基于动态切片和关联分析的错误定位方法

曹鹤玲[1),2)] 姜淑娟[1)] 鞠小林[1),3)] 王兴亚[1)]

[1)](中国矿业大学计算机科学与技术学院 江苏 徐州 221116)

[2)](河南工业大学信息科学与工程学院 郑州 450001)

[3)](南通大学计算机科学与技术学院 江苏 南通 226019)

摘 要 错误定位是软件调试中非常耗时费力的活动之一,自动错误定位技术可以提高调试效率,降低调试成本.该文提出一种把动态切片、关联分析及排序策略相结合的错误定位方法.首先,收集程序执行的动态切片及相应的执行结果构建混合谱矩阵;然后,基于混合谱矩阵进行关联分析,随后依据提出的排序策略对语句进行排序,得出较合理的语句优先级次序,从而进行错误定位.为验证该方法有效性,作者设计并实现了一个错误定位原型工具DSFL,针对一组 Java 基准程序开展错误定位实验,并与 12 种错误定位技术进行对比.实验结果表明该方法可以在一定程度上提高错误定位精度和效率.

关键词 动态切片;关联分析;错误定位;排序策略

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02188

Fault Localization Based on Dynamic Slicing and Association Analysis

CAO He-Ling[1),2)] JIANG Shu-Juan[1)] JU Xiao-Lin[1),3)] WANG Xing-Ya[1)]

[1)](*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

[2)](*College of Information Science and Engineering, Henan University of Technology, Zhengzhou* 450001)

[3)](*School of Computer Science and Technology, Nantong University, Nantong, Jiangsu* 226019)

Abstract Fault localization is one of the most difficult and time-consuming activities of debugging. Automatic fault localization technique can improve the effectiveness of debugging and decrease the cost of debugging. This paper proposes an approach, combining dynamic slicing with association analysis and rank strategy to locate faults within programs. First, dynamic slices are computed and the corresponding test result is obtained, in which hybrid spectrum matrix is constructed. Second, association analysis is carried out based on hybrid spectrum matrix and rank strategy we proposed is used to rank statements to give a more reasonable priority sequence of statements. Moreover, a prototype tool, DSFL, has been implemented to evaluate our approach. In addition, our approach is compared with 12 fault localization techniques across a set of Java programs. The experimental results show that our approach is more precise and effective than the compared techniques.

Keywords dynamic slicing; association analysis; fault localization; rank strategy

1 引 言

现代技术的发展使得软件广泛应用于社会的各个领域,随着软件规模越来越大,结构越来越复杂,存在错误的几率也越来越高.软件错误的存在可能会导致软件系统失效或崩溃,甚至会造成巨大损失和灾难.这对软件调试效率提出了较高要求.软件调

收稿日期:2014-03-04;最终修改稿收到日期:2015-03-12.本课题得到国家自然科学基金(60970032,61202006,61340037)和广西可信软件重点实验室研究课题(kx201530,kx201532)资助.**曹鹤玲**,女,1980 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为软件分析与测试、数据挖掘.**姜淑娟**(通信作者),女,1966 年生,博士,教授,博士生导师,主要研究领域为编译技术、软件工程等.E-mail:shjjiang@cumt.edu.cn.**鞠小林**,男,1976 年生,博士研究生,讲师,主要研究方向为软件分析与测试.**王兴亚**,男,1990 年生,博士研究生,主要研究方向为软件分析与测试.

试代价高、耗时长，占软件开发维护过程代价的50%～80%[1]. 错误定位，即在程序执行失败的情况下找出可能错误位置的过程[2]，是软件调试中不可缺少的关键性工作之一. 错误定位有利于软件错误的快速检测与修复，可以降低软件调试成本[2]. 因此，研究错误定位具有重要意义.

软件错误定位方法主要通过缩小错误搜索范围来提高效率[3-5]. 早期程序员通过调试工具设置断点来缩小错误查找范围，但这种手工调试方式效率比较低. Delta 调试技术[4]采用迭代运行程序的方式，不断交换失败与成功运行的内存状态，从而缩小错误定位范围，但迭代搜索代价较大. 动态切片技术[5]则通过去除与程序错误不相关的语句来缩小错误搜索范围. 然而，计算动态切片时，很少考虑变量定义时实际引用的变量，使得切片结果包含部分不相关变量的切片，结果冗余. 为此，本文提出了一种改进的动态切片方法，该方法在计算动态切片时，找出当前变量在定义时的实际引用变量，排除未引用变量，使得动态切片结果更加精确.

基于覆盖信息的错误定位以程序中语句或基本块为研究对象，统计不同执行轨迹的覆盖信息进行怀疑度计算. 如 Jones 等人[6-7]提出了 Tarantula 方法；Abreu 等人[8]使用分子生物学领域的相似系数提出了 Ochiai 方法；Naish 等人[9]提出了 Naish1、Naish2 方法. 这类方法通过计算语句怀疑度得出检查语句的先后次序以定位错误. 然而，这类方法对执行轨迹进行了高度简化并忽略了程序内部固有的依赖关系，定位精度受限. 通过分析程序执行过程，可以发现可疑代码与程序执行失败有着某种关联，找到这种关联能更有效地定位错误. 关联分析技术能够从大量程序执行信息中获取语句错误的相关性信息，反映出执行轨迹中语句与执行结果的关系. 为此，本文提出了一种关联分析与排序策略相结合确定语句检查优先级次序的方法.

基于上述思路，本文提出了动态切片、关联分析及排序策略相结合的错误定位方法 DS-FLAR (Dynamic Slicing-Fault Localization based on Association analysis and Rank strategy). 该方法有如下优点：(1) 动态切片技术可以缩小错误定位范围，将可疑语句缩小为与失败输出变量相关的语句；(2) 关联分析及排序策略相结合的方法(FLAR)可以提高错误定位的精度，即提高错误语句在 Rank 列表中的次序.

本文主要贡献包括：

(1) 提出了一种排序策略，对关联分析后的语句进行排序，生成更加合理的语句检查序列.

(2) 提出了基于动态切片、关联分析及排序策略的错误定位方法，从而得到更加精确的错误定位结果.

本文第 2 节介绍研究动机；第 3 节详细描述本文方法；第 4 节介绍实验结果及相关分析；第 5 节介绍相关工作；第 6 节总结全文，并明确今后进一步的研究工作.

2　研究动机

软件错误定位存在搜索域过大与精度不高的问题. 错误定位方法常使用动态切片技术缩小错误定位范围，而现有前向计算动态切片方法[10-11]计算语句定义变量的切片时，考虑语句中所有引用变量的切片，而不是变量定义时实际引用变量的切片，造成切片结果冗余. 为此，本文对定义变量进行影响集分析，考虑变量定义时实际引用变量的切片结果，从而对原动态切片方法[11]进行改进. 动态切片技术在一定程度上缩小了错误定位范围，但切片后的程序为语句集合，集合内元素无先后次序，测试人员需检查切片中所有语句以定位错误. 关联分析技术能找到执行轨迹中语句与程序执行失败的关联，这种关联能有效地定位错误. 为此，本文提出了关联分析及排序策略相结合的方法(FLAR)来确定语句检查优先级次序，用于提高定位错误精度. 下面以图 1 所示程序(错误在 s_{10})为例来说明本文的研究动机.

```
s1  sub(i, j, k) {                 s9   m=sub(x, y, z)+z;
s2    if (k >0)                    s10  x=x/z; //correct: x=x×z;
s3      return i;                  s11  y=x+1;
      else                         s12  if(x>1)
s4      return j; }                s13    x=x×z;
s5  main() {                       s14  if(m>0)
    int x, y, z, r;                s15    r=x−z;
s6  x=read( );                           else
s7  y=read( );                     s16    r=y+z;
s8  z=read( );                     s17  print(r);}
```

图 1　示例程序

$$\text{Tarantula}=\frac{\text{failed}(s)/\text{totalfailed}}{\text{passed}(s)/\text{totalpassed}+\text{failed}(s)/\text{totalfailed}} \quad (1)$$

针对图 1 中示例程序，我们构造 8 个测试用例如下 $t_1(5,2,5)$，$t_2(4,1,4)$，$t_3(-4,2,-2)$，$t_4(1,4,1)$，$t_5(1,0,-1)$，$t_6(-2,-4,1)$，$t_7(0,4,1)$和$t_8(0,0,-1)$，并分别收集其执行的覆盖信息、(原/改进)动态切片

信息(〈I,s_{17},r〉为切片准则).比较 Tarantula[6-7]和本文 FLAR 方法(详见 3.3 节)在覆盖信息、原动态切片[11]和改进动态切片上的错误定位效果,如表 1 所示("●"表示语句被覆盖,"○"表示语句未被覆盖).第 1 列为语句编号,第 2 列为覆盖信息及 Tarantula(简写为 Tar.)和 FLAR 在其上计算得到的语句排序结果.定位错误按序号从小到大检查语句.Tarantula 式(1)中 failed(s),passed(s)表示语句 s 被失败和成功执行轨迹覆盖的次数,totalfailed 和 totalpassed 分别表示失败和成功执行轨迹总数.第 3 列为原有动态切片及 Tarantula 和 FLAR 在其上计算得到的语句排序结果,第 4 列为改进的动态切片及 Tarantula 和 FLAR 在其上计算得到的语句排序结果.倒数第 2 行"T/F"表示程序执行结果成功(T)或失败(F).最后一行"fault R."表示各方法定位到错误需检查的语句数,例如"4-14"表示定位到错误"最好"情况需检查 4 条语句,"最坏"情况需检查 14 条语句.

