# 一种基于模型结合的错误定位方法

唐启锋[1,2]，许蕾[1,2]，钱巨[3]，陈林[1,2]，张震宇[4]
(1. 南京大学计算机软件新技术国家重点实验室，南京 210093;
2. 南京大学计算机科学与技术系，南京 210093;
3. 南京航空航天大学计算机科学与技术学院，南京 210016;
4. 中国科学院软件研究所，计算机科学国家重点实验室，北京 100080)

**摘要：**目前大多数错误定位技术的研究均基于单一类型的程序节点（如语句、谓词等），其效果往往只在定位相应类型的错误时表现较好，而定位其他类型的程序错误时则表现不佳。为此，借鉴机器学习领域中集成学习的思想，建立多错误定位方法相结合的错误定位模型，并综合了基于语句覆盖信息和程序谓词信息这 2 种错误定位方法，提出了 3 种新的错误定位方法。实验结果表明，相对于此前单一的方法，所提出的 2 种方法具有更高的错误定位效率和更强的适应性。

**关键词：**错误定位；集成学习；程序分析；软件调试
**中图分类号：**TP311.5 **文献标志码：**A

# A fault localization method based on model combination

Tang Qifeng[1,2], Xu Lei[1,2], Qian Ju[3], Chen Lin[1,2], Zhang Zhenyu[4]
(1.State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, China; 2.Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China; 3.College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; 4.State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

**Abstract:** Automated fault-localization technique has become a hot topic in research fields of software engineering. However, most of techniques are based on single type program node(e.g. statement, predicate, etc.) until now, when these techniques utilized to locate the corresponding types of errors, it has good performance in generally. But in other situation, the opposite is true. Therefore, we draws on the method of Integrated Learning in machine learning to contribute a fault-localization model which combines the information of statement-based coverage and predicate-based coverage method. In addition, three new fault-localization methods are proposed. The experimental results show that our methods have better effectiveness and adaptability for fault-localization than previous methods.

**Key words:** fault localization; ensemble learning; program analysis; debugging

## 1 引言

提高软件质量是软件工程亟需解决的重大问题之一。软件调试作为一种提高软件质量的重要手段而备受重视。传统的调试方法主要包括设置断点，重复运行程序和检查相应的程序状态点。但是，这些调试方法通常会耗费大量的时间和金钱，并且已经成为软件发展的一个瓶颈。因此，软件工程研究者提出了很多自动化调试技术来辅助。而错误定位作为软件调试任务之一也备受研究者关注，并由此产生了多种自动化方法。这些自动化的测试方法主要包括静态的错误定位技术和动态的错误定位技术。由于程序语言具有较高的复杂性，如果应用静态错误定位技术则往往导致定位的时间复杂度非常高，难以用于较大规模的程序。所以，目前很多学者更青睐动态的错误定位技术。

当前，大部分的动态错误定位技术是针对某种类型程序节点的频谱信息来建立统计模

**基金项目：**国家自然科学基金(90818027, 61170071, 60903026)；江苏省自然科学基金(BK2011190)
**作者简介**：唐启锋(1986- )，男，硕士生，主要研究方向：软件自动化测试，qifeng_tang@126.com
**通信联系人：**陈林，博士，主要研究方向：软件分析与重构，lchen@nju.edu.cn

型。例如，有些是根据语句的覆盖信息进行建模（如 Tarantula 方法[1]），有些则是基于谓词信息进行建模（如 SOBER 方法[2]），等。一般情况下，某种类型的程序错误往往仅与一种或多种程序节点的频谱信息相关程度比较大。例如，计算型错误与语句覆盖信息关联明显，而控制流错误则更多反映在程序谓词信息。然而，在大多数情况下，基于某种程序节点频谱信息建立的错误定位模型难以处理与其关联不大的程序错误。在实际调试中，由于程序错综复杂，各种类型的程序错误均可能出现。因而，目前还没有一种错误定位模型针对任何一种类型的程序错误其错误定位效果总是最好的。

为了适用于多种类型的程序错误，借鉴集成学习的思想，提出了综合多种错误定位方法的错误定位模型。该模型主要是对多种已有的错误定位方法所输出的结果进行特定的组合以获得更好的错误定位效果。在此模型的基础上，提出了对基于语句覆盖信息和基于程序谓词信息这两种频谱模型结果相结合的组合算法——简单组合算法和基于排序层组合算法，并将其应用到 Tarantula 方法和 SOBER 方法的结合上。西门子程序测试集表明，相比结合前的 SOBER 方法和以 Tarantula 为基础的 T_Tarantula 方法，所提出的模型结合方法的错误定位效果更好，并具有更广泛的适应性。

## 2 问题的提出

基于程序频谱的错误定位方法是程序执行测试用例，对其产生特定的程序节点信息（频谱信息）进行收集和统计分析。所谓的统计分析是指通过比较程序节点信息在正确执行和错误执行时所产生不同的特征，来预测程序的可能出错的位置。这里，正确执行是指程序运行一个测试用例所得到的结果与预想结果是一致；相反，如果得到的结果与预想结果不一致，那么就是错误执行。由于统计的特性，这些方法都需要相对充足的测试用例。

