doi:10.11959/j.issn.1000-436x.2015095

基于参数–值替换的错误定位方法

王兴亚[1]，姜淑娟[1]，鞠小林[1,2]，曹鹤玲[1]
(1. 中国矿业大学 计算机科学与技术学院, 江苏 徐州 221116；2. 南通大学 计算机科学与技术学院, 江苏 南通 226019)

摘 要：现有错误定位方法难以检测程序遗漏错误，提出一种方法层次的基于参数–值替换的错误定位方法。首先将出现在失败执行中的方法作为可疑方法候选集，计算该集合中元素对程序执行结果的影响度，再利用值替换技术计算高影响度方法的兴趣参数–值映射对，然后依据方法是否包含兴趣参数–值映射对对候选集中方法进行分类，并依据其影响度分别进行排序，最后生成一个可疑方法序列进行错误定位。实验结果表明该方法较其他错误定位方法具有更好的定位效果。

关键词：错误定位；遗漏错误；值替换；兴趣参数–值映射对
中图分类号：TP311 **文献标识码**：A

Effective fault localization technique using parameter-value replacement

WANG Xing-ya[1], JIANG Shu-juan[1], JU Xiao-lin[1, 2], CAO He-ling[1]
(1. School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China;
2. School of Computer Science and Technology, Nantong University, Nantong 226019, China)

Abstract: Statistic-based fault localization and slice-based fault localization cannot locate the omission faults effectively. A method-level fault localization approach based on parameter-value replacement was proposed. Those methods appearing in failed executions were treated as the fault methods candidate set (FMCS). For each element contained in FMCS, its impact to the execution result was measured and then the FMCS elements based on the impact were classified. For each method with high impact, its interesting parameter value mapping pair (IPVMP) was searched by using parameter-value replacement. According to the impact and IPVMP, a method list of FMCS elements was finally provided to debuggers. Empirical results show that the proposed approach performs better than other fault localization approaches.

Key words: fault localization; omission error; value replacement; interesting parameter-value mapping pair

1 引言

程序调试是软件开发过程中的重要步骤。当程序执行失败时，开发人员需要进行程序调试来定位并修正错误。纯手工进行程序调试十分耗时耗力，自动化调试技术可以有效地减轻开发人员的负担，并提高错误被修正的可能性，从而保证软件具有更高的可靠性和顽健性[1]。因此，针对程序调试自动化技术的研究具有重要意义。错误定位是程序调试的第一步，研究人员对其开展了广泛而深入的研究，并取得了一些研究成果，其中主要包括基于统计技术的错误定位方法[2~6]以及基于程序切片的错误定位方法[7~11]。

基于统计技术的错误定位方法通过搜集和比

收稿日期：2014-01-18；**修回日期**：2014-04-08
基金项目：国家自然科学基金资助项目(61202006, 60970032)；江苏省 333 工程基金资助项目；江苏省高校自然科学研究基金资助项目(12KJB520014)；江苏省研究生培养创新工程基金资助项目(CXZZ12_0935, KYLX_1390)；南通市应用研究计划基金资助项目(BK2011025, BK2012023)
Foundation Items: The National Natural Science Foundation (61202006, 60970032); The 333 Project of Jiangsu Province; The University Natural Science Research Project of Jiangsu Province (12KJB520014); The Graduate Training Innovative Projects Foundation of Jiangsu Province (CXZZ12-0935, KYLX_1390); The Nantong Application Research Plan (BK2011025, BK2012023)

较程序在失败执行和成功执行中的状态信息来对程序实体(如语句、谓词、方法等)进行怀疑度计算及排名来完成错误定位[12]。然而，当程序执行失败是由本该执行的语句未执行引起时(如遗漏执行错误[13]、语句遗漏错误[14])，错误语句没有包含在执行轨迹中，因而难以定位此类错误。基于程序切片的错误定位方法通过语句间的依赖关系找到与状态异常语句相关的语句集，并以此为开发人员提供一个可理解的上下文环境进行错误定位[11]。当程序发生遗漏执行错误时，通过语句间潜在的相关依赖关系可以找到错误语句所在位置[13]。当程序发生语句遗漏错误时，通过依赖关系无法找到遗漏语句，导致该类方法无法适用于语句遗漏错误的定位。

一般认为，程序执行失败是由执行了错误的程序实体导致的[15]。开发人员在进行错误定位时，通常基于完美错误检测假设，检测到错误程序实体时即可发现错误[16]。然而在实际定位过程中，开发人员往往需要阅读和理解异常状态的上下文信息来理解和定位错误[17]。为了可以定位程序遗漏错误，同时为开发人员提供良好的定位信息，将错误定位的粒度设为方法。