表 1 示例程序的覆盖信息、(原/改进)动态切片及排序结果

语句	覆盖信息										(原)动态切片										(改进)动态切片									
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	Tar.	FLAR	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	Tar.	FLAR	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	Tar.	FLAR
s_1	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_2	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_3	●	●	○	●	○	●	●	○	3	14	●	●	○	●	○	●	●	○	2	10	●	●	○	●	○	●	●	○	2	9
s_4	○	○	●	○	●	○	○	●	15	15	○	○	●	○	●	○	○	●	11	11	○	○	●	○	●	○	○	●	10	10
s_6	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_7	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	○	○	●	○	●	○	○	●	10	10
s_8	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_9	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
$s_{10}(f)$	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_{11}	●	●	●	●	●	●	●	●	4	2	○	○	●	○	●	●	○	●	12	12	○	○	●	○	●	●	○	●	12	12
s_{12}	●	●	●	●	●	●	●	●	4	2	○	○	○	○	○	○	○	○			○	○	○	○	○	○	○	○		
s_{13}	○	○	●	○	○	○	○	○	1	1	○	○	○	○	○	○	○	○			○	○	○	○	○	○	○	○		
s_{14}	●	●	●	●	●	●	●	●	4	2	●	●	●	●	●	●	●	●	3	1	●	●	●	●	●	●	●	●	3	1
s_{15}	●	●	○	●	○	○	●	○	2	13	●	●	○	●	○	○	●	○	1	9	●	●	○	●	○	○	●	○	1	8
s_{16}	○	○	●	○	●	●	○	●	16	16	○	○	●	○	●	●	○	●	12	12	○	○	●	○	●	●	○	●	12	12
s_{17}	●	●	●	●	●	●	●	●	4	2	○	○	○	○	○	○	○	○			○	○	○	○	○	○	○	○		
T/F	F	F	F	T	T	T	T	T			F	F	F	T	T	T	T	T			F	F	F	T	T	T	T	T		
fault R.									4-14	2-12									3-10	1-8									3-9	1-7

由表 1 第 2,3 列可知,原有动态切片比覆盖信息规模小,例如 t_1 执行时,动态切片去除了无关语句 s_{11},s_{12},s_{17}.Tarantula 方法在原有动态切片和覆盖信息上定位到错误需要检查的语句数分别为"3-10"和"4-14",可以看出动态切片提升了错误定位的精度.由第 3,4 列可知,改进的动态切片比原有动态切片规模小,例如 t_1 执行时,改进的动态切片去除了无关语句 s_7,其原因是考虑了定义变量的实际引用变量,去除了无关变量 y 的切片从而缩小了切片规模.Tarantula 方法在改进动态切片上定位到错误需要检查"3-9"语句,由此可见改进动态切片比原有动态切片进一步提高了错误定位精度.

在覆盖信息上应用本文提出的 FLAR 方法需检查"2-12"语句定位错误,优于 Tarantula 方法;在原有动态切片上需检查"1-8"条语句,在改进动态切片上需检查"1-7"条语句定位错误,同样优于 Tarantula 方法.由此可见,使用关联分析及排序策略给出的语句检查次序更有效,并且能提高错误定位精度.

3 本文方法

本文方法主要步骤如下:(1)计算程序执行的动态切片信息;(2)将每次测试用例执行的动态切片信息和相应的测试结果构造成混合谱矩阵;(3)在混合谱矩阵上进行关联分析,随后依据提出的排序策略对关联分析后的语句进行排序,生成错误定位报告.

3.1 动态切片

动态切片技术是一种有效的程序调试技术,已广泛应用于软件调试中的错误定位研究.当程序执行产生错误输出或异常时,调试人员通常希望快速确定错误语句可能存在的范围,动态切片技术可以通过计算错误输出语句的后向动态切片以缩小错误定位范围.动态切片计算方法根据计算过程不同分为后向计算方法[12-13]和前向计算方法[10-11].后向计算方法采用回溯程序执行时产生的动态依赖关系的方式来寻找程序中兴趣点的依赖节点,从而获得兴

趣点的动态切片，该方法需遍历程序一次执行中的全部动态依赖关系[12-13]. 前向计算方法通过程序中兴趣点的直接动态依赖关系从而获得程序中兴趣点的动态切片，该方法需要维护程序中每个执行实例的切片结果[10-11]. 相比后向计算方法，前向计算方法在错误发生后可以根据错误输出语句的直接动态依赖关系生成该语句的动态切片，不需要后向遍历动态依赖关系，因而能在程序出错时更加快速的计算出动态切片结果.

鉴于此，本文采用前向计算方法计算程序后向动态切片，并且对定义变量进行影响集分析以提高切片精度，将得到的动态切片结果用于错误定位.

首先介绍相关定义.

定义 1.　执行轨迹. 测试用例 t 执行源代码的执行轨迹 T 是程序执行过程中跟踪到的所有语句的序列，记为 $T=\langle s_1,s_2,\cdots,s_n\rangle$. 用时间戳 stamp 表示语句在执行轨迹 T 中执行的先后次序，标记为 s^{stamp}.

按程序执行成功与否，将执行轨迹分为成功执行轨迹与失败执行轨迹两类，分别表示为 $T_p=\{T_{p1},T_{p2},\cdots,T_{pm}\}$ 和 $T_f=\{T_{f1},T_{f2},\cdots,T_{fn}\}$.

定义 2.　动态切片. 对于给定的切片准则 $C=\langle I,s,V\rangle$，其中 I 为程序的输入，s 为程序中的兴趣点，V 为变量集. 基于切片准则 C 和输入 I，动态切片是由程序 P 中的所有影响兴趣点 s（一条语句或语句块）中变量集 V 的语句和谓语构成的集合[5].

定义 3.　影响集 $Influence[v]$. 对于给定语句 s，变量 $v\in Def[s]$，影响集 $Influence[v]=\{v'\mid v'$ 是 v 定义时实际引用的变量，$v'\in Use[s]\}$. 其中，定义集 $Def[s]$ 是语句 s 中被定义的变量集合，引用集 $Use[s]$ 是语句 s 中引用的变量集合.

本文对定义变量进行影响集分析，去除无关变量，以降低切片结果的冗余程度.

定义 4.　混合谱矩阵（Hybrid Spectrum Matrix，HSM）. 对于由 n 条语句 s 组成的程序 P 和给定的 m 个测试用例，测试用例每次执行得到一个动态切片 $slice$ 和相应执行结果. 混合谱矩阵是由多个动态切片和执行结果构成的二维矩阵 $\boldsymbol{HSM}=[M_{m\times n}N_{m\times 1}]$，$\forall a_{ij}\in M_{m\times n}$，$\forall b_{ij}\in N_{m\times 1}$，其中

$$a_{ij}=\begin{cases}1, & s\in slice\\ 0, & s\notin slice\end{cases},\ 1\le i\le m,\ 1\le j\le n \quad (2)$$

$$b_{ij}=\begin{cases}\text{T}, & \text{执行成功}\\ \text{F}, & \text{执行失败}\end{cases},\ 1\le i\le m,\ j=n+1 \quad (3)$$

本文中混合谱矩阵采用语句的击谱[3]形式，若某条语句被覆盖到为 1，否则为 0. 文中提到的某条语句被程序执行轨迹覆盖多次是指由不同的程序执行轨迹多次覆盖，而不是被一条程序执行轨迹多次覆盖.

算法 1 描述了改进的前向计算动态切片算法. 第 1～3 行计算语句 s 的引用集 $Use[s]$ 中变量 use 的动态切片 $slice[use]$. 根据当前执行实例 s^i 的动态数据依赖 $dynDD[s^i]$ 获得引用变量的动态切片 $slice[use]$. 第 4～10 行根据定义 3 计算定义变量 def 的影响集 $Influence[def]$. 第 11～18 行为计算定义集 $Def[s]$ 中变量 def 的动态切片 $slice[def]$. 第 11 行由 $dynCD[s^i]$ 得到控制依赖谓词 $pred$ 的切片 $slicePred$. 第 12～18 行计算定义变量的动态切片 $slice[def]$，其切片由当前执行语句 s、调用语句 $call$、控制依赖谓词切片 $slicePred$ 以及定义变量影响集 $Influence[def]$ 中变量 $influence$ 的切片 $slice[influence]$ 共 4 部分组成.

算法 1.　改进的前向计算动态切片算法.

输入：s^i//当前执行实例
　　$call$//当前执行方法调用语句
　　$dynCD[s^i]$//s^i 的动态控制依赖
　　$dynDD[s^i]$//s^i 的动态数据依赖
　　$slicePred$//谓词 $Pred$ 的切片
输出：$slice[use]$//语句 s 引用变量 use 的动态切片
　　$slice[def]$//语句 s 定义变量 def 的动态切片

```
begin
1.  for each use∈Use[s] do
2.      find slice[use] by dynDD[s^i];
3.  end for
4.  for each def∈Def[s] do
5.      Influence[def]={};
6.    for each use∈Use[s] do
7.      if use affect the value of def then
8.        add use to Influence[def];
9.    end for
10. end for
11. find slicePred by dynCD[s^i];
12. for each def∈Def[s] do
13.     slice_T={s,call}∪slicePred;
14.   for each influence∈Influence[def] do
15.       slice_T=slice_T∪slice[influence];
16.   end for
17.   slice[def]=slice_T;
18. end for
end
```

以图 1 程序为例，改进的前向计算动态切片算法计算过程如表 2 所示. 对于给定输入 I 为 $t_1(5,2,5)$ 和切片准则 $\langle I,s_{17},r\rangle$，则程序执行轨迹 $T_f=\{s_5^1,s_6^2,s_7^3,s_8^4,s_1^5,s_2^6,s_3^7,s_9^8,s_{10}^9,s_{11}^{10},s_{12}^{11},s_{14}^{12},s_{15}^{13},s_{17}^{14}\}$. 其中，函数 $read()$ 和 $sub()$ 的返回值抽象为变量 ret_read 和

ret_sub,*pred_2* 表示语句 s_2 是谓词.以表 2 第 9 行为例说明动态切片的计算过程,语句 s_9 引用集 $Use[s]$ 中变量 x,y,z 和 *ret_sub* 的切片结果通过动态数据依赖即可获得;定义集 $Def[s]$ 变量 m 受函数 *sub*() 返回值变量 *ret_sub* 和变量 z 的影响,与变量 x,y 无直接依赖关系,而在此次执行中 *ret_sub* 实际引用变量 x,与变量 y 无关,其动态切片 $slice[m]=\{s_9\}\cup slice[z]\cup slice[ret\_sub]$,不包含变量 y 的切片.现有前向计算动态切片方法[10,11]计算语句 s_9 中定义变量 m 的动态切片时,包含引用变量 y 的动态切片,结果存在冗余.而本文方法得到的动态切片更加精确,从而使错误定位更加准确.