M. Renieris 等给出了基于程序频谱定位程序错误的一般步骤[3]：首先，执行测试用例，产生程序的执行轨迹信息，记为集合 $T$；其次，根据程序代码的某种特征 $k$ 把执行信息转化为某种抽象表示，如语句覆盖信息、谓词信息等。这些信息就是程序的频谱信息，记为 $T_k$，可知 $T_k \subset T$；然后，比较执行结果和预测结果，把集合 $T_k$ 分为 2 类：正确执行信息 $T_{kp}$ 和失败的执行信息 $T_{kf}$；最后，根据频谱信息 $T_{kp}$ 和 $T_{kf}$ 进行建模，得到某种程序节点的可疑值列表。依据该节点可疑值的大小进行排序，从而得到可疑度序列 $\text{Rank}_k$，以辅助程序员来调试错误。

目前，大多数基于程序频谱的错误定位方法是针对语句覆盖信息或谓词信息。这些方法在实践中已经表现出比较好的错误定位效果。然而，这些错误定位技术大都仅与特定类型的程序错误密切关联，而对其他类型的程序错误的定位效果并不理想。

可以通过一个用于处理字符串相关操作的程序 function()来说明这一点：

```
    function() {
1     int t=0;
2     char str[10];
3       while(str[t] != '\0')
4       {
5          If (str[t] >= '0' && str[t] < '5')
6             str[t] += 5;
7          else    if (str[t] >= '5' && str[t] <= '9')
8                      str[t] = 8;
9          t++;
10      }
11    }
```

为清楚起见，把示例程序、测试用例与测试结果等放在同一图中（见图 1），其中共“设计”

了 2 个错误：将语句 8“str[t] = 8;”误为“str[t] -= 1;”的计算型错误，将语句 5“if (str[t] >= ’5’&&str[t] <= ’9’)”误为“if (str[t] >= ’5’)”的控制流错误。

图 1 主要分为 2 部分：function()的语句覆盖信息及其 Tarantula 方法的语句可疑值和谓词信息及其 SOBER 方法的谓词可疑值。从图 1 可以看出，对于错误 1，如果使用 Tarantula 方法，则只要检查 1 条可疑语句就能定位程序中的错误。而如果用 SOBER 方法则需要检查 4 条可疑语句才能定位程序中的错误。然而，对于错误 2，如果使用 Tarantula 方法则需要检查 4 条可疑语句才能定位程序中的错误，而如果使用 SOBER 方法则只需要检查 1 条语句就能定位程序中的错误。由这个例子可以看出，针对同一种类型的程序错误，不同的频谱方法其定位效果可能会有极大的不同。

R. Santelices 等提出了基于多个频谱信息的错误定位模型，并得到了较好的效果[4]。然而，该模型仅仅对多个频谱信息使用同一种语句覆盖信息方法，如 Tarantula。由于其他频谱信息的特征与语句覆盖信息的特征不尽相同，所以，这些基于语句覆盖信息的方法在其他频谱信息应用中效果不一定好。应尝试把针对不同频谱信息中比较典型的方法结合起来，提高错误定位的效果。

错误 *1*：误将语句 *8* 写为 *str*[*t*] -= *1*；　　错误 *2*：误将语句 *7* 写为 *if*(*str*[*t*] >= ’*5*’)

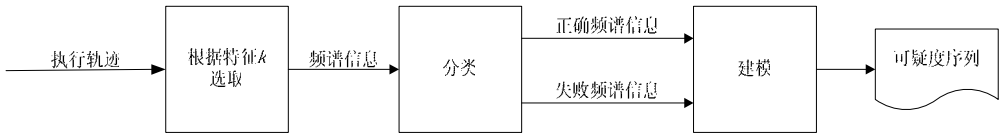
| 语句编号 | | *t1* 空 | *t2* ‘*2*’,’+’ | *t3* ‘*2*’,’*9*’,’*a*’ | *t4* ‘*8*’,’*9*’ | 可疑值 | *t1* 空 | *t2* ‘*2*’,’+’ | *t3* ‘*2*’,’*9*’,’*a*’ | *t4* ‘*8*’,’*9*’ | 可疑值 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *1* | | ● | ● | ● | ● | *0.57* | ● | ● | ● | ● | *0.57* | |
| *2* | | ● | ● | ● | ● | *0.57* | ● | ● | ● | ● | *0.57* | |
| *3* | | ● | ● | ● | ● | *0.57* | ● | ● | ● | ● | *0.57* | |
| *4* | | | | | | *0* | | | | | *0* | |
| *5* | | | ● | ● | ● | *0.67* | | ● | ● | ● | *0.67* | |
| *6* | | | ● | ● | | *0* | | ● | ● | | *0.8* | *Tarantula* 方法 |
| *7* | | | ● | ● | ● | *0.67* | | ● | ● | ● | *0.67* | |
| *8* | | | | ● | ● | *0.8* | | | ● | ● | *0.8* | |
| *9* | | | ● | ● | ● | *0.67* | | ● | ● | ● | *0.67* | |
| *10* | | | | | | *0* | | | | | *0* | |
| *11* | | | | | | *0* | | | | | *0* | |
| 谓词 | | | | | | | | | | | | |
| *3* | 正 | *0* | *3* | *3* | *2* | *-1.71* | *0* | *3* | *2* | *3* | *-1.30* | |
| | 错 | *1* | *1* | *1* | *1* | | *1* | *1* | *1* | *1* | | |
| *5* | 正 | *0* | *2* | *1* | *0* | *10.61* | *0* | *2* | *0* | *1* | *-2.12* | *SOBER* 方法 |
| | 错 | *0* | *1* | *2* | *2* | | *0* | *1* | *2* | *2* | | |
| *7* | 正 | *0* | *0* | *1* | *2* | *5.66* | *0* | *0* | *2* | *2* | *-0.29* | |
| | 错 | *0* | *1* | *1* | *0* | | *0* | *1* | *0* | *0* | | |
| 通过/失败 | | *P* | *P* | *P* | *F* | | *P* | *P* | *F* | *P* | | |

