软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(7):1492−1504 [doi: 10.13328/j.cnki.jos.004518]

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

结合用例约简与联合依赖概率建模的错误定位*

苏小红, 龚丹丹, 王甜甜, 马培军

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

通讯作者: 龚丹丹, E-mail: gongdandan0418@126.com

摘 要: 现有的测试用例约简方法不能有效提高错误定位精度,现有的软件错误定位方法不能充分分析元素间的依赖关系.针对以上问题,提出结合测试用例约简和联合依赖概率建模的软件错误自动定位方法,将测试用例约简与软件错误定位统一为一个整体.不同于一般的测试用例约简方法,所提出的测试用例约简方法在程序执行路径的基础上充分考虑了错误测试用例对错误定位的影响,能够为错误定位提供有效的测试用例,为快速、准确地定位软件错误奠定基础.定义了一种新的统计模型——联合依赖概率模型,充分分析了程序元素间的控制依赖、数据依赖以及语句执行状态,并提出基于联合依赖概率模型的错误自动定位方法.通过计算联合依赖关系的可疑度,对可疑节点进行排序,准确定位错误语句.实验结果表明:与 SBI,SOBER,Tarantula,SF 和 RankCP 方法相比,该算法可以更加有效地定位软件错误.

关键词: 程序分析;错误定位;测试用例约简;程序切片;统计分析

中图法分类号: TP311

中文引用格式: 苏小红,龚丹丹,王甜甜,马培军.结合用例约简与联合依赖概率建模的错误定位.软件学报,2014,25(7):1492−1504. http://www.jos.org.cn/1000-9825/4518.htm

英文引用格式: Su XH, Gong DD, Wang TT, Ma PJ. Automatic fault localization approach combining test case reduction and joint dependency probabilistic model. Ruan Jian Xue Bao/Journal of Software, 2014,25(7):1492−1504 (in Chinese). http://www.jos.org.cn/1000-9825/4518.htm

Automatic Fault Localization Approach Combining Test Case Reduction and Joint Dependency Probabilistic Model

SU Xiao-Hong, GONG Dan-Dan, WANG Tian-Tian, MA Pei-Jun

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Corresponding author: GONG Dan-Dan, E-mail: gongdandan0418@126.com

Abstract: The current test case reduction methods can not improve the effectiveness of fault localization, and the current fault localization approaches do not fully analyze the dependency of program elements. To solve these problems, this study proposes an automatic fault localization approach combining test case reduction and joint dependency probabilistic model. Different from the usual test case reduction approach, the failed test cases are fully considered in the proposed test cases reduction method based on execution path in order to provide effective test cases for fast and accurate fault localization. This paper defines a novel statistical model—Joint dependency probabilistic model. In this model, the control dependency and data dependency between program elements, the execution states of each statement are analyzed. An automatic fault localization approach is presented based on joint dependency probabilistic model. It ranks the suspicious statements by calculating the joint dependency suspicion level of the statement. Experimental results show that this approach is more effective than current state-of-art fault-localization methods such as SBI, SOBER, Tarantula, and RankCP.

Key words: program analysis; fault localization; test case reduction; program slicing; statistical analysis

随着软件规模越来越大,逻辑越来越复杂,对软件可靠性的要求也越来越高.影响软件可靠性的一个主要因

* 基金项目: 国家自然科学基金(61173021, 61202092); 教育部博士点基金(20112302120052)
收稿时间: 2012-04-28; 修改时间: 2012-10-19; 定稿时间: 2013-11-05

素是软件中潜在的错误,这些潜在的软件错误不仅给计算机应用系统带来不利影响,甚至可能造成巨大的经济损失和灾难性的后果.传统的软件错误定位方法大多采用设置断点等人工分析的方法,人工定位错误不仅难度大,而且极其耗时.因此,如何有效检测并快速准确地定位和消除软件错误,得到了研究人员的广泛关注.

错误定位[1−4]的精度高度依赖于使用的测试用例,测试用例的数量和质量是决定错误定位的成本和有效性的关键因素.因此,为错误定位提供有效的测试用例,不仅有助于分析软件错误产生的原因,还能极大地提高软件错误定位的效率.目前,测试用例约简的研究大多用于软件测试,而用于错误定位的测试用例约简方法的研究较少.如:Abreu 等人[5]研究了正确测试用例和错误测试用例的数目对错误定位的影响;Baudry 等人[6]将相同测试用例所覆盖的语句定义为动态基本块,研究表明,增加动态基本块的数量有助于提高错误定位精度;Zhang 等人[7]将相对冗余的思想用于错误定位约简方法中,并指出,均衡语句覆盖有助于错误定位;Chen 等人[8]指出软件错误之间存在着相互联系,在此基础上提出了一种轻量级测试用例约简方法,以定位更多的软件错误; Renieris 等人[9]提出“近邻模型”,该模型的主要思想是,在失效测试用例的邻域附近选择成功测试用例,利用失效测试用例和成功测试用例的测试信息进行差异分析以定位故障,但选择的方法是通过对比两条路径所执行的基本语句块的集合来实现的,未考虑在循环分支条件下语句块的执行序列;Wang 等人[10]提出根据错误路径生成最相似的正确路径,但算法的复杂度很高并只能生成一条成功路径,且对赋值错误的定位不太敏感;北京大学的郝丹等人[11,12]主要研究基于语句覆盖的测试用例约简方法,提出了不同的测试用例约简策略,为基于测试用例的故障定位提供高语句覆盖的测试集合,并指出测试用例的冗余或相似降低了错误定位的精度,但它只考虑了语句覆盖,没有考虑路径覆盖;Yu 等人[13]将相对冗余思想用于测试用例约简,提出基于向量的测试用例约简方法,通过实验证明,与基于语句覆盖的测试用例约简方法相比,基于向量的测试用例约简方法更有利于错误定位.目前的测试用例约简方法都只研究了程序运行时的覆盖情况,忽略了程序运行时的路径信息以及错误测试用例对错误定位的影响,因此,约简后的测试用例对错误定位的精度没有明显提高.