表 2 改进的前向计算动态切片计算过程

s^{stamp}	$Def[s]$	$Use[s]$	$slice[use]$	$dynCD[s^i]$	*call*	*pred*	*def*	$Influence[def]$	*influence*	$dynDD[influence]$	$slice[def]$
s_5^1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
s_6^2	$\{x\}$	$\{ret\_read\}$	∅	∅	∅	∅	x	$\{ret\_read\}$	*ret_read*	∅	$\{s_6\}$
s_7^3	$\{y\}$	$\{ret\_read\}$	∅	∅	∅	∅	y	$\{ret\_read\}$	*ret_read*	∅	$\{s_7\}$
s_8^4	$\{z\}$	$\{ret\_read\}$	∅	∅	∅	∅	z	$\{ret\_read\}$	*ret_read*	∅	$\{s_8\}$
s_1^5	$\{i,j,k\}$	$\{x,$ $y,$ $z\}$	$\{s_6\}$ $\{s_7\}$ $\{s_8\}$	∅	s_9	∅	i j k	$\{x,$ $y,$ $z\}$	x y z	s_6^2 s_7^3 s_8^4	$\{s_1,s_6,s_9\}$ $\{s_1,s_7,s_9\}$ $\{s_1,s_8,s_9\}$
s_2^6	$\{pred\_2\}$	$\{k\}$	$\{s_1,s_8,s_9\}$	∅	s_9	∅	*pred_2*	$\{k\}$	k	s_1^5	$\{s_1,s_2,s_8,s_9\}$
s_3^7	$\{ret\_sub\}$	$\{i\}$	$\{s_1,s_6,s_9\}$	s_2^6	s_9	*pred_2*	*ret_sub*	$\{i\}$	i	s_1^5	$\{s_1,s_2,s_3,$ $s_6,s_8,s_9\}$
s_9^8	$\{m\}$	$\{ret\_sub,$ $x,$ $y,$ $z\}$	$\{s_1,s_2,s_3,$ $s_6,s_8,s_9\}$ $\{s_6\}$ $\{s_7\}$ $\{s_8\}$	∅	∅	∅	m	$\{ret\_sub,$ $z\}$	*ret_sub* z	s_3^7 s_8^4	$\{s_1,s_2,s_3,$ $s_6,s_8,s_9\}$
s_{10}^9	$\{x\}$	$\{x,$ $z\}$	$\{s_6\}$ $\{s_8\}$	∅	∅	∅	x	$\{x,$ $z\}$	x z	s_6^2 s_8^4	$\{s_6,s_8,s_{10}\}$
s_{11}^{10}	$\{y\}$	$\{x\}$	$\{s_6,s_8,s_{10}\}$	∅	∅	∅	y	$\{x\}$	x	s_{10}^9	$\{s_6,s_8,s_{10},s_{11}\}$
s_{12}^{11}	$\{pred\_12\}$	$\{x\}$	$\{s_6,s_8,s_{10}\}$	∅	∅	∅	*pred_12*	$\{x\}$	x	s_{10}^9	$\{s_6,s_8,s_{10},s_{12}\}$
s_{14}^{12}	$\{pred\_14\}$	$\{m\}$	$\{s_1,s_2,s_3,$ $s_6,s_8,s_9\}$	∅	∅	∅	*pred_14*	$\{m\}$	m	s_9^8	$\{s_1,s_2,s_3,s_6,$ $s_8,s_9,s_{14}\}$
s_{15}^{13}	r	$\{x,$ $z,$ $pred\_14\}$	$\{s_6,s_8,s_{10}\}$ $\{s_8\}$ $\{s_1,s_2,s_3,s_6,$ $s_8,s_9,s_{14}\}$	s_{14}^{12}	∅	*pred_14*	r	$\{x,$ $z,$ $pred\_14\}$	$x,$ $z,$ $pred\_14$	s_{10}^9 s_8^4 s_{14}^2	$\{s_1,s_2,s_3,s_6,$ $s_8,s_9,s_{10},$ $s_{14},s_{15}\}$
s_{17}^{14}	∅	$\{r\}$	$\{s_1,s_2,s_3,s_6,s_8,$ $s_9,s_{10},s_{14},s_{15}\}$	∅	∅	∅	∅	∅	∅	∅	∅

3.2 关联分析和排序策略

动态切片技术能在一定程度上缩小错误定位的范围.为了得到更加精确的错误定位结果,本文在动态切片信息及相应的执行结果上构建混合谱矩阵 ***HSM***,然后采用关联分析及排序策略生成语句检查的优先级次序,从而定位错误.

3.2.1 关联分析

定义 5. 布尔判别函数 $B(s)$.若某条语句 s 只出现在失败执行轨迹中,则 $B(s)$ 值为 1;若某条语句 s 既出现在失败执行轨迹中又出现在成功执行轨迹中,或者语句 s 只出现在成功执行轨迹中,则 $B(s)$ 值为 0,定义如下:

$$B(s)=\begin{cases}1, & s\in T_f \wedge s\notin T_p\\ 0, & \text{其他}\end{cases} \tag{4}$$

定义 6. 关联规则(association rules).为 $A\Rightarrow B$ 形式的逻辑蕴涵式,其中 $A\subset D$,$B\subset D$ 且 $A\cap B=\varnothing$,则 A,B 分别称为关联规则 $A\Rightarrow B$ 的前件和后件[14].

在错误定位中,程序的一条执行轨迹及相应执行结果作为一个事务,所有执行轨迹及相应执行结果为事务集 D;A 表示执行轨迹中语句,B 表示程序执行结果(成功 T 或失败 F).A 表示关联规则的前件,B 表示关联规则的后件[14].本文中的混合谱矩阵存储了软件错误定位中的所有事务.

定义 7. 支持度(support).规则 $A\Rightarrow B$ 的支持度是指 A 和 B 两者在事务集 D 中同时出现的概率,记为 $support(A\Rightarrow B)$,它是概率 $P(A\cup B)$[14],即

$$support(A\Rightarrow B)=support(A\cup B)=P(A\cup B) \tag{5}$$

在错误定位的上下文环境中,$support(A\Rightarrow B)$ 表示某条语句 A 出现在事务集 D 中且同时程序执行结果为 B 的概率.最小支持度阈值 $min\_support=\frac{1}{N}$,其中 $N=|T_p|+|T_f|$,因定位到错误至少需要一个

失败执行轨迹，所以最小支持度阈值为$\frac{1}{N}$.

定义 8.　置信度(confidence). 规则 $A\Rightarrow B$ 的置信度表示在出现 A 的事务集 D 中 B 也同时出现的条件概率[14]，即包含 A 和 B 的事务数与包含 A 的事务数的比值，记为

$$confidence(A\Rightarrow B)=P(B|A)=\frac{support(A\cup B)}{support(A)} \tag{6}$$

在错误定位的上下文环境中，$confidence(A\Rightarrow B)$ 表示语句 A 出现的事务集 D 中执行结果 B 也同时出现的条件概率，$support(A)$表示包含语句 A 的事务数与所有事务数的比率. 最小置信度阈值 $min\_confidence=\frac{1}{N}$，因定位到错误至少出现一个失败执行轨迹，且最极端情况下错误语句又出现在所有执行轨迹中，所以最小置信度阈值为$\frac{1}{N}$.

定义 9.　语句错误相关性. 对于执行轨迹中的每条语句 A 和程序执行结果 B(成功 T 或失败 F)，$A\Rightarrow B$ 有 3 种规则：$A\Rightarrow$F$(A\in T_f\wedge A\notin T_p)$，$A\Rightarrow$T$(A\in T_p\wedge A\notin T_f)$，$A\Rightarrow$F 或 T$(A\in T_p\wedge A\in T_f)$.

3.2.2　排序策略

在软件错误定位中，已有研究表明语句的怀疑度与它被失败执行轨迹覆盖的次数正相关，与它被失败执行轨迹没有覆盖的次数负相关，与它被成功执行轨迹覆盖次数负相关；并且一个程序至少被一个失败测试执行，且所有失败测试的贡献率大于所有成功测试的贡献率[6-7,15-16].

根据以上分析，本文设计如下 3 层排序策略：(1) $B(s)$值为 1 的语句优先被检查. 程序执行失败必定触发了错误语句，只出现在失败执行轨迹中的语句出错概率更高，应该优先被检查；(2) $support$ 值越大的语句越优先被检查. $support$ 值越大表明语句被失败执行轨迹覆盖的次数越多，则此语句应该优先被检查；(3) $confidence$ 值越大的语句越优先被检查. 在语句出现在失败执行轨迹中次数相同情况下(即 $support$ 值相同)，出现在成功执行轨迹中次数越少，则 $confidence$ 值越大，越优先被检查.