值替换方法[9, 18]通过修改程序运行时的状态信息，如谓词输出、语句变量值等，找到影响程序执行输出的程序实体供开发人员进行错误定位。但是当程序复杂且执行周期比较长时，对谓词输出、语句变量值进行替换会带来很大的开销。同时，这些方法难以完成程序遗漏错误的定位。选择程序中的方法作为研究对象，通过替换方法的参数信息找到影响程序执行结果的方法。这不仅可以降低替换对象的数目，同时还可以进行程序遗漏错误的定位以及为开发人员提供一个良好的定位上下文环境。

本文提出一种基于参数–值替换的错误定位方法: 该方法将所有失败执行中出现的方法作为可疑方法候选集，对其中的每一个元素根据其在成功执行和失败执行时参数输入的差异计算元素对程序执行结果的影响度，接着利用参数–值替换方法检测该元素是否包含兴趣参数–值映射对，之后按照影响度以及是否包含兴趣参数–值映射对对可疑方法候选集中的元素进行分类和排序，最终生成一个可疑方法序列协助开发人员完成错误定位工作。

本文贡献如下：

1) 分析说明了现有错误定位方法在定位程序遗漏错误时的局限性，提出一种基于参数–值替换的错误定位方法；

2) 设计实现了本文提出的错误定位方法，与一组错误定位方法展开对比验证本文方法的有效性。

2 研究动机

本节以一个代码片段为例，说明基于统计技术的错误定位方法和基于程序切片的错误定位方法存在的问题，并分析问题产生的原因，进而引出本文错误定位方法。

图 1 是 SIR 数据集[19]中 NanoXML 程序的代码片段，该段代码用于对 XML 文档的开始元素进行解析。为了避免同时使用多个 XML 文档时发生命名冲突，开发人员通常为 XML 的开始元素设置命名空间并为其添加前缀，从而使解析器能够区分不同 XML 文档中具有相同名称的元素。NanoXML 在对 XML 文档开始元素进行解析时完成元素的全命名工作，该工作由类 net.n3.nanoxml.StdXMLBuilder 中的 startElement 方法完成。当 XML 文件解析完成后，程序调用类 net.n3.nanoxml.XMLWriter 中的 write 方法完成输出功能。

对 XML 开始元素进行解析时，如果参数 nsPrefix 为空，说明待解析的 XML 文件没有设置命名空间前缀，采用默认命名空间；否则，说明需要为该 XML 文档中的元素添加前缀，对临时变量 fullName 进行修改。由于程序遗漏了语句 2 和语句

StdXMLBuilder.java

```
public void startElement(String name, String
        nsPrefix, String nsURI) {
1       String fullNamc = namc;
2   //  if (nsPrefix != null)
3   //      fullName = nsPrefix + ':' + name;

4       this.stack.push(new XMLElement(name,
                fullName, nsURI));
        ...
    }
```

XMLWriter.java

```
    public void write(XMLElement xml){
5       if (xml.getName().equals(xml.getFullName()))
6           this.writer.print("xmlns=\"" +
                    xml.getNamespace() + '"');
        else {
7           String prefix = xml.getFullName();
8           prefix=prefix.substr(0,prefix.indexOf(':'));
9           this.writer.print(" xmlns:"+prefix+
                "=\""+xml.getNamespace()+"\"");
    } ... }
```

图 1 NanoXML 代码片段

3，startElement 方法没有对变量 fullName 进行判断。此时，无论 nPrefix 是否为空，变量 fullName 都不会被修改。在 XML 输出时，方法 write 通过比较 name 与 fullName 的值是否相等来判断 XML 中元素是否包含前缀，若相等，则说明元素不包含前缀，否则元素包含前缀。由于 fullName 未做修改，导致其在任何时候都与 name 相等。随机抽取 SIR 提供的测试用例进行测试，收集方法的运行时状态和执行结果。部分数据如表 1 所示，当 nPrefix 值为 null 时，fullName 值无需进行修改，程序实际输出与期望输出相符，程序执行成功；当 nPrefix 值不为 null 时，程序实际输出与期望输出相悖，程序执行失败。

使用基于统计技术的方法对该程序进行错误定位时，如果定位粒度为语句，可以得到语句 1、4、5、6 的覆盖信息，无法得到语句 2 和语句 3 的覆盖信息，因此不能定位出错误语句；如果定位粒度为方法，可以看到程序在失败执行和成功执行中都执行过方法 startElement，错误定位的精度不高。使用基于程序切片的错误定位方法对该程序进行错误定位时，首先找到状态异常语句 6，然后找到与语句 6 有依赖关系的语句作为可疑语句候选集协助开发人员进行调试。但是由于程序遗漏了语句 2 和语句 3，与语句 6 存在依赖关系语句集合中只包含语句 1、4、5 和 6。因此，开发人员无法在可疑语句候选集中找到错误。