软件错误定位近年来已成为一个热点研究问题,主要方法有程序切片、程序谱、统计分析、状态变更、模型检验等.程序切片方法[14−16]所需测试用例较少,但不提供语句的可疑程度描述,并且切片规模仍然可能很大,在切片中观察程序的行为代价较大.程序谱方法[17−24]提供语句的可疑程度描述,但缺少对错误行为的分析,仍需开发人员确定应用程序的期望行为,以确认和修正错误,主要方法包括 Tarantula[17],WHITHER[18],Zoltar[19]等.状态变更方法[25−28]和模型检验方法[29−31]提供了错误行为信息,可以辅助开发人员理解软件错误.但是对于复杂系统而言,这两种方法的开销很大,限制了它们的有效性.统计分析方法[32−40]具有计算复杂度低、准确性较高的优点,是一种很有发展前景的方法,例如 SBI[32],Holmes[33],SOBER[34]等.然而,统计分析的方法缺少对程序元素间的相互影响的分析,对测试用例的数量和质量要求高.CP[38]方法分析了程序的控制依赖关系,通过分析程序的执行状态定位语句错误.在以前的工作中,我们提出了基于状态依赖概率的错误定位方法(后面将其简称为 SF).该方法在控制依赖图的基础上分析了语句的状态依赖关系,进而定位错误语句.Jiang 等人[39]提出结合特征选择、聚类、控制流图遍历来辅助查找生成错误路径,辅助诊断潜在的错误.RankCP[40]方法分析了程序的控制依赖以及数据依赖信息,通过分析元素间的条件依赖关系定位错误语句.

尽管在软件错误自动定位领域已有很多研究,但仍存在以下关键问题没有得到良好的解决:

(1) 如何将软件测试与错误定位有机结合,为错误定位提供有效的测试用例,以提高错误定位的精度;

(2) 如何提供一种自动的软件错误定位方法,使其充分分析程序元素间的相互影响,以有效地定位软件错误.

针对以上问题,本文提出结合测试用例约简和联合依赖概率建模的软件错误自动定位方法,将测试用例约简、错误定位有机地统一为一个整体.首先,针对软件错误定位自身的特点,提出面向错误定位需求的测试用例约简方法,为错误定位提供有效的测试用例,降低错误定位的复杂度,并且提高错误定位的精度;然后,定义一种新的统计分析模型——联合依赖概率模型;在此基础上,提出可疑语句的定位和排序方法,以有效定位错误语句.

1 结合测试用例约简与联合依赖概率建模的软件错误自动定位模型

如图 1 所示,本文将测试用例约简与软件错误定位有机结合,提出面向错误定位需求的测试用例约简方法,同时提出结合测试用例约简和联合依赖概率建模的软件错误自动定位方法.

(1) 采用面向错误定位需求的测试用例约简方法,为错误定位提供合适的测试用例,降低错误定位的复杂度,并且提高错误定位的精度;

(2) 通过运行精简的测试用例,获取节点的控制依赖信息和数据依赖信息.结合程序的控制流图和数据依赖图,分析节点在不同状态下的联合依赖关系,进而建立联合依赖概率模型;

(3) 采用基于联合依赖概率模型的错误定位方法,对程序语句按可疑度进行降序排列.

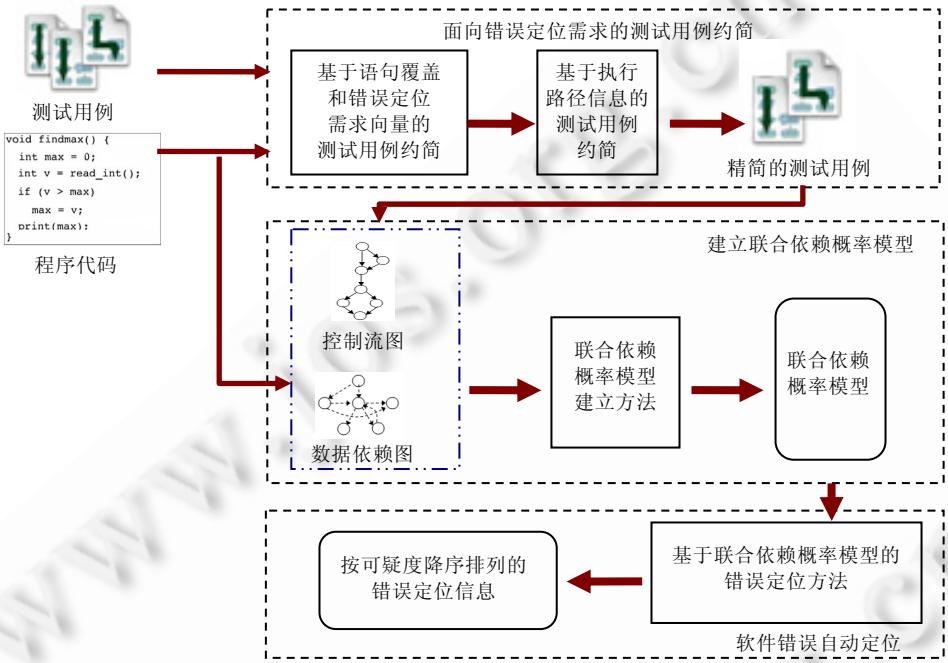


Fig.1 Model of automatic fault localization approach combining test cases reduction and joint dependency probabilistic model

图 1 结合测试用例约简与联合依赖概率建模的软件错误自动定位模型

2 面向错误定位需求的测试用例约简

2.1 面向错误定位需求的测试用例约简模型

本文提出的面向错误定位需求的测试用例约简模型如图 2 所示.该模型主要分为 3 个模块:数据预处理模块、基于语句覆盖和错误定位需求向量约简模块、基于执行路径信息约简模块.

(1) 数据预处理:利用插装器向程序中插入探针,通过动态执行测试用例,捕获插装信息(主要包括语句覆盖信息和具体的执行路径信息);

(2) 基于语句覆盖和错误定位需求向量的测试用例约简:面向错误定位的需求,通过执行错误测试用例并计算错误定位需求向量,删除与错误测试用例无关和关系较小的正确测试用例;

(3) 基于执行路径信息的测试用例约简:对具有相同语句覆盖的测试用例,分析其对应的执行路径信息,删除其中执行路径完全相同的冗余的测试用例.对剩余的路径进行循环标准化,删除冗余路径对应的测试用例.

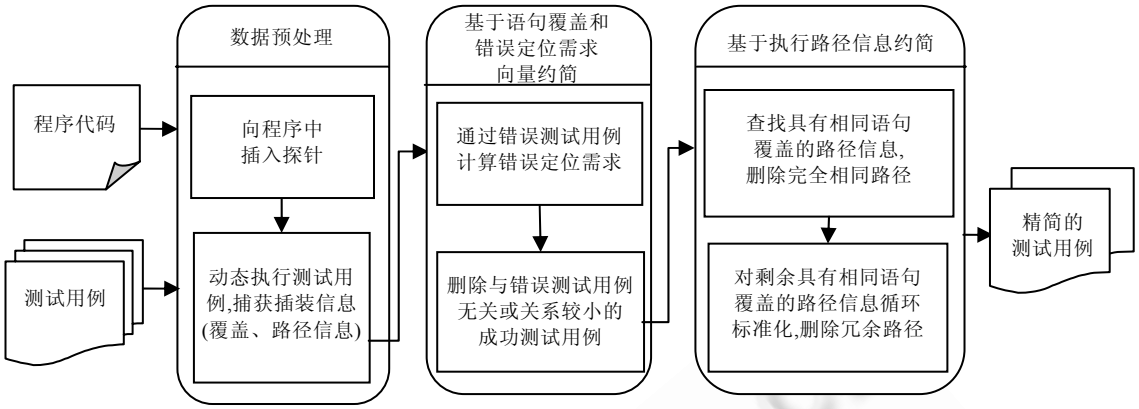


Fig.2 Model of test cases reduction approach oriented to fault localization requirement

