功与失败测试用例下服从正态分布的假设并不合理,这也直接影响了SOBER算法的最终定位效果.于是,Zhang等人[7]提出,在总体分布未知的前提下,应使用统计学中非参数假设检验法来比较谓词evaluation bias在成功与失败两类分布中是否相同.实验结果显示,非参数假设检验法能定位到更多的缺陷.

在此之前的研究中,我们曾提出Nbit-Pesla算法[8],并在实验中证明,2bit-Pesla算法的定位精度要明显优于SOBER算法和非参数假设检验法,但扩展算法(例如:3bit-Pesla、4bit-Pesla等)的定位精度反而呈现出下降的趋势.为此我们提出一种解决办法,采取对不同缺陷版本程序使用不同"N"值的Nbit-Pesla算法进行计算,最终定位效果获得了提高.但是该方法对同一程序中的所有谓词采用相同的信息利用强度,对一些谓词执行信息的利用仍然不够合理,因此定位精度的提高有限.

3 研究动机

在本节中,我们在对Nbit-Pesla算法基本思想进行讨论的基础上,归纳出本文的研究动机.

假如谓词P在两个测试用例下的执行情况分别为"10011100"和"10001101",P判断为"真"记为"1",判断为"假"记为"0".可以发现,在上述两次运行中,谓词判断为"真"和判断为"假"的次数是相同的,但其判断顺序不同.SOBER算法虽然比较了这两次执行中P被判断为"真"和"假"的次数,但是忽略了判断顺序,因而无法找到其中的差别.相比之下,考虑谓词判断顺序的Nbit-Pesla算法则可以发挥很好的定位作用,有关该算法的详细描述参见文献[8].然而,该算法没有考虑谓词执行信息量多少与算法信息利用强度大小这两者之间是否能够动态匹配的问题,导致在某些情况下定位精度不高.

为此,我们将借助Siemens程序包中的一个实例程序,从以下两个角度阐述本文的研究动机:

(1)程序运行时谓词提供信息的能力存在着差别.

这主要表现在两个方面:① 不同谓词在同一次运行中提供信息的能力不同;② 同一谓词在不同运行中提供信息的能力也不同.

图1是"replace"程序包v17版本的部分代码,共包含5个谓词.其中,缺陷位于第7行,右侧注释为正确的代码.

```
L1:  char esc(char *s, int *i){
L2:      char result;
L3:      if(s[*i]!='@')                                  P1
L4:          result=s[*i];
L5:      else{
L6:          if(s[*i+1]=='\0')                           P2
L7:              result=10; /*fault: result='@'*/
L8:          else{
L9:              *i= *i+1;
L10:             if(s[*i]=='n')                          P3
L11:                 result=10;
L12:             else{
L13:                 if(s[*i]=='t')                      P4
L14:                     result=9;
L15:                 else
L16:                     result=s[*i]; }}}
L17:     return result; }                                P5
```

图1 replace程序包中v17版本的部分代码

上述各谓词在所有成功和失败测试用例下的执行情况分别如图2和图3所示.为了便于展示,我们对执行次数采取以2为底的对数运算.其中,"Max"表示最大的执行次数,"Min"表示最小的执行次数,"Mean"表示执行次数的平均值.从图中我们可以直观地看到,不同谓词之间执行情况不同.例如P_2和P_5,无论是在成功测试用例还是在失败测试用例下,这两个谓词的执行情况都相差较大.

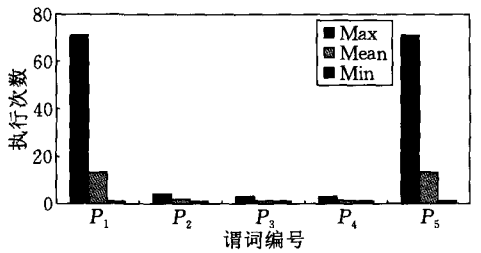


图 2　各谓词在成功测试用例下的执行次数

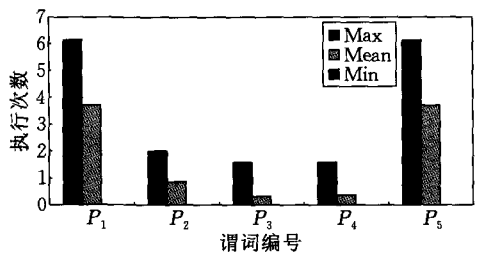


图 3　各谓词在失败测试用例下的执行次数

即使是同一个谓词，在不同测试用例下的执行情况也存在着很大的差异. 例如，图 4 总结了谓词 P_5 在所有 24 个失败测试用例下的执行次数情况(横坐标表示测试用例号，纵坐标表示执行次数). 可以看到，执行次数分布跨越的范围很大.

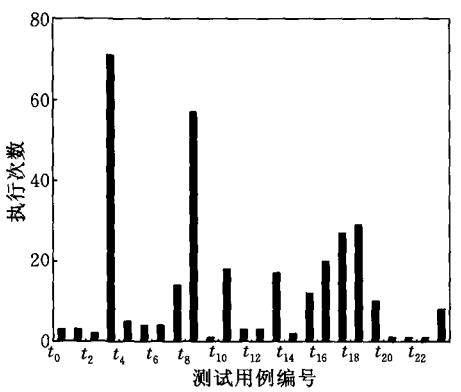


图 4　P_5 在所有失败测试用例下的执行次数

如果以执行次数多少来衡量谓词提供信息的能力，可以看到，由于程序执行剖面不同，不同谓词在实际程序运行时提供信息的能力存在着差别.

(2) 对所有谓词采用固定信息利用强度不合理.

通过分析图 1 中的程序，我们发现，谓词 P_2 和 P_5 与缺陷直接相关，我们希望缺陷定位方法能够将这两个谓词都排在排位表中相对靠前的位置. 在使用 Nbit-Pesla 算法进行缺陷定位($N=1,2,3,4,5$)时，1bit-Pesla 算法或 2bit-Pesla 算法对 P_2 的定位效果最优(均将 P_2 排在第 1 位). 但是，无论是 1bit-Pesla 算法还是 2bit-Pesla 算法对 P_5 都无法取得很好的定位效果(1bit-Pesla 算法将 P_5 排在第 57 位，2bit-Pesla 算法将 P_5 排在第 63 位). 相反，增大算法中信息利用的强度，使用 5bit-Pesla 算法反而可以取得最优的定位效果(将 P_5 排在第 33 位).

