软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(5):977−992 [doi: 10.3724/SP.J.1001.2013.04342]

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

一种基于层次切片谱的软件错误定位技术*

文万志, 李必信, 孙小兵, 刘翠翠

(东南大学 计算机科学与工程学院,江苏 南京 211189)

通讯作者: 李必信, E-mail: bx.li@seu.edu.cn, http://cse.seu.edu.cn/people/bx.li/index.htm

摘 要: 传统的软件错误定位技术通常利用测试覆盖信息计算程序语句发生错误的可疑度进行软件错误定位,但是这种定位技术没有充分考虑程序本身固有的依赖信息,缺乏语句筛选,从而使错误定位的精度受限.提出了一种基于层次切片谱的错误定位技术,以提高面向对象程序中的错误定位效率.这种技术首先分析程序不同粒度层次元素(包、类、方法以及语句)之间的依赖信息,对可能发生错误的元素进行筛选,缩小错误查找范围;在此基础上,建立了层次切片谱模型,并定义了一种可疑度度量方法;最后根据该可疑度结果从大到小的顺序进行错误定位.通过实验验证了基于层次切片谱的错误定位技术的有效性,且比基于程序谱的 Tarantula 技术、Union 技术、Intersection 技术效率更高.

关键词: 层次切片模型;层次切片谱;可疑度;错误定位;程序谱

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 文万志,李必信,孙小兵,刘翠翠.一种基于层次切片谱的软件错误定位技术.软件学报,2013,24(5):977−992. http://www.jos.org.cn/1000-9825/4342.htm

英文引用格式: Wen WZ, Li BX, Sun XB, Liu CC. Technique of software fault localization based on hierarchical slicing spectrum. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):977−992 (in Chinese). http://www.jos.org.cn/1000-9825/4342.htm

Technique of Software Fault Localization Based on Hierarchical Slicing Spectrum

WEN Wan-Zhi, LI Bi-Xin, SUN Xiao-Bing, LIU Cui-Cui

(School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

Corresponding author: LI Bi-Xin, E-mail: bx.li@seu.edu.cn, http://cse.seu.edu.cn/people/bx.li/index.htm

Abstract: A commonly-used software fault localization technique mainly utilizes testing coverage information to calculate the suspiciousness of each program statement to find the faulty statement. However, this technique does not fully take dependences of the program into account; thus, its capacity for precisely locating faults is limited. This paper proposes a more effective hierarchical slicing spectrum-based software fault localization (HSS-SFL) technique for object-oriented programs. HSS-SFL first analyzes the dependence among the elements at different granularity levels including package level, class level, method level and statement level, and deletes some elements which are unrelated to the failed testing outputs. Next, the technique builds the hierarchical slicing spectrum model for these hierarchical slices obtained from the previous step, and further defines a suspiciousness method for each slice element. Finally, the faulty statement is located according to the suspiciousness results. Experimental results on real application programs show that HSS-SFL is more effective to locate the fault than program spectrum-based Tarantula technique, Union technique and Intersection technique.

Key words: hierarchical slicing model; hierarchical slicing spectrum; suspiciousness; fault localization; program spectrum

在软件开发和维护过程中,程序员需要不断地调试程序,而软件错误定位是调试活动中最困难、最耗时的任务之一.传统的软件错误定位方法主要有插入输出语句法和设置断点法,主要通过手动完成,效率不高.近年

* 基金项目: 国家自然科学基金(60973149, 61202006); 高等学校博士学科点专项科研基金(20100092110022); 中国科学院计算机科学国家重点实验室开放基金(SYSKF1110)

收稿时间: 2012-03-27; 修改时间: 2012-06-29; 定稿时间: 2012-10-19

来,软件正变得越来越复杂多样,这给软件调试者带来了巨大的挑战.自动化、半自动化的软件错误定位技术因而也变成最热门的研究方向之一[1−3].

软件错误定位技术主要通过缩小错误的搜索域来提高其效率.经典的Delta调试技术[4,5]通过不断地迭代运行程序,交换正确运行的内存状态和错误运行的内存状态来缩小错误查找的范围以提高错误定位的效率.这种技术虽然能够逐步缩小搜索域,但迭代运行和内存交换的代价太大.程序切片技术[6]是一种根据特定计算提取程序中相关语句的分解技术,它可以通过单一的错误运行来缩小错误的搜索域.其思想是:给定一个程序 P,可疑语句 S 及 S 中变量 V,切片给出了影响 S 中变量 V 值的语句,它删除了与可疑语句 S 中变量 V 不相关的部分,在一定程度上缩小了错误的搜索域.程序切片技术主要包括静态切片技术[7]和动态切片技术[8].基于静态程序切片的软件错误定位主要是通过静态分析数据流和控制流,对程序进行分解,从而降低程序错误定位的搜索域,但是由于静态切片过于保守,错误定位的精度不高.基于动态切片的错误定位技术是在静态切片基础之上使用更加精确的切片准则使搜索域进一步减小,但这样的切片技术比较复杂,成本比较高,对于大规模软件来说,搜索域依然很大,不是很适用.层次切片技术[9]是根据面向对象 Java 语言的层次结构特性提出的一种更加高效的程序切片技术.这种技术通过分析程序不同粒度层次元素(包、类、方法以及语句)之间的依赖信息,逐层对可能发生错误的元素进行筛选,从而可有效地缩小错误定位的范围.

近年来人们提出了一种更适合于大规模程序,容易实现以及自动化的错误定位技术——基于程序谱的错误定位技术[10,11].程序谱通常用来刻画一个程序的行为,是一个执行轨迹的投影,它给出了程序执行时的活跃部分[12].通常情况下,对程序谱信息的收集比较简单、容易实现,而且便于存储,适用于资源有限的环境.基于程序谱的错误定位技术通过对程序谱的分析,计算程序中各个元素可能产生错误的概率,一般称其为可疑度;然后根据各个元素的可疑度从大到小依次检查,直到精确定位到错误所在的元素为止.研究表明,基于程序谱的错误定位方法的平均查错精度不足整个程序的 20%[11,13].然而,由于该技术没有充分考虑到程序语句或语句块之间相应的依赖关系,错误定位过程中存在着一些语句冗余.

本文结合程序切片和程序谱的优点提出了一种基于层次切片谱的软件错误定位技术(hierarchical slicing spectrum-based software fault localization,简称 HSS-SFL),以提高面向对象程序中的错误定位的效率.主要贡献如下:

(1) 提出了面向对象程序错误逐层定位的思想,并利用层次切片技术提取各层次错误相关元素,删除程序中与错误无关的元素,减小了传统的程序谱的规模,提高了语句搜索的效率;
(2) 在传统的可疑度度量方法基础上,通过引入程序元素的执行频度和贡献度,提出了一种可疑度度量方法;
(3) 构建了在新的可疑度基础上的软件错误定位算法;
(4) 实验验证了 HSS-SFL 技术的有效性,且比当今流行的基于程序谱的 Tarantula 技术、Union 技术和 Intersection 技术错误定位精度更高.

本文第 1 节介绍层次切片模型和基于程序谱的错误定位模型的相关基础知识.第 2 节介绍 HSS-SFL 技术.第 3 节实验验证 HSS-SFL 技术的有效性.第 4 节介绍一些错误定位相关工作.第 5 节给出结论及下一步的工作.

1 背景知识

1.1 层次切片模型

程序切片技术[6]最早在软件维护领域中的应用即为错误定位.它通过删除程序中与某个出错语句中变量不相关的一些代码来缩小查找错误的范围,从而降低查找错误的工作量,提高错误定位的效率.因而基于切片技术的错误定位方法的效率主要依赖于切片技术的效率.层次切片模型(hierarchical slicing model,简称 HSM)[9]是根据面向对象 Java 语言的层次结构特性提出的一种程序切片技术,它可以根据不同层次(包层、类层、方法层、语句层)的切片标准,计算得到不同层次的切片,并且这些层次切片结果根据其粒度层次的不同,其精度也不同,即粒度越细,精度越高.图 1 描述了层次切片的提取过程.HSM 包括 3 部分:层次切片标准、层次依赖图以及逐步

求精算法.