图 2 面向错误定位需求的测试用例约简模型

为了更好地理解语句覆盖向量和路径向量,下面加以举例说明.程序如图 3 所示,此函数的功能是求 n 个数的最大数和最小数,执行测试用例 t:n=3,a[·]={11,13,12},动态执行得到的语句覆盖向量是{1,1,1,1,1,1,1,0,1,1},路径向量是{0,1,2,3,4,6,8,3,4,5,6,8,3,4,6,8,3,9}.

```
0: void max_min(int a[·],int n,int *max,int *min){
1:    *max=*min=a[0];
2:    int i;
3:    for (i=0; i<n; i++){
4:          if (a[i]>*max)
5:                *max=a[i];
6:          else if (a[i]<*min)
7:                *min=a[i];
8:       }
9: }
```

Fig.3 Example program max_min()

图 3 程序实例 max_min()

2.2 基于语句覆盖和错误定位需求向量的测试用例约简

与未被错误测试用例执行的语句相比,被错误测试用例执行的语句更有可能是错误的.而所有正确测试用例和错误测试用例都执行的语句(如函数入口等),出错的可能性比较小.基于这一思想,本文定义了两类错误定位需求:一类是面向含有单个错误程序的错误定位需求,此时,被所有错误测试用例都执行的语句,出错的可能性更大,这些语句更有助于定位错误,则约简后的测试用例应该执行这些语句,因此,错误定位需求向量由对所有测试用例对应的覆盖向量求交得到;另一类是面向包含多个错误程序的错误定位需求,此时,每个错误测试用例可能只执行其中一个或几个错误语句,每个错误测试用例执行的语句都可能是错误语句,则被错误测试用例执行过的所有语句都将有利于定位错误,因此,错误定位需求向量由对所有错误测试用例对应的覆盖向量求并得到.根据以上分析,本文提出基于语句覆盖和错误定位需求向量的测试用例约简方法,删除与错误定位需求无关或关系较小的测试用例,如图 4 所示.

(1) 通过执行错误测试用例,得到每个错误测试用例所对应的覆盖向量.由于错误测试用例覆盖向量直观地反映了错误测试用例对各个语句的覆盖情况,因此,错误语句必然包含在错误测试用例对应的覆盖向量中.对于含有单个错误的程序,对所有错误测试用例对应的覆盖向量求交,即可得到错误定位需求向量;而对于含有多个错误的程序,对所有错误测试用例对应的覆盖向量求并,可得到错误定位需求向量;

(2) 通过执行正确测试用例,得到正确测试用例覆盖向量集;

(3) 被所有正确测试用例和错误测试用例都执行的语句,对错误定位的贡献不大.因此,屏蔽错误定位需

求向量和正确用例覆盖向量集中都执行的语句,将屏蔽后的各个正确测试用例向量与屏蔽后的错误定位需求向量求交,如果结果为全 0,表明该正确测试用例与错误定位需求无关或关系较小,加以删除,最终得到基于语句覆盖和错误定位需求向量的测试用例约简后的测试用例.

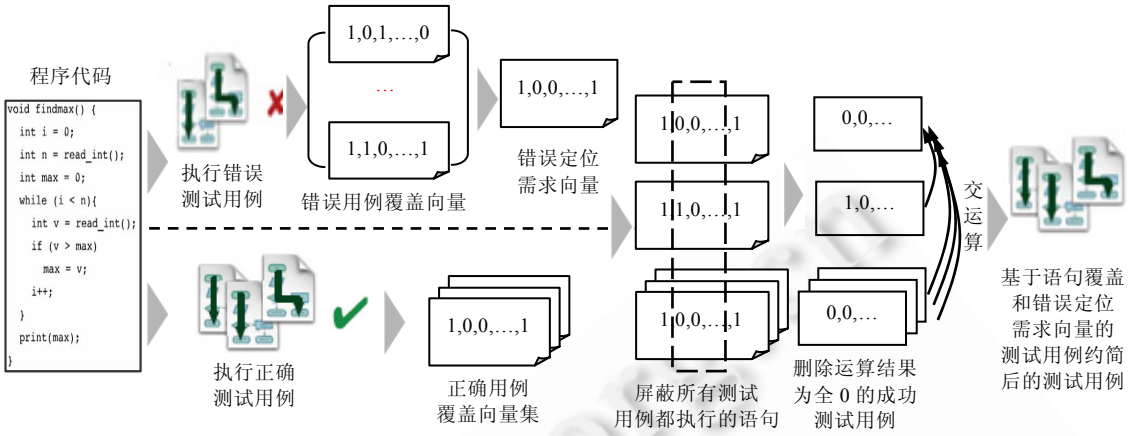


Fig.4 Model of test cases reduction based on statement coverage and fault localization requirement vector

图 4 基于语句覆盖和错误定位需求向量的测试用例约简方法模型

2.3 基于执行路径信息的测试用例约简方法

已有研究表明:均衡语句覆盖有助于错误定位[7],但具有相同语句覆盖的测试用例,所对应的执行路径未必相同.仅根据语句覆盖集合约简测试用例,可能会丢失很多有用的路径信息.因此,本文提出基于执行路径信息的测试用例约简方法,以均衡执行路径的覆盖,如图 5 所示.

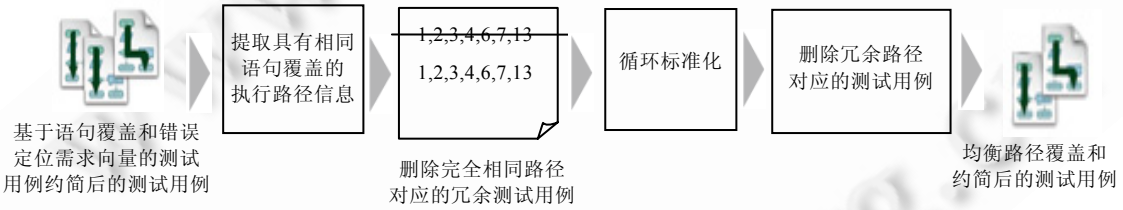


Fig.5 Model of test cases reduction based on path information

图 5 基于执行路径信息的测试用例约简方法模型

对经过基于语句覆盖和错误定位需求向量的测试用例约简后的测试用例进行如下操作:

(1) 提取具有相同覆盖的测试用例所对应的执行路径,删除完全相同路径对应的冗余测试用例;

(2) 对步骤(1)中的剩余路径进行循环标准化,删除冗余路径对应的测试用例,最终得到均衡路径覆盖和约简后的测试用例.