图 1 Tarantula 和 SOBER 应用实例
Fig1 Examples of Tarantula and SOBER

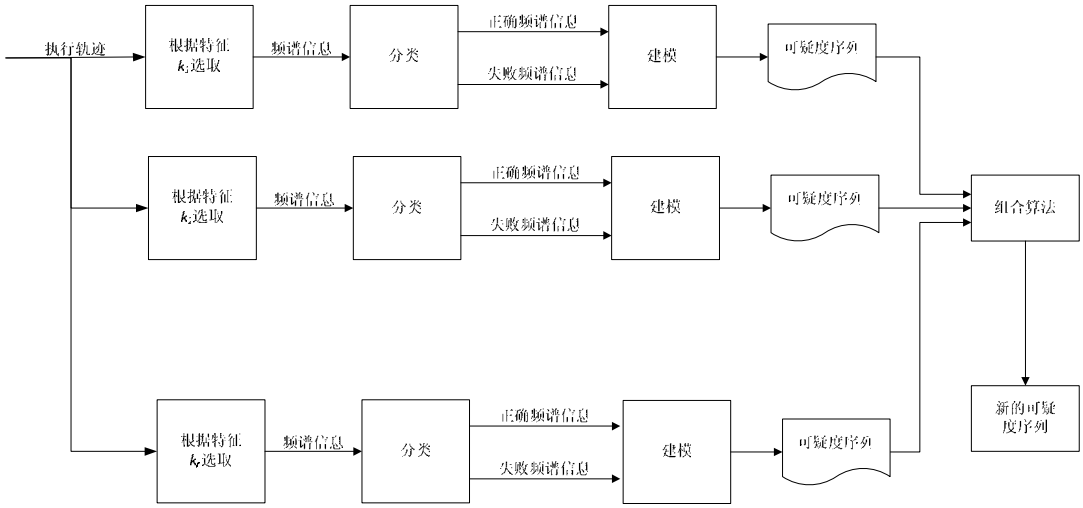
## 3 基于模型集成的错误定位

如图 1 所示，不同类型的程序错误其关联的频谱信息可能各不相同。为了达到更好的错误定位效果，将集成学习方法结论合成的思想运用到程序的错误定位上，提出模型集成的错误定位。

所谓集成学习就是对同一个问题用多种方法建模，并对各个模型所得到的结果进行集成输出。一般而言，为了得到更好的集成效果，所要集成的学习器效果要好，而且学习器的种类要尽可能地不同[5]。




(a) 错误定位建模流程




(b) 多模型组合的错误定位建模流程

图 2 单模型和多模型组合的错误定位建模流程

Fig 2 The model process of fault localization with single-model and muti-model

基于单一频谱信息的错误定位模型相当于集成学习里面的一个学习器，其建模流程如图2(a)所示。而模型集成的错误定位模型就是对不同的频谱信息分别建模，并对各个模型的输出结果进行组合，形成新的结果输出。如图 2(b)所示，其建模流程如下：

1) 待测程序执行测试用例产生其相应的执行轨迹信息 $T$；
2) 从众多程序代码的特征中选取要集成的特征集合 $S_k$。令 $ki \in S_k$，则根据 $ki$ 从 $T$ 中获取相应的频谱信息 $T_{ki}$；
3) 比较执行结果和预测结果，把 $T_{ki}$ 分为成功执行信息 $T_{kp}$ 和失败的执行信息 $T_{kf}$；
4) 对 $T_{kp}$ 和 $T_{kf}$ 建模，得到 $ki$ 相应的可疑度序列 $\mathrm{Rank_{ki}}$；
5) 通过特定的算法 $F$，对所得到不同的 $\mathrm{Rank_{ki}}$ 进行结论合成，从而得到新的排序 $\mathrm{Rank_{mix}}$；

如果同时对过多的模型进行组合，不仅会导致计算的开销越来越大，而且模型的差异性变得更难以获得，这样会影响集成效果。因而，选取了 2 种模型进行组合：基于谓词信息的错误定位模型和基于语句覆盖信息错误定位模型。这 2 个模型目前使用广泛，建模简单，而且具有一定的差异性，满足集成的需求。

1) 基于谓词信息的错误定位模型

所谓程序谓词是代表程序某些状态的布尔表达式。基于谓词信息的错误定位模型是比较程序执行正确和执行失败时谓词信息的特征不同，从而进行错误定位，其代表方法主要有 2 种：CBI 方法和 SOBER 方法[6-7]。

CBI 方法是使用了谓词的中值和它的改变量去评估程序谓词与程序错误的相关度。该方

法首先要计算一个谓词在程序错误执行时被观察为 true 和（无论正确的还是错误的）被观察为 true 的概率。CBI 方法是将后者比前者的增加值作为可疑值，可疑值越高代表该谓词与错误的关联度越高[8]。