通过以上实验结果，我们可以发现，在谓词提供信息能力有强弱差别的前提下，按照之前做法对所有谓词采用相同的信息利用强度是不合理的，这将会造成信息的浪费或信息冗余，降低缺陷相关谓词与程序失效之间的联系，从而影响算法的定位精度. 因此，针对每个谓词选择合适的信息利用强度是十分必要的.

此外，由于程序运行之前很难对谓词的执行情况进行静态分析，如何提高算法对不同谓词执行情况的适应性也是一个需要解决的难题. 在本文中，我们通过对测试用例运行后每个谓词的执行情况进行分析，得到谓词执行次数分布情况来动态地选择适合于每个谓词的信息利用强度.

4　基于谓词执行信息分析的自适应缺陷定位算法

4.1　基本定义

定义 1(谓词执行序列).　设 P_i 是程序中第 i 个谓词，在使用一个测试用例 t_l 来执行程序时，谓词 P_i 的执行序列是一个 n 维向量，可表示为

$$\boldsymbol{S}(P_i,t_l)=(o^1_{P_i,t_l},o^2_{P_i,t_l},\cdots,o^n_{P_i,t_l}),$$

其中，$o^j_{P_i,t_l}$ 为测试用例 t_l 执行过程中，程序第 j 次($j\in[1,n]$)运行到谓词 P_i 所在语句时该谓词的取值，即

$$o^j_{P_i,t_l}=\begin{cases}0, & P_i\text{第 }j\text{ 次取值为假}\\ 1, & P_i\text{第 }j\text{ 次取值为真}\end{cases}.$$

定义 2(信息利用强度).　谓词 P_i 的谓词执行序列 $\boldsymbol{S}(P_i,t_l)$ 的第 j 个 $\omega(\omega\leq n)$ 维连续子向量称为 $\boldsymbol{S}(P_i,t_l)$ 的第 j 个长度为 ω 的子序列，表示为

$$\boldsymbol{S}^j_\omega(P_i,t_l)=(o^j_{P_i,t_l},o^{j+1}_{P_i,t_l},\cdots,o^{j+\omega-1}_{P_i,t_l}),$$

其中 ω 称为信息利用强度.

例如，执行序列 $\boldsymbol{S}(P_i,t_l)=(1,1,1,0,1)$，其子序列长度可为 1 到 5，即其信息利用强度可取 1 到 5. 在已有方法中，SOBER 算法对于信息的利用强度为 1，Nbit-Pesla 算法对信息的利用强度为 N.

定义 3(值集合).　长度为 ω 的子序列共有 2^ω 种取值，其集合 $\{v_1,v_2,\cdots,v_{2^\omega}\}$ 称为值集合.

上例中长度为 2 的子序列可能的取值为(0,0)，(0,1)，(1,0)，(1,1)，则值集合为

$\{(0,0),(0,1),(1,0),(1,1)\}$.

定义 4(执行信息向量). 对于执行序列$\boldsymbol{S}(P_i,t_l)$,信息利用强度为ω时,向量$\boldsymbol{N}^{\omega}_{P_i,t_l}=(n_1,n_2,\cdots,n_{2^\omega})$,其中$n_k=$size of $\{j|\boldsymbol{S}^j_\omega(P_i,t_l)=v_k\}$. 将$\boldsymbol{N}^{\omega}_{P_i,t_l}$归一化后的向量称为执行信息向量,表示为$\boldsymbol{E}^{\omega}_{P_i,t_l}$.

在上例中,可以根据定义 4 得到$\boldsymbol{N}^2_{P_i,t_l}=(0,1,1,2)$,$\boldsymbol{E}^2_{P_i,t_l}=\left(0,\frac{1}{4},\frac{1}{4},\frac{1}{2}\right)$.

需要注意的是:

(1) 如果选择的信息利用强度ω大于执行序列的长度n,执行序列与值集合中对应取值相比,将会有$\omega-n$位的取值未知. 而由于每个未知位取值为"0"或"1"的概率各占 0.5,因此,在这种情况下,执行信息向量中相应位置的$n_k=\frac{1}{2^{\omega-n}}$.

(2) 如果谓词在某个测试用例运行时没有被执行到,则由该测试用例得到的执行信息向量中各位均为零.

定义 5(缺陷关联度). 谓词P_i的缺陷关联度定义为在信息利用强度ω下,所有成功测试用例中P_i的执行信息向量总和均值$\overline{ES}^{\omega}_{P_i}$与所有失败测试用例中$P_i$的执行信息向量总和均值$\overline{EF}^{\omega}_{P_i}$之间的距离,用 2-范数计算为

$$R_{P_i}=\|\overline{ES}^{\omega}_{P_i}-\overline{EF}^{\omega}_{P_i}\| \tag{1}$$

其中,$\overline{ES}^{\omega}_{P_i}=\frac{1}{TotalPass}\sum_{l=1}^{TotalPass}\boldsymbol{E}^{\omega}_{P_i,t_l}$,$\overline{EF}^{\omega}_{P_i}=\frac{1}{TotalFailed}\sum_{l=1}^{TotalFailed}\boldsymbol{E}^{\omega}_{P_i,t_l}$,$TotalPass$表示所有成功测试用例,$TotalFailed$表示所有失败测试用例.

4.2 两个性质及其证明

在本节中,我们将在提出并证明两个性质的基础上,对谓词执行信息量和信息利用强度之间的关系进行一定程度的理论分析.

性质 1. 假设谓词P_i在失败测试用例下执行序列的长度均为n位,并且执行序列都相同,而成功测试用例均没有经过谓词P_i. 如果选择信息利用强度m进行信息提取,则谓词P_i的缺陷关联度R_{P_i}满足如下性质:

$R_{P_i}<1$, 当$m>n$时,

$R_{P_i}=1$, 当$m=n$时,

$R_{P_i}\leq 1$, 当$m<n$时.

证明. 我们将m和n的关系分为如下 3 种情况进行分析.

情况 1. $m>n$.

假设信息利用强度为$m=n+x(1\leq x<+\infty)$,长度为m的子序列共有2^{n+x}种取值,则失败测试用例t_l下执行信息向量$\boldsymbol{E}^{n+x}_{P_i,t_l}=(n_1,\cdots,n_{2^{n+x}})$. 其中,由于失败测试用例下执行序列的长度均为$n$位,在信息利用强度$m$下,$n+x$维子序列中共有$x$位的取值未知,按照定义 4 中等概率赋值的操作,在n_k($k\in\{1,2,\cdots,2^{n+x}\}$)的取值中,将会有$2^x$个取值为$\frac{1}{2^x}$,其余位均为 0,即$\boldsymbol{E}^{n+x}_{P_i,t_l}=\underbrace{(n_1,\cdots,n_{2^{n+x}})}_{\text{一共有}2^x\text{个}\frac{1}{2^x}\text{,其余位为0}}$. 因此,所有失败测试用谓词$P_i$执行信息向量总和均值$\overline{EF}^{n+x}_{P_i}=\frac{1}{TotalFailed}\times\sum_{l=1}^{TotalFailed}\boldsymbol{E}^{n+x}_{P_i,t_l}=\underbrace{(n_1,\cdots,n_{2^{n+x}})}_{\text{一共有}2^x\text{个}\frac{1}{2^x}\text{,其余位为0}}$.