循环标准化的实质就是对每一条路径,将连续的重复片段进行去重,如果几条路径经过标准化后的结果相同,则只保留循环次数最少的路径所对应的测试用例.例如,测试用例 t_1 和 t_2 所对应的执行路径为{0,1,2,3,4,6,8,3,4,5,6,8,3,4,6,8,3,9}和{0,1,2,3,4,6,8,3,4,6,8,3,4,6,8,3,4,5,6,8,3,4,5,6,8,3,4,6,8,3,4,6,8,3,9},二者标准化后结果都为{0,1,2,3,4,6,8,3,4,5,6,8,3,4,6,8,3,9}.而 t_1 所对应的执行路径循环次数较少,因此删除测试用例 t_2,只保留 t_1.

3 基于联合依赖概率模型的错误定位方法

3.1 联合依赖概率模型

程序的控制依赖关系描述了一条程序语句语义的变化是否影响其他语句的执行状况.数据依赖描述了变

量的定义、使用和引用关系.将控制依赖和数据依赖相结合,可以更好地分析程序元素间的相互影响.因此,在分析程序控制依赖和数据依赖的基础上,本文提出了联合依赖概率模型用于错误定位.

定理 1(联合依赖概率模型). 程序代码 P 的联合依赖概率模型是一个三元组(D,S,R),它记录了节点在不同状态下的依赖概率,其中,

(1) $D=(N,E)$是 P 的数据依赖图,N 为节点集合,E 为数据依赖边集合,表示程序的数据依赖信息;

(2) S 是节点到状态的映射.每个节点都对应一个与程序执行相关的状态集合:分支节点和循环节点的状态是其谓词的值(真或假);其他节点的状态表明该节点是否被执行,若节点被执行则为真,否则为假;

(3) R 是各节点的联合依赖概率.节点联合依赖概率是根据运行测试用例得到的节点状态依赖概率和节点条件依赖概率计算得到的.

联合依赖概率模型的建立方法如图 6 所示.

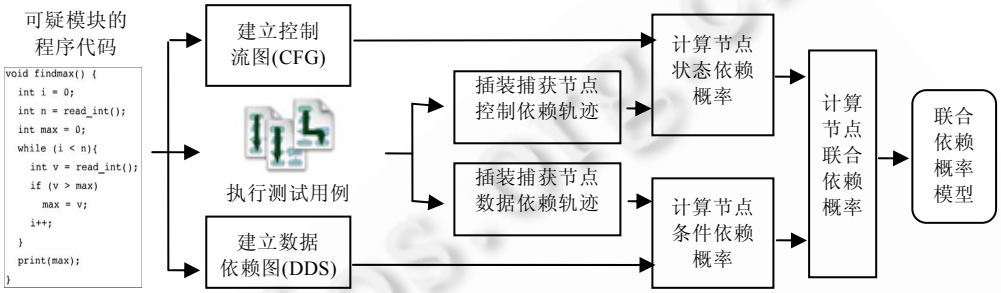


Fig.6 Establishment of joint dependency probabilistic model

图 6 联合依赖概率模型建立方法

(1) 通过执行多组测试用例,捕获节点控制依赖轨迹和数据依赖轨迹.

例如,对于程序实例 max_min(·)(如图 3 所示),执行测试用例 t:n=3,a[·]={11,13,12},捕获的控制依赖轨迹为0,1,2,3(true),4(false),6(false),8,3(true),4(true),5,6(false),8,3(true),4(false),6(false),8,3(false),9;捕获的数据依赖轨迹为 3(true,($d3[i]$,$d0[n]$)),4(false,($d3[i]$,$d1$[max])),6(false,($d3[i]$,$d1$[min])),3(true,($d3[i]$,$d0[n]$)),4(true,($d3[i]$,$d1$[max])),5($d3[i]$),6(false,($d3[i]$,$d1$[min])),3(true,($d3[i]$,$d0[n]$)),4(false,($d3[i]$,$d5$[max])),6(false,($d3[i]$,$d1$[min])).其中,4(false,($d3[i]$,$d1$[max]))表示节点 4 的执行状态是 false,并且节点 4 数据依赖于节点 3 和节点 1,数据依赖变量分别为 i 和 max.

(2) 根据控制流图(CFG)和控制依赖轨迹,计算节点状态依赖概率,即,节点被执行的概率 $P(state)$.分支节点和循环节点记录还需记录状态为真和状态为假的概率,即,$P(state$=true)和 $P(state$=false);

(a) 对于程序中的每个节点,计算节点被执行的概率:

$$P(state)=\frac{n(node)}{n(para(node))}\times P(para(state)) \tag{1}$$

(b) 对于分支节点和循环节点,节点的状态分为真和假.因此,除计算节点被执行的概率外,还需计算节点状态为真和状态为假的概率,则有,

$$P(state)=P(state=\text{true})+P(state=\text{false}) \tag{2}$$

$$P(state=\text{true}))=\frac{n(node(\text{true}))}{n(node)}\times P(state)=\frac{n(node(\text{true}))}{n(node(\text{true}))+n(node(\text{false}))}\times P(state) \tag{3}$$

$$P(state=\text{false}))=\frac{n(node(\text{false}))}{n(node)}\times P(state)=\frac{n(node(\text{false}))}{n(node(\text{true}))+n(node(\text{false}))}\times P(state) \tag{4}$$

其中,$n(node)$为节点在控制依赖轨迹中出现的次数;$n(para(node))$为该节点的父节点在控制依赖轨迹中出现的次数;$n(node(\text{true}))$为当节点状态为 true 时,在控制依赖轨迹中出现的次数;$n(node(\text{false}))$为当节点状态为 false 时,在控制依赖轨迹中出现的次数;$P(para(state))$为其控制依赖父节点在状态 $state$ 下被执行的概率.

(3) 根据数据依赖图(DDG)和数据依赖轨迹,计算节点条件依赖概率(节点在某一状态下,与其数据依赖节点之间的概率),即,$P(data\ dependency|state)$:

$$P(data\ dependency \mid state)=\frac{n(state,data\ dependency)}{n(state)} \tag{5}$$

其中,$n(state,data\ dependency)$为节点在数据依赖轨迹中,状态为 *state*、数据依赖为 *data dependency* 出现的次数;$n(state)$为节点在数据依赖轨迹中,状态为 *state* 出现的总次数.

(4) 根据节点状态依赖概率和节点条件依赖概率,计算节点联合依赖概率,联合依赖概率为 $P(state,data\ dependency)$,表示节点状态为 *state*、同时数据依赖为 *data dependency* 的概率.