SOBER 方法定义了一个评估偏差来代表谓词在程序运行过程中观察为 ture 的情况。如果 $P$ 是一个谓词，该谓词在程序运行过程中观察为 ture 的概率值为 $\pi(P)$，即

$$\pi(P)=\frac{n_t}{n_t+n_f},$$

其中，$n_t$ 是程序运行是，$P$ 为 true 的次数，而 $n_f$ 是 $P$ 为 false 的次数。SOBER 方法则应用中心极限定理去评估 $\pi(P)$在正确运行和错误运行时的差异度，两者的差异度越大，表示该谓词与错误的关联度越高。

2) 基于语句覆盖信息的错误定位模型

现有语句覆盖信息的错误定位方法是基于一个先验知识：如果一个语句给更多的错误测试用例所覆盖，那么这个语句与错误的相关度越高；反之，如果一个语句给更多的正确测试用例所覆盖，那么这个语句与程序错误的相关度越低。以 Tarantula 方法为例[1]，代表语句与程序错误关联度的值为

$$T(s)=\frac{\text{fail\%}}{\text{fail\%}+\text{success\%}}$$

其中，success%为被正确测试用例覆盖的比率；fail%为被错误测试用例覆盖的比率。该方法根据 $T(s)$的值对语句进行排名从而定位错误。

# 4 模型结合算法

最大的难点是如何将基于不同度量标准的 2 种方法相结合。L. Xu 等[9]深入研究了模型结合的问题，把模型的整合方式分为抽象层次、排位层次和度量层次 3 个层次。其中，排序层次和度量层次的模型结合方式更适合用于错误定位模型。因此，基于这 2 种结合的方式提出了简单组合和基于排序层的结论合成 2 个合成方法。

## 4.1 简单组合方法

针对不同类型的程序错误，Tarantula 和 SOBER 方法的错误定位效果具有较大的差异性。为了综合这 2 个方法的优点，本方法主要针对这 2 个方法中可疑值最高的节点进行组合。

Tarantula 和 SOBER 方法的可疑度序列的度量标准不同，但每个序列的节点都能对应 PDG 图上的一个节点。因此，首先对这些方法可疑值高的节点进行适当组合，然后结合使用的搜索错误语句的方法（基于 PDG 的 T-score 方法）来查找程序的错误。这里的适当组合是指把所要结合的模型其输出的可疑度高节点集合起来作为候选节点，具体方法如下：

1) 对覆盖信息应用 Tarantula 方法得到语句的可疑度排序 $\text{Rank}_s$，对谓词信息应用 SOBER 方法得到谓词的可疑度排序 $\text{Rank}_p$；

2) 选取 $\text{Rank}_s$ 中具有最高可疑值的语句节点组成节点集合，记为 $V_{sh}$。选取 $Rank_p$ 中可疑值高的谓词节点组成节点集合，记为 $V_{ph}$。

3) 令节点集合 $V_{sp}=V_{sh}\bigcup V_{ph}$，将 $V_{sp}$ 的节点当作 T-score 方法中的候选可疑节点。

## 4.2 基于排序层的结论合成

在大部分情况下，不同模型所输出的可疑度序列对应的程序节点是不同的，也就是说，他们的度量标准不同。为此，必须对这些方法的输出进行度量同一的转化。所要结合的两个

模型分别是基于语句和谓词这 2 种不同程序节点。为了让两者有统一的度量标准，设计了一个从语句映射到谓词的工具 P-Mapper。

语句的 Tarantula 可疑值之所以不同是因为程序中存在大量的控制节点。在执行测试用例时，由于控制节点的存在，有些程序基本块被执行，有些程序基本块未被执行，这可能导致不同的基本块出现不同的 Tarantula 可疑值。因此，如果一个语句的可疑值比较高，则其控制节点出错的概率也相对其他控制节点出错的概率要高。为此， P-Mapper 制定了 3 个准则:

**准则1：** 对于条件分支语句，由于该语句本身就是一个谓词，所以它的可疑值就是自身的 Tarantula 可疑值。

**准则2：** 对于被程序节点控制的语句，则应该将其可疑值赋予该控制点谓词语句。

**准则3：** 对于程序剩下还没对应的语句，则检查这些语句是否存在与控制节点的数据流。如果存在数据流则把该语句的可疑值赋该控制节点。

对程序的各个语句同时应用以上准则，得到程序谓词的可疑值。如果同一个谓词语句同时存在多个可疑值，则选择值最大的可疑值。根据谓词的可疑值对程序中的谓词进行排序，得到谓词的排序序列。这里把由 Tarantula 方法得到的语句可疑值经映射工具 P-Mapper 得到谓词排序序列的方法记为 T-Predicate 方法。

为了更好地解释映射工具，以表 1 的错误 1 为例。对语句 7 应用规则 1，由于语句 7 原本就是一个谓词语句节点，所以语句 7 这谓词节点的可疑值就是其本身 Tarantula 生成的可疑值 0.67。语句 8 是受语句 7 控制，则应用规则 2 于语句 8，因此语句 8 由 Tarantula 生成的可疑值 0.8 赋给语句 7。语句 1 没被其他控制语句所控制，则应用规则 3 于语句 1 上，由于语句 1 与其他谓词节点没有数据流关系，所以，对语句 1 未做任何操作。经过 P-Mapper 处理后，就能得出错误 1 各谓词的可疑值为<0.67,0.67,0.8>。错误 2 中各谓词的可疑值为<0.67, 0.67, 0.8>。

一个谓词排序是指按照谓词的可疑度由低到高进行排序。对于每个排序，每个谓词都有相应的排序序号。这里规定把该排序序号看作为这个谓词在该排序中的权值。如果存在多个排序，那么每个谓词就对应着多个权值。这里把谓词的多个权值组成一个集合，并记为该谓词的权值集合。