由于成功测试用例没有经过谓词P_i,那么成功测试用例t_l下执行信息向量$\boldsymbol{N}^{n+x}_{P_i,t_l}=\boldsymbol{0}$,所以$\overline{ES}^{n+x}_{P_i}=\frac{1}{TotalPass}\sum_{l=1}^{TotalPass}\boldsymbol{E}^{\omega}_{P_i,t_l}=\boldsymbol{0}$. 代入定义 5 中可以得到,

$$R_{P_i}=\|\overline{ES}^{n+x}_{P_i}-\overline{EF}^{n+x}_{P_i}\|=\sqrt{\left(\frac{1}{2^x}\right)^2\times 2^x}=\sqrt{\frac{1}{2^x}}<1.$$

情况 2. $m=n$.

在这种情况下,长度为n的子序列共有2^n种取值,失败测试用例t_l下执行信息向量$\boldsymbol{E}^{n}_{P_i,t_l}=(n_1,\cdots,n_{2^n})$. 由于失败测试用例下执行序列的长度均为$n$位,则在$n$维子序列$2^n$种取值中,有且仅有一个取值与失败测试用例下的执行序列相对应,即$\boldsymbol{E}^{n}_{P_i,t_l}=\underbrace{(n_1,\cdots,n_{2^n})}_{\text{只有一位为1,其余位为0}}$. 因此,$\overline{EF}^{n}_{P_i}=\frac{1}{TotalFailed}\times\sum_{l=1}^{TotalFailed}\boldsymbol{E}^{n}_{P_i,t_l}=\underbrace{(n_1,n_2,\cdots,n_{2^n})}_{\text{只有一位1,其余位0}}$,而$\overline{ES}^{n}_{P_i}=\boldsymbol{0}$,那么$R_{P_i}=\|\overline{ES}^{n}_{P_i}-\overline{EF}^{n}_{P_i}\|=1$.

情况 3. $m<n$.

假设信息利用强度为$m=n-x(1\leq x\leq n-1)$,长度为m的子序列共有2^{n-x}种取值,则失败测试用例t_l下执行信息向量$\boldsymbol{E}^{n-x}_{P_i,t_l}=(n_1,n_2,\cdots,n_{2^{n-x}})$. 由于信息利用强度$m<n$,在该信息利用强度下,最多可以形成$x+1$个$n-x$维的子序列,根据定义 4,$\boldsymbol{E}^{n-x}_{P_i,t_l}=\left(\frac{n_1}{x+1},\cdots,\frac{n_{2^{n-x}}}{x+1}\right)$. 因此,所有失败测试用例中谓词$P_i$执行信息向量总和均值$\overline{EF}^{n-x}_{P_i}=\frac{1}{TotalFailed}\times\sum_{l=1}^{TotalFailed}\boldsymbol{E}^{n-x}_{P_i,t_l}=\left(\frac{n_1}{x+1},\frac{n_2}{x+1},\cdots,\frac{n_{2^{n-x}}}{x+1}\right)$,而$\overline{ES}^{n-x}_{P_i}=\boldsymbol{0}$,所以

$$R_{P_i}=\|\overrightarrow{ES}_{P_i}^{n-x}-\overrightarrow{EF}_{P_i}^{n-x}\|=\sqrt{\left(\frac{n_1}{x+1}\right)^2+\cdots+\left(\frac{n_{2^{n-x}}}{x+1}\right)^2}$$

$$=\sqrt{\frac{(n_1+\cdots+n_{2^{n-x}})^2-2\left(\sum\limits_{i,j\in\{1,\cdots,2^{n-x}\}}^{i\neq j}n_i\times n_j\right)}{(x+1)^2}},$$

又因为 $n_1+\cdots+n_{2^{n-x}}=x+1$，而 $n_j\in\{0,1,\cdots,x+1\}$，$j\in\{1,2,\cdots,2^{n-x}\}$，因此，$R_{P_i}\leq 1$.　　证毕.

从性质 1 可以看出，当 $m=n$ 时，谓词 P_i 的缺陷关联度 R_{P_i} 的计算值最大. 虽然性质 1 具有较强的假设，但是根据该性质，我们可以推断出：在谓词提供信息量固定的前提下，过分利用谓词执行信息或是对谓词信息利用不足，都将会导致缺陷谓词的可疑度计算值降低.

性质 2.　假设谓词 P_i 在失败测试用例下执行序列的长度均为 n 位，并且执行序列全部相同. 谓词 P_j 在失败测试用例下执行序列的长度均为 $n+x$ $(1\leq x<+\infty)$ 位，并且执行序列全部相同. 而成功测试用例都没有经过谓词 P_i 和谓词 P_j，如果选择信息利用强度 m 进行信息提取，则谓词 P_i 的缺陷关联度 R_{P_i} 满足如下性质：

$R_{P_i}\leq 1$，　当 $m<n$ 时，

$R_{P_i}=1$，　当 $m=n$ 时，

$R_{P_i}<1$，　当 $n<m<n+x$ 时，

$R_{P_i}<1$，　当 $m=n+x$ 时，

$R_{P_i}<1$，　当 $m>n+x$ 时.

谓词 P_j 的缺陷关联度 R_{P_j} 满足如下性质：

$R_{P_j}\leq 1$，　当 $m<n$ 时，

$R_{P_j}\leq 1$，　当 $m=n$ 时，

$R_{P_j}\leq 1$，　当 $n<m<n+x$ 时，

$R_{P_j}=1$，　当 $m=n+x$ 时，

$R_{P_j}<1$，　当 $m>n+x$ 时.

证明.　我们将 m 和 n 以及 x 的关系分为如下 5 种进行分析.

情况 1.　$m<n$.

如果信息利用强度 $m<n<n+x$，则

$$\boldsymbol{E}_{P_i,t_l}^m=\left(\frac{n_1}{n-m+1},\cdots,\frac{n_{2^m}}{n-m+1}\right),\text{且}$$

$$\boldsymbol{E}_{P_j,t_l}^m=\left(\frac{n_1}{n+x-m+1},\cdots,\frac{n_{2^m}}{n+x-m+1}\right).$$

依据性质 1 中情况 3 的讨论，同理可以推断出可疑度 $R_{P_i}\leq 1$，$R_{P_j}\leq 1$.