为了能够定位程序遗漏错误，选择程序中的方法作为定位对象。失败执行和成功执行中的方法覆盖信息并没有反映出方法与执行结果的关系，不能有效地用于错误定位。方法的执行行为(如方法内语句覆盖、方法输出等)受到方法输入的影响，同时方法的执行行为能够影响程序的执行结果，因此方法输入的不同可能会导致程序执行结果的不同。参数输入是方法最基本的输入。在程序执行过程中，替换方法的参数输入，判断程序的执行结果是否发生变化可以判断方法是否对程序执行结果产生影响，以此进行错误定位。从上面分析可知，方法在成功执行和失败执行时方法参数输入的差异可以反映该方法对程序执行结果的影响。由此，本文提出一种基于参数−值替换的错误定位方法。

3　本文方法

本节首先描述了错误定位过程中使用的基本概念以及兴趣参数−值映射对的计算方法，在此基础上，提出基于参数−值替换的错误定位方法。

3.1　基本概念

给定一个方法执行实例，该实例的参数−值映射反映了当前执行上下文中该方法所有参数与其值的一一映射关系。

定义 1　参数−值映射(PVM, parameter-value mapping)。给定一个方法执行实例 m_i，其参数序列 $\text{params}(m_i) = \{a_j \mid j \in [1, n]\}$，参数值序列 $\text{values}(m_i) = \{v_j \mid j \in [1, n]\}$，则 m_i 的参数−值映射 $\text{PVM}(m_i) = \{(a_j, v_j) \mid j \in [1, n]\}$。

在程序一次执行过程中，同一个方法可能会被多次调用，且每次调用时方法的参数值可能会不同。因此程序一次执行过程中可能会产生多个关于方法 m 的参数−值映射。用 $\text{PVM}(t, m)$表示方法 m 在给定测试用例 t 时出现的所有参数−值映射。

在程序失败执行时，对于其中的一个方法执行实例，若存在一个参数−值映射，替换该方法执行实例的原始参数−值映射后，程序由失败执行变为成功执行，称该实例包含兴趣参数−值映射对。否则，该实例不包含兴趣参数−值映射对。

定义 2　兴趣参数−值映射对(IPVMP, interesting parameter-value mapping pair)。给定一个方法执行实例 m_i，关于该实例的兴趣参数−值映射对 $\text{IPVMP}(m_i)$是一个参数−值映射二元组(*origPVM*, *altPVM*)。其中 *origPVM* 是 m_i 的初始参数−值映射，*altPVM* 是待替换的参数−值映射。将 *origPVM* 替换为 *altPVM* 后，程序由失败执行变为成功执行。

表 1　测试用例及测试结果

测试用例	参数输入	预计输出	实际输出	结果
A	startElement("FOO", null, null)write(non-null)	“xmlns=”	“xmlns=”	passed
B	startElement("Bar", null, "http://.../bar") write(non-null)	"xmlns=http://…/bar"	"xmlns=http://…/bar"	passed
C	startElement("template", "xsl", "http://.../trans") write(non-null)	"xmlns:xsl=http://…/trans	"xmlns=http://…/trans	failed
D	startElement("Bar", "ns", "http://.../bar") write(non-null)	"xmlns:ns=http://…/bar	"xmlns=http://…/bar	failed

不同的方法输入会导致程序产生不同的执行结果，如表 1 所示，参数 nsPrefix 值为空时程序正确执行，值为非空时会触发方法 startElement 中的错误导致程序执行失败。如果同一方法在成功执行和失败执行中的输入不同，说明该方法在不同执行中的行为有差异，该差异可能影响程序的执行结果。输入差异越大表明该方法对程序执行结果影响越大，包含错误的可能性越高。参数输入是程序执行中接收的最基本参数，本文记录方法在成功执行和失败执行中的参数输入信息并将两者的差异作为该方法对执行结果的影响度。

在数据分析过程中，经常需要计算个体间差异的大小从而来评价个体的差异性。Jaccard 距离是衡量 2 个集合区分度的一种指标，它用 2 个集合中不同元素占所有元素的比例来衡量 2 个集合的区分度。使用 Jaccard 距离来表示成功执行参数−值映射集与失败执行参数−值映射集的区分度，并将其作为方法对程序执行结果的影响度。其中成功执行参数−值映射集(PPVM, passed parameter-value mappings)表示方法在成功执行中出现的参数−值映射集合，失败执行参数−值映射集(FPVM, failed parameter-value mappings)表示方法在失败执行中出现的参数−值映射集合。

定义 3 影响度(*impact*)。表示方法对程序执行结果的影响程度。给定方法 *m* 及其成功执行参数−值映射集 PPVM(*m*)和失败执行参数−值映射集 FPVM(*m*)，方法影响度 *impact*(*m*)采用 Jaccard 距离来计算，计算公式如下。

$$impact(m)=1-\frac{\left|\mathrm{PPVM}(m)\cap\mathrm{FPVM}(m)\right|}{\left|\mathrm{PPVM}(m)\cap\mathrm{FPVM}(m)\right|} \quad (1)$$