3.2.3　关联分析及排序策略算法

算法 2 描述关联分析及排序策略算法. 该算法将混合谱矩阵 ***HSM*** 作为输入，其中 ***HSM*** 存储语句击谱信息，即 1 表示语句被执行轨迹覆盖，0 表示语句未被执行轨迹覆盖；将排序后语句序列 $sList$ 作为输出. 第 1 行初始化链表 S'，用于存储语句 s_i 及 $B(s_i)$，$support(s_i\Rightarrow$F$)$，$confidence(s_i\Rightarrow$F$)$值，$i=1,2,\cdots,n$. 第 2～6 行在混合谱矩阵上计算语句 s 的 $B(s)$，$support(s\Rightarrow$F$)$和 $confidence(s\Rightarrow$F$)$值，其中 $B(s)$值按定义 5 计算，$support(s\Rightarrow$F$)$按定义 7 计算，即语句 s 被失败执行轨迹覆盖次数与所有执行轨迹数量的比值，$confidence(s\Rightarrow$F$)$按定义 8 计算，即语句 s 的 $support(s\Rightarrow$F$)$与 $support(s)$的比值. 第 7～23 行进行关联分析与排序，选取 $support(s_i\Rightarrow$F$)\geq min_support\left(\frac{1}{N}\right)$且 $confidence(s_i\Rightarrow$F$)\geq min_confidence\left(\frac{1}{N}\right)$的关联规则 $A\Rightarrow B$，其中 $A=s_i\in\{s_1,s_2,\cdots,s_n\}$，$B=$F(程序执行失败)，其中，第 7 行中 $S'.size$ 为链表中元素个数；第 9、10 行为取出$S'[j]$，$S'[j+1]$中存储的语句 s_j，s_{j+1}；第 13、16 和 19 行 $swap()$函数为两条语句的交换函数；然后按照本文设计的排序策略进行排序. 第 24～27 行取出排序后语句序列 $sList$ 中的语句；第 28 行返回语句序列 $sList$.

算法 2.　关联分析及排序策略算法.

```
输入：HSM                 //混合谱矩阵
      min_support         //最小支持度阈值
      min_confidence      //最小置信度阈值
输出：sList               //有优先级次序的语句序列
begin
1.   list S'=InitInput( );
2.   for each s∈HSM do
3.    compute B(s);
4.    support(s⇒F)=P(s∪F);
5.    confidence(s⇒F)=P(s∪F)/P(s);
6.   end for
7.   for i=0: S'.size do
8.    for j=0: (S'.size-i-1) do
9.     s_j=S'[j].getStatement();
10.    s_{j+1}=S'[j+1].getStatement();
11.    if(support(s_j⇒F)≥min_support&&
         confidence(s_j⇒F)≥min_confidence)
12.      {if B(s_j)<B(s_{j+1})
13.            swap(S'[j], S'[j+1]);
14.       else if B(s_j)==B(s_{j+1})
15.            if support(s_j⇒F)<support(s_{j+1}⇒F)
16.                 swap(S'[j],S'[j+1]);
17.            else if support(s_j⇒F)==
                       support(s_{j+1}⇒F)
18.                 if confidence(s_j⇒F)<
                       confidence(s_{j+1}⇒F)
19.                 swap(S'[j], S'[j+1]); }
20.    else
21.     delete s_i;
22.   end for
```

```
23. end for
24. for each s′∈S′ do
25.   if s′.getStatement()!=null then
26.         sList.add(s′.getStatement());
27. end for
28. return sList;
end
```

3.3 错误定位实现

本文方法实现过程:首先,基于改进的动态切片算法,计算得到一个更加精确的动态切片;其次,将动态切片及相应执行结果构建混合谱矩阵;最后,进行关联分析,并根据排序策略给出语句检查优先级次序,生成错误定位报告.

下面以图 1 程序为例来分析如何定位错误.我们以〈I,s_{17},r〉为切片准则,以 8 个测试用例为输入运行程序,根据算法 1 计算动态切片如表 1 第 4 列所示.将动态切片和相应执行结果构建混合谱矩阵 ***HSM*** 如图 2 所示.根据算法 2 对混合谱矩阵中数据进行关联分析及排序(FLAR),其中按本文设计排序策略对语句排序如表 3 所示.以语句 s_1 为例说明算法 2 计算过程.首先,计算语句 s_1 的布尔判别函数 $B(s_1)$、支持度 $support(s_1\Rightarrow F)$ 和置信度 $confidence(s_1\Rightarrow F)$ 值:对于 $B(s_1)$ 值,因语句 s_1 既出现在失败执行轨迹也出现在成功执行轨迹中,根据定义 5,$B(s_1)=0$;对于 $support(s_1\Rightarrow F)$,因语句 s_1 出现在失败轨迹中 3 次,出现在成功轨迹中 5 次,根据定义 7,$support(s_1\Rightarrow F)=3/(3+5)=3/8$;对于 $confidence(s_1\Rightarrow F)$ 值,因语句 s_1 出现在失败执行轨迹中概率为 3/8,$support(s_1)=1$,根据定义 8,$confidence(s_1\Rightarrow F)=(3/8)/1=3/8$.其他语句指标计算同 s_1.其次,选取 $support(s_i\Rightarrow F)\geq 1/8$ 和 $confidence(s_i\Rightarrow F)\geq 1/8$ 进行关联分析,并对关联分析后的语句进行排序.

表 3 展示了本文设计的 3 层排序策略排序过程:(1)因为没有只出现在失败执行轨迹中的语句,所以 $B(s)$ 值都为 0,见第 2 行;(2)在语句 $B(s)$ 值相同的情况下,*support* 值越大的语句越优先被检查,因此语句 s_1 比语句 s_3 优先被检查,见第 3 行;(3)在语句 *support* 值相同情况下,*confidence* 值越大的语句越优先被检查.对于有相同 *support* 值的语句 s_3 和 s_{15},根据 *confidence* 值大小则先检查 s_{15} 再检查 s_3,见第 4 行.对于有相同 *support*、*confidence* 值的语句 s_4 和 s_7,则具有相同检查等级;最后生成语句 *Rank* 如表 3 最后一行所示.按此序列定位错误,"最好"情况需要检查 1 条语句,"最坏"情况需要检查 7 条语句即可定位到错误.在覆盖信息和原有动态切片上进行关联分析和排序(FLAR)过程同上.

$$
\boldsymbol{HSM}=\begin{array}{c}
\begin{matrix} s_1 & s_2 & s_3 & s_4 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{14} & s_{15} & s_{16} & \mathrm{R} \end{matrix} \\
\left[\begin{array}{cccccccccccc:c}
1&1&1&0&1&0&1&1&1&0&1&1&0&\mathrm{F}\\
1&1&1&0&1&0&1&1&1&0&1&1&0&\mathrm{F}\\
1&1&0&1&1&1&1&1&1&1&1&0&1&\mathrm{F}\\
1&1&1&0&1&0&1&1&1&0&1&1&0&\mathrm{T}\\
1&1&0&1&1&1&1&1&1&1&1&0&1&\mathrm{T}\\
1&1&1&0&1&0&1&1&1&1&1&0&1&\mathrm{T}\\
1&1&1&0&1&0&1&1&1&0&1&1&0&\mathrm{T}\\
1&1&0&1&1&1&1&1&1&1&1&0&1&\mathrm{T}
\end{array}\right]
\end{array}
\begin{matrix} t_1\\ t_2\\ t_3\\ t_4\\ t_5\\ t_6\\ t_7\\ t_8 \end{matrix}
$$

图 2 混合谱矩阵

表 3 错误定位结果

指标	s_1	s_2	s_3	s_4	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{14}	s_{15}	s_{16}
$B(s_i)$	0	0	0	0	0	0	0	0	0	0	0	0	0
$support(s_i\Rightarrow F)$	3/8	3/8	2/8	1/8	3/8	1/8	3/8	3/8	3/8	1/8	3/8	2/8	1/8
$confidence(s_i\Rightarrow F)$	3/8	3/8	2/5	1/3	3/8	1/3	3/8	3/8	3/8	1/4	3/8	1/2	1/4
Rank	1	1	9	10	1	10	1	1	1	12	1	8	12

4 实验结果及分析

为了评估本文方法的有效性,我们进行了对比实验.实验运行环境为 Intel(R) core(TM) 3.10 GHz 处理器和 4 GB 内存的计算机,Eclipse 开发平台.

4.1 实验对象

本文选取 13 个 Java 程序作为测试基准程序,如表 4 所示,其中第 1~3 列为程序名称、描述及代码行数(不含空白行和注释行),第 4,5 列为实验使用的错误版本数和测试用例数.每个程序累积代码行数为所有错误版本代码行数的累加.其中 Siemens Suite 中后 5 个程序是由 Santelices 等人[17]把 C 版本程序转换为 Java 版本程序①,jtcas,Nanoxml 和 XML-security 来源于 SIR(Subject Infrastructure Repository)②.本文方法适用于单错误版本程序的错误定位.我们从 jtcas 的 41 个错误版本中去除无测试用例执行失败的 2 个版本和多错误的 8 个版本,选取其余 31 个版本进行测试;随机选取 tot_info 程序 23 个错误版本中的 10 个版本进行测试;选取 schedule 的 9 个错误版本;选取 schedule2 的 10 个错误版本中 8 个版本进行测试,去除了 v4 版本(错误为去除 350 行处注释语句)和多错误的 v6 版本;选取 print_tokens 的 7 个错误版本和 print_tokens2 的 5 个错误版本进行测试;另外选取 NanoXML 共 28 个错误版本和 XML-security 共 18 个错误版本进行测试.因此,我们共测试 116 个单错误版本程序.