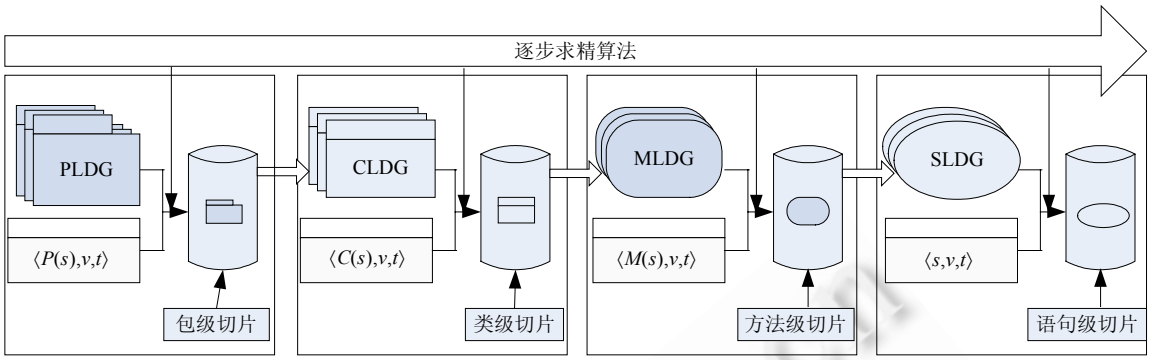


Fig.1 Hierarchical slicing model

图 1 层次切片模型

层次切片标准包括从包层、类层、方法层到语句层的各层次切片标准,它是计算不同层次切片的依据和出发点.给定一个 Java 源程序以及程序的一个输入集 T,P,C,M,S 分别表示该程序中的包集、类集、方法集、语句集.对于某一个语句 $s\in S,P(s),C(s),M(s)$分别表示包含该语句的包、类和方法,V_s 表示 s 语句中包含的变量的集合,则对于任意 $t\in T,v\in V_s$,从包层、类层、方法层到语句层次的切片标准, 分别为$\langle P(s),v,t\rangle,\langle C(s),v,t\rangle,\langle M(s),v,t\rangle,\langle s,v,t\rangle$.

层次依赖图包括:包层次依赖图(package-level dependence graph,简称 PLDG)、类层次依赖图(class-level dependence graph,简称 CLDG)、方法层次依赖图(method-level dependence graph,简称 MLDG)以及语句层次依赖图(statement-level dependence graph,简称 SLDG).它们分别表示不同粒度层次元素之间的依赖关系.其中,包层次依赖图表示的是包之间的引用关系;类层次依赖图表示的是类之间的继承以及使用关系;方法层次依赖图表示的是方法之间的相互调用关系;而语句层次依赖图即是传统的程序依赖图[14],表示元素之间的控制流依赖以及变量之间的数据依赖关系.

逐步求精切片算法(stepwise slicing algorithm,简称 SSA)是一种混合算法,在同一粒度层次上,基于依赖图的可达性计算切片结果,而在不同的粒度层次之间采用逐步求精的方法进行计算.具体来说,首先根据包切片标准,在包层次依赖图上进行可达性计算,得到包层次的切片结果;然后在类层次上,先构造包层次切片结果中各个包内的类层次依赖图,再根据类层次切片标准,在类层次依赖图上计算类层次的切片结果;然后采用类似的方法,计算得到方法层次的切片结果,直到最后计算得到语句层次的切片结果.

在我们前期的研究工作中,我们提出了基于程序切片谱的软件错误定位技术[3,15],但一次性提取整个程序的切片则效率过低.本文提出的基于层次切片谱的软件错误定位技术逐层提取了面向对象程序中各层次错误相关切片,可有效解决传统的切片过大、效率过低的问题.

1.2 基于程序谱的错误定位模型

由于程序谱信息便于收集与存储,适用于资源有限、甚至是大规模软件维护的环境,当前很多研究人员提出了各种各样的基于程序谱的错误定位技术[10−13,16,17].一般来说,基于程序谱的错误定位过程包括 3 步:(1) 通过运行测试用例收集程序中的语句覆盖信息;(2) 根据语句覆盖信息计算出覆盖语句的出错可疑度;(3) 根据覆盖语句出错可疑度大小顺序逐句排查来进行错误定位.

定义 1(程序谱). 已知程序有 n 条语句,测试执行集合 $T=T_F\cup T_P$,其中 $T_F=\{T_1,\ldots,T_s\}$表示测试运行结果错误的测试用例集合,$T_P=\{T_{s+1},\ldots,T_m\}$表示测试执行结果正确的测试用例集合,则程序谱是一个二维矩阵 $M_{n,m}$,矩阵元素 $b_{i,j}$ 为

$$b_{i,j}=\begin{cases}1, & T_j\text{的测试执行路径覆盖语句}s_i\\ 0, & \text{否则}\end{cases},\ 1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m \tag{1}$$

一个形象化的基于程序谱的错误定位模型见表 1,其中行 $s_1,\ldots,s_n$ 表示程序中对应的语句,列 $T_1,\ldots,T_m$ 表示 m 个测试执行,包括 s 个失效的运行和 $m-s$ 个成功的运行,$b_{i,j}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$ 是程序谱矩阵元素,其值为 1 或 0,值为 1 表示测试 T_j 经过语句 s_i,值为 0 表表示测试 T_j 不经过语句 s_i,最后一列表示相应行语句包含错误的可疑度(suspiciousness),可疑度通常是基于这样的假设:经过某条程序语句 s_i 的失效测试数目越多,而成功测试数目越少,该模块发生错误的可能性就越大,即在表 1 中,$\sum_{j=1}^{s}b_{i,j}$ 越大,$\sum_{j=s+1}^{m}b_{i,j}$ 越小,$suspiciousness(s_i)$就越大.因此,对于某条语句 s_i,$suspiciousness(s_i)$的可疑度一般定义如下[10]:

定义 2(可疑度). 可疑度

$$suspiciousness(s_i)=\frac{failed(s_i)\%}{failed(s_i)\%+passed(s_i)\%} \tag{2}$$

其中,$failed(s_i)$%表示经过语句 s_i 的失效测试数占失效测试总数之比,$passed(s_i)$%表示经过语句 s_i 的成功测试数占成功测试总数之比.

通常,不同的研究人员对于 $failed(s_i)$%和 $passed(s_i)$%有着不同的定义方法,总的目标希望通过对 $failed(s_i)$%和 $passed(s_i)$%定义尽可能无限地接近语句 s_i 出错的可疑度,从而判断语句是否真正出错.本文通过引入程序元素的执行频度和贡献度重新定义了 $failed(s_i)$%和 $passed(s_i)$%,并验证其有效性.

Table 1 Program spectrum-based software fault localization model

表 1 基于程序谱的错误定位模型

Program	Test cases						Suspiciousness
	Failed			Passed			
	T_1	…	T_s	T_{s+1}	…	T_m	
s_1	$b_{1,1}$	…	$b_{1,s}$	$b_{1,s+1}$	…	$b_{1,m}$	p_1
s_2	$b_{2,1}$	…	$b_{2,s}$	$b_{2,s+1}$	…	$b_{2,m}$	p_2
…	…	…	…	…	…	…	…
s_n	$b_{n,1}$	…	$b_{n,s}$	$b_{n,s+1}$	…	$b_{n,m}$	p_n

2 基于层次切片谱的软件错误定位

鉴于程序切片技术能够有效地缩小程序错误定位的范围,而程序谱在操作上比较简单,且适用于大规模程序错误定位,本文将这两种技术有效地结合起来进行错误定位,提出了一种新型的程序谱模型——层次切片谱模型(hierarchical slicing spectrum,简称 HSS).HSS-SFL 技术主要包括以下 4 个过程:

(1) 根据失效的测试输出,计算从包层次、类层次、方法层次到语句层次的错误相关切片;

(2) 根据测试覆盖信息建立层次切片谱矩阵;

(3) 根据可疑度模型计算各个层次错误相关元素的可疑度;

(4) 根据可疑度结果大小进行错误定位分析.

下面我们将详细介绍 HSS-SFL 技术.

2.1 层次切片谱的构造

为了更为准确地定位程序中的错误,本文首先针对失效的测试执行收集错误相关切片来缩小错误查找的范围,然后在此基础之上构造了程序切片谱进行软件的错误定位.

程序切片的提取关键在于切片准则的定义.切片准则一般主要包含 3 个元素:兴趣点、兴趣变量、特定的输入[6−9].兴趣点一般为程序的输出语句,兴趣变量一般为输出语句中的输出变量,特定的输入一般为某次程序执行的输入.根据层次切片技术和测试执行历史,本文的层次切片准则定义如下:

定义 3(层次切片准则). 已知程序 SP 及其测试执行 T.令 T 的输入为 I_T,T 的输出语句为 S_T,输出变量为 V_T,$P(S_T)$为包含 S_T 的包,$C(S_T)$为包含 S_T 的类,$M(S_T)$为包含 S_T 的方法,则程序 SP 的包层切片准则、类层切片准则、方法层切片准则、语句层切片准则分别是三元组$\langle I_T,P(S_T),V_T\rangle$,$\langle I_T,C(S_T),V_T\rangle$,$\langle I_T,M(S_T),V_T\rangle$,$\langle I_T,S_T,V_T\rangle$.