3.2 参数−值替换技术

给定程序的一次失败执行，依次考察每一个方法执行实例，若能找到一个参数−值映射，用它替换该执行实例的初始参数−值映射后，程序执行变为成功执行，则说明该实例包含兴趣参数−值映射对。否则，该实例不包含兴趣参数−值映射对。处理完成所有的执行实例后，该次执行中所有的兴趣参数−值映射对 IPVMP 查找完成。算法 1 给出 IPVMP 生成算法。

算法 1 IPVMP 生成算法

输入

P: 源程序

f: 程序执行结果为失败的测试用例

T: 测试用例集

输出

IPVMP: 兴趣参数−值映射对集

步骤

开始

1) 对于测试用例集 T 中的每一个测试用例 *t*
2) 记录程序 P 在给定测试用例 *t* 时执行产生的参数−值映射轨迹 $trace_t$
3) 如果程序执行成功
4) 将 $trace_t$ 中的元素记录到相应方法的成功执行参数−值映射集 PPVM
5) 否则
6) 将 $trace_t$ 中的元素记录到相应方法的失败执行参数−值映射集 FPVM
7) 返回步骤 1)
8) 将 P 在给定测试用例 *f* 时执行产生的参数−值映射轨迹记为 $trace_f$
9) 对于 $trace_f$ 中的每一个方法实例 *i*
10) 将方法实例 *i* 对应的方法记为 *m*
11) 如果 IPVMP 中包含关于 *m* 的兴趣参数−值映射对
12) 返回步骤 9);
13) 将 *i* 处的参数−值映射记为初始参数−值映射 *origPVM*
14) 如果 PPVM(*m*)为空
15) 将 IPVMP(*origPVM*, null) 添加到 IPVMP 中
16) 对于 PPVM 中的每一个元素 *altPVM*
17) 如果 *altPVM* ∈ FPVM
18) 返回步骤 16);
19) 根据测试用例 *f* 执行程序 P 并在 *i* 处将 *altPVM* 替换为 *origPVM*
20) 如果程序执行成功
21) 将 IPVMP(*origPVM*, *altPVM*) 添加到 IPVMP 中;
22) 返回步骤 9);
23) 返回步骤 16)
24) 返回步骤 9)

结束

生成 IPVMP 需要程序的一次失败执行以及与失败执行相关的期望输出及实际输出。根据测试用例集 *T* 执行程序，收集程序中方法的成功执行

参数−值映射集信息和失败执行参数−值映射集信息。收集完成后，选择一个失败测试用例 f 驱动程序 P 执行，记录下该次执行产生的所有执行实例 $trace_f$，用 m 表示执行实例 i 对应的方法。行 11、行 12 判断替换的方法是否已经包含相应的兴趣参数−值映射对。如果包含，说明在之前的替换中已经找到该方法的兴趣参数−值映射对，继续分析下一个方法实例；反之，说明还未找到该方法的兴趣参数−值映射对，需要继续对该方法进行分析。行 14、行 15 判断 m 的可替换参数−值映射集合 PPVM(m)是否为空。若 PPVM(m)为空，说明 m 仅在程序失败执行中出现过。此时，可以认为该方法出错的可能性大，并将(*origPVM*, null)作为当前实例的 IPVMP 记录下来。依次选择 PPVM(m)中的元素作为替换参数−值映射 *altPVM*。行 17、行 18 判断 *altPVM* 是否属于 FPVM(m)，若属于，说明 *altPVM* 在失败执行中出现过，不使用其进行替换操作；否则，说明 *altPVM* 仅在成功执行中出现过。行 19 至行 22 在完成参数−值替换后，检测程序执行结果。若执行变为成功，则说明(*origPVM*, *altPVM*)是该实例的 IPVMP，记录到 IPVMP 中。所有执行实例处理完成后，该次执行的参数−值映射集 IPVMP 生成。

在 IPVMP 生成过程中，需要记录方法的参数−值映射信息，以及在一次失败执行中对每个方法执行实例进行参数−值替换并记录程序执行结果。如果程序在不同的执行过程中没有占用相同的系统资源，则可以使用多线程技术来提高 IPVMP 的生成效率。可以从以下 3 个方面使用多线程技术：1) 单独执行每个测试用例以记录方法的参数−值映射信息；2) 在一次失败执行中，单独测试每个方法执行实例并判断其是否包含兴趣参数−值映射对；3) 对执行实例的每个可替换参数−值映射，单独进行替换并判断其是否能够组成兴趣参数−值映射对。

理论上，方法执行实例的可替换参数−值映射数目是无限的,需要能够使失败执行变为成功执行的 PVM，因此需要一个数目有限的且有效的可替换参数−值映射集合。由于成功执行中出现的 PVM 没有导致程序执行失败，因此将在成功执行中出现的值映射作为可替换参数−值映射。