情况 2.　$m=n$.

如果信息利用强度 $m=n<n+x$，则

$$\boldsymbol{E}_{P_i,t_l}^m=\underbrace{(n_1,\cdots,n_{2^m})}_{\text{只有一位为1，其余位为0}},\text{且}$$

$$\boldsymbol{E}_{P_j,t_l}^m=\left(\frac{n_1}{x+1},\cdots,\frac{n_{2^m}}{x+1}\right).$$

依据性质 1 中相应情况的讨论，同理可以推断出可疑度 $R_{P_i}=1$，$R_{P_j}\leq 1$.

情况 3.　$n<m<n+x$.

如果信息利用强度 $n<m<n+x$，则

$$\boldsymbol{E}_{P_i,t_l}^m=\underbrace{(n_1,\cdots,n_{2^m})}_{\text{一共有}2^{m-n}\text{个}\frac{1}{2^{m-n}}\text{，其余位为0}},\text{且}$$

$$\boldsymbol{E}_{P_j,t_l}^m=\left(\frac{n_1}{n+x-m+1},\cdots,\frac{n_{2^m}}{n+x-m+1}\right).$$

依据性质 1 中相应情况的讨论，同理可以推断出可疑度 $R_{P_i}<1$，$R_{P_j}\leq 1$.

情况 4.　$m=n+x$.

如果信息利用强度 $m=n+x>n$，则

$$\boldsymbol{E}_{P_i,t_l}^m=\underbrace{(n_1,\cdots,n_{2^m})}_{\text{一共有}2^x\text{个}\frac{1}{2^x}\text{，其余位为0}},\text{且}$$

$$\boldsymbol{E}_{P_j,t_l}^m=\underbrace{(n_1,\cdots,n_{2^m})}_{\text{只有一位为1，其余位为0}}.$$

依据性质 1 中相应情况的讨论，同理可以推断出可疑度 $R_{P_i}<1$，$R_{P_j}=1$.

情况 5.　$m>n+x$.

如果信息利用强度 $m>n+x>n$，则

$$\boldsymbol{E}_{P_i,t_l}^m=\underbrace{(n_1,\cdots,n_m)}_{\text{一共有}2^{m-n}\text{个}\frac{1}{2^{m-n}}\text{，其余位为0}},\text{且}$$

$$\boldsymbol{E}_{P_j,t_l}^m=\underbrace{(n_1,\cdots,n_m)}_{\text{一共有}2^{m-n-x}\text{个}\frac{1}{2^{m-n-x}}\text{，其余位为0}}.$$

依据性质 1 中相应情况的讨论，同理可以推断出可疑度 $R_{P_i}<1$，$R_{P_j}<1$.　　证毕.

从性质 2 可以看出，对于谓词 P_i，当 $m=n$ 时，缺陷关联度 R_{P_i} 的计算值最大. 但对于谓词 P_j，当 $m=n+x$ 时，缺陷关联度 R_{P_j} 的计算值最大. 虽然性质 2 的假设较强，但是根据该性质，我们可以推断出：在不同谓词提供信息量有差异的前提下，针对每个谓词选择恰当的信息利用强度是获得最佳缺陷关联度计算值的必要条件.

借助这两个性质，我们可以发现，谓词执行信息量和信息利用强度都是影响缺陷定位的重要因素，这两者均会对缺陷定位过程产生影响（这一结论将会在实验部分进一步得到验证）. 因此，基于谓词执行信息进行软件缺陷定位的算法需要根据程序实际运行时谓词提供的信息量合理地选择信息利用强

度，才能发挥较好的定位作用，这也是 Safla 算法的基本思想.

4.3 算法基本流程

Safla 的实现分为数据收集、数据分析、数据处理和数据统计 4 个部分：

(1) 数据收集部分. 本文选择对 if/while/for 这 3 类分支语句以及函数返回值语句(return 语句)进行插桩，然后输入程序的全部测试用例，收集插桩谓词的输出结果.

(2) 数据分析部分. 针对谓词 P_i，通过对所有失败测试用例运行时谓词 P_i 的执行情况进行分析，确定对 P_i 所采用的信息利用强度.

(3) 数据处理部分. 根据确定的信息利用强度，对所有成功测试用例下 P_i 的谓词执行序列集合 $INFO_s$ 和所有失败测试用例下 P_i 的谓词执行序列 $INFO_f$ 进行变换处理，得到执行信息向量总和均值 $\overline{ES^{\omega}_{P_i}}$ 和 $\overline{EF^{\omega}_{P_i}}$，代入式(1)中计算 P_i 的缺陷关联度. 不断重复数据分析和数据处理两部分的工作，直到计算出所有谓词的缺陷关联度.

(4) 数据统计部分. 将谓词按照缺陷关联度的大小排序，输出最终的谓词排序结果.

下面我们给出 Safla 的算法描述，具体如下.

算法 1. Safla 算法.

输入：$INFO_s$，$INFO_f$

输出：谓词的最终排序结果

符号：$TotalPass$：成功测试用例数目；

$TotalFailed$：失败测试用例数目；

P：程序中所有插桩谓词的集合；

$\boldsymbol{S}(P_i,t_l)$：测试用例 t_l 运行时，谓词 P_i 的谓词执行序列；

$\boldsymbol{E}^{\omega}_{P_i,t_l}$：测试用例 t_l 运行时，在信息强度 ω 下，谓词 P_i 的执行信息向量；

$evaluate(INFO_f)$：根据谓词在所有失败测试用例下的执行情况，选择信息利用强度大小的函数；

$stat(\boldsymbol{S}(P_i,t_l),\omega)$：根据信息利用强度 ω，计算谓词 P_i 执行信息向量的函数；

1. For all $P_i \in P$ do
2. $\quad \omega \leftarrow evaluate(INFO_f)$
3. $\quad$ For all $INFO_s$ and $INFO_f$ do
4. $\quad\quad$ If $\boldsymbol{S}(P_i,t_l) \in INFO_s$ then
5. $\quad\quad\quad \boldsymbol{E}^{\omega}_{P_i,t_l} \leftarrow stat(\boldsymbol{S}(P_i,t_l),\omega)$
6. $\quad\quad$ else
7. $\quad\quad\quad \boldsymbol{E}^{\omega}_{P_i,t_l} \leftarrow stat(\boldsymbol{S}(P_i,t_l),\omega)$
8. $\quad$ End For
9. $\quad \overline{ES^{\omega}_{P_i}} \leftarrow \frac{1}{TotalPass}\sum_{l=1}^{TotalPass} \boldsymbol{E}^{\omega}_{P_i,t_l}$
10. $\quad \overline{EF^{\omega}_{P_i}} \leftarrow \frac{1}{TotalFailed}\sum_{l=1}^{TotalFailed} \boldsymbol{E}^{\omega}_{P_i,t_l}$
11. $\quad R_{P_i} = \| \overline{ES^{\omega}_{P_i}} - \overline{EF^{\omega}_{P_i}} \|$
12. End For
13. 根据缺陷关联度的大小对所有谓词进行排序

4.4 自适应数据分析与处理技术

本节主要是对上述算法中所涉及的自适应数据分析与处理技术进行论述，相关技术主要体现在滑动窗口的选择与操作上.