① http://www3.nd.edu/~rsanteli/subjects/
② http://sir.unl.edu/portal/index.html

表 4　实验对象描述

程序		描述	代码行数	错误版本数	测试用例数
Siemens Suite	jtcas	避免相撞程序	165	31	1608
	tot_info	数据统计程序	283	10	1052
	schedule	优先级调度器	290	9	2650
	schedule2	优先级调度器	317	8	2710
	print_tokens	词法分析器	478	7	4130
	print_tokens2	词法分析器	410	5	4115
NanoXML v1		XML 解析器	3497	7	214
NanoXML v2			4009	6	214
NanoXML v3			4608	8	216
NanoXML v5			4782	7	216
XML-security v1		XML 加密器	21613	7	92
XML-security v2			22318	5	94
XML-security v3			19895	6	84

4.2　实验工具

本文在 Eclipse 实验平台上使用 Java 语言开发了基于动态切片、关联分析及排序策略的错误定位原型工具 DSFL(Dynamic Slicing Fault Locator). DSFL 使用 Java 优化框架 Soot 分析计算目标程序的数据流和控制流信息，使用 Java 调试接口(Java Debugger Interface，JDI)获取目标程序执行时的轨迹信息和覆盖信息，结合上述信息利用本文提出的切片算法进行动态程序切片计算，并进行错误定位.

4.3　实验设计

本文方法主要从动态切片缩小错误定位范围和关联分析及排序策略相结合提高错误定位精度两个方面来提高错误定位效率. 本文设计了 3 个层次对比实验：(1)验证关联分析及排序策略相结合 FLAR 方法(假设不使用动态切片技术)的有效性，见 4.4.1 节；(2)验证提出的动态切片技术提升错误定位精度的程度，见 4.4.2 节；(3)验证本文 DS-FLAR 方法有效性，见 4.4.3 节. 并选取 4 组方法进行错误定位对比实验. 第 1 组为基于覆盖信息的经典错误定位方法 Tarantula[6-7]和 Ochiai[8]；第 2 组为 Xie 等人[18]理论证明的两组最优怀疑度计算公式：其一为 Naish1[9]和 Naish2[9]，其二为 Wong1[19]、Russel&Rao[9]和 Binary[9]，这两组公式理论上无法证明哪一组更优；第 3 组为 Xie 等人[20]理论证明的基于遗传算法机器推导的 3 个最优怀疑度计算公式 GP02、GP03 和 GP19；第 4 组为基于动态切片的错误定位方法 FS[21]和 PSS-SFL[22].

4.4　实验结果分析

本文从累积检查语句数[15]、错误定位代价[23]和 *EXAM* 指标[15,32] 3 个指标对错误定位效率进行评估. 检查结果中可能有多条语句检查等级相同，“最好”情况是检查第 1 条语句就定位到错误，“最坏”情况为直到检查到最后 1 条语句才定位到错误，本文选取“最好”和“最坏”情况平均值进行评估.

4.4.1　基于覆盖信息对比实验

为验证提出的关联分析及排序策略相结合的 FLAR 方法有效性，在覆盖信息上进行对比实验.

对每个基准程序统计累积检查语句数——定位出多个错误版本中所有错误需检查的总语句数[15]. 实验结果可能受不同程序结构影响，因此，我们测试了不同程序，其结果如表 5 所示，第 1 列为基准程序，其余各列为不同方法定位错误需检查总语句数. 对于每个程序，FLAR 检查语句数少于对比方法(Ochiai 除外)；FLAR 检查语句数在 tot_info 程序上与 Ochiai 相当，在 print_tokens 程序上定位效果稍差于 Ochiai. 对于所有程序(由表 5 最后一行可知)，FLAR 累积检查语句数是 Tarantula、Ochiai、Naish1、Naish2、Wong1、Russel&Rao、Binary、GP02、GP03 和 GP19 累积检查语句数的 69.41%，73.92%，80.97%，70.49%，70.62%，70.49%，70.45%，71.27%，67.59%和 67.16%. 由此可知，FLAR 方法累积检查语句数少于对比方法.

表 5　FLAR 与基于覆盖信息的各种方法累积检查语句数对比

程序	FLAR	Tarantula	Ochiai	Naish1	Naish2	Wong1	Russel&Rao	Binary	GP02	GP03	GP19
jtcas	674	955	849	933	1248	933	933	932	921	1081	1173
tot_info	501	550	504	530	701	674	674	674	580	690	659
schedule	130	175	187	150	160	252	251	250	240	230	175
schedule2	650	797	802	794	802	938	938	938	722	876	1090
print_tokens	401	604	292	359	475	501	445	486	407	686	619
print_tokens2	192	500	515	503	930	516	516	517	874	529	580
NanoXMLv1	1250	1760	1650	1400	1750	1950	1955	1945	1920	2100	2130
NanoXML v2	840	1100	950	880	950	1120	1121	1121	900	1190	1100
NanoXML v3	1300	2520	2444	1661	1870	1615	1614	1614	2498	1809	1880
NanoXML v5	2287	2832	2516	2300	2619	2584	2579	2580	2302	2598	2650
XML-security v1	2300	3601	3401	2702	3213	3498	3498	3499	3400	3592	3404
XML-security v2	1650	2450	2400	2450	2686	2500	2548	2546	2350	2608	2720
XML-security v3	3300	4450	4426	4450	4550	4833	4883	4863	4600	4905	4863
累积检查总语句数	15475	22294	20936	19112	21954	21914	21955	21965	21714	22894	23043

注：数字表示定位到错误的累积检查语句数，越小越好.

同时,使用定位错误代价从相对指标的角度对所有方法进行评估.错误定位代价是找到一个程序所有测试版本中错误的累积检查语句数与所有版本可执行代码总行数的比值[23].某方法的错误定位代价越小,则该方法定位效果越好.由表 6 最后一行可知,FLAR 比 Tarantula、Ochiai、Naish1、Naish2、Wong1、Russel&Rao、Binary、GP02、GP03 和 GP19 错误定位代价分别降低了 3.58%,2.50%,2.33%,5.49%,4.21%,4.08%,4.17%,4.38%,4.82%和 5.40%.由此可知,FLAR 的错误定位平均代价较小.FLAR 定位效果优于对比方法是因为本文方法考虑了程序执行轨迹中语句与执行结果的关联关系,并结合排序策略,使得生成的语句检查序列更有效.

表 6 FLAR 与基于覆盖信息的各种方法错误定位代价对比 (单位:%)

程序	FLAR	Tarantula	Ochiai	Naish1	Naish2	Wong1	Russel&Rao	Binary	GP02	GP03	GP19
jtcas	13.18	18.67	16.60	18.24	24.40	18.24	18.24	18.22	18.01	21.13	22.93
tot_info	17.70	19.43	17.81	18.73	24.77	23.82	23.82	23.82	20.49	24.38	23.29
schedule	4.98	6.70	7.16	5.75	6.13	9.66	9.62	9.58	9.20	8.81	6.70
schedule2	25.63	31.43	31.62	31.31	31.62	36.99	36.99	36.99	28.47	34.54	42.98
print_tokens	11.98	18.05	8.73	10.73	14.20	14.97	13.30	14.52	12.16	20.50	18.50
print_tokens2	9.37	24.39	25.12	24.54	45.37	25.17	25.17	25.22	42.63	25.80	28.29
NanoXML v1	5.11	7.19	6.74	5.72	7.15	7.97	7.99	7.95	7.84	8.58	8.70
NanoXML v2	3.49	4.57	3.95	3.66	3.95	4.66	4.66	4.66	3.74	4.95	4.57
NanoXML v3	3.53	6.84	6.63	4.51	5.07	4.38	4.38	4.38	6.78	4.91	5.10
NanoXML v5	6.83	8.46	7.52	6.87	7.82	7.72	7.70	7.71	6.88	7.76	7.92
XML-security v1	1.52	2.38	2.25	1.79	2.12	2.31	2.31	2.31	2.25	2.37	2.25
XML-security v2	1.48	2.20	2.15	2.20	2.41	2.24	2.28	2.28	2.11	2.34	2.44
XML-security v3	2.76	3.73	3.71	3.73	3.81	4.05	4.09	4.07	3.85	4.11	4.07
平均定位代价	8.27	11.85	10.77	10.60	13.76	12.48	12.35	12.44	12.65	13.09	13.67

注:数字表示错误定位代价,越小定位效果越好.

为更直观比较不同方法优劣,本文还使用 *EXAM* 指标进行评估.*EXAM* 指标为错误检出率与代码检查率的比值[15,32].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[32].从图 3(a)、(b)和(d)可知,FLAR 定位效果明显优于 Tarantula、Ochiai、Naish1、Naish2、GP02、GP03 和 GP19.由图 3(c)可知,在代码检查率

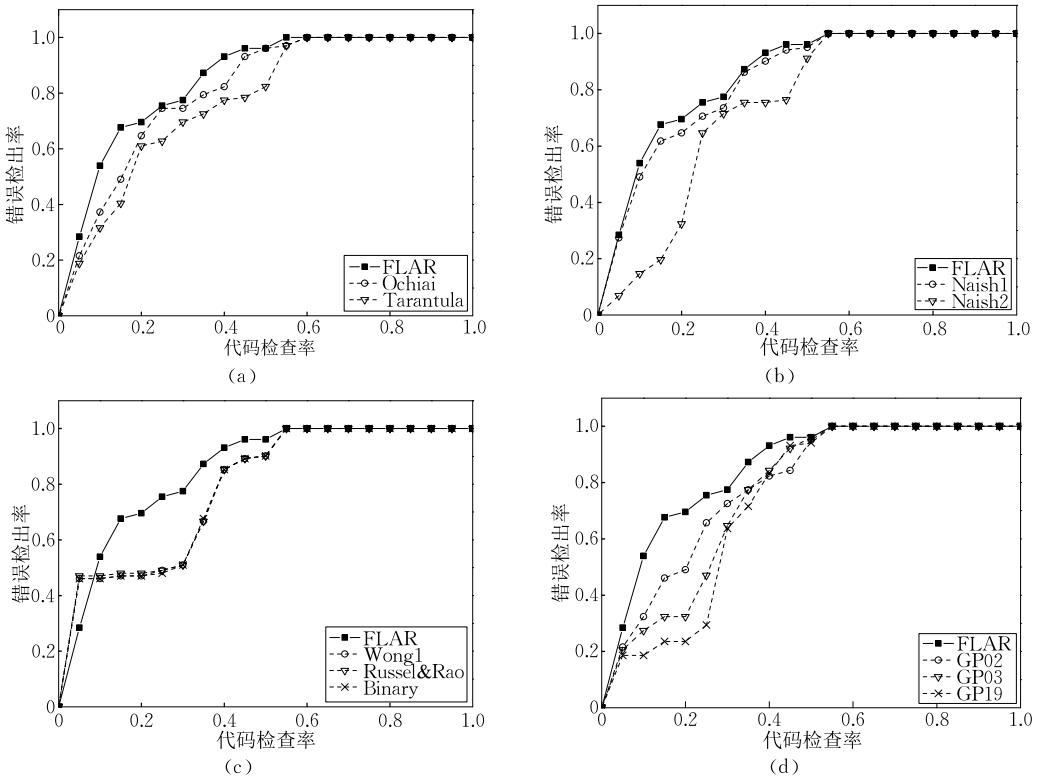

图 3 FLAR 与基于覆盖信息的各种方法效率对比

小于 5%时，Wong1、Russel&Rao 和 Binary 定位效果优于 FLAR，这是因为这 3 种方法检查出多条有相同最大怀疑度值的语句(错误语句包含其中)，若能接近最好情况定位到错误，则使得在较低代码检查率时就能定位到错误.但在代码检查率大于 5%时，FLAR 定位效果优于对比方法；由表 5 和表 6 可知，总体上 FLAR 定位效果优于 Wong1、Russel&Rao 和 Binary.