一般来说,程序切片可根据切片准则在程序依赖图上通过两步图可达性算法取得.它可以提取影响兴趣点处兴趣变量的部分[6−9].本文基于已知的测试执行历史,选取运行结果错误的测试的输出语句为兴趣点,输出变量为兴趣变量,从而可以提取仅对错误输出有影响的部分,删除错误无关部分,以缩小错误定位的搜索域.具体来说,根据层次切片准则,我们定义了各层次错误相关切片.

定义 4(错误相关包切片). 已知程序 SP 及其测试执行集合 $T=T_F\cup T_P$,其中 $T_F=\{T_1,\ldots,T_s\}$表示测试运行结果错误的测试用例集合,$T_P=\{T_{s+1},\ldots,T_m\}$表示测试运行结果正确的测试用例集合.令 $GetPSlice(SP,T)$为根据测试执行包层切片准则$\langle I_T,P(S_T),V_T\rangle$在程序 SP 中提取的包层切片,则错误相关包切片

$$PSlice=GetPSlice(SP,T_1)\cup GetPSlice(SP,T_2)\cup\ldots\cup GetPSlice(SP,T_s).$$

在定义 4 中,$GetPSlice$ 通过在包级依赖图上运用图可达性算法取得包层切片,在我们前期的研究工作(文献[9])中有详细的介绍.错误相关包切片 $PSlice$ 提取了整个程序 SP 中影响错误输出的包.同理,我们可在某个可疑的包中提取相应的错误相关类切片 $CSlice$,在可疑的类中提取错误相关方法切片 $MSlice$,在可疑的方法中提取错误相关语句切片 $SSlice$.下文中我们用 $XSlice$ 分别表示包层切片 $PSlice$、类层切片 $CSlice$、方法层切片 $MSlice$ 和语句层切片 $SSlice$.算法 1 给出了各层次错误相关切片的提取算法.

算法 1. 错误相关切片提取算法.

Function: ***GetRelatedXSlice*(*E*,*T*)**

Inputs: E: a program element, in which the fault-related slice is computed. At different levels, it represents a program, a package, a class or a method.

T: a list of test cases $\{T_1,T_2,...,T_m\}$.

Outputs: $XSlice$: a list of elements in the fault-related slice, which is $\{e_1,e_2,\ldots,e_n\}$.

```
begin
    XSlice=∅
    for each test case T_i in T do
        if T_j is failed
            XSlice=XSlice∪GetXSlice(E,T_j)
        end if
    end for
    return XSlice
end begin
```

定义 5(层次切片谱). 已知错误相关切片集合 $XSlice$ 中元素个数为 n(即 $n=|XSlice|$),测试执行集合 $T=T_F\cup T_P$,其中 $T_F=\{T_1,\ldots,T_s\}$表示测试运行结果错误的测试用例集合,$T_P=\{T_{s+1},\ldots,T_m\}$表示测试执行结果正确的测试用例集合,则层次切片谱表示一个二维矩阵 $M_{n,m}$,

$$\forall f_{i,j}\in M_{n,m}, f_{i,j}=\begin{cases}k,\ T_j\text{的测试执行路径经过}XSlice\text{中元素}e_i\text{的次数}\\0,\ T_j\text{的测试执行路径未经过}XSlice\text{中元素}e_i\end{cases},\ 1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m \tag{3}$$

错误相关切片删除了程序中的冗余部分,减小了层次切片谱的规模,层次切片谱引入了程序元素的执行频度,便于程序元素可疑度更加精确地度量.算法 2 给出了层次切片谱的构造算法.在包层、类层、方法层和语句层,$GetXHSS$ 函数名称中的 X 分别代表 P,C,M 和 S.

算法 2. 层次切片谱构造算法.

Function: ***GetXHSS*(*XSlice*,*T*)**

Inputs: $XSlice$: a list of elements in the fault-related slice, which is $\{e_1,e_2,\ldots,e_n\}$;

T: a list of test cases $\{T_1,T_2,...,T_m\}$.

Outputs: $XHSS[e_i,T_j]$: a matrix of the program spectra, where e_i is the element of the fault-related slice and T_j is a test case.

```
begin
    for each test case Tj in T do
        for each ei in XSlice do
            XHSS[ei,Tj]=the frequency that Tj passed ei
        end for
    end for
    return XHSS
end begin
```

2.2 可疑度度量

基于程序谱的错误定位技术是根据程序语句的可疑度进行错误定位的.目前较流行的可疑度模型是定义 2 中的可疑度模型,可疑度的度量依赖于对 $failed(s_i)$%和 $passed(s_i)$%的计算.本文对 $failed(s_i)$%和 $passed(s_i)$%的定义主要是建立在层次切片谱基础上,相对于传统的基于谱的错误定位技术,本文通过引入程序元素的执行频度和贡献度来提高可疑度度量的精确性.当今较流行的 Tarantula 技术[10]对 $failed(s_i)$%和 $passed(s_i)$%的定义如下:

$$failed(s_i)\% = \frac{n_{11}(s_i)}{n_{11}(s_i)+n_{01}(s_i)},\ passed(s_i)\% = \frac{n_{10}(s_i)}{n_{10}(s_i)+n_{00}(s_i)} \tag{4}$$

其中,$n_{11}(s_i)$表示经过语句 s_i 的失效测试数目,$n_{01}(s_i)$表示未经过语句 s_i 的失效测试数目,$n_{10}(s_i)$表示经过语句 s_i 的成功测试数目,$n_{00}(s_i)$表示未经过语句 s_i 的成功测试数目.Tarantula 技术通过统计经过程序块的失效测试数目和成功测试数目来计算程序错误的可疑度.它将单个块对不同测试结果的贡献程度看作是相同的,也就是说,在表 1 中,Tarantula 技术在计算 s_i 的可疑度时只考虑了表中其被测试执行时覆盖的信息,且只统计 s_i 行中非 0 的数目.但是,实际中由于每个测试执行规模大小不一,这些语句的贡献程度也是不一样的.例如,如果某次测试执行经过了 1 条语句,那么这条语句对执行结果的贡献度可以达到 100%;而如果这次执行经过了 2 条语句,且每条语句仅被执行 1 次,那么我们认为这两条语句对测试结果有相同的贡献度,各为 50%.

下面我们用集合论的知识给出 $failed(s_i)$%和 $passed(s_i)$%的计算方法.令 f 是映射失效的测试 T 到测试执行历史 H 的函数,即

$$f:T\to H \tag{5}$$

其中,执行历史 H 是一个集合,集合中的元素 e_{i_k} 表示程序元素 e_i 的第 k 次执行.对于一次失效的执行,我们称 f 给出了程序的一种错误解释.元素 e_i 对这次失效执行的贡献度为

$$Contribution(e_i)=\frac{|\{e_{i_k}\mid e_{i_k}\in H\}|}{|H|} \tag{6}$$

其中,绝对值符号表示集合元素个数(下同),分子表示元素 e_i 在测试 T 中被执行的次数,分母表示所有的元素在测试 T 中被执行的次数和.我们扩展 f 到 s 次失效的测试执行,即

$$F:\{f_1:T_1\to H_1,...,f_j:T_j\to H_j,...,f_s:T_s\to H_s\},\ 1\leqslant j\leqslant s \tag{7}$$

元素 e_i 对所有的失效测试的贡献度为

$$Contribution(e_i)'=\sum_{j=1}^{s}\frac{|\{e_{i_k}\mid e_{i_k}\in H_j\}|}{|H_j|} \tag{8}$$

由于 F 给出了所有失效测试的错误解释的集合,$failed(e_i)$%的定义如下:

$$failed(e_i)\% = \frac{\sum_{j=1}^{s}\frac{|\{e_{i_k}\mid e_{i_k}\in H_j\}|}{|H_j|}}{|F|} \tag{9}$$

同理,若 p 是映射正确的测试 T 到测试执行历史 H'的函数,P 是 p 的集合,则

$$passed(e_i)\% = \frac{\sum_{j}\frac{|\{e_{i_k}\mid e_{i_k}\in H'_j\}|}{|H'_j|}}{|P|} \tag{10}$$