联合依赖概率=条件依赖概率×状态依赖概率 (6)

即:

$$P(state,data\ dependency)=P(data\ dependency|state)\times P(state) \tag{7}$$

3.2 基于联合依赖概率模型的错误定位方法

如果某个节点的联合依赖关系在错误测试用例执行过程中出现的频率较高,而在正确测试用例执行过程中出现的频率较低或没有出现,则该节点的联合依赖关系很可能是错误的.基于这种思想,本文提出了基于联合依赖概率模型的错误定位方法,如图 7 所示.

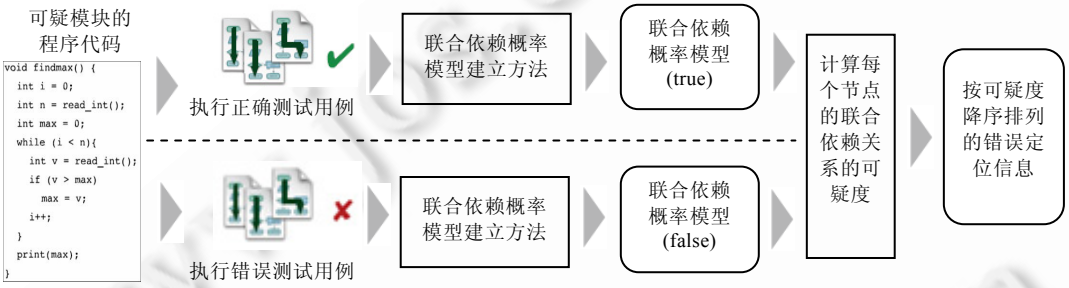


Fig.7 Fault localization approach based on joint dependency probabilistic model

图 7 基于联合依赖概率模型的错误定位方法

(1) 分别根据正确测试用例和错误测试用例建立联合依赖概率模型.分别记为“联合依赖概率模型(true)”和“联合依赖概率模型(false)”;

(2) 根据“联合依赖概率模型(true)”和“联合依赖概率模型(false)”计算各联合依赖关系的可疑度(*suspicious_score*(*JD*)),并根据可疑度对联合依赖关系所对应的节点进行排序.

$$suspicious\_score(JD)=\frac{P_{failed}(JD)}{P_{passed}(JD)} \tag{8}$$

其中,$P_{failed}(JD)$为错误测试用例对应的联合依赖概率模型中,该联合依赖的概率;$P_{passed}(JD)$为正确测试用例对应的联合依赖概率模型中,该联合依赖的概率.

对于公式(8),以下两点需要注意:

(1) 对于 $P_{failed}(JD)$为 0 的节点,不计算其联合依赖关系的可疑度,因为概率为 0 表示该联合依赖关系在错误测试用例中未出现,则此联合依赖关系出错的可能性极小;

(2) 若分母为 0,表示该联合依赖关系在正确测试用例中未出现,则该联合依赖关系出错的可能性极大.对于分母同时为 0 的节点,比较其分子的大小.分子越大,表明该联合依赖关系在错误测试用例中出现的次数越多,则,更有可能是错误.

对于分支节点和循环节点,还需要计算状态为真和假时的联合依赖关系的可疑度,即:

suspicious_score(*JD*|*state*=true)和 *suspicious_score*(*JD*|*state*=false).

在实验中,选择 *suspicious_score*(*JD*|*state*=true)和 *suspicious_ score*(*JD*|*state*=false)的较大者作为 *suspicious_ score*,进行错误排名.

4 实验结果分析

4.1 实验数据

本文的实验数据使用了 Siemens Suite[41]和 space[42,43],具体信息见表 1.表中第 1 列为程序名称,第 2 列为每个程序对应的错误版本个数,第 3 列为程序可执行代码的行数,第 4 列为每个程序提供的测试用例个数,第 5 列为每个程序的功能描述.Siemens Suite 包含 7 组实现不同功能的 C 程序,每组程序通过人工注入的方式创建了基本程序的错误版本,这些错误通常通过修改程序中的一行代码来注入,包括语句的增删和判断条件的修改等,以模拟实际中可能存在的错误.space 由欧洲航天局开发,与 Siemens Suite 相比,space 具有更多的代码行数和测试用例.在这 170 个版本中,删除了其中的 21 个错误版本,包括:(1) 无错误测试用例:schedules2 的第 9 个版本,replace 的第 32 个版本,space 的第 1,2,3,12,32,34 个错误版本;(2) 仅头文件错误:print_tokens 的第 4 和第 6 个错误版本; (3) 程序段错误:replace 的第 27 个版本,print_token2 的第 10 个版本,schedule 的第 5,6,9 个版本,space 的第 25,26,30,35,36,38 个版本.删除 21 个版本后,对 Siemens Suite 的 123 个版本和 space 的 26 个版本进行了实验.

Table 1 Experimental subject programs

表 1 实验数据

Program	No.	Line	Num	Description
print_tokens	7	472	4 130	Lexical analyzer
print_tokens2	10	399	4 115	Lexical analyzer
replace	32	512	5 542	Pattern replacement
schedule	9	292	2 650	Priority scheduler
schedule2	10	301	2 710	Priority scheduler
tcas	41	141	1 608	Altitude separation
tot_info	23	440	1 052	Information measure
space	38	6 218	13 585	Array definition interpreter

4.2 实验结果及分析

4.2.1 测试用例约简方法实验结果分析

本文对 Siemens Suite 提供的 7 个程序的 123 个错误版本和 space 程序的 26 个错误版本进行了实验.箱式图可直观分析、比较多组数据平均水平和变异程度,具体包括数据的 5 个特征值:最小值、最大值、中位数、四分之一位数和四分之三位数.其中,四分之一位数和四分之三位数构成箱子的上下两个边,箱子中间的点为中位数,上下两个点为最大值和最小值.

本文采用箱式图统计了 8 组程序约简率(reduction rate)的分布情况,如图 8 所示.