为了直观反映对谓词执行信息的利用强度，我们设计滑动窗口来对谓词执行序列中蕴含的信息进行提取，如图 5 所示. 参数 ω 是滑动窗口的大小，控制着信息利用的强度. 滑动窗口每次只可沿着谓词执行序列移动一位.

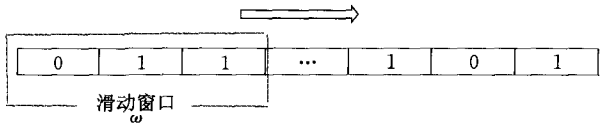


图 5 使用滑动窗口提取信息

在设计滑动窗口大小选择方案时，考虑到失败测试用例对于缺陷定位的重要作用[13]，Safla 选择谓词在失败测试用例下的执行情况作为分析对象，从中挑选出现频率最高的执行次数，并根据该执行次数选择滑动窗口 ω 的大小，具体方案如下：

$$\omega = \begin{cases} W_d, & \text{当 } W \le W_d \text{时} \\ W, & \text{当 } W_d < W < W_u \text{时}, \\ W_u, & \text{当 } W \ge W_u \text{时} \end{cases}$$

其中，W 表示挑选出的执行次数，W_d 和 W_u 分别是滑动窗口的下限和上限. 按照该方案选择滑动窗口的大小，可以使信息利用的粒度最大化，对谓词执行信息的提取也更加充分，同时尽量避免文献[8]中等概率赋值情况的出现.

需要说明的是，和大多数自适应策略一样，如何选择信息利用强度的大小是一种启发式的策略. 因此，虽然本文的方法在制定信息利用强度选择策略时重点考虑失败测试用例，但是也可以存在其它操作情形. 例如，综合考虑成功测试用例以及失败测试用例下的执行情况选择信息利用强度，或者当遇到成功测试用例较少的情况时，我们可以选择根据成功测试用例下的执行情况选择信息利用强度. 对于这些策略的深入分析和比较，将是我们下一步的研究工作.

对滑动窗口的操作包括以下两步：

(1) 将滑动窗口放在谓词执行序列 $\boldsymbol{S}(P_i,t_l)$ 的

起始位置，逐位移动滑动窗口，共获得 $\delta=Len(S)-\omega+1$ 个长度为 ω 的子序列（其中 $Len(S)$ 表示谓词执行序列 $\boldsymbol{S}(P_i,t_l)$ 的长度，要求 $Len(S)\geq\omega$）.

（2）针对滑动窗口中长度为 ω 的子序列的值集合中每一种可能取值，统计其在 δ 个子序列中出现的次数，进行归一化处理，便构成信息强度 ω 下的执行信息向量 $\boldsymbol{E}^{\omega}_{P_i,t_l}$. 例如当 $\omega=1$ 时，记录滑动窗口中“0”和“1”分别出现的次数 n_0、n_1，则执行信息向量为 $\boldsymbol{E}^{1}_{P_i,t_l}=\left(\frac{n_0}{\delta},\frac{n_1}{\delta}\right)$. 其中，$\frac{n_1}{\delta}$ 为 SOBER 中使用的 evaluation bias；当 $\omega=2$ 时，记录滑动窗口中“00”、“01”、“10”和“11”分别出现的次数 n_{00}、n_{01}、n_{10}、n_{11}，则执行信息向量为 $\boldsymbol{E}^{2}_{P_i,t_l}=\left(\frac{n_{00}}{\delta},\frac{n_{01}}{\delta},\frac{n_{10}}{\delta},\frac{n_{11}}{\delta}\right)$.

5　实　验

在本节中，我们将进行如下 3 部分实验：（1）信息利用强度选择有效性验证实验；（2）多种算法定位效果比较实验；（3）算法稳定性实验，以此来充分证明 Safla 的优越性. 首先，我们对实验中使用的目标程序、算法评价准则以及实验设置进行描述，随后对实验结果展开分析.

5.1　目标程序

参照大量相关工作，本文选取 Siemens 程序包中的全部 7 个程序以及一个实际程序 space 作为目标程序进行实验. 表 1 中列举了目标程序的主要信息. 所有这些程序以及缺陷版本都是由 Software-artifact Infrastructure Repository（SIR）网站上下载①.

表 1　目标程序的主要信息

程序	描述	版本数	可执行代码数	测试用例数
print-tokens（2 个程序）	词汇分析器	17	341～355	4130
schedule（2 个程序）	优先排队	19	261～294	2710
replace	模式识别	32	505～518	5542
tot-info	信息测量	23	272～274	1052
tcas	高度分离	41	133～137	1608
space	ADL 解释器	38	6218	13496

5.2　评价方法

本文选用 P-score[9] 作为比较各算法定位精度的主要评价方法. 首先给出 P-score 的计算公式：

$$P\text{-score}=\frac{\text{index of }P}{L}\times 100\% \qquad (2)$$

其中，L 为目标程序中的谓词数量，P 表示程序中与缺陷最为相关的谓词，index of P 为该谓词在算法输出的谓词排位表中所排的位置. P-score 可以解释为所需检查的谓词数量越少，则算法的定位精度越高.

5.3　实验设置

Siemens 程序包一共包含 132 个缺陷版本，在本文实验中共排除掉 21 个缺陷版本. 其中，replace-v27 和 schedule2-v9 没有失败测试用例，其余 19 个版本由于无法清晰确定缺陷谓词的位置，因此也予以排除，分别是 print_tokens-v4，v5，v6；replace-v12；tcas-v7，v8，v13，v14，v16，v17，v18，v19，v33，v36，v38；tot_info-v6，v10，v19，v21. 按照相同的原则，在 space 程序包中共排除掉 10 个缺陷版本，分别是 v1，v2，v25，v26，v30，v32，v34，v35，v36，v38.