结合层次切片谱的定义,本文对各层次元素可疑度的 $failed(e_i)\%$和 $passed(e_i)\%$的定义如下:

$$failed(e_i)\%=\frac{\sum_{j=1}^{m}(p_{T_j}\times C_{i,j})}{\sum_{j=1}^{m}p_{T_j}},\ passed(e_i)\%=\frac{\sum_{j=1}^{m}[(1-p_{T_j})\times C_{i,j}]}{\sum_{j=1}^{m}(1-p_{T_j})} \tag{11}$$

其中,

$$C_{i,j}=\frac{f_{i,j}}{\sum_{k=1}^{n}f_{k,j}},\ p_{T_j}=\begin{cases}1, & T_j \text{ is failed}\\ 0, & T_j \text{ is passed}\end{cases}.$$

在上述公式中,$C_{i,j}$ 表示元素 e_i 对测试 T_j 的贡献度.其中,程序切片谱矩阵元素 $f_{i,j}$ 如定义 5 所述.它表示测试 T_j 中元素 e_i 的执行频度. p_{T_j} =1 表示 T_j 是失效的测试,否则, p_{T_j} =0.$failed(e_i)\%$和 $passed(e_i)\%$的分母 $\sum_{j=1}^{m}p_{T_j}$ 和 $\sum_{j=1}^{m}(1-p_{T_j})$ 分别表示失效的测试数目和成功的测试数目,而分子对经过 e_i 的失效测试的数目和成功的测试数目分别进行加权(权值即贡献度).算法 3 给出了可疑度计算算法.

算法 3. 可疑度算法.

Function: ***GetSuspiciousness***(*XSlice*,*XHSS*,*T*)

Inputs: *XSlice*: a list of elements in the fault-related slice ,which is {e_1,e_2,…,e_n};

T: a list of test cases {T_1,T_2,...,T_m};

XHSS: a hierarchical slicing spectrum matrix *XHSS*[e_i,T_j].

Outputs: *Suspiciousness*(*XSlice*): a vector that denotes the suspiciousness of each element in *Xslice*.

Declare: *PN*: the number of passed test cases in *T*

FN: the number of failed test cases in *T*

PC_i: the sum of the contribution of e_i to the passed test cases

FC_i: the sum of the contribution of e_i to the failed test cases

PT_i: a variable with value 1 or 0 which means T_i is failed or passed

N_i : the sum of the count of elements in *XSlice* that test case T_i covers

```
begin
  PN=0
  FN=0
  for each test case in Ti in T do
    Ni=0
    if Ti is passed
      PTi=0
      PN=PN+1
    else
      PTi=1
      FN=FN+1
    end if
    for each ej in XSlice
      Ni=Ni+XHSS[ej,Ti]
    end for
  end for
```

for each e_i in *XSlice* **do**
  PC_i=0
  FC_i=0
  for each test case T_j in T **do**
    $PC_i=PC_i+(1-PT_j)\times XHSS[e_i,T_j]/N_j$
    $FC_i=FC_i+PT_j\times XHSS[e_i,T_j]/N_j$
  end for
  $Suspiciousness(e_i)=FC_i/FN/(FC_i/FN+PC_i/PN)$
end for
return *Suspiciousness*(*XSlice*)
end begin

2.3 基于层次切片谱的错误定位

根据层次切片谱的定义和可疑度模型,一个形象化的基于层次切片谱的错误定位模型见表2.表2中行表示各层次错误相关切片中的元素,列 $T_1,\ldots,T_m$ 表示 m 个测试执行,包括 s 个失效的运行和 $m-s$ 个成功的运行,$f_{i,j}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$ 是层次切片谱矩阵元素,其值为 k 或0,表示测试 T_j 执行元素 e_i 的频度.最后一列表示相应元素包含错误的可疑度,可疑度度量方法如第2.2节所示.基于层次切片谱的错误定位模型在谱的规模、谱矩阵元素值以及度量方法上与传统的基于程序谱的方法都有了较大的改进.算法 4 描述了基于层次切片谱的错误定位的逐层定位过程.

Table 2 Hierarchical slicing spectrum-based software fault localization model

表 2 基于层次切片谱的错误定位模型

XSlice	Test cases						Suspiciousness
	Failed			Passed			
	T_1	…	T_s	T_{s+1}	…	T_m	
e_1	$f_{1,1}$	…	$f_{1,s}$	$f_{1,s+1}$	…	$f_{1,m}$	p_1
e_2	$f_{2,1}$	…	$f_{2,s}$	$f_{2,s+1}$	…	$f_{2,m}$	p_2
…	…	…	…	…	…	…	…
e_n	$f_{n,1}$	…	$f_{n,s}$	$f_{n,s+1}$	…	$f_{n,m}$	p_n

算法 4. 基于层次切片谱的错误定位算法.

Function: ***GetBugLocation***(***SP***,***T***)

Inputs: *SP*: a program that includes bugs;
  T: a list of test cases $\{T_1,T_2,...,T_m\}$.

Outputs: *location*: a location in the program where a bug is located.

Declare: *PSlice*: a list of fault-related packages in the program *SP*, which is $\{P_1,P_2,...,P_{|PSlice|}\}$
  $CSlice_i$: a list of fault-related classes in the package P_i, which is $\{C_{i1},C_{i2},..., C_{i|CSlice_i|}\}$
  $MSlice_{ij}$: a list of fault-related methods in the class C_{ij}, which is $\{M_{ij1},M_{ij2},..., M_{ij|MSlice_{ij}|}\}$
  $SSlice_{ijk}$: a list of fault-related statements in the method M_{ijk}, which is $\{S_{ijk1},S_{ijk2},..., S_{ijk|SSlice_{ijk}|}\}$
  Suspiciousness(*XSlice*): a vector that denotes the suspiciousness of each element in *XSlice*

begin
  PSlice=*GetRelatedPSlice*(*SP*,*T*)
  Suspiciousness(*PSlice*)=*GetSuspiciousness*(*PSlice*,*GetPHSS*(*PSlice*,*T*),*T*)
  Sort *PSlice* to make $Suspiciousness(P_r)\geqslant Suspiciousness(P_t)$,where $1\leqslant r<t\leqslant|PSlice|$
for i=1; $i\leqslant|PSlice|$; i++ **do**

$CSlice_i$=*GetRelatedCSlice*(P_i,T)
Suspiciousness($CSlice_i$)=*GetSuspiciousness*($CSlice_i$,*GetCHSS*($CSlice_i$,T),T)
Sort $CSlice_i$ to make *Suspiciousness*(C_{ir})≥*Suspiciousness*(C_{it}),where 1≤r<t≤|$CSlice_i$|
for j=1; j≤|$CSlice_i$|; j++ **do**
$MSlice_{ij}$=*GetRelatedMSlice*(C_{ij},T)
Suspiciousness($MSlice_{ij}$)=*GetSuspiciousness*($MSlice_{ij}$,*GetMHSS*($MSlice_{ij}$,T),T)
Sort $MSlice_{ij}$ to make *Suspiciousness*(M_{ijr})≥*Suspiciousness*(M_{ijt}), where 1≤r<t≤|$MSlice_{ij}$|
for k=1; k≤|$MSlice_{ij}$|; k++ **do**
$SSlice_{ijk}$=*GetRelatedSSlice*(M_{ijk},T)
Suspiciousness($SSlice_{ijk}$)=*GetSuspiciousness*($SSlice_{ijk}$,*GetSHSS*($SSlice_{ijk}$,T),T)
Sort $SSlice_{ijk}$ to make *Suspiciousness*(S_{ijkr})≥*Suspiciousness*(S_{ijkt}), where 1≤r<t≤|$SSlice_{ijk}$|
for l=1; l≤|$SSlice_{ijk}$|; l++ **do**
if (S_{ijkl} inlucludes a bug)
{*location*=S_{ijkl}
Return *location*
}
end if
end for
end for
end for
end for
end begin