图 3 所示为结合语句覆盖信息和谓词信息的错误定位流程。

1) 对语句覆盖信息应用 Tarantula 方法得到语句可疑值，并通过 P-Mapper 把语句可疑值转化为谓词可疑值，从而得到谓词的可疑度排名，记为 $Sort_s$。

2) 对谓词信息应用 SOBER 方法得到谓词可疑值，并得到谓词的可疑度的排名，记为 $Sort_p$。

3) 对得到的谓词可疑值排名 $Sort_s$ 和 $Sort_p$，应用排序层组合算法得到新的谓词可疑值排名，记为 $Sort_{mix}$。这里的组合算法将在 4.3 节讨论。

4) 根据 $Sort_{mix}$ 利用 T-score 方法对程序进行错误定位。

## 4.3 排序层的组合算法

基于结论合成的典型应用有最大规则、最小规则和加法规则等，结合这些典型的应用给出模型结合的组合算法：

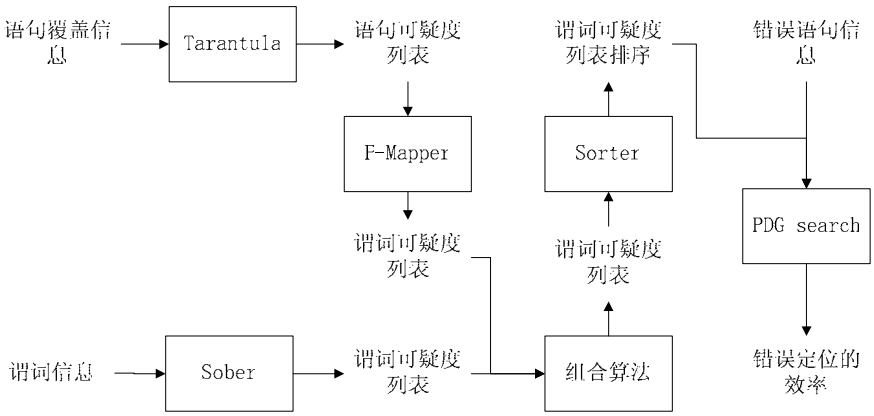



图 3 语句覆盖信息应用 P-Mapper 的多模型结合错误定位模型

Fig 3 The model process of fault localization with P-Mapper

1) 最大规则算法（Max-mix Rank）

最大规则算法是在每个谓词对应的权值集合中选取最大的一个权值赋给这个谓词，然后根据谓词的权值对谓词进行排序得出新的谓词排序。以表 1 的错误 2 为例，错误 1 的程序存在谓词组<$p_1$, $p_2$, $p_3$ >，其根据 SOBER 排序后得到相应的权值是<2, 3, 1>，而根据 P-Mapper 排序后得到相应的权值是<2, 1, 3>，则对于谓词 $p_2$ 权值集合是<3, 1>，故根据最大规则算法 $p_2$ 的权值应该为大的那个就是 3。同理，谓词组<$p_1$, $p_2$, $p_3$ >相应的权值是<2, 3, 3>。因此，得到新的谓词排序为 $p_2$, $p_3$, $p_1$。

2) 加法规则算法（Avg-max Rank）

加法规则算法是对每个谓词相对应权值集合的权值取平均值赋给这个谓词，然后根据谓词的权值对谓词进行排序得出新的谓词排序。以表 1 的错误 2 为例，错误 1 的程序存在谓词组<$p_1$, $p_2$, $p_3$>，其根据 SOBER 排序后得到相应的权值是<2, 3, 1>，而根据 P-Mapper 排序后得到相应的权值是<2, 1, 3>。则对于谓词 $p_2$ 权值集合是<3, 1>，则根据加法规则算法 $p_2$ 的权值应该为 2。同理，谓词组< $p_1$, $p_2$, $p_3$ >相应的权值是<2, 2, 2>。因此，得到新的谓词排序为 $p_1$, $p_2$, $p_3$。

## 5 实验

本实验采用西门子 132 个版本程序集[10]进行测试，其中有 6 个版本需要抛弃（其中有 2 个版本是运行测试用例集中时没有错误，其它 4 个版本是运行测试用例集时出现段错误不能收集覆盖信息）。西门子程序集一共包含 7 个单错误程序，分别为 print_tokens、print_tokens、replace、schedule、schedule2、tcas 和 tot_info。这些程序的基本信息如表 1 所示。

表 1 西门子程序集的基本信息

Table 1 The infomation of siemens suite

| 程序 | 描述 | 错误版本数 | 源代码行数 | 可执行代码行数 | 测试用例数 |
|---|---|---|---|---|---|
| print_tokens | 词法分析程序 | 7 | 565 | 204 | 4130 |
| print_tokens2 | 词法分析程序 | 10 | 510 | 202 | 4115 |
| replace | 模式代换 | 32 | 563 | 274 | 5542 |
| schedule | 优先级调度 | 9 | 412 | 166 | 2650 |
| schedule2 | 优先级调度 | 10 | 307 | 146 | 2710 |
| tcas | 高度区分 | 41 | 173 | 74 | 1608 |
| tot_info | 信息估量 | 23 | 406 | 139 | 1052 |