4.4.2　基于动态切片信息的对比实验

为了评估本文提出的动态切片技术相对于覆盖信息提升错误定位精度程度进行了如下实验：基于覆盖信息应用 FLAR、Tarantula、Naish1 和 Wong1 生成语句检查序列，同样在动态切片信息上应用上述方法，并分别命名为 DS-FLAR、DS-Tarantula、DS-Naish1 和 DS-GP02，比较在这两种情况下的错误定位精度差异.

通过比较表 5 和表 7 可知，累积检查总语句数 DS-FLAR 比 FLAR 减少了 34.71%，DS-Tarantula 比 Tarantula 减少了 34.32%，DS-Naish1 比 Naish1 减少了 27.72%，DS-GP02 比 GP02 减少了 35.51%.由此可见，动态切片技术的应用使得错误定位方法的累积检查总语句数有所减少.

比较表 6 和表 8 最后一行数据计算可知，动态切片技术在一定程度上可以减少错误定位代价，例如，FLAR 定位错误总体平均代价降低了 4.09%(从 8.27%下降为 4.18%)，Tarantula 降低了 6.28%(从 11.85%下降为 5.57%)，Naish1 降低了 5.33%(从 10.60%下降为 5.27%)，GP02 降低了 7.30%(从 12.65%下降为 5.35%).动态切片技术对不同方法定位代价影响不同，如在 jtcas 程序上，FLAR 定位错误代价降低了 7.12%，Tarantula 降低了 9.09%，Naish1 降低了 10.77%，GP02 降低了 9.70%，这是受方法不同程序的影响.

图 4 虚线表示基于覆盖信息上述 4 种方法的错误检出率随代码检查率的变化趋势，实线表示动

表 7　动态切片使用后各方法的累积检查语句数对比

程序	DS-FLAR	DS-Tarantula	DS-Naish1	DS-GP02
jtcas	310	490	382	425
tot_info	220	260	276	280
schedule	109	150	140	162
schedule2	176	159	210	216
print_tokens	149	182	159	129
print_tokens2	158	211	216	184
NanoXML v1	902	1350	1101	1208
NanoXML v2	750	820	762	792
NanoXML v3	528	1906	1401	1445
NanoXML v5	1650	2100	1629	1922
XML-security v1	1603	2200	2400	1901
XML-security v2	1240	1702	1938	2015
XML-security v3	2309	3112	3200	3325
累积检查总语句数	10104	14642	13814	14004

注：数字表示定位到错误的累积检查语句数，越小越好.

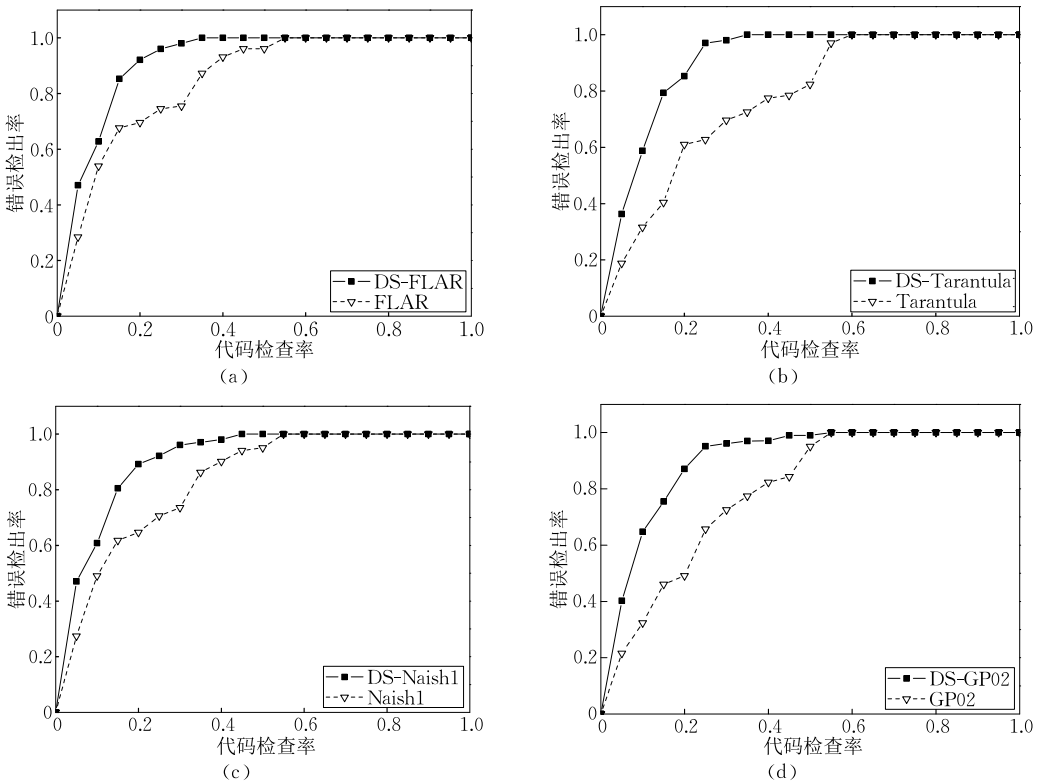

图 4　动态切片相对覆盖信息提升错误定位精度对比

表 8 动态切片使用后各方法的错误定位代价对比

（单位：%）

程序	DS-FLAR	DS-Tarantula	DS-Naish1	DS-GP02
jtcas	6.06	9.58	7.47	8.31
tot_info	7.77	9.19	9.75	9.89
schedule	4.18	5.75	5.36	6.21
schedule2	6.94	6.27	8.28	8.52
print_tokens	4.45	5.44	4.75	3.86
print_tokens2	7.71	10.29	10.54	8.98
NanoXML v1	3.68	5.51	4.50	4.93
NanoXML v2	3.12	3.41	3.17	3.29
NanoXML v3	1.43	5.17	3.80	3.92
NanoXML v5	4.93	6.27	4.87	5.74
XML-security v1	1.06	1.45	1.59	1.26
XML-security v2	1.11	1.53	1.74	1.81
XML-security v3	1.93	2.61	2.68	2.79
平均定位代价	4.18	5.57	5.27	5.35

注：数字表示错误定位代价，越小定位效果越好.

态切片技术使用后上述 4 种方法的错误检出率随代码检查率的变化趋势. 由图 4 可知，DS-FLAR、DS-Tarantula、DS-Naish1 和 DS-GP02 在定位效果上优于原始方法. 文献[24,25]认为在较低代码检查率（如 10%[25]或 20%[24]）时定位出的错误数越多，则该方法越有效. 本文比较不同方法在 20%代码检查率时定位出的错误数百分比. DS-FLAR 定位出总错误数的 92.08%，而 FLAR 定位出 69.61%；DS-Tarantula 定位出 85.26%，而 Tarantula 为 60.99%；DS-Naish1 定位出 89.24%，而 Naish1 为 64.71%；DS-GP02 定位出 87.10%，而 GP02 为 49.12%. 这种错误定位准确度提高主要是由于动态切片技术的使用缩小了错误定位范围. 从总体上可以看出，本文提出的动态切片技术从不同程度上提升了错误定位精度.

4.4.3 本文 DS-FLAR 方法与各种方法对比

为了评估本文提出的 DS-FLAR 方法的错误定位效果，将其与基于动态切片错误定位方法 FS[21]和 PSS-SFL[22]对比，并与基于覆盖信息的 10 种怀疑度计算错误定位方法对比.

首先，评估本文改进动态切片对整体定位精度改善的贡献大小. 在原动态切片上应用 FLAR（记为 D-FLAR）与 DS-FLAR 相比较，两者差异即为改进动态切片对整体错误定位精度的贡献. 由表 9 第 2，3 列比较可知，对于每个程序，改进动态切片减少了检查语句数，如对 jtcas 程序定位错误检查语句数减少了 78，对 tot_info 程序减少了 92，总体上累积检查语句数减少了 1732. 由表 10 第 2，3 列比较可知，对于每个程序，改进动态切片比原动态切片减少了平均定位代价，如对 jtcas 程序减少 1.53%，对 tot_info 程序减少 3.25%，总体上平均定位代价减少 1.01%. 由此可见，改进动态切片对整体定位精度有一定程度的提升.