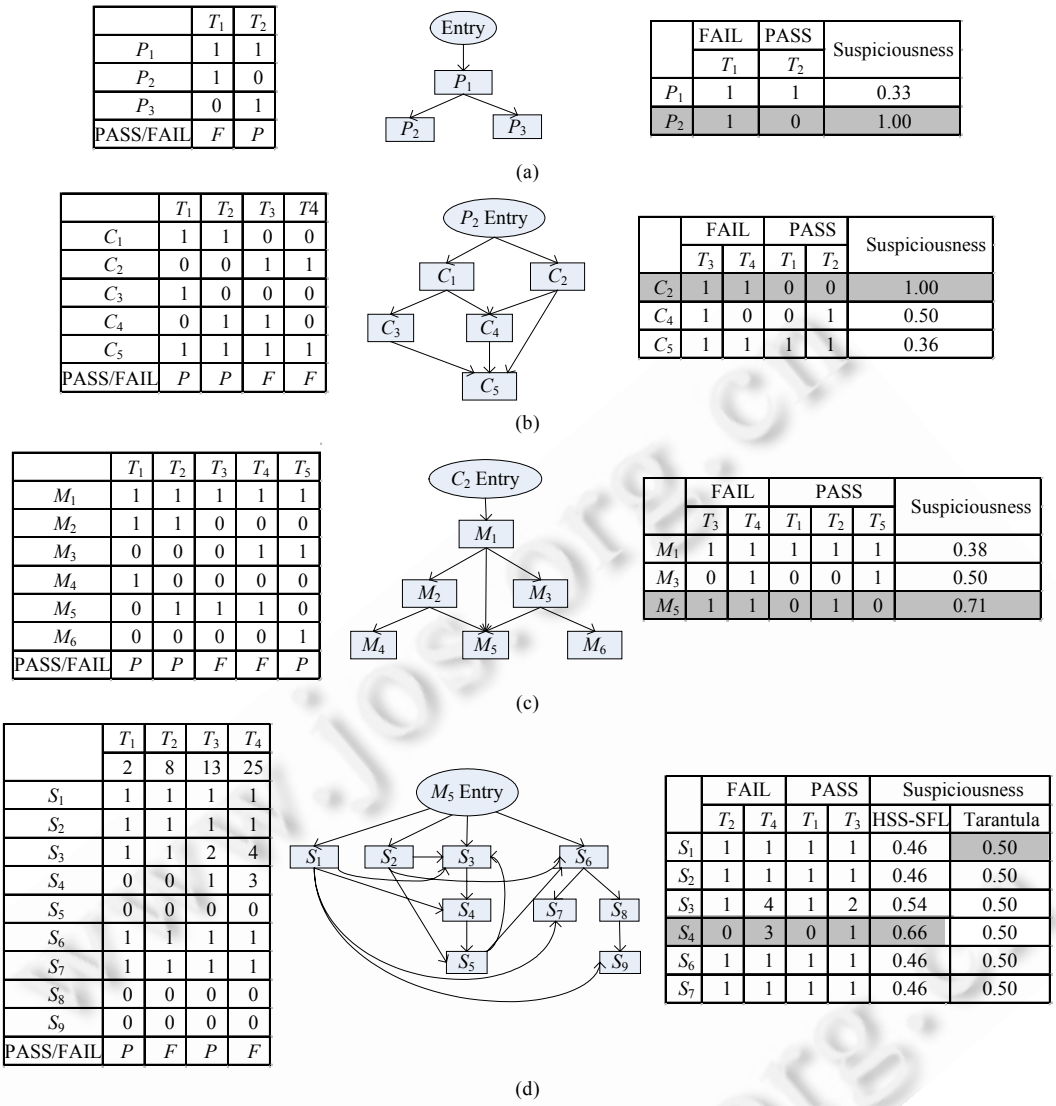
2.4 实例分析

本节通过一个简单实例来分析如何基于层次切片谱来进行面向对象程序的错误定位.图 2 是实例的具体流程.图中第 1 列和第 2 列给出了测试覆盖信息和各层次依赖图,在此基础之上,我们基于层次切片谱进行错误定位(图中第 3 列).在语句层,我们给出一个简单的实例程序来说明 HSS-SFL 进行错误定位的语句搜索顺序,并比较了 HSS-SFL 度量方法与 Tarantula 度量方法.

图 2(a)第 1 列是收集的包层测试覆盖表,表的内容描述的是测试 T_i 经过包 P_j 的次数,表的最后一行是测试的结果:P 表示测试结果正确,F 表示测试结果错误.图 2(a)第 2 列是根据源程序构造的包依赖图 PLDG.由于 T_1 的测试结果是错误的,我们在包依赖图上提取了错误相关包 *Slice*(T_1)={P_1,P_2},去除了错误无关包,并生成了包切片谱(如图 2(a)第 3 列所示).

根据可疑度算法,*failed*(P_1)%=(1×1/2)/1=0.5,*passed*(P_1)%=(1×1)/1=1,*suspiciousness*(P_1)=0.5/(0.5+1)=0.33.同理,由于 *passed*(P_2)%=0,我们得到 *suspiciousness*(P_2)=1.00.错误定位到可疑度最大的包 P_2,然后建立包 P_2 的依赖图.同样的原理,将错误定位到类 C_2,然后再定位到方法 M_5 中.

为了更好地表示 HSS-SFL 进行错误定位的语句搜索过程,我们给出了方法 M_5 的实例程序(如图 3 所示),其功能是接收一个大于 1 的正整数,判断它是否为素数.我们将错误植入在语句 S_3 中,这种错误也是现实程序开发中比较常见的错误.图 2(d)第 1 列是程序的测试覆盖信息表,表的第 1 行表示测试的参数 *num* 值分别为 2,8,13,25,图 2(d)第 2 列是相应的语句层次依赖图,第 3 列给出了 HSS-SFL 度量方法和 Tarantula 度量方法计算的可疑度值.根据可疑度大小顺序,HSS-SFL 技术语句搜索顺序是 S_4,S_3,通过 2 次搜索,最终将错误定位在 S_3 中.如果使用 Tarantula 技术的度量方法,并将相同可疑度的语句按语句被执行的自然顺序进行定位,则错误定位的次序依次是 S_1,S_2,S_3,需要进行 3 次才能准确定位到错误语句.



	T_1	T_2
P_1	1	1
P_2	1	0
P_3	0	1
PASS/FAIL	F	P

	FAIL	PASS	Suspiciousness
	T_1	T_2	
P_1	1	1	0.33
P_2	1	0	1.00

	T_1	T_2	T_3	$T4$
C_1	1	1	0	0
C_2	0	0	1	1
C_3	1	0	0	0
C_4	0	1	1	0
C_5	1	1	1	1
PASS/FAIL	P	P	F	F

	FAIL		PASS		Suspiciousness
	T_3	T_4	T_1	T_2	
C_2	1	1	0	0	1.00
C_4	1	0	0	1	0.50
C_5	1	1	1	1	0.36

	T_1	T_2	T_3	T_4	T_5
M_1	1	1	1	1	1
M_2	1	1	0	0	0
M_3	0	0	0	1	1
M_4	1	0	0	0	0
M_5	0	1	1	1	0
M_6	0	0	0	0	1
PASS/FAIL	P	P	F	F	P

	FAIL		PASS			Suspiciousness
	T_3	T_4	T_1	T_2	T_5	
M_1	1	1	1	1	1	0.38
M_3	0	1	0	0	1	0.50
M_5	1	1	0	1	0	0.71

	T_1	T_2	T_3	T_4
	2	8	13	25
S_1	1	1	1	1
S_2	1	1	1	1
S_3	1	1	2	4
S_4	0	0	1	3
S_5	0	0	0	0
S_6	1	1	1	1
S_7	1	1	1	1
S_8	0	0	0	0
S_9	0	0	0	0
PASS/FAIL	P	F	P	F

	FAIL		PASS		Suspiciousness	
	T_2	T_4	T_1	T_3	HSS-SFL	Tarantula
S_1	1	1	1	1	0.46	0.50
S_2	1	1	1	1	0.46	0.50
S_3	1	4	1	2	0.54	0.50
S_4	0	3	0	1	0.66	0.50
S_6	1	1	1	1	0.46	0.50
S_7	1	1	1	1	0.46	0.50

Fig.2 Fault locating process of HSS-SFL

图 2 HSS-SFL 错误定位过程

```
S1  public void  M5 (int num)
     {
S2    int i, result=1;
S3    for (i=2; i<(int)math.sqrt(num); i++)
      //BUG, i≤(int)math.sqrt(num);
        {
S4      if (num%i==0)
S5      result=0;
        }
S6    if (result==1)
S7      System.out.println(num+"is a prime number");
S8    else
S9      System.out.println(num+"is not a prime number");
     }
```

Fig.3 Program example at the statement level

图 3 语句层程序实例

3 实验分析

HSS-SFL 技术首先通过提取错误相关切片缩小了错误查找的范围,降低了层次切片谱的规模,然后通过度量程序元素的可疑度来更加精确地定位程序中的错误.故本文的实验主要研究如下两个问题:

(1) 错误相关切片能够在多大程度上删除面向对象程序中与错误不相关的部分;

(2) HSS-SFL 错误定位效率及与相关的错误定位技术比较.