本文选用文献[7]所采用的最相关谓词的选择策略：首先确定缺陷位置，若缺陷在可执行语句上，标记该可执行语句；若缺陷为可执行语句的缺失，则标记相距缺失语句最近的下一条可执行语句. 其次，根据标记的可执行语句，选择距离该可执行语句最近的谓词作为程序中与缺陷最相关的谓词. 通常，该谓词所在分支语句与标记的可执行语句之间不超过 3 行代码.

在设置滑动窗口上下限时，由于窗口打开方能进行谓词执行信息的提取，因此设置 $W_d=1$. 另外，由于当 $\omega\geq 6$ 时，算法的时间复杂度太高，对一些程序而言，运行时间难以接受，所以在实验中我们设置 $W_u=5$.

5.4　实验结果分析

5.4.1　信息利用强度选择有效性验证实验

在研究动机中，我们指出，针对每个谓词选择合适的信息利用强度，不仅可以充分利用谓词的执行信息，还能够提高算法的定位精度. 因此，在本节中，我们通过对 Safla 与 N bit-Pesla 算法[8]（$N=2,3,4,5$）的定位效果进行比较，来验证 Safla 中加入的动态信息利用强度选择方案的有效性.

在各统计节点处，我们首先给出 N bit-Pesla 算法分别在 Siemens 和 space 程序包上算得的 P-score 的最大值、最小值以及平均值结果，分别如图 6 和图 7 所示（图中横坐标表示所需检查谓词占全部谓词的百分比，纵坐标表示发现缺陷占全部缺陷的百分比. 之后各图横纵坐标所表示的物理意义均与本图相同，故不再赘述）. 由于 space 程序包较大，其中

① http://sir.unl.edu/portal/index.php

包含的谓词数量较多，而且当检查少量谓词时就可以发现大部分的缺陷，因此图中 space 与 Siemens 的横坐标设置不同.

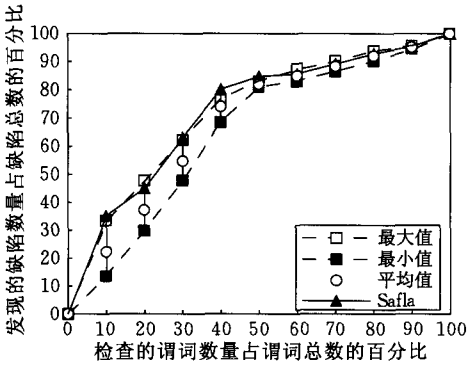


图 6 Safla 与 Nbit-Pesla 算法在 Siemens 上定位效果对比

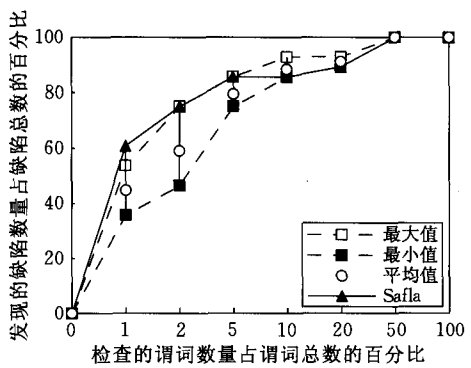


图 7 Safla 与 Nbit-Pesla 算法在 space 上定位效果对比

在此基础上，我们又给出 Safla 的定位结果. 由图 6 和图 7 我们可以看到，只有在图 7 的 10%统计节点处，Safla 的定位效果不如 Nbit-Pesla 算法中的最优结果. 但是，如果考虑到如下两个因素：(1) 对于 space 程序而言，检查排位表中前 10%的谓词就意味着需要检查大约 91 个谓词 ($10\% \times 914 = 91.4$)，代价较大；(2) 当检查前 5%的谓词时，就可以发现超过 85%的缺陷 ($24/28 = 0.8571$)，继续增加检查的谓词数量，所发现的缺陷数量却增加并不明显. 我们可以认为，Safla 的定位效果在整体上能够接近甚至超出 Nbit-Pesla 算法 ($N=2,3,4,5$) 中最优的定位效果.

上述实验验证了本文所提出的针对不同谓词选择信息利用强度的方法是有效的，它能够较好地适应谓词在不同测试用例上的执行情况，大大提高算法的适应性，并能在一定程度上提高算法的定位精度.

5.4.2 算法定位效果比较实验

在本部分中，我们将给出 Safla、CBI[5]、SOBER[6]、Mann-Whitney、Wilcoxon、F-test、t-test 算法[7]应用在 Siemens 程序包以及 space 程序包上得到的整体定位效果对比，分别如图 8 和图 9 所示. 这些参与比较的算法都是近年来研究者所提出的基于谓词的统计学缺陷定位算法.

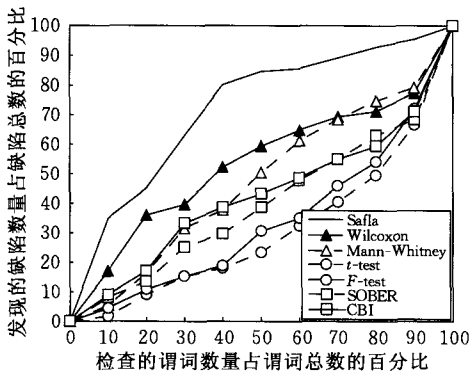


图 8 各算法在 Siemens 上整体定位效果对比

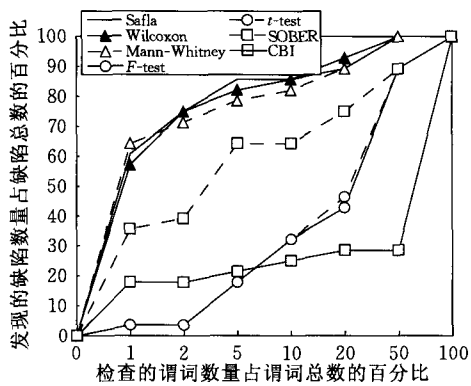


图 9 各算法在 space 上整体定位效果对比

由图 8 我们可以发现，在 9 个统计节点中，相比其它的 6 种算法，Safla 的定位效果都是最好的. 举例说明如下：当检查 10%的谓词时，Safla 可以定位出 35.14%的缺陷，其它 6 种算法中表现最好的 Wilcoxon 也仅能定位出 17.11%的缺陷. Safla 定位到的缺陷数量接近 Wilcoxon 的 2 倍，Mann-Whitney 的 5 倍，是其它算法的 7～30 倍.