在某些情况下，失败执行的期望输出是未知的。对此，Jeffrey 等[18]认为与替换前失败执行的输出结果相比，替换后的输出结果更接近期望结果即可说明该方法包含 IPVMP。例如，程序发生异常导致程序执行失败，如果替换后程序执行不再发生异常，此时即可认为程序实际输出得到改善，进而说明该方法包含 IPVMP。

3.3　错误定位方法

监控程序的执行情况，记录程序中方法在不同执行中的参数输入信息，并将这些信息作为下一步定位分析时的基础。本文方法框架如图 2 所示。

第 1 步　基于一组测试用例执行源程序，监控并记录程序的执行信息，记录程序的执行结果以及程序执行时每个方法的参数输入信息。当测试用例执行结束后，记录所有失败执行中出现的方法，并将其作为可疑方法候选集(FMCS, fault method candidate set)。

第 2 步　根据第 1 步收集的参数输入信息，对 FMCS 中的元素进行影响度计算。

第 3 步　根据第 2 步的分析结果对 FMCS 的元素进行分类。设置一个影响度阈值 t(threshold)，按照影响度是否高于 t 将 FMCS 中的元素分为 2 类，其中影响度高于 t 的元素组合生成高影响度方法集(HIM, high impact method)，影响度不高于 t 的元素组合生成低影响度方法集(LIM, low impact method)。阈值 t 的值在不同执行情况下不同，本文选择 0 作为 t 的值。

第 4 步　选择一次失败执行，对包含在 HIM 中的方法进行参数−值替换，计算该次执行中所有的 IPVMP。根据方法是否包含 IPVMP 对 HIM 中元素进行分类，生成包含 IPVMP 的方法集(IM, IPVMP method) 和不包含 IPVMP 的方法集(NIM, non-IPVMP method)。

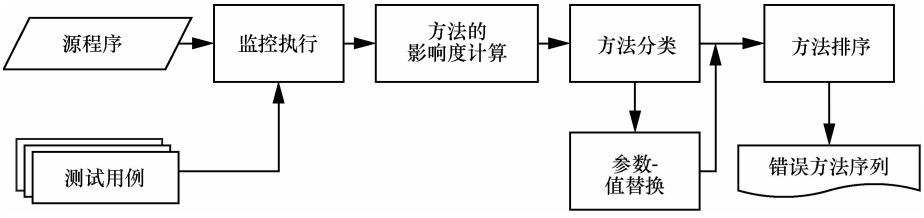


图 2　本文方法框架

第 5 步 对第 3 步生成的 LIM 和第 4 步生成的 IM 以及 NIM 按照影响度分别进行降序排序，同时为 IM 赋予最高优先级，NIM 次之，LIM 最低，组合生成一个错误方法序列为程序错误定位提供帮助。

基于上述框架，利用 Java 平台调试体系结构 JPDA 提供的 Java 调试接口实现了一个错误定位工具。该工具监听并记录程序的执行情况，当 MethodEntry 事件发生时，记录与该事件相关的方法及方法参数输入生成 PVM。一次执行结束，该次执行中所有的 PVM 信息记录下来。测试用例执行完成后，所有的 PVM 信息记录下来。根据 PVM 信息完成方法的影响度计算、分类、排序等工作，并以此完成错误定位工作。

4 实验

为了验证本文方法的有效性，设计并实现了本文提出的错误定位方法，并在一组开源程序上开展错误定位实验。

4.1 实验对象

本文选取 SIR 网站提供的 NanoXML 和 Siena 程序作为研究对象[19]，其中 NanoXML 是一个 XML 解析器，Siena 是一个发布/订阅系统。与 Masri 等[2]的工作一样，在实验中去除了不能被测试用例触发错误的版本，且只处理单错误的故障定位，共处理错误版本 25 个。

表 2 给出程序版本、程序的方法数目、语句数目、测试用例数目、错误数目和语句遗漏错误数目。可以看到本文研究的 25 个错误对象有 9 个是语句遗漏错误。实验目的是验证本文方法是否能有效的定位程序错误所在位置。

表 2 实验对象

版本	方法数	语句数	测试用例数	错误数	遗漏错误数
NanoXML v1	201	4 351	167	4	0
NanoXML v3	354	6 838	191	8	5
NanoXML v5	370	7 160	205	6	3
Siena v1	198	8 295	560	2	0
Siena v2	198	8 296	565	1	0
Siena v5	209	8 524	566	2	1
Siena v6	209	8 498	565	1	0
Siena v7	205	8 472	565	1	0

4.2 实验设计

选择 Tarantula[3]、Ochiai[4]、Naish1[5]和 Wong1[6] 4 种错误定位方法与本文方法进行对比实验。其中 Tarantula 和 Ochiai 是基于语句覆盖信息的经典的错误定位方法，Naish1 和 Wong1 则是分别选自谢等[20]理论证明最优的 2 组怀疑度公式。使用 EXAM 指标来评判错误定位代价，EXAM 指标表示定位到错误时需要检查的程序实体数占可执行程序实体数的百分比。