3.1 实验对象

当前,大部分实验对象都是选用西门子程序进行(http://sir.unl.edu/php/showfiles.php),但是西门子程序规模比较小(100行~500行代码),而且主要是由一些面向过程的C语言程序构成,不适合本文的面向对象程序的错误定位技术研究.本文选取的 3 个面向对象程序都是用 Java 实现,见表 3.表 3 的第 1 列是对象名称;第 2 列~第 6 列分别统计了各个程序所包含的包数、类数、方法数和代码行数;第 7 列是实验中我们在 3 个对象程序中分别植入的错误数,我们一共植入了 54 个不同的错误;第 8 列是程序的简单描述.其中,Tetris 是大家熟悉的俄罗斯方块游戏程序(http://www.percederberg.net/games/tetris/tetris-1.2-src.zip);SimpleJavaApp 是代码覆盖工具 CodeCover 的例子程序(http://codecover.org/documentation/tutorials/SimpleJavaApp.zip),主要用来显示、编辑书目列表,并可以将列表以 XML 文件形式保存并加载;JHSA 是我们小组所开发的一个 Java 层次切片工具,用于构造 Java 程序的层次依赖图,计算 Java 程序的层次切片,以及在此基础之上的一些软件维护应用.

Table 3 Experiment subject

表 3 实验对象

面向对象程序	包	类	方法	语句	错误数	描述
Tetris	1	6	98	2 397	16	俄罗斯方块游戏
SimpleJavaApp	4	7	128	1 784	15	显示、编辑书目列表
JHSA	3	26	521	11 201	23	Java 层次切片工具

3.2 实验度量方法

本实验研究的第 1 个问题是各层次错误相关切片能够删除多少错误无关部分以帮助程序开发者在不同层次缩小错误查找的范围.我们收集了从包层次、类层次、方法层次到语句层次的错误相关切片,并计算了各个层次的错误相关切片平均覆盖度,即错误相关切片规模占元素总规模的平均百分比(P_{XSlice}),定义如下:

$$P_{XSlice}=\frac{|XSlice|}{|E|}\times 100\% \tag{12}$$

其中,|*XSlice*|表示各个层次(包层次、类层次、方法层次、语句层次)错误相关切片中包含的元素个数,|*E*|表示错误相关切片所在层次的元素总规模.错误相关切片平均覆盖度 P_{XSlice} 越小,说明平均删除的错误无关部分越多.

错误定位技术最终要通过逐条检查语句来确定错误语句,所以,同理,我们用查找错误所需要访问的语句数占程序总规模之比(P_{search})来衡量错误定位的效率,定义如下:

$$P_{search}=\frac{s}{|SP|}\times 100\% \tag{13}$$

其中,*s* 表示成功定位到错误所需要访问的语句数,分母表示程序 *SP* 中所包含总的语句条数.P_{search} 越小,定位错误所需要查找的语句数越少,效率越高.

3.3 实验过程

本实验通过如下几步进行实验数据的收集与分析:

(1) 利用工具 codecover(http://www.codecover.org/)收集各个测试覆盖程序元素(包、类、方法、语句)的信息;

(2) 利用层次切片工具 JHSA 计算失效测试的包层、类层、方法层、语句层各个层次的错误相关切片结果,并计算各个层次错误相关切片的覆盖度 P_{XSlice};

(3) 构造从包层次到语句层次的各粒度层次的层次切片谱;

(4) 计算各个层次错误相关切片元素的可疑度;

(5) 根据可疑度大小在各个层次进行错误定位,记录定位每个错误需要查找的语句数,并计算 P_{search};

(6) 分析统计了当前流行的 3 种错误定位技术的效率,并与 HSS-SFL 技术进行了比较.

3.4 实验结果与分析

本节针对问题 1 和问题 2 这两项研究收集了相关数据,并分析了 HSS-SFL 错误定位技术的有效性.

3.4.1 问题 1

本文使用程序切片技术进行错误定位,主要就是因为程序切片能够在错误定位过程中删除一些与错误无关的元素,能够解决传统的基于程序谱的错误定位技术因未充分考虑程序本身固有的依赖信息而使定位精度受限的问题.本文使用逐步求精的层次切片技术计算从包层次到语句层次的切片结果.所以我们研究的第 1 个问题即是层次切片技术在各个粒度层次能够删除元素的程度,实验中用前面定义的平均覆盖度 P_{XSlice} 来表示,它表示各层次切片所包含元素的规模与总元素规模的百分比.层次切片技术在逐步求精的过程中,到达语句层次时,已经删除了很大一部分错误无关语句,这样就缩小了我们错误定位的范围.表 4 列出了各层次错误相关切片的平均覆盖度.对象程序 Tetris 和 SimpleJavaApp 由于包之间的依赖和类之间的依赖很强,他们的包层和类层平均覆盖度依然很大,删除的错误无关部分比例很小;在方法层上,方法之间的依赖明显降低,因而平均覆盖度下降很大;到了语句层,平均覆盖度分别达到了 0.34 和 0.23.对象程序 JHSA 在包层、类层和方法层平均覆盖度比前两个程序明显减小,主要是因为在执行构造 SDG 图时,仅用到 SDG 和 AST 两个包,这在很大程度上降低了包、类和方法的平均覆盖度;在语句层,平均覆盖度与前两个对象程序相当,但由于程序规模相对较大,绝对代码量比前两个对象大很多.这主要是因为层次切片的构造过程相对复杂,远高于俄罗斯方块中一个简单的动作,也高于对一个书目列表的简单操作.表 4 中最后一行列出 3 个对象程序各层次覆盖度的平均值.在语句层,平均覆盖度达到了 0.28,这有效地缩小了错误定位的范围.

Table 4 Average coverage of the fault-related slice at each level

表 4 各个层次错误相关切片平均覆盖度

对象程序	P_{Psilce}	P_{Cslice}	P_{Mslice}	P_{Sslice}
Tetris	1	0.83	0.57	0.34
SimpleJavaApp	0.89	0.8	0.56	0.23
JHSA	0.78	0.53	0.32	0.27
平均	0.85	0.69	0.48	0.28

3.4.2 问题 2

本节分析统计了 HSS-SFL 技术的效率,并与当前盛行的 Tarantula 技术、并集谱定位技术(Union)和交集谱定位技术(Intersection)[18]进行了比较.Tarantula 技术度量方法在前面已有介绍.给定一个失效的测试运行 f 和正确的测试运行集合 P,Union 技术通过计算 $f-\bigcup_{p\in P}p$ 来进行错误定位,而 Intersection 技术是通过其补集运算 $\bigcap_{p\in P}p-f$ 来进行错误定位.

实验中,由于 3 个对象程序规模大小不一,我们采用第 3.2 节定义的 P_{search},即成功定位到错误查找的语句数占整个程序规模的百分比来衡量错误定位的效率.表 5 将 P_{search} 分成了 12 段,然后分别统计了 4 种技术在不同的 P_{search} 段成功定位到的错误数.如果在越小的段内,定位到越多的错误,那么错误定位技术的效果越好.具体来说,因为 HSS-SFL 技术成功定位到的错误需要查找的代码量大多数集中在程序总代码量的 10%以内,为了更加精确地研究错误定位技术的效率,我们将 0~10%分成了 3 段,10%以后每间隔 10%分一段,然后分别统计了上述 4 种技术成功定位到错误数在不同的段内的分布情况.这 4 种技术进行错误定位的顺序是:HSS-SFL 以及 Tarantula 技术根据可疑度大小的顺序依次定位错误,Union 和 Intersection 技术先顺序查找集合内的程序代码,如果错误不在集合内,则定位错误的次序根据失效测试的语句执行顺序进行错误定位.根据这种方法,我们得到了表 4 中的统计数据.其中,HSS-SFL 和 Tarantula 技术在前 10%段成功定位到的错误数最多,Union 和

Intersection 技术成功定位到错误的版本主要集中在 20%~70%段.