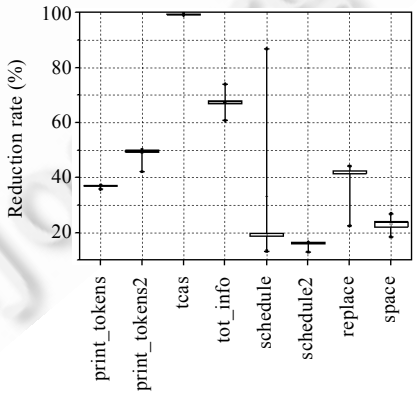


Fig.8 Distribution of percentage in test cases size reduction

图 8 约简率分布情况

由图 8 可以看出,8 组程序的箱式图的覆盖范围都比较小,表明约简的测试用例数目比较集中.schedule 的最大值比较大,远远偏离了箱式图,这是因为对于 schedule 的第 4 个错误版本,在 2 650 个测试用例中,大部分测试用例所对应的路径基本相同,经循环标准化后删减了大部分测试用例,最终仅剩 357 个

测试用例.尽管约简后的测试用例数目较少,但并未影响错误定位的精度.采用 Tarantula 方法进行错误定位时,使用约简前 2 650 个测试用例,错误定位排名为第 161 位,而使用约简后的 357 个测试用例,错误定位排名提高到第 66 位.其中,约简率(reduction rate)使用 Yu 等人[13]提出的公式(9)进行计算.

$$约简率=1-\frac{约简后剩余测试用例个数}{测试用例总个数}\times 100\% \tag{9}$$

对于程序 space,Yu 等人[13]在其实验中人工加入了 20 个错误版本,与本文的实验版本数目不同.因此,本文仅将 Siemens Suite 的结果与 Yu 等人提出的方法进行比较,见表 2.其中,SA 为基于语句覆盖的测试用例约简方法,VA 为基于向量的测试用例约简方法.由表 2 可以看出,SA 方法具有最大的约简率,本文方法与 VA 方法的约简率相似.

Table 2 Experimental results of mean reduction rate on Siemens Suite (%)

表 2 Siemens Suite 测试集中平均约简率实验结果(%)

Program	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	average
SA	96.654	97.123	94.426	97.657	97.495	97.728	97.043	96.875
VA	24.330	28.006	25.939	54.678	35.702	95.100	68.241	47.428
本文方法	36.934	48.821	41.411	25.094	15.978	99.440	67.395	47.868

为了与 Yu 等人提出的方法作对比,本文将未约简的测试用例和约简后的测试用例作为 Tarantula[17]的输入,并记录错误定位结果.结果见表 3,其中,消耗增量(*ExpenseChange*)定义为

$$ExpenseChange=\frac{UnreducedRank-ReducedRank}{程序可执行代码行数}\times 100 \tag{10}$$

其中,*UnreducedRank* 为运行未约简的测试用例时,可疑语句的排名;*ReducedRank* 为运行约简后的测试用例时,可疑语句的排名.则有:*ExpenseChange* 为正值时,表示约简后的测试用例可以提高错误定位的精度;反之,*ExpenseChange* 为负值时,表示约简后的测试用例降低了错误定位的精度.

Table 3 Experimental results of mean *ExpenseChange* on Siemens Suite

表 3 Siemens Suite 测试集中 *ExpenseChange* 均值实验结果

Program	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	average
SA	−4.934	−4.597	−4.747	−8.805	−6.081	−6.854	−4.895	−5.845
VA	−0.062	0.408	−0.31	0.367	−0.600	−0.019	1.075	0.123
本文方法	2.077	1.723	1.172	5.022	3.322	0.921	0.614	2.122

由表 3 可以看出:SA 方法的 *ExpenseChange* 皆为负值,表明 SA 约简后的测试用例降低了错误定位的精度;VA 方法的 *ExpenseChange* 有正有负,说明 VA 方法约简后的测试用例,有些版本可以提高错误定位的精度,有些版本却降低了错误定位的精度;而本文提出的测试用例约简方法的 *ExpenseChange* 都为正值,且值相对较大,说明本文提出的测试用例约简方法可以有效提高错误定位的精度.

表 4 为本文方法在 space 测试集中的实验结果.由于 space 程序的可执行代码行数较大,因此,尽管 *ExpenseChange* 值相对较小,为 0.129,根据公式(10),可疑语句排名平均上升 8 位,可以有效节省开发人员定位软件错误的时间.

Table 4 Experimental results of test cases reduction on space program

表 4 space 测试集中测试用例约简实验结果

Program	Mean reduction rate	Mean *ExpenseChange*
Space	23.025%	0.129

4.2.2 错误定位方法实验结果分析

为了与已有研究进行比较,本文采用被广泛应用的 *T*-score[9,13,17,26,34]来描述错误定位的精度,如公式(11)所示:

$$T=\frac{\text{可疑语句的错误定位排名}}{\text{程序可执行代码行数}}\times 100\% \tag{11}$$

为了验证本文提出的错误定位方法的有效性,本文方法与 SBI,SOBER,Tarantula,SF,RankCP 方法的实验结果如图 9 所示,X 轴为找到错误语句需检查语句的百分比,Y 轴为成功定位错误版本的比率,记为 *fault_percent*,如公式(12)所示:

$$fault\_percent=\frac{\text{成功定位错误版本数}}{\text{总版本数}}\times 100\% \tag{12}$$

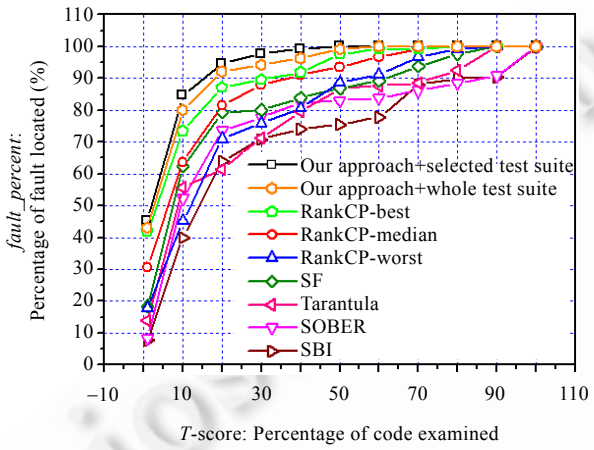


Fig.9 Comparisons on the effectiveness of each fault localization technique

图 9 错误定位效果对比图

由图 9 可以看出,本文方法具有最高的错误定位精度.SBI 方法通过谓词覆盖信息计算语句的可疑度,能够隔离程序中的多重错误,但它只记录实体在执行过程中是否被覆盖到,不足以捕获测试用例的行为信息.SOBER 改进了 SBI,分析了执行过程中谓词为真和假的执行次数.然而,SBI 和 SOBER 方法只能定位与谓词相关的错误,对于与谓词无关的错误,定位效果较差.Tarantula 方法利用程序执行时语句的覆盖信息计算所有执行语句的可疑度,进而定位错误语句.虽然 Tarantula 可以定位所有错误语句,但它只利用了程序运行时各个语句的覆盖信息,并未分析程序执行时的路径信息,也未考虑谓词的具体状态.SF 方法在程序具体执行路径的基础上,分析了程序元素间的控制依赖关系以及程序执行时各元素的具体执行状态,其错误定位精度高于 SBI,SOBER 和 Tarantula,但该方法并未分析程序元素间的数据依赖信息.本文方法和 RankCP 方法不仅分析了程序元素间的控制依赖关系,还分析了程序元素间的数据依赖关系.RankCP 将依赖网络与程序分析相结合,提出程序依赖图(PPDG),统计元素间的条件依赖关系.但它假定所有语句被执行的概率都相等,而实际上,程序在执行过程中,每条语句被执行的概率并不相同,对于分支语句和循环语句,每个状态(谓词为真、假)被执行的概率也不同.因为 RankCP 方法每次只使用一条错误测试用例进行错误定位,因此,该方法统计了最好、最坏以及平均错误定位结果,分别为 RankCP-best,RankCP-worst,RankCP-median.本文提出了联合依赖概率模型,该模型不仅能分析每个节点不同状态被执行的概率,而且可以统计节点在不同状态下与其数据依赖父节点间的联合依赖概率,在程序执行路径的基础上充分分析了控制依赖和数据依赖信息,能够更准确地分析程序的错误行为状态,从而能够更有效地定位错误语句.由图 9 可以看出,使用未约简的测试用例,本文方法的错误定位效果优于 RankCP-best.在检查相同数目的程序语句时,本文方法可以定位更多错误版本.在运行本文方法约简后的测试用例时,检查程序 50%的语句,本文方法可以定位所有语句错误.

表 5 为本文方法和已有方法在 space 测试集中的实验对比结果,其中,排名增量 *RankChange* 定义为

$$RankChange=\frac{PreRank-OurRank}{\text{程序可执行代码行数}}\times 100 \tag{13}$$

其中,*PreRank* 为使用前人方法(如 SOBER,Tarantula,SF,RankCP-best)错误语句的排名,*OurRank* 为使用本文方法错误语句的排名.由此可以看出:*RankChange* 为正,表明与前人方法相比,本文方法可以提高错误定位的精度;反之,*RankChange* 为负,表明与前人方法相比,本文方法降低了错误定位的精度;*RankChange* 的值越大,表明与前人方法相比,本文方法对提高错误定位精度的效果越明显.

Table 5 Experimental results of fault localization on space program

表 5 space 测试集中错误定位实验结果

Program	*RankChange*				
	本文方法:SBI	本文方法:SOBER	本文方法:Tarantula	本文方法:SF	本文方法:RankCP-best
Space	0.241 2	0.193 0	0.209 1	0.160 8	0.080 4

4.2.3 小测试集实验结果分析

在现实情况中,错误测试用例往往很难得到,因此,本文探讨了在错误测试用例数目较少时的错误定位精度.图 10 分别为使用 1,3,5 以及所有的错误测试用例时的错误定位精度.使用的测试用例集为本文方法约简后的测试用例集合.测试对象为 Siemens Suite 程序集.RankCP 方法每次只能使用 1 条错误测试用例进行错误定位,而本文方法不受错误测试用例个数的限制.图 10 表明:同样使用 1 条错误测试用例,本文方法的错误定位精度高于 RankCP-best;随着错误测试用例数目的增多,本文方法具有更高的错误定位精度.

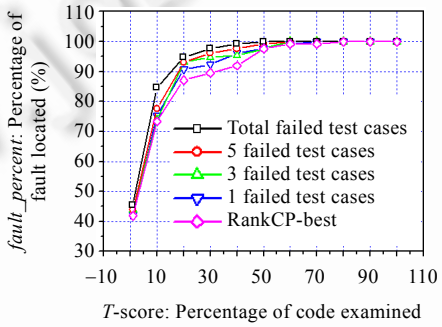


Fig.10 Effect of the number of failed test cases on fault localization

图 10 错误测试用例数目对错误定位的影响

为了验证本文提出的测试用例约简方法对包含多个错误的程序的有效性,在 Siemens Suite 测试集上进行了错误注入实验.对于每个程序,只随机选择其中的一个版本进行实验.将相邻的两个错误版本进行错误合并.例如,选择 tcas 的第 8 个版本进行实验,错误注入的方式为:将第 9 个版本的错误注入到第 8 个版本中.根据多个错误语句,可以计算多个 *T*-score 值.选取这些 *T*-score 的最大值作为最终的 *T*-score.通过实验发现,面向多个错误的错误定位需求向量包含了较多的语句,约简后的测试用例相对较多.但与未约简的测试用例相比,约简后的测试用例仍然可以提高错误定位的精度.

5 结束语

本文将测试用例约简与软件错误定位统一为一个整体,提出了结合测试用例约简与联合依赖概率建模的软件错误自动定位方法.首先,本文充分考虑了错误测试用例对错误定位的重要性,根据软件错误定位的需求,提出面向错误定位需求的测试用例约简方法,充分考虑了程序运行时的路径信息,均衡路径覆盖率,为错误定位提供有效的测试用例,并降低错误定位分析的复杂度;然后,本文定义了一种新的统计分析模型——联合依赖概率模型,并在此基础上提出了基于联合依赖概率模型的错误定位方法,分析了元素间控制依赖、数据依赖以及

语句的执行状态.实验结果表明,本文方法能够有效地约简测试用例,并可以有效地定位软件错误.

References:

[1] Zhang ZY, Chanb WK, Tsec TH, Yub YT, Hud PF. Non-Parametric statistical fault localization. Journal of Computer and System Sciences, 2011,84:885−905. [doi: 10.1016/j.jss.2010.12.048]

[2] Jobstmann B, Staberb S, Griesmayerc A, Bloem R. Finding and fixing faults. Journal of Computer and System Sciences, 2012,78: 441−460. [doi: 10.1016/j.jcss.2011.05.005]

[3] Wong WE, Debroy V, Xu D. Towards better fault localization: A crosstab-based statistical approach. IEEE Trans. on Systems, Man, and Cybernetics, 2012,42(3):378−396. [doi: 10.1109/TSMCC.2011.2118751]

[4] Yu K, Lin M, Gao Q, Zhang H, Zhang X. Locating faults using multiple spectra-specific models. In: Proc. of the SAC. 2011. [doi: 10.1145/1982185.1982490]

[5] 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, 2007. [doi: 10.1109/TAIC.PART.2007.13]

[6] Baudry B, Fleurey F, Traon YL. Improving test suites for efficient fault localization. In: Proc. of the Int'l Conf. on Software Engineering. Shanghai, 2006. 82−91. [doi: 10.1145/1134285.1134299]

[7] Zhang X, Gu Q, Chen X, Qi J, Chen D. A study of relative redundancy in test-suite reduction while retaining or improving fault-localization effectiveness. In: Proc. of the SAC. 2010. 2229−2236. [doi: 10.1145/1774088.1774556]

[8] Chen Z, Xu B, Zhang X, Xie C. A novel approach for test suite reduction based on requirement relation contraction. In: Proc. of the SAC. 2008. 390−394. [doi: 10.1145/1363686.1363782]