由图 9 可以看出，在 space 程序包上，Safla 总体定位效果接近非参数假设检验方法中最好的 Mann-Whitney 和 Wilcoxon，远高于其它的 4 种算法. 当检查少量谓词时，例如检查 1%的谓词时，Safla 定位到的缺陷数量略少于 Mann-Whitney (平均少定位出 1 个缺陷)，而在 2%节点处，Safla 可以取得相比其它所有比较算法最好的定位效果.

图 10 还给出了 Safla、CBI[5]、SOBER[6]、Mann-Whitney、Wilcoxon、F-test、t-test[7] 算法在 Siemens 7 个独立程序上的定位效果对比. 可以看到，除了在 schedule(2 programs) 上的定位效果还有待提高外，在其它程序上 Safla 都能取得很好的定位效果.

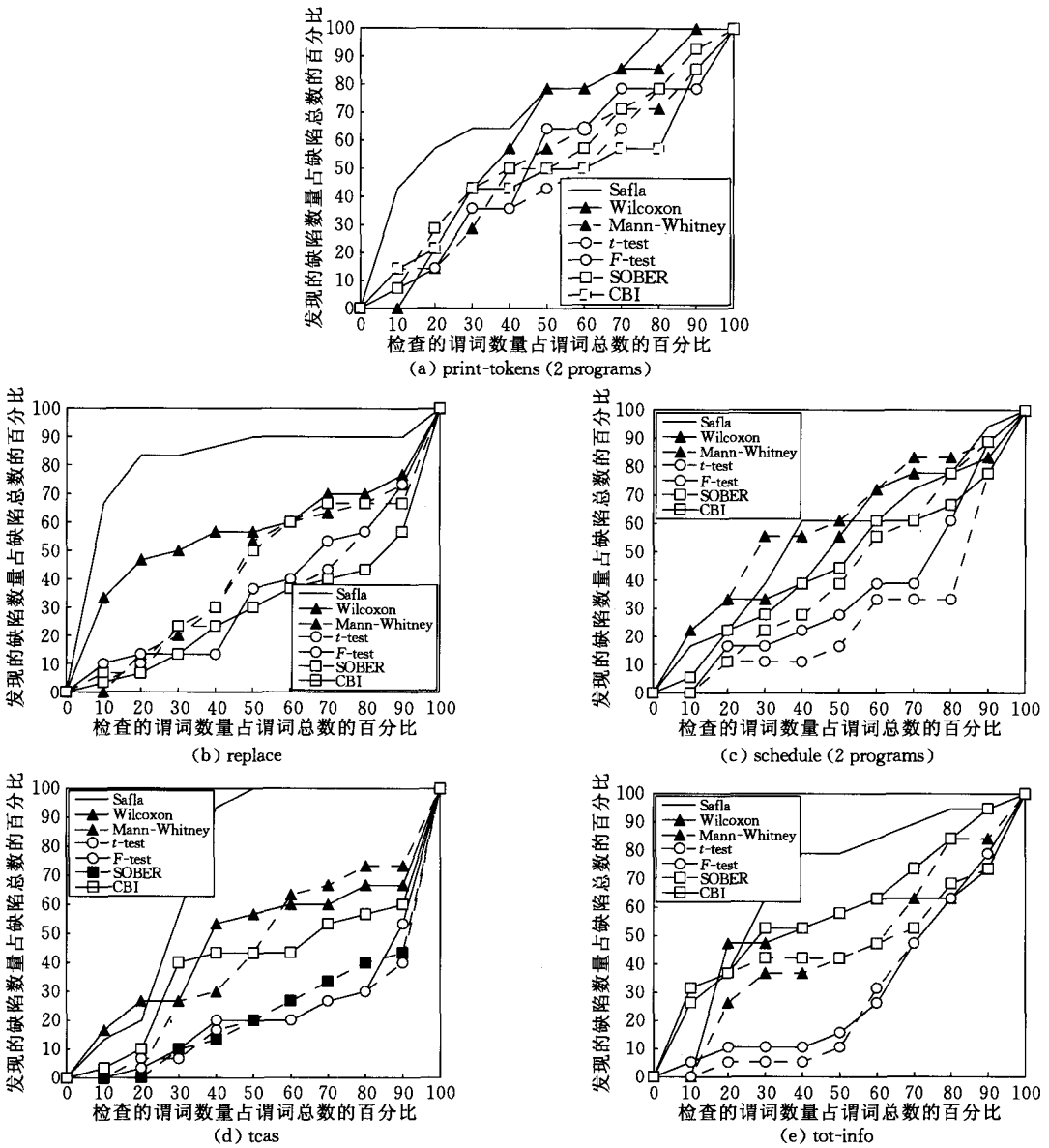


图 10　7 个算法在 Siemens 程序包各独立程序上的定位效果对比

5.4.3　算法稳定性实验

在本节中，我们主要对 7 种算法在单个缺陷版本上定位效果的差别进行实验比较，这些实验反映了算法在不同类型程序和缺陷上定位效果的稳定性.

在表 2 中，“Min”表示使用相应算法定位缺陷时所得到的最小 P-score；“Max”表示最大的 P-score；“Mean”表示 P-score 的均值；“Stdev”表示 P-score 的标准差. 在这 4 个指标中，后两个体现了算法在定位不同缺陷时所表现出的稳定性，其值越小，表明该算法的稳定性越好.

表 2　各种算法在 Siemens 和 space 程序包上定位效果的详细对比

	Safla/%		Wilcoxon/%		Mann-Whitney/%		F-test/%		t-test/%		SOBER/%		CBI/%	
	Siemens	space	Siemens	space	Siemens	space	Siemens	space	Siemens	space	Siemens	space	Siemens	space
Min	**1.59**	**0.11**	0.89	0.11	3.45	0.11	4.44	0.77	4.50	0.77	3.70	0.11	0.91	0.77
Max	**100.00**	**45.51**	100.00	46.78	100.00	48.46	100.00	95.40	100.00	94.97	100.00	95.68	100.00	94.97
Mean	**29.17**	**5.29**	46.91	5.66	53.38	5.57	67.71	26.40	70.74	25.81	60.64	17.23	58.58	25.81
Stdev	**26.24**	**12.24**	35.71	15.26	30.56	15.79	29.09	26.96	28.23	26.95	32.06	31.18	34.34	26.95

从表 2 中可以看出，对于 Siemens 程序包的 111 个缺陷版本，在“Mean”和“Stdev”这两个指标中，Safla 都是表现最好的算法，即使用 Safla 进行定位缺陷时，可以在很大程度上获得更高的定位精

度.对于 space 程序包中 28 个缺陷版本,相比于 CBI[5]、SOBER[6]、Wilcoxon、Mann-Whitney、F-test 和 t-test[7]这 6 种算法,Safla 无论是在定位精度上还是在定位稳定性上都具有很大的优势.