图 4 是根据表 5 中数据得到的直观的错误定位技术的效率图,横坐标是 P_{search}×100,纵坐标 *Bugs* 是随着 P_{search} 的增加成功定位到错误数.当横坐标值相同时,纵坐标值越大说明效率越高.从图 4 可以看出,Intersection 技术的效率最低,当查找到程序代码的 5%时,我们仍然未能正确定位到一个错误,即使查找到代码的 30%,能够成功定位到的错误数仍然很少.这主要是因为我们的实验中没有错误语句位于 Intersection 集合中,所有的错误语句都是在集合外根据程序语句执行的顺序定位到的.即便如此,我们对 Intersection 技术的统计结果仍优于文献[18]中的统计结果,这主要是因为我们错误语句在交集外的统计方法不同.Union 技术的效率有所提高,主要是因为本实验在很多时候,我们的正确运行执行不到错误语句.在这种情况下,Union 集合包含错误的可能性很大,但由于没有对集合内的语句给出不同的优先级,错误定位效率低于另外两种技术.实验中,HSS-SFL 技术展示了较高的定位效率.从实验统计数据来看,在 0~10%段,HSS-SFL 成功定位到 31 个错误,而其他 3 种技术分别是 25,14 和 2.这说明 HSS-SFL 技术对错误进行定位时,能够较快地定位到错误位置,在其他 P_{search} 段,图 4 给出的效率曲线也展示了 HSS-SFL 较高的错误定位效率.

Table 5 Distribution of successfully locating faults at different P_{search} ranges

表 5 不同 P_{search} 段成功定位到的错误数分布表

P_{search} (%)	HSS-SFL	Tarantula	Union	Inter
0~1	9	5	2	0
1~5	13	11	7	0
5~10	9	9	5	2
10~20	7	7	9	3
20~30	5	8	12	5
30~40	7	6	11	8
40~50	1	4	3	11
50~60	0	1	1	9
60~70	2	2	3	10
70~80	0	0	0	4
80~90	1	1	1	2
90~100	0	0	0	0

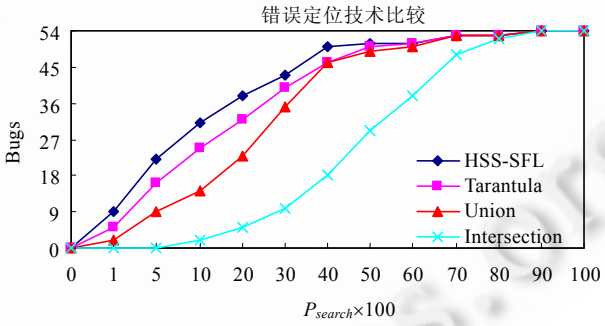


Fig.4 Comparison of software fault localization techniques

图 4 4 种错误定位技术的效率比较

3.5 讨 论

通过上述实验的结果与分析,文中使用层次切片技术能够有效地缩小错误定位的范围.另外,HSS-SFL 错误定位技术比基于程序谱的 Tarantula 技术、Union 技术和 Intersection 技术的效率有了进一步的提高.但我们的实验过程中也还存在着一些问题.

首先,Tarantula 可疑度度量方法随着程序顺序块的不断增多,精度会不断下降.本文的 HSS-SFL 可疑度度量方法虽然优化了 Tarantula 度量方法,引入程序切片技术和各层次元素执行的频度以及贡献度在一定程度上减小了这种退化,但这种退化现象仍然存在于 HSS-SFL 技术中.其次,当程序错误是由于程序中遇到语句丢失而引起时,通过我们的可疑度度量算法,最终可以将错误定位到丢失语句位置正常执行序列中的上一条语句,从而可

以根据这条语句来找到相应的错误.如果程序错误是由程序结构不合理引起的,我们的错误定位算法可以将错误定位在不合理结构的更高层次上,这两种情况需要配合更多的人工分析.最后,因为基于程序谱的错误定位技术是通过统计程序元素的失效测试和成功测试来度量的,如果程序中同时包含多个错误,那么基于程序谱的错误定位技术的效率会随着错误的增多而降低,在本文的错误定位技术中也存在这样的情况.

4 相关工作

错误定位技术因其重要性与困难性一直受到软件工程研究者的广泛关注与研究.本文首先根据层次切片技术逐层提取了错误相关切片,然后对错误相关切片中的元素进行了统计度量,这主要涉及两个方面的错误定位技术:基于程序切片技术的错误定位技术和基于程序谱的错误定位技术.

在一个实际的软件中,单一的程序切片由于规模较大而不能帮助程序员有效地定位错误.因此,主流的基于程序切片的技术采用集合运算方式进行错误定位.削片(dice)[19]技术通过计算包含错误的切片与正确的切片之差来初步确定错误的位置.执行切片[20]是根据特定的输入得到的程序执行基本块,这在提取效率上比传统的切片有所提高.通过执行切片的并集和交集运算来进行错误定位,能够进一步提高错误定位的效率.执行切片的并集错误定位思想是计算包含某条错误的执行切片与所有正确的执行切片的并集之差来确定错误的位置;执行切片的交集错误定位思想是计算所有正确切片的交集与某条包含错误的执行切片之差来进行错误定位的方法.文献[18]提出了一种将某条错误的执行切片与这条错误的执行距离最近的正确的执行切片之差来进行错误定位,并与交集与并集错误定位错位进行了比较.另外,DeMillo 等人根据变异测试的“语句删除”变异操作的思想提出了关键切片错误定位的方法[21];Burger 等人更进一步地结合了动态切片、事件切片以及 Delta 调试技术来降低错误定位的平均搜索域[22].我们的工作与这些技术是不同的.我们利用程序切片技术在面向对象程序中逐层提取了错误相关的切片,对程序的不同执行是以谱矩阵的形式存储,且对错误相关切片中元素的可疑度进行了度量,然后根据可疑度进行错误的搜索定位.

基于程序谱的软件错误定位技术一般根据测试覆盖信息计算程序元素发生错误的可疑度,然后根据可疑度大小顺序来进行错误定位.最初,Jones 提出了 Tarantula 技术计算程序语句的可疑度[10].其基本思想是,通过某条语句的失效的测试越多,而成功的测试越少,这条语句的可疑度就越大,发生错误的概率也就越高.随后,Abreu 等人将 Pinpoint 工具中使用的 Jaccard 度量方法与分子生物学领域的 Ochiai 度量方法应用到语句可疑度的计算中,并与 Tarantula 可疑度计算方法进行了比较[11].Masri 等人将 Tarantula 可疑度计算方法应用到程序信息流的度量上,提出了一种基于信息流覆盖的错误定位方法,并分别与基于语句、分支、定义使用对的程序谱错误定位方法进行了比较[23];更进一步地,Santelices 等人通过将分支、定义使用对映射到语句,通过统一计算语句的可疑度分析了基于语句、分支、定义使用对的程序谱错误定位方法的优劣,然后组合了这些方法,提出了使用多覆盖类型的错误定位方法[24].基于 Tarantula 可疑度计算方法,Jones 等人随后又提出一种并行错误定位方法[25].这种方法创建了不同的测试用例集合,使不同的开发者能够同时进行多错误的定位.另外,Liblit 和 Liu 等人结合了程序谱和统计学方法对程序中样本谓词元素进行度量从而实现错误的分离和定位[26,27].Abreu 等人从 MBD(model based diagnosis,一种行为模型上的逻辑推理方法)中获得启发,将程序语句视为模型组件,从而将 Bayasian 方法应用到基于程序谱的错误定位定位中,以计算每个错误候选(语句组成)的可疑度[17,28].Park 等人提出了一种并发程序的错误定位技术[29].这种技术使用 Jaccard 可疑度计算方法,对并发程序中不良的并发序列(pattern)进行了可疑度度量.进一步地,Artzi 等人将基于程序谱的错误定位技术应用到动态 Web 应用程序中[30].该技术通过使用源映射函数以及扩展条件语句和函数调用语句的语句域来提高 Tarantula,Ochiai,Jaccard 技术在动态 Web 应用程序中的错误定位效率.在测试用例优化方面,Wong 等人指出,成功的测试对基于程序谱的错误定位贡献不同,提出一种使用权重函数来统计成功测试的基于程序谱的错误定位方法[16];Yu 等人通过实验分析了 10 种测试用例的削减策略对错误定位效率的影响,以帮助开发者权衡削减和定位的效率[31];Hao 等人提出了 3 种可以削减测试用例但仍然能够有效进行错误定位的策略[32].本文的技术与上述技术是不同的,主要体现在:首先,我们使用程序切片技术缩小了错误定位搜索域,然后仅对错误相关部分的元素进行了度量;其次,本文

针对面向对象程序采用了逐层定位的方法;最后,本文的可疑度度量模型与传统的可疑度度量模型是不同的,我们在删除与错误无关程序元素的基础上,将元素的执行频度以及贡献度引入可疑度度量模型中,以改进错误定位的效率.