NanoXML 程序在不同执行中没有占用同一系统资源，因此可以按照 3.2 节的方法使用多线程技术来提高错误定位效率。而 Siena 程序在测试过程中需要占用 7 000 和 2 000 这 2 个端口，因此无法使用多线程技术来提高定位效率。

4.3 实验结果分析

实验结果如图 3 所示。横坐标表示需要检查的代码行数占总代码行数的百分比，纵坐标表示发现的缺陷版本数占总版本数的百分比。当横坐标值相同时，纵坐标值越大表明定位效果越好。图 3 中的 5 条曲线分别给出 Tarantula、Ochiai、Naish1、Wong1 以及本文方法(IPVMP)的错误定位效果。

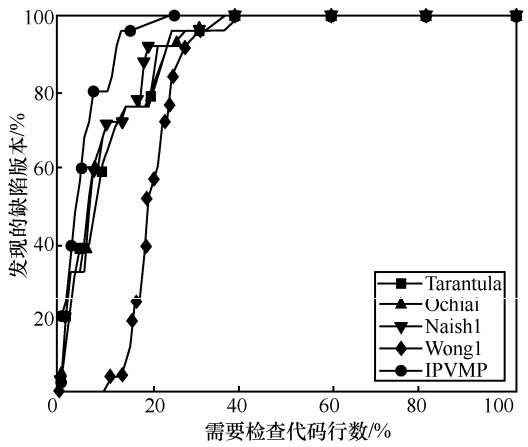


图 3 多种方法的错误定位效果[0%, 100%]

由图 3 可以看出，本文方法一直具有良好的定位效果，并始终领先于其他 4 种方法。当代码检查率达到 17%时，本文方法可以找到程序中全部错误；达到 38%时，其他 4 种方法可以找到程序中全部错误。图 3 给出的效率曲线也展示了本文方法具有较高的错误定位效率。

在错误定位过程中，当定位代价超过 20%时，错误定位结果并不能显著提高调试效率，说明此次错误定位过程是没有意义的[21]。如图 4 所示，比较本文方法与 4 种对比方法在 0%~20%段的错误定位效果。可以发现通过检查程序前 20%的代码，本文

方法可以成功定位所有的错误，而其他 4 种方法分别可以定位到 76%、76%、92%和 52%的错误，这说明本文方法的错误定位代价比较合理，能够提供有效的错误定位信息。

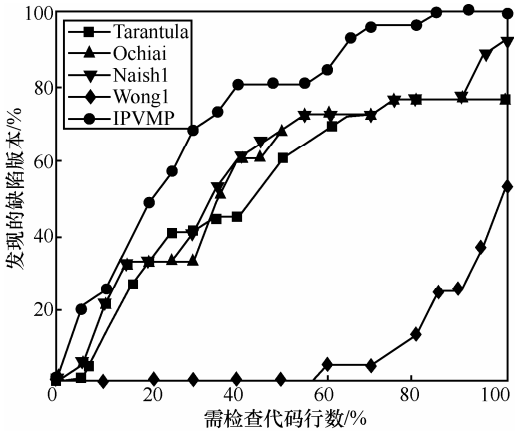


图 4　多种方法的错误定位效果[0%, 20%]

表 3 给出实验的具体对比数据。列 1 和列 2 分别说明了程序的版本以及错误的版本号，列 3~列 7 分别给出 Tarantula、Ochiai、Naish1、Wong1 这 4 种错误定位方法以及本文方法在方法层次的错误定位代价。列 8 表示错误方法是否包含兴趣参数–值映射对。可以看到，在 25 个对比实验中，本文方法在 17 个对比实验中定位效果最好。同时，有 18 个错误方法包含兴趣参数–值映射对。

表 4 给出本文方法与 4 种对比方法在错误定位代价上的最大值、最小值、中位数、平均值、标准差，通过这 5 个指标对实验结果作进一步分析。可以看到，在这 5 个指标中，本文方法均优于其他 4 种方法，因此可以看出本文方法的实验效果更好，稳定性更高。

表 4　　方法有效性对比

指标	Tarantula	Ochiai	Naish1	Wong1	IPVMP
最大值	37.31	35.82	35.32	35.82	16.42
最小值	1.13	0.28	0.28	11.62	0.27
中位数	8.61	7.30	6.76	19.77	4.32
平均值	10.84	10.01	9.34	21.22	5.40
标准差	9.33	9.14	8.57	5.25	4.50