## 5.1 实验实现

在 llvm 的基础上分别实现了收集谓词信息和覆盖信息的插装工具。其中，收集的谓词信息主要包括程序的分支信息和函数返回信息，而不包含标量对信息。因为在文献 2 中提到不收集标量对信息对实验的效果影响不大，而且会大大降低实验的复杂度[2]。

## 5.2 评测标准

Renieris 和 Reiss[3]提出用一种度量方式（T-score）评价错误定位的效果。而该方法也被 Liu et 等[2]和 Cleve[11]在评价他们的方法中采用。

T-score 方法评测方法如下：首先，构建一个待测程序 $P$ 的程序状态图 $G(N,E)$，其中，$N$ 是程序语句的集合，而 $E$ 是存在数据依赖或者控制依赖关系的语句对集合；其次，把程序错误的语句节点记为错误节点，错误节点的集合记为 $V_{\text{defect}}$；接着，给定一组可疑语句的集合，其相应的节点标记为可疑节点，这些可疑节点的集合记为 $V_{\text{blamed}}$，它可能包括一个节点或者是多个节点；然后，程序员可以从可疑节点开始寻找错误并对 PDG 进行广度优先遍历直到达到错误节点。在 PDG 中遍历所得语句集的集合记为 $V_{\text{examined}}$。用评分函数 T 表示检查语句数所占的百分比。

$$T = \frac{|V_{\text{examined}}|}{|V|}$$

其中$|V|$是程序依赖图的节点数目；$|V_{\text{examined}}|$是 $V_{\text{examined}}$ 集合中的节点个数。实际上，这个公式是计算找出程序错误需要检查程序代码量的百分比。

一个高质量错误定位方法是希望检查少量的代码就能定位程序错误。如果一个方法需要检测的代码量越少，则它的错误定位效果越高。T-score 方法是以程序依赖图为基础来评价错误定位的效率。该方法能客观评价与错误相关谓词序列的质量和错误定位的效果，同时，由于其他一些谓词的方法（SOBER 和 CBI）都使用 T-score 方法作为评价标准的，所以这里也使用 T-score 方法。

## 5.3 实验分析和比较

由于本实验是针对谓词的统计方法，与以 Tarantula 方法为代表的基于语句方法的评价标准不太相同，所以效果比较主要针对基于谓词的 SOBER 方法和 CBI 方法。

在这一节当中，用标签 SOBER 和 CBI 分别代表 SOBER 方法和 CBI 方法，T_Predicate 代表 T-Predicate 方法，Simple_mix 代表简单结合方法，而 Max_mix 和 Avg_mix 分别代表排序层结合中 Max-mix Rank 方法和 Avg-mix Rank 方法，如图 4 所示，图中的横坐标表示定位程序错误时需检测的语句数占总语句数的百分比，纵坐标表示找到错误的版本个数占总错误版本个数的百分比。

为了更全面地评估一个方法的效率，首先应比较各方法应用在西门子程序集的平均效果，如图 4 所示。

图 4 给出的是简单结合方法、Max-mix Rank 方法、Avg-mix Rank 方法、T_Predicate 方法、SOBER 和 CBI 这 6 种方法的总体性能效果的比较。可以看出，简单结合方法和 Max-mix Rank 方法都比原方法的性能要好。在所有版本中，简单结合方法和 Max-mix Rank 方法只要检查平均 10%的代码就能分别找到 53.08%和 54.03%的错误，超过一半的错误均能检测出来。

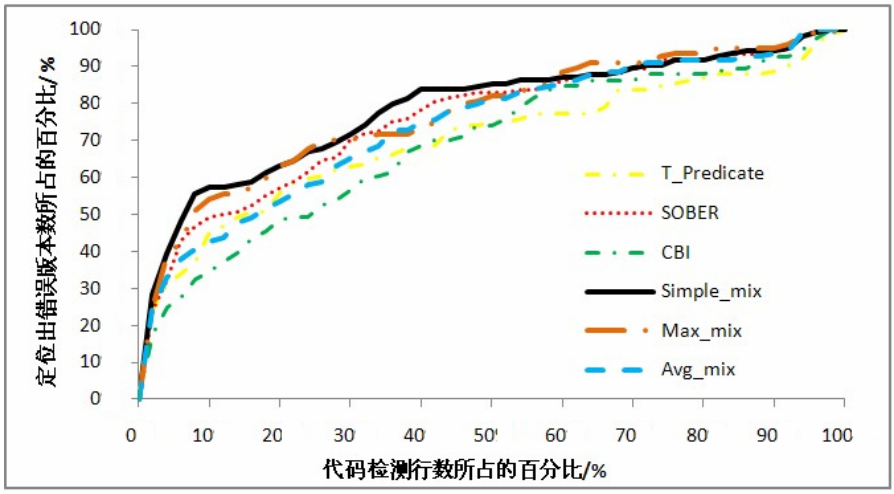



图 4 应用西门子程序集时 6 种方法的总体效果
Fig 4 The six methods effectiveness comparison in Siemens suite