其次，DS-FLAR 与 FS 和 PSS-SFL 进行对比分析. 由表 9 可知，对于每个程序，DS-FLAR 检查语句数比 FS 少，也比 PSS-SFL 少（NanoXML v2，XML-security v2 除外）. 对于所有程序（由表 9 最后一行可知），定位到错误 DS-FLAR 累积检查语句数比 FS、PSS-SFL 少，且是 FS、PSS-SFL 累积检查语句数的 54.41%（10104/18570）和 77.01%（10104/13120）. 由表 10 最后一行可知，DS-FLAR 平均定位代价比 FS、PSS-SFL 分别降低了 4.79%、2.48%. 直观上，由图 5(d)可知，DS-FLAR 定位效果优于 FS、PSS-SFL. 总体上，DS-FLAR 定位效果优于基于动态切片的错误定位方法 FS 和 PSS-SFL. 这是因为本文动态切片对定义变量进行影响集分析提高了切片的精度，并且关联分析及排序策略生成的语句检查序列更有效.

表 9 基于动态切片的各种方法累积检查语句数对比

程序	DS-FLAR	D-FLAR	FS	PSS-SFL
jtcas	310	388	850	550
tot_info	220	312	560	398
schedule	109	158	380	214
schedule2	176	202	340	243
print_tokens	149	198	277	280
print_tokens2	158	191	266	294
NanoXML v1	902	984	1400	1150
NanoXML v2	750	823	1201	730
NanoXML v3	528	700	1195	750
NanoXML v5	1650	1790	3400	2160
XML-security v1	1603	1860	3200	2109
XML-security v2	1240	1461	1601	1140
XML-security v3	2309	2769	3900	3102
累积检查总语句数	10104	11836	18570	13120

注：数字表示定位到错误的累积检查语句数，越小越好.

表 10 基于动态切片的各种方法错误定位代价对比

（单位：%）

程序	DS-FLAR	D-FLAR	FS	PSS-SFL
jtcas	6.06	7.59	16.62	10.75
tot_info	7.77	11.02	19.79	14.06
schedule	4.18	6.05	14.56	8.20
schedule2	6.94	7.97	13.41	9.58
print_tokens	4.45	5.92	8.28	8.37
print_tokens2	7.71	9.32	12.98	14.34
NanoXML v1	3.68	4.02	5.72	4.70
NanoXML v2	3.12	3.42	4.99	3.03
NanoXML v3	1.43	1.90	3.24	2.03
NanoXML v5	4.93	5.35	10.16	6.45
XML-security v1	1.06	1.23	2.12	1.39
XML-security v2	1.11	1.31	1.43	1.02
XML-security v3	1.93	2.32	3.27	2.60
平均定位代价	4.18	5.19	8.97	6.66

注：数字表示错误定位代价，越小定位效果越好.

最后，DS-FLAR 与基于覆盖信息的错误定位方法进行对比分析. 由表 5 和表 9 可知，对于每个程序，DS-FLAR 检查语句数均少于相比较的基于覆盖

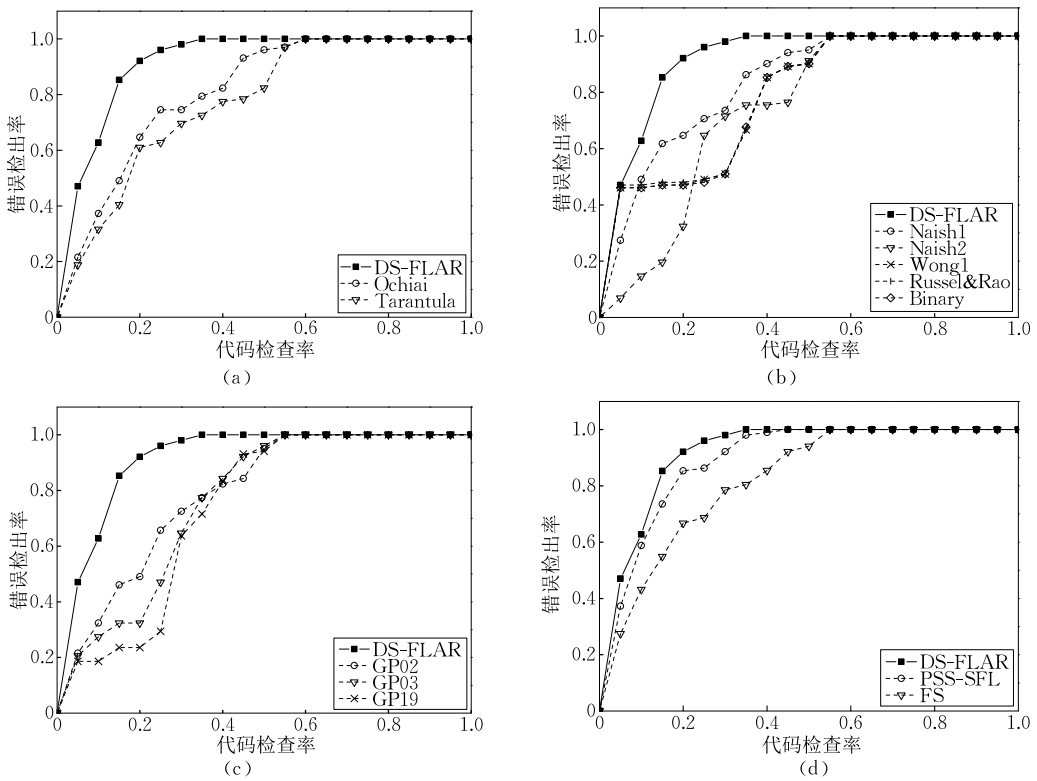

图 5　DS-FLAR 与各种方法效率对比

信息的 10 种错误定位方法. 对于所有程序(由表 5 和表 9 最后一行可知),DS-FLAR 能够减少错误定位时需要检查的总语句数,且 DS-FLAR 累积检查总语句数是 Tarantula、Ochiai、Naish1、Naish2、Wong1、Russel&Rao、Binary、GP02、GP03 和 GP19 累积检查总语句数的 45.32%,48.26%,52.87%,46.02%, 46.11%, 46.02%, 46.00%, 46.53%,44.13%和 43.85%. 由此可知,本文方法错误定位准确度有一定程度的提升. 由表 6、表 10 可知,DS-FLAR 相比 Tarantula、Ochiai、Naish1、Naish2、Wong1、Russel&Rao、Binary、GP02、GP03 和 GP19 平均错误定位代价分别降低了 7.67%,6.59%,6.42%,9.58%,8.30%,8.17%,8.26%,8.47%,8.91%和 9.49%. 由图 5(a)、图 5(c)可知,DS-FLAR 错误定位效果明显优于 Tarantula、Ochiai、GP02、GP03 和 GP19. 由图 5(b)可知,DS-FLAR 定位效果明显优于 Naish1 和 Naish2;当代码检查代码率小于 5%时,DS-FLAR 与 Wong1、Russel& Rao、Binary 错误定位效果相当,当代码检查代码率大于 5%时,DS-FLAR 定位效果明显优于 Wong1、Russel&Rao 和 Binary. 由表 6、表 10 可知,总体上 DS-FLAR 定位效果优于 Wong1、Russel&Rao 和 Binary.

通过前面 3 个层次的对比实验,可得出以下结论:

(1)本文提出的关联分析及排序策略相结合的 FLAR 方法(假设不使用动态切片技术),定位精度高于 Tarantula、Ochiai、Naish1、Naish2、Wong1、Russel&Rao、Binary、GP02、GP03 和 GP19.

(2)本文提出的改进动态切片应用于 FLAR、Tarantula、Naish1 和 GP02,其错误定位精度都有明显提升,说明本文提出的动态切片技术缩小错误定位范围效果较好.

(3)DS-FLAR 定位效果优于相比较的基于动态切片的错误定位方法和基于覆盖信息的 10 种怀疑度计算错误定位方法.

5　相关工作

目前,主要有以下两类基于动态测试的错误定位方法:(1)基于程序切片的错误定位方法;(2)基于覆盖信息的错误定位方法.

研究人员提出了多种基于程序切片的错误定位方法. 第 1 类方法是程序切片结果直接用于错误定位. 如 Zhang 等人[21]提出了一种基于动态切片的错误定位方法,该方法采用前向计算方法分别计算和比较了数据切片、全切片和相关切片的错误定位效果,并实验证明了全切片和相关切片错误定位效果较好. 我们之前的研究[26]是针对空指针异常问题,

应用程序切片技术,结合实时堆栈信息,在切片后程序上进行空指针与别名分析,从而定位空指针异常.第2类方法是程序切片和程序谱方法相结合的错误定位.例如,Wen 等人[22]使用动态切片提取程序元素间依赖关系,缩减程序执行轨迹,然后在语句频谱信息上构造了类似于 Tarantula 的公式统计语句怀疑度.Alves 等人[27]通过动态切片和修改影响分析来提高Tarantula 的错误定位效率.与文献[22,27]不同,本文采用改进的动态切片技术来提高动态切片精度;然后采用关联分析及排序策略生成可疑语句的优先级次序,而他们采用 Tarantula 公式生成可疑语句检查次序.Yu 等人[28]基于覆盖信息和静态控制流图构建执行流图,从而得到半动态切片(semi-dynamic slicing),然后使用 Tarantula 公式计算怀疑度.Lei 等人[29]使用静态后向切片和执行切片的交集构成近似动态切片(approximate dynamic slice),然后统计语句的怀疑度.与文献[28-29]构造半动态切片和近似动态切片的方法不同,本文方法得到的是动态切片,切片结果更加准确,且在动态切片信息上进行关联分析及排序生成语句检查序列.