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

[10] Wang T, Abhik R. Automated path generation for software fault localization. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York, 2005. 347−351. [doi: 10.1145/1101908.1101966]

[11] Hao D, Zhang L, Sun J, Mei H. VIDA: Visual interactive debugging. In: Proc. of the ICSE. 2009. 583−586. [doi: 10.1109/ICSE.2009.5070561]

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

[13] Yu Y, Jones JA, Harrold MJ. An empirical study of the effects of test suite reduction on fault localization. In: Proc. of the ICSE. 2008. 201−210. [doi: 10.1145/1368088.1368116]

[14] Sterling CD, Olsson RA. Automated bug isolation via program chipping. In: Proc. of the 6th Int'l Symp. on Automated Analysis-Driven Debugging. 2005. 23−32. [doi: 10.1145/1085130.1085134]

[15] Zhang X, Gupta N. Locating faulty code by multiple points slicing. Software Practice & Experience, 2007,37(9):935−961. [doi: 10.1002/spe.795]

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

[17] 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 (ASE). 2005. 273−282. [doi: 10.1145/1101908.1101949]

[18] Renieres M, Reiss SP. Fault localization with nearest neighbor queries. In: Proc. of the 18th ASE. 2003. 30−39. [doi: 10.1109/ASE.2003.1240292]

[19] Janssen T, Abreu R, van Gemund AJC. Zoltar: A toolset for automatic fault localization. In: Proc. of the 24th ASE. 2009. 662−664. [doi: 10.1109/ASE.2009.27]

[20] Wong WE, 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. [doi: 10.1016/j.jss.2009.09.037]

[21] Naish L, Lee HJ, Ramamohanarao K. Statements versus predicates in spectral bug localization. In: Proc. of the 17th Asia Pacific Software Engineering Conf. (APSEC). 2010. 375−384. [doi: 10.1109/APSEC.2010.50]

[22] Lo LD, Jiang L, Budi A. Comprehensive evaluation of association measures for fault localization. In: Proc. of the 2010 IEEE Int'l Conf. on Software Maintenance (ICSM). 2010. 1−10. [doi: 10.1109/ICSM.2010.5609542]

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

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

[25] Jeffrey D, Nagarajan V, Gupta R, Gupta N. Execution supression: An automated iterative technique for locating memory errors. ACM Trans. on Programming Languages and Systems, 2010,32(5):1−36. [doi: 10.1145/1745312.1745314]

[26] Cleve H, Zeller A. Locating causes of program failures. In: Proc. of the ICSE. 2005. 342−351. [doi: 10.1145/1062455.1062522]

[27] Gupta N, He H, Zhang X, Gupta R. Locating faulty code using failure-inducing chops. In: Proc. of the ASE. 2005. 263−272. [doi: 10.1145/1101908.1101948]

[28] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching. In: Proc. of the ICSE. 2006. 272−281. [doi: 10.1145/1134285.1134324]

[29] Dallmeier V, Zeller A, Meyer B. Generating fixes from object behavior anomalies. In: Proc. of the ASE. 2009. 550−554. [doi: 10.1109/ASE.2009.15]

[30] Mariani L, Pezzè M. Dynamic detection of COTS components incompatibility. IEEE Software, 2007,24(5):76−85. [doi: 10.1109/MS.2007.138]

[31] Leonardo M, Pezze M, Riganelli O, Santoro M. SEIM: Static extraction of interaction models. In: Proc. of the 2nd Int'l Workshop on Principles of Engineering Service Oriented Systems (PESOS) Workshop Colocated with ICSE. 2010. 22−28. [doi: 10.1145/1808885.1808891]

[32] Liblit B, Naik M, Zheng A, Aiken A, Jordan M. Scalable statistical bug isolation. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2005. 15−26. [doi: 10.1145/1065010.1065014]

[33] Chilimbi TM, Liblit B, Mehra KK, Nori AV, Vaswani K. Holmes: Effective statistical debugging via efficient path profiling. In: Proc. of the ICSE. 2009. 34−44. [doi: 10.1109/ICSE.2009.5070506]

[34] Liu C, Yan X, Fei L, Han J, Midkiff S. Sober: Statistical model-based bug localization. In: Proc. of the 10th European Software Engineering Conf./13th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (ESEC/FSE 2005). 2005. 286−295. [doi: 10.1145/1081706.1081753]

[35] Wong WE, Wei T, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: Proc. of the 1st Int'l Conf. on Software Testing, Verification and Validation. 2008. 42−51. [doi: 10.1109/ICST.2008.65]

[36] Feng M, Gupta R. Learning universal probabilistic models for fault localization. In: Proc. of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2010. 81−88. [doi: 10.1145/1806672.1806688]

[37] Nainar PA, Liblit B. Adaptive bug isolation. In: Proc. of the ICSE. 2010. 255−264. [doi: 10.1145/1806799.1806839]

[38] Zhang Z, Chan W, Tse T, Jiang B, Wang X. Capturing propagation of infected program states. In: Proc. of the ESEC/FSE. 2009. [doi: 10.1145/1595696.1595705]

[39] Jiang LX, Su ZD. Context-Aware statistical debugging: From bug predictors to faulty control flow paths. In: Proc. of the ASE. 2007. [doi: 10.1145/1321631.1321660]

[40] Baah GK, Podgurski A, Harrold MJ. The probabilistic program dependence graph and its application to fault diagnosis. IEEE Trans. on Software Engineering, 2008,87:189−200. [doi: 10.1145/1390630.1390654]

[41] Hutchins M, Foster H, Goradia T, Ostrand T. Experiments on the effectiveness of dataflow and controlflow-based test adequacy criteria. In: Proc. of the Int'l Conf. on Software Engineering. 1994. 191−200. [doi: 10.1109/ICSE.1994.296778]

[42] Vokolos F, Frankl P. Empirical evaluation of the textual differencing regression testing techniques. In: Proc. of the Int'l Conf. on Software Maintenance. 1998. [doi: 10.1109/ICSM.1998.738488]

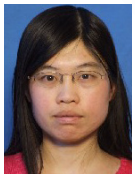
[43] Aristotle Research Group. Aristotle analysis system. 2007. http://www.cc.gatech.edu/aristotle/

苏小红(1966－),女,辽宁海城人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件缺陷检测,软件错误定位,智能信息处理与信息融合.

E-mail: sxh@hit.edu.cn

王甜甜(1980－),女,博士,副教授,主要研究领域为程序分析,计算机辅助教学.

E-mail: sweetwtt@126.com



龚丹丹(1982－),女,博士生,主要研究领域为软件错误定位,软件缺陷检测,程序理解.

E-mail: gongdandan0418@126.com

马培军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,智能信息处理与信息融合.

E-mail: ma@hit.edu.cn