图 5 给出了 6 种方法检查前 20%代码的结果比较，显示了如果只检查 2%的代码量，组合方法 Simple_mix Rank、Max_mix Rank 和 Avg-mix Rank 分别定位了 28.46%、24.19%和 24.19%的错误。而 SOBER、CBI 和 T_Predicate 分别定位了 24.19%、24.19%和 25.38%的错误。当检查 10%和 20%的代码时，Simple_mix Rank 能定位到 53.66%和 61.79%的错误；Max_mix Rank 能定位到 54.03%和 62.90%的错误；Avg_mix Rank 能定位到 42.74%和 53.23%的错误；SOBER 能定位到 49.19%和 57.23%的错误；CBI 能定位到 34.68%和 48.39%的错误；T_Predicate 能定位到 45.16%和 55.65%的错误。由此可以看出，在检查 2%~20%的代码时，组合方法 Simple_mix Rank 和 Max_mix Rank 比集成前单一方法（SOBER 和 T_Predicate 方法）的定位效果都要好。但是，有些组合方法的错误定位效果却不让人满意，如 Avg-mix Rank 方法。

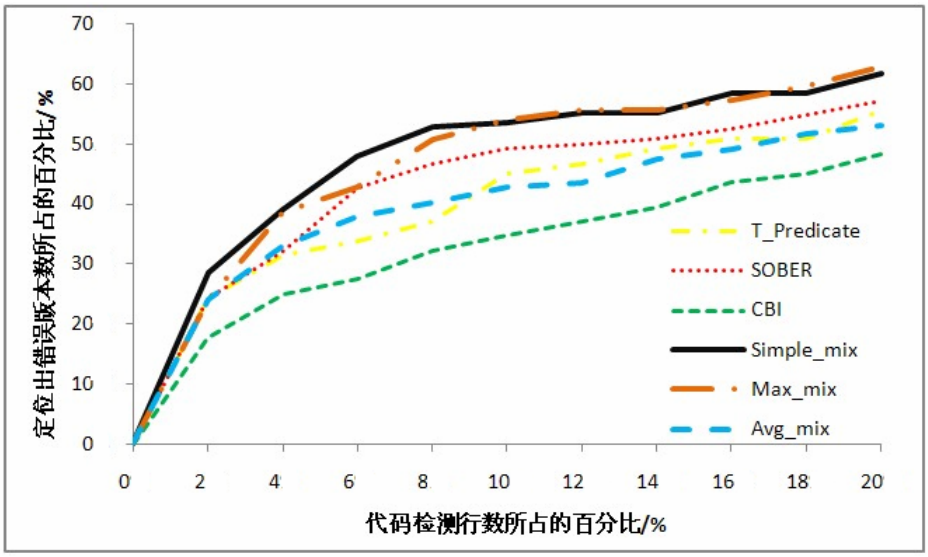
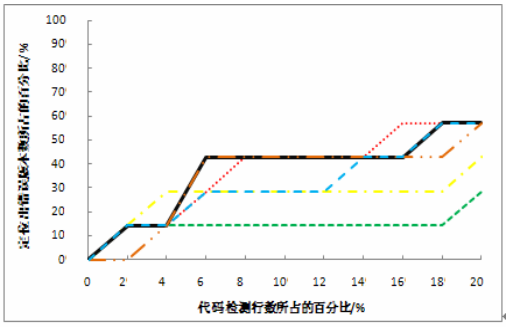


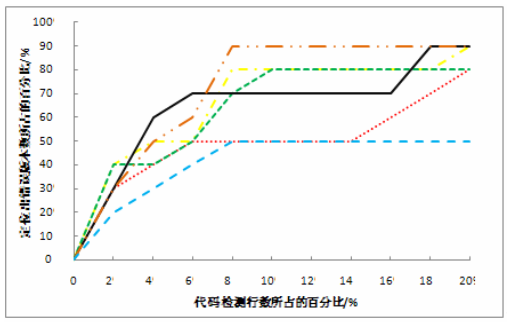

图 5 6 种方法在检查 0%~20%代码时的效果
Fig 5 Six methods results comparison in range of [0%, 20%]

图 6 分别给出了 6 种方法在 7 个程序中检查前 20%代码的结果比较。可以看出，SOBER 和 T_Predicate 方法不够稳定。对于有些程序，SOBER 方法的错误定位效果是最好的，而有些方法的效果却是最差的。T_Predicate 方法也存在这些情况。而结合算法 Simple_mix Rank

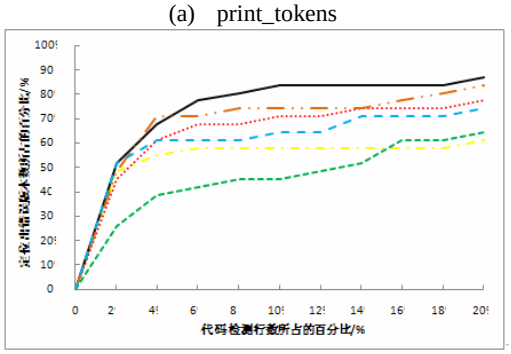
和 Max_mix Rank 方法总比其中一个集成前方法的定位效果要好。这说明对于西门子程序集，适当集成后的方法要比集成前的方法要稳定，更适应新的需检测的程序。