基于覆盖信息的错误定位方法可分为两类.第1类是比较覆盖信息的差异.如 Renieris 等人[30]提出了最近邻查询错误定位方法(Nearest Neighbor Queries,NNQ).该方法假定存在一个失败运行与多个成功运行,使用距离度量方法从成功运行中统计出与失败运行最相似的程序谱,比较其差异,去除被失败和成功运行都执行的语句,生成错误定位报告.第2类是统计覆盖信息的特征.如 Jones 等人[6-7]提出了 Tarantula 怀疑度计算错误定位方法,计算出每条语句怀疑度,然后按照怀疑度大小检查可疑语句.Abreu 等人[8]提出了 Ochiai 错误定位方法,引入分子生物学领域相似系数 Ochiai 来计算语句怀疑度,实验表明 Ochiai 定位效果优于 Tarantula.Xie 等人[18]理论证明了 Naish1 和 Naish2,Wong1、Russel&Rao 和 Binary 为两组最优的怀疑度计算方法,Xie 等人[20]还理论证明了基于遗传算法通过机器推导出来的3个最优公式 GP02,GP03 和 GP19.上述方法实质上是按照语句怀疑度大小检查可疑语句,从而定位错误.Nessa 等人[31]通过构造 N 长度的执行序列,利用统计信息关联挖掘 N 长度执行子序列与执行结果的关联,进而提出了 N 长度序列和关联规则相结合的错误定位方法.赵磊等人[32]提出了在覆盖向量(每次执行的代码基本块覆盖信息)中,求解与某个基本块 b_0 一致的分量所对应的基本块,这些分量即为基本块 b_0 的频繁集;求出频繁集后,语句的怀疑度按 Tarantula,Ochiai 等公式进行计算.与文献[31]相比,本文方法不但使用支持度和置信度指标进行关联分析,而且使用布尔判别函数和排序策略加以筛选排序.与文献[32]相比,本文采用关联分析与排序策略直接筛选排序可疑语句,从而定位错误,无需进一步计算语句怀疑度,从而减少时间代价.实验结果表明,本文方法定位错误的准确度更高.

6 结束语

本文提出了一种基于动态切片、关联分析及排序策略相结合的错误定位方法.该方法首先采集程序的动态切片信息和相应执行结果,构建混合谱矩阵;然后在此基础上进行关联分析,随后依据本文设计的排序策略对语句进行排序,进而生成语句检查的优先级次序,开发人员利用此序列定位错误.本文从3个层次在13个 Java 基准程序上进行了对比实验,与基于动态切片的 FS、PSS-SFL 以及基于覆盖信息的10种怀疑度计算错误定位方法进行对比.实验结果表明,本文提出的改进动态切片方法能有效的缩小错误定位范围,提出的关联分析与排序策略相结合生成的语句优先级次序定位错误效果很好.因此,本文方法定位错误的效率较高.

虽然我们在不同规模基准程序上验证了本文方法的有效性,但本文方法同其他错误定位方法一样,处理多错误的故障定位时,效率会有所降低.下一步的工作中,我们将重点研究本文方法如何提高多错误程序中定位的效率.

致 谢 在此,我们对审稿人和编辑表示感谢,对本文提出建议的同行表示感谢!

参 考 文 献

[1] Collofello J S, Woodfield S N. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software, 1989, 9(3): 191-195

[2] Agrawal H, Horgan J R, London S, Wong W E. Fault localization using execution slices and dataflow tests// Proceedings of the 6th International Symposium on Software Reliability Engineering. Toulouse, France, 1995: 143-151

[3] Wen Wan-Zhi, Li Bi-Xin, Sun Xiao-Bing, Liu Cui-Cui. Technique of software fault localization based on hierarchical slicing spectrum. Journal of Software, 2013, 24(5): 977-992 (in Chinese)

(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)

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

[5] Korel B，Laski J. Dynamic program slicing. Information Processing Letters，1988，29(3)：155-163

[6] 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，USA，2005：273-282

[7] 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. New York，USA，2002：467-477

[8] Abreu R，Zoeteweij P，Van Gemund A J C. An evaluation of similarity coefficients for software fault localization//Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing. Riverside，USA，2006：39-46

[9] Naish L，Lee H J，Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology，2011，20(3)：11-43

[10] Korel B，Yalamanchili S. Forward computation of dynamic program slices//Proceedings of the 1994 International Symposium on Software Testing and Analysis. New York，USA，1994：66-79

[11] Zhang X，Gupta R，Zhang Y. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams//Proceedings of the 26th International Conference on Software Engineering. Edinburgh，UK，2004：502-511

[12] Zhang X，Gupta R，Zhang Y. Precise dynamic slicing algorithms//Proceedings of the 25th International Conference on Software Engineering. Portland，USA，2003：319-329

[13] Nagarajan V，Jeffrey D，Gupta R，Gupta N. A system for debugging via online tracing and dynamic slicing. Software：Practice and Experience，2012，42(8)：995-1014

[14] Tan P-N，Steinbach M，Kumar V. Introduction to Data Mining. New Jersey：Addison Wesley，2006

[15] Wong W E，Debroy V，Li Y，Gao R. Software fault localization using dstar (d^*)//Proceedings of the 6th IEEE International Conference on Software Security and Reliability. Gaithersburg，USA，2012：21-30

[16] Wong W E，Debroy V，Choi B. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software，2010，83(2)：188-208

[17] Santelices R，Jones J A，Yu Y，Harrold M J. Lightweight fault-localization using multiple coverage types//Proceedings of the 31st International Conference on Software Engineering. Vancouver，Canada，2009：56-66

[18] Xie X Y，Chen T Y，Kuo F C，Xu B W. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering and Methodology，2013，22(4)：31-70

[19] Wong W E，Qi Y，Zhao L，Cai K Y. Effective fault localization using code coverage//Proceedings of the 31st Annual International Conference on Computer Software and Applications. Beijing，China，2007：449-456

[20] Xie X，Kuo F C，Chen T Y，Yoo S，Harman M. Provably optimal and human-competitive results in SBSE for spectrum based fault localisation. Search Based Software Engineering，2013：224-238

[21] Zhang X，He H，Gupta N，Gupta R. Experimental evaluation of using dynamic slices for fault location//Proceedings of the 6th International Symposium on Automated Analysis-driven Debugging. New York，USA，2005：33-42

[22] Wen W，Li B，Sun X，Li J. Program slicing spectrum-based software fault localization//Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering. Miami，USA，2011：213-218

[23] Wong W E，Debroy V，Golden R，Xu X F，Thuraisingham B. Effective software fault localization using an RBF neural network. IEEE Transactions on Reliability，2012，61(1)：149-169

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

[25] Xu J，Chan W K，Zhang Z，Li S. A dynamic fault localization technique with noise reduction for java programs//Proceedings of the 11th International Conference on Quality Software. Madrid，Spain，2011：11-20

[26] Jiang S，Li W，Li H，Zhang Y，Zhang H，Liu Y. Fault localization for null pointer exception based on stack trace and program slicing//Proceedings of the 12th International Conference on Quality Software. Xi'an，China，2012：9-12

[27] Alves E，Gligoric M，Jagannath V，d'Amorim M. Fault-localization using dynamic slicing and change impact analysis//Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence，USA，2011：520-523

[28] Yu R，Zhao L，Wang L，Yin X. Statistical fault localization via semi-dynamic program slicing//Proceedings of the 10th IEEE International Conference on Trust，Security and Privacy in Computing and Communications. Changsha，China，2011：695-700

[29] Lei Y，Mao X，Dai Z，Wang C. Effective Statistical Fault Localization Using Program Slices//Proceedings of the 36th International Conference on Computer Software and Applications. Izmir，Turkey，2012：1-10

[30] Renieris M，Reiss S P. Fault localization with nearest neighbor queries//Proceedings of the 18th IEEE International Conference on Automated Software Engineering. Montreal，Canada，2003：30-39

[31] Nessa S, Abedin M, Wong W E, Latifur K, Yu Q. Software fault localization using N-gram analysis. Wireless Algorithms, Systems, and Applications, 2008: 548-559

[32] Zhao Lei, Wang Li-Na, Gao Dong-Ming, et al. Mining association to improve the effectiveness of fault localization. Chinese Journal of Computers, 2012, 35(12): 2528- 2540(in Chinese)
(赵磊，王丽娜，高东明等. 基于关联挖掘的软件错误定位方法. 计算机学报，2012，35(12)：2528-2540)

CAO He-Ling, born in 1980, Ph. D. candidate. Her research interests include software analysis and testing, data mining.

JIANG Shu-Juan, born in 1966, professor, Ph. D. supervisor. Her research interests include compilation techniques, software engineering.

JU Xiao-Lin, born in 1976, Ph. D. candidate, lecturer. His research interests include software analysis and testing.

WANG Xing-Ya, born in 1990, Ph. D. candidate. His research interests include software analysis and testing.

Background

Program debugging is estimated to cost 50% or more of the total maintenance effort. Fault localization is one of the most expensive and time-consuming activities of program debugging. To locate faults, debuggers need to detect and find the statements involved in program failure.

Automatic fault localization approaches always try to narrow the search range of faults. Dynamic slicing can narrow the search space to the statements which affect an output variable. We improved dynamic slices by finding the influence set of definition variables to obtain more accurate dynamic slices.

Coverage-based fault localization approaches usually use statistical analysis to compute the suspiciousness of the statements being faulty, and produce a ranking list according to the suspiciousness. However, these approaches only focus on the suspiciousness of the statements and do not consider association information on program entity, leading to the inaccuracy of locating faults. Moreover, there is association relationship between the suspicious statements and the failed results under test. Association analysis is used to get the relationship between the statements and test results from lots of execution traces.

In this paper, we proposed a new approach based on dynamic slicing and association analysis for locating faults. First, we used dynamic slicing to narrow the scope of the faults. Second, we utilized association analysis and ranking strategy to produce a more reasonable rank of statements to improve the accuracy of fault localization. In addition, we developed a prototype tool to implement our approach. To evaluate the effectiveness of our approach, we performed a series of empirical studies across 13 Java programs. The experimental results indicated that our approach outperformed the compared ones in fault localization.

This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 60970032, 61202006 and 61340037, and Guangxi Key Laboratory of Trusted Software nos. kx201530 and kx201532.