5　相关工作

基于状态替换的错误定位方法是一种有效的

表 3　　实验结果

程序版本	错误版本	Tarantula	Ochiai	Naish1	Wong1	IPVMP	contains
NanoXML v1	F_NV_HD_1	4.98	7.46	7.46	21.89	5.97	yes
NanoXML v1	F_NV_HD_2	2.99	2.99	2.99	27.86	12.94	no
NanoXML v1	F_NV_HD_4	37.31	35.82	35.32	35.82	0.5	yes
NanoXML v1	F_SP_HD_3	27.36	27.36	24.38	24.88	16.42	no
NanoXML v3	F_SB_HD_1	21.19	21.19	19.21	19.77	3.67	yes
NanoXML v3	F_SB_HD_2	9.89	9.6	9.6	22.32	7.91	yes
NanoXML v3	F_SP_HD_1	22.03	21.19	18.36	18.93	0.57	yes
NanoXML v3	F_XEL_HD_2	3.39	0.28	0.28	19.49	2.54	yes
NanoXML v3	F_XEL_HD_3	1.7	1.41	1.41	24.86	0.28	yes
NanoXML v3	F_XEL_HD_4	6.22	6.21	6.21	22.6	3.39	yes
NanoXML v3	F_XEL_HD_5	1.13	1.13	1.13	24.29	0.57	yes
NanoXML v3	F_XEL_HD_6	1.41	1.13	1.13	20.62	1.41	yes
NanoXML v5	F_CR_HD_1	8.38	6.49	5.14	11.62	4.32	yes
NanoXML v5	F_CR_HD_2	11.08	7.3	5.14	16.76	7.03	yes
NanoXML v5	F_NV_HD_1	4.87	6.76	6.76	15.68	5.95	no
NanoXML v5	F_SP_HD_1	3.24	1.35	1.08	18.11	11.08	no
NanoXML v5	F_SP_HD_3	23.51	20.27	18.65	19.19	0.27	yes
NanoXML v5	F_XER_HD_1	20.81	20.27	18.65	19.19	12.16	no
Siena v1	F_SP_HD_1	10.61	7.58	7.07	16.16	2.53	yes
Siena v1	F_SP_HD_2	2.02	2.02	2.02	22.22	2.02	no
Siena v2	F_SP_HD_1	12.12	10.10	10.10	27.27	3.54	yes
Siena v5	F_SP_HD_1	9.57	9.09	8.13	16.75	4.78	yes
Siena v5	F_SP_HD_2	8.61	6.70	6.70	14.83	6.22	yes
Siena v6	F_AV_HD_1	2.39	2.39	2.39	18.66	5.74	yes
Siena v7	F_SP_HD_2	14.15	14.15	14.15	30.73	13.17	no

错误定位方法。Zhang 等[9]提出了基于关键谓词的错误定位方法，通过强制改变谓词的取值结果，找到可以使失败执行变为成功执行的关键谓词，并以此作为识别故障的根源。Jeffrey 等[18]提出了值替换方法，该方法寻找兴趣值映射对，然后对包含兴趣值映射对的语句使用 Tarantula 公式计算怀疑度值并进行排序。实验表明，值替换方法可以有效地定位出错误语句或和错误语句存在直接依赖关系的语句。但是，值替换方法在程序执行时替换语句的状态，无法识别语句遗漏错误，同时，值替换方法需要对程序中的每一条语句进行替换，时空开销比较大。而本文提出的参数−值替换方法在方法层次上进行替换，不仅可以节省开销，而且可以进行语句遗漏错误的定位。

Jones 等[3]认为语句的故障怀疑度与其执行失败的次数正相关，提出了用不同颜色表示语句故障可疑程度的方法。Abreu 等[4]引入分子生物学领域的相似系数 Ochiai，并通过实验证明了 Ochiai 的定位精度稍优于 Tarantula 方法。Naish 等[5]描述故障发生的 2 种情形，在此基础上提出了 2 种理想度量怀疑度公式 Naish1 和 Naish2。 Yilmaz 等[22]提出 TWT 方法，该方法以时间光谱作为程序执行特征的抽象，根据成功执行的时间光谱建立程序行为模型，通过该模型识别失败执行和成功执行的偏离来进行错误定位。实验表明统计错误定位方法是一类有效的错误定位方法。但是，该类方法只收集程序的运行时信息，对于程序遗漏信息不能进行有效地收集，造成错误的漏报。本文方法在增加定位对象粒度的同时，考虑方法参数输入对程序执行结果的影响，并利用参数−值替换验证方法对程序执行结果的影响程度，从而找到可疑的方法，以此在一定程序上提高错误定位的效率。

减少切片中错误无关语句的数量而不降低定位错误代码的能力是基于程序切片错误定位方法的研究目标。Gupta 等整合了 Delta 调试技术以及前向、后向切片技术，提出程序砍片方法[9]和多点切片方法[7]。实验表明，上述 2 种方法可以有效减少错误无关语句的数量。Zhang 等[13]从动态角度提出隐式依赖的概念，它将已经观测到的发生在谓词和变量使用上的依赖关系加入切片中。 利用该切片结果可以完成程序遗漏执行错误的定位。基于程序切片的错误定位方法可以为开发人员提供一个上下文信息进行调试，但是该上下文信息只能包含存在的语句，不能提供语句遗漏错误的信息。