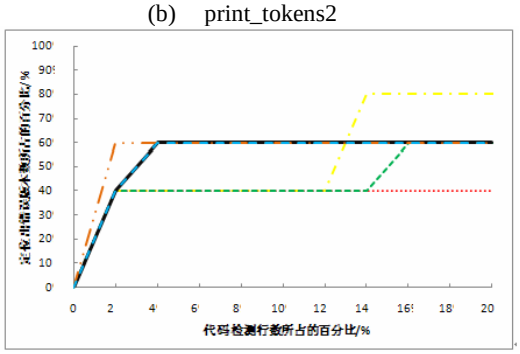

(a) print_tokens



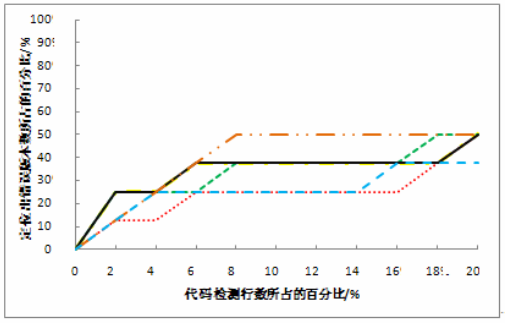

(b) print_tokens2



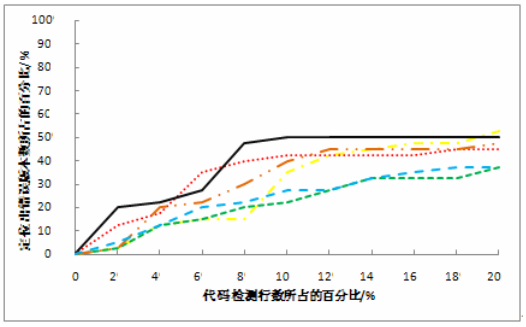

(c) replace



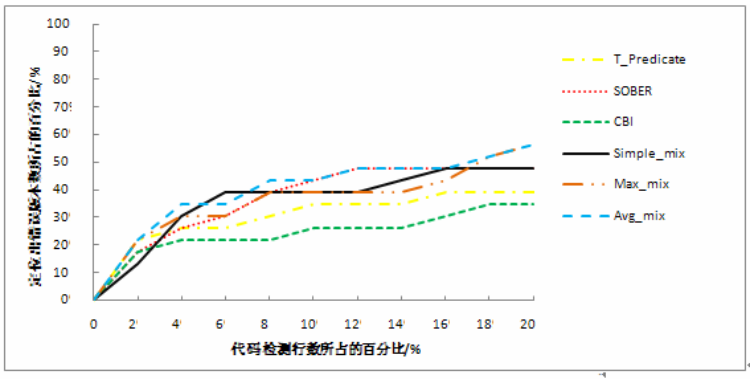

(d) schedule




(e) schedule2




(f) tcas




(g) tot_info

图 6 六种方法在西门子七个程序应用时的性能

Fig 6 Six methods effectiveness comparison on individual programs in Siemens suite

## 5.4 实验讨论

基于上述结果，针对西门子程序集得到以下结论：

1) 这里提出的方法能使度量标准不同基于语句信息和基于谓词信息的2种方法相结合，并且有比较好的错误定位效果。

2)，没有一个方法在任何情况下得到的错误定位结果是最好的。每种错误类型都有更合适的建模方法。

3) 经过适当组合的方法总体上比原方法的效果要好且稳定。

4) 有些组合方法不但不能提高错误定位的效果，反而会降低其检测错误的效果。

## 6 结 语

基于语句覆盖信息的错误定位方法和基于谓词信息的错误定位方法在实践中取得了比较好的效果。但是，针对任意类型的程序错误，基于单一程序节点类型的错误定位方法其效果不总是最佳的。为此，引入集成学习的思想，并提出模型结合的错误定位方法。该思想是通过选择几个错误定位模型，对其输出的结果进行结论合成，从而提高错误定位模型上的泛化能力和错误定位的效率。基于该思想，选取了 Tarantula 和 SOBER 方法这 2 个方法进行组合，并提出了 3 个结论合成的方法：Simple_mix Rank、Max_mix Rank 和 Avg_mix Rank 方法。实验表明，组合算法 Simple_mix Rank 与 Max_mix Rank 方法相比，集成前单一的算法有更好的错误定位效果和更强的适应性。

### 参考文献(References)

[1] Jones J A, Harrold M J. Empirical evaluation of the Tarantula automatic fault-localization technique [C]// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005). New York: ACM Press, 2005: 273–282.

[2] Liu C, Yan X, Fei L, et al. Midkiff. SOBER: statistical model-based bug localization [C]// Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundation of Software Engineering (ESEC 2005/FSE-13). New York: ACM Press, 2005: 342–351.

[3] Renieres M, Reiss S P. Fault localization with nearest neighbor queries [C]//Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003). San Franciso: IEEE Computer Society Press, 2003: 30–39.

[4] Santelices R, Jones J A, Yu Y, et al. Lightweight fault-localization using multiple coverage types [C]// Proceedings of the 31st International Conference on Software Engineering (ICSE 2009). San Franciso: IEEE Computer Society Press, 2009: 56–66.

[5] Dietterich T G. Machine-learning research [J]. AI Magazine, 1997,18(4): 97–136

[6] Liblit B, Aiken A, Zheng A X, et al. Bug isolation via remote program sampling [C]// Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003). New York: ACM Press, 2003: 141–154.

[7] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation [C]// Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005). New York: ACM Press, 2005:15–26.

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

[9] Xu L, Krzyzak A, Suen C Y. Method of combining multiple classifiers and their applications to handwriting recognition [J]. IEEE Trans. Systems, Man, and Cybernetics, 1992, 22(3):418-435.

[10] Siemens suite [OL] [2006], http://sir.unl.edu/portal/index.html

[11] Cleve H, Zeller A. Locating causes of program failures [C]// Proceedings of the 27th International Conference on Software Engineering (ICSE 2005). San Franciso: IEEE Computer Society Press, 2005: 342-351.