6 有效性讨论

影响本文结论有效性的因素主要有以下 3 个:

(1)各算法的复现是否正确.

(2)P-score 评价方法是否有效.

(3)目标软件的选取是否合适.

首先,对于 CBI 算法和 SOBER 算法,本文按照文献[5-6]所述步骤进行复现,并用文献[7]中所列举的例子验证了其正确性.对于 Wilcoxon、Mann-Whitney、F-test,t-test 算法,我们直接利用 ALGLIB 数理统计软件进行实现,并用文献[7]中公布的实验数据进行了验证.因此可以认为本文对于各算法的复现应该是正确的,所得到的实验结果和数据也是有效的.

P-score[9]是由 Zhang 等人于近几年提出,主要适用于基于谓词的方法,而本文所提出的算法以及选取出作为比较对象的算法均属于基于谓词的方法.因此,选用 P-score 作为评价各算法定位精度的标准也是合理的.

Siemens 程序包一直以来就是研究软件缺陷定位必选的实验对象之一,但是由于其中所包含的缺陷都为人工植入,很难代表真实的软件缺陷.而且代码行较少,属于小型程序.为此,本文增加了 space 程序包作为实验对象,得到了相同的结论,但这依旧不能排除对于其它实验对象,本文的研究会得到不同实验结果的情况.因此,在今后的研究中,考虑增加更大规模的程序作为实验对象仍旧是一个重要的方面.

7 结束语

软件缺陷定位在软件调试过程中是一项异常费时费力的工作,在现有方法中,基于谓词的统计学缺陷定位方法在定位精度上已有良好的表现.但之前方法中信息利用的强度相对固定,无法针对谓词执行信息量的变化进行动态调节,这样就会不可避免的导致信息利用不足或过分利用情况的发生,从而影响定位精度的提高.在本文中,我们提出一种基于谓词执行信息分析的自适应缺陷定位算法,并用实验证明,根据测试用例运行时每个谓词提供信息量的多少来选择算法中信息利用的强弱是有效而且必要的.由于需要对不同谓词选择不同的信息利用强度,因此,针对不同谓词,如何更加公平地比较不同信息利用强度下的计算结果以及完善信息利用强度选择方案将是我们下一步研究工作的重点.

参 考 文 献

[1] Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization//Proceedings of the 24th International Conference on Software Engineering. Orlando, Florida, USA, 2002: 467-477

[2] Jones J A, Harrold M J. Empirical evaluation of the tarantula automatic fault-localization technique//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York, NY, USA, 2005: 273-282

[3] Abreu R, Zoeteweij P, van Gemund A J C. On the accuracy of spectrum-based fault localization//Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION. Washington, DC, USA, 2007: 89-98

[4] Abreu R, Zoeteweij P, van Gemund A J C. Spectrum-based multiple fault localization//Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington, DC, USA, 2009: 88-99

[5] Liblit B. Scalable statistical bug isolation//Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. Chicago, New York, USA, 2005: 15-26

[6] Liu C, Fei L, Yan X et al. Statistical debugging: A hypothesis testing-based approach. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848

[7] Zhang Z, Chan W K, Tse T, et al. Non-parametric statistical fault localization. The Journal of Systems and Software, 2011, 84(6): 885-905

[8] Li Wei, Zheng Zheng, Hao Peng, et al. Predicate execution-sequence based fault localization algorithm. Chinese Journal of Computers, 2013, 36(12): 2406-2419(in Chinese)
(李伟,郑征,郝鹏等.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)

[9] Zhang Z, Chan W K, Tse T H, et al. Capturing propagation of infected program states//Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY, USA, 2009: 43-52

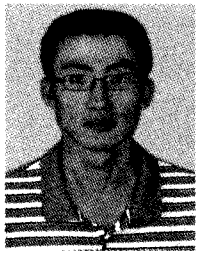
[10] Masri W. Fault localization based on information flow coverage. Software Testing, Verification and Reliability, 2010, 20(2): 121-147

[11] Zhang Z, Chan W K, Tse T H, et al. Is non-parametric hypothesis testing model robust for statistical fault localization? Information Software Technology, 2009, 51(11): 1573-1585

[12] Gong C, Zheng Z, Zhang Z, et al. Factorising the multiple fault localization problem//Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC 2012). Hong Kong, China, 2012: 4-7

[13] Zhang Z, Chan W K, Tse T H. Fault localization based only on failed runs. IEEE Computer, 2012, 45(6): 64-71

[14] Jiang B, Zhai K, Chan W K, et al. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization. Journal of Information and Software Technology, 2013, 55(5): 897-917



HAO Peng, born in 1986, M. S. candidate. His research interests include software fault localization, software testing.

ZHENG Zheng, born in 1980, Ph.D., associate professor. His research interests mainly focus on intelligence decision and software testing.

ZHANG Zhen-Yu, born in 1979, Ph. D., associate professor. His research interests include software testing, software debugging, etc.

GAO Yi-Chao, born in 1988, M. S. candidate. His research interests include software fault localization, software testing.

GONG Cheng, born in 1987, M. S. candidate. His research interests include software fault localization, software testing.

XUE Yun-Zhi, born in 1979, Ph.D., associate professor. His main research interests include software testing automation, compiler optimization.

Background

Fault localization is important in software debugging, which is to find out where the faulty code lies before fix it out. Manual debugging often requires much time and is easy to make mistakes. As a result, how to use the computer to predict the fault automatically has become a crucial research and practical problem. Many automatic techniques have been developed in recent years. Among them, coverage-based one has been widely proved to be useful. It tries to calculate suspiciousness for each entity (predicate, statement, branch, etc.) by comparing the behavior difference on running of correct and failed test cases, and then produces a rank list of all the entities according to their suspicious values in an ascending or descending order.

In this paper, we only focus on the predicate-based technique. Our experimental results show that some classical predicate-based methods, like CBI, SOBER, fail to locate some faults because they use predicate execution information in a fixed intensity, which may cause insufficient or excessive usage. To solve the problem, we propose a new method, called self-adaptive fault localization algorithm based on predicate execution information analysis, which dynamically select the intensity of information utilization for each predicate through the analysis of test cases run. Experimental results demonstrate that our approach performs well in both accuracy and stability for localizing faults in subject programs of the Siemens and space suites.

This work was supported by the National Natural Science Foundation of China (Nos. 60904066, 61003027) and the National Science and Technology Major Project of China (Grant No. 2012ZX01039-004).