6 结束语

本文分析方法输入对程序执行结果的影响，提出基于参数−值替换的错误定位方法。首先记录失败执行中出现的方法，并将其作为可疑方法候选集，根据方法在不同执行中参数输入的差异计算候选集方法对程序执行结果的影响度，然后计算高影响度方法的参数−值映射对，依据是否包含参数−值映射对分别对候选集中发方法进行排序，并完成错误定位。实验表明本文方法具有很好的错误效果。下一步，将针对参数−值替换过程中的时空效率问题做进一步分析，尽量减少替换次数，同时考虑方法外部输入对方法执行行为的影响，以提高错误定位的准确性和效率。

参考文献：

[1] PARNIN C, ORSO A. Are automated debugging techniques actually helping programmers?[A]. Proceedings of the 2011 International Symposium on Software Testing and Analysis[C]. Toronto, 2011. 199-209.

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

[3] JONES J A, HARROLD M J, STASKO J. Visualization of test information to assist fault localization[A]. Proceedings of the 24th International Conference on Software Engineering[C]. Orlando, 2002. 467-477.

[4] ABREU R, ZOETEWEIJ P, VAN GEMUND A J. An evaluation of similarity coefficients for software fault localization[A]. Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing[C]. Riverside, 2006. 39-46.

[5] NAISH L, LEE H J , RAMAMOHANARAO K. A model for spectra-based software diagnosis[J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 11.

[6] WONG W E, QI Y, ZHAO L. Effective fault localization using code coverage[A]. Proceedings of the 31st Computer Software and Applications Conference[C]. Beijing,China, 2007. 449-456.

[7] ZHANG X Y, GUPTA N , GUPTA R. Locating faulty code by multiple points slicing[J]. Software: Practice and Experience, 2007, 37(9): 935-961.

[8] GUPTA N, HE H, ZHANG X. Locating faulty code using failure-inducing chops[A]. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering[C]. Long Beach, 2005. 263-272.

[9] ZHANG X Y, GUPTA N , GUPTA R. Locating faults through automated predicate switching[A]. Proceedings of the 28th International Conference on Software Engineering[C]. Shanghai,China, 2006. 272-281.

[10] ZHANG X Y, GUPTA N, GUPTA R. Pruning dynamic slices with confidence[J]. ACM SIGPLAN Notices, 2006, 41(6):169-180.

[11] WEISER M. Program slicing[A]. Proceedings of the 5th International

Conference on Software Engineering[C]. San Diego, 1981. 439-449.

[12] YU Y B, JONES J A, HARROLD M J. An empirical study of the effects of test-suite reduction on fault localization[A]. Proceedings of the 30th International Conference on Software Engineering[C]. Leipzig, 2008. 201-210.

[13] ZHANG X Y, TALLAM S, GUPTA N. Towards locating execution omission errors[J]. ACM SIGPLAN Notices, 2007: 415-424.

[14] CHILLAREGE R. Orthogonal defect classification[J]. Handbook of Software Reliability Engineering, 1996: 359-399.

[15] VOAS J M. PIE: A dynamic failure-based technique[J]. IEEE Transactions on Software Engineering, 1992, 18(8): 717-727.

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

[17] SHU G, SUN B Y, PODGURSKI A. MFL: Method-Level Fault Localization with Causal Inference[A]. Proceedings of the 6th International Conference on Software Testing, Verification, and Validation[C]. Luxembourg, 2013. 124-133.

[18] JEFFREY D, GUPTA N, GUPTA R. Fault localization using value replacement[A]. Proceedings of the 2008 International Symposium on Software Testing and Analysis[C]. Seattle, 2008. 167-178.

[19] DO H, ELBAUM S, ROTHERMEL G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact[J]. Empirical Software Engineering, 2005, 10(4): 405-435.

[20] XIE X, CHEN T Y, KUO F C. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J]. ACM Transactions on Software Engineering and Methodology, 2013, 22(4): 31.

[21] LIU C, FEI L, YAN X. Statistical debugging: a hypothesis testing-based approach[J]. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848.

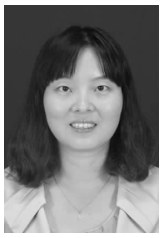
[22] YILMAZ C, PARADKAR A , WILLIAMS C. Time will tell: fault localization using time spectra[A]. Proceedings of the 30th International Conference on Software Engineering[C]. Leipzig, 2008. 81-90.

作者简介：

王兴亚（1990-），男，山东淄博人，中国矿业大学博士生，主要研究方向为软件分析与测试。

姜淑娟（1966-），女，山东莱阳人，中国矿业大学教授、博士生导师，主要研究方向为编译技术、软件工程等。

鞠小林（1976-），男，江苏南通人，南通大学讲师，主要研究方向为软件分析与测试。



曹鹤玲（1980-），女，河南郑州人，中国矿业大学博士生，主要研究方向为软件分析与测试、数据挖掘。