5 结论与下一步的工作

本文提出了一种有效的面向对象程序的错误定位技术——HSS-SFL 技术.这种技术首先利用层次切片技术提取了各层次错误相关切片,减小了程序谱的规模;其次,文中将程序元素的执行频度和贡献度引入到可疑度度量中,改进了传统的基于程序谱的度量方法;最后,由于建立整个程序的语句级程序切片谱规模过大,我们采用了逐层定位的方式,依次将错误定位到包、类、方法,最后定位到相应的语句.通过实验研究与分析,HSS-SFL 技术能够有效减小程序谱规模,并能高效地定位程序中的错误.

我们的实验虽然在现实中的面向对象程序上验证了我们的技术的有效性,但是由于 HSS-SFL 技术像其他错误定位技术一样,随着程序中错误的增多,效率会不断降低.我们下一步的工作将着重研究在层次切片谱基础上如何提高多错误程序中错误定位的效率.

References:

[1] Bandyopadhyay A. Improving spectrum-based fault localization using proximity-based weighting of test cases. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lawrence: IEEE Computer Society, 2011. 660−664. [doi: 10.1109/ASE.2011.6100150]

[2] Jose M, Majumdar R. Cause clue clauses: Error localization using maximum satisfiability. In: Proc. of the 32nd ACM SIGPLAN Conf. on Programming Language Design and Implementation. San Jose: ACM, 2011. 437−446. [doi: 10.1145/1993316.1993550]

[3] Wen W, Li B, Sun X, Li J. Program slicing spectrum-based software fault localization. In: Proc. of the 23rd Int'l Conf. on Software Engineering and Knowledge Engineering. Miami: Knowledge Systems Institute Graduate School, 2011. 213−218.

[4] Zeller A, Hildebrandt R. Simplifying and isolating failure-inducing input. IEEE Trans. on Software Engineering, 2002,28(2): 183−200. [doi: 10.1109/32.988498]

[5] Cleve H, Zeller A. Locating causes of program failures. In: Proc. of Int'l Conf. on Software Engineering. St. Louis: IEEE Computer Society, 2005. 342−351. [doi: 10.1145/1062455.1062522]

[6] Weiser M. Program slicing. IEEE Trans. on Software Engineering, 1984,10(4):352−357. [doi: 10.1109/TSE.1984.5010248]

[7] Weiser M. Programmers use slices when debugging. Communications of the ACM, 1982,25(7):446−452. [doi: 10.1145/358557.358577]

[8] Agrawal H, DeMillo RA, Spafford EH. Debugging with dynamic slicing and backtracking. Software-Practice and Experience, 1993, 23(6):589−616. [doi: 10.1002/spe.4380230603]

[9] Li B, Fan X, Pang J, Zhao J. A model for slicing java programs hierarchically. Journal of Computer Science and Technology, 2004, 19(6):848−858. [doi: 10.1007/BF02973448]

[10] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the 24th Int'l Conf. on Software Engineering. Orlando: ACM, 2002. 467−477. [doi: 10.1145/581339.581397]

[11] Abreu R, Zoeteweij P, van Gemund AJC. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing: Academic and Industrial Conf.—Practice and Research Techniques. Windsor: IEEE Computer Society, 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[12] Harrold MJ, Rothermel G, Wu R. An empirical investigation of program spectra. In: Proc. of the 1998 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. Montreal: ACM, 1998. 83−90. [doi: 10.1145/277633.277647]

[13] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault localization technique. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering. Long Beach: IEEE Computer Society, 2005. 273−282. [doi: 10.1145/1101908.1101949]

[14] Ferrante J, Ottenstein J, Warren JD. The program dependence graph and its use in optimization. ACM Trans. on Programming Languages and Systems, 1987,9(3):319−349. [doiI: 10.1145/24039.24041]

[15] Wen W. Software fault localization based on program slicing spectrum. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE Computer Society, 2012. 1511−1514. [doi: 10.1109/ICSE.2012.6227049]

[16] Wong E, Qi Y, Zhao L, Cai KY. Effective fault localization using code coverage. In: Proc. of the Int'l Computer Software and Applications Conf. Beijing: IEEE Computer Society, 2007. 449−456. [doi: 10.1109/COMPSAC.2007.109]

[17] Abreu R, Gonzlez A. Exploiting count spectra for bayesian fault localization. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. Timisoara: Association for Computing Machinery, 2010. 1−10. [doi: 10.1145/1868328.1868347]

[18] Renieris M, Reiss SP. Fault localization with nearest neighbor queries. In: Proc. of the 18th IEEE Int'l Conf. on Automated Software Engineering. Montreal: IEEE Computer Society, 2003. 30−39. [doi: 10.1109/ASE.2003.1240292]

[19] Chen TY, Cheung YY. Dynamic program dicing. In: Proc. of the Conf. on Software Maintenance. Montreal: IEEE Computer Society, 1993. 378−385. [doi: 10.1109/ICSM.1993.366925]

[20] Wong E, Qi Y. Effective program debugging based on execution slices and inter-block data dependency. Journal of Systems and Software, 2006,79(7):891−903. [doi: 10.1016/j.jss.2005.06.045]

[21] DeMillo RA, Pan H, Spafford EH. Critical slicing for software fault localization. In: Proc. of the 1996 ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. San Diego: ACM 1996. 121−134. [doi: 10.1145/226295.226310]

[22] Burger M, Zeller A. Minimizing reproduction of software failures. In: Proc. of the Int'l Symp. on Software Testing and Analysis. Toronto: Association for Computing Machinery, 2011. 221−231. [doi: 10.1145/2001420.2001447]

[23] Masri W. Fault localization based on information flow coverage. Software Testing, Verification Reliability, 2010,20(2):121−147. [doi: 10.1002/stvr.v20:2]

[24] Santelices R, Jones JA, Yu YB, Harrold MJ. Lightweight fault-localization using multiple coverage types. In: Proc. of the 31st Int'l Conf. on Software Engineering. Vancouve: IEEE Computer Society, 2009. 56−66. [doi: 10.1109/ICSE.2009.5070508]

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

[26] Liblit B, Naik M, Zheng AX, Aiken A, Jordan MI. Scalable statistical bug isolation. In: Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation. Chicago: Association for Computing Machinery, 2005. 15−26. [doi: 10.1145/ 1064978.1065014]

[27] Liu C, Fei L, Yan XF, Han JW, Midkiff SP. Statistical debugging: A hypothesis testing-based approach. IEEE Trans. on Software Engineering, 2006,32(10):831−848. [doi: 10.1109/TSE.2006.105]

[28] Abreu R, Zoeteweij P, Gemund AV. Spectrum-Based multiple fault localization. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. Auckland: IEEE Computer Society, 2009. 88−99. [doi: 10.1109/ASE.2009.25]

[29] Park S, Vuduc RW, Harrold MJ. Falcon: Fault localization in concurrent programs. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. Cape Town: ACM, 2010. 245−254. [doi: 10.1145/1806799.1806838]

[30] Artzi S, Dolby J, Tip F, Pistoia M. Fault localization for dynamic Web applications. IEEE Trans. on Software Engineering, 2012, 38(2):314−335. [doi: 10.1109/TSE.2011.76]

[31] Yu YB, Jones JA, Harrold MJ. An empirical study of the effects of test suite reduction on fault localization. In: Proc. of the Int'l Conf. on Software Engineering. Leipzig: ACM, 2008. 201−210. [doi: 10.1145/1368088.1368116]

[32] Hao D, Xie T, Zhang L, Wang X, Sun J, Mei H. Test input reduction for result inspection to facilitate effective fault localization. Automated Software Engineering, 2010,17(1):5−31. [doi: 10.1007/s10515-009-0056-x]

文万志(1982－),男,安徽桐城人,博士生,主要研究领域为程序切片技术及其应用,软件错误定位,软件演化与维护.
E-mail: wen@seu.edu.cn

孙小兵(1985－),男,博士,CCF 会员,主要研究领域为程序分析,软件维护及演化,修改分析.
E-mail: xbsun@yzu.edu.cn

李必信(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为程序切片技术及其应用,软件演化与维护,软件建模、分析、测试与验证.
E-mail: bx.li@seu.edu.cn

刘翠翠(1987－),女,硕士生,主要研究领域为 Web 服务可信性评估.
E-mail: liucuicui@seu.edu.cn