软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006081]

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

基于信息检索的软件缺陷定位技术研究进展*

张芸[1], 刘佳琨[2], 夏鑫[3], 吴明晖[1], 颜晖[1]

[1](浙江大学城市学院 计算机与计算科学学院,浙江 杭州 310015)

[2](浙江大学 计算机科学与技术学院,浙江 杭州 310007)

[3](Monash University, Australia Melbourne VIC 3800, Australia)

通讯作者: 夏鑫,E-mail: Xin.Xia@monash.edu

摘 要: 缺陷定位是软件工程研究最活跃的领域之一.大部分软件缺陷都会被提交到类似于 Bugzilla 和 Jira 的缺陷追踪系统中.由于提交的缺陷报告数量过多,开发人员不能及时地处理,因而迫切需要一个自动化工具来帮助开发人员识别缺陷相关源代码文件.研究人员已经提出了大量的缺陷定位技术.基于信息检索的软件缺陷定位技术（Information Retrieval-based Bug Localization,简称 IRBL）利用了缺陷报告的文本特性,并且由于计算成本低、对不同的程序语言更具有普适性,成为缺陷定位领域的研究热点,取得了一系列研究成果.然而,IRBL 技术也在数据预处理、相似度计算和工程应用等方面存在诸多挑战.鉴于此,本文对现有的 IRBL 技术进行梳理总结.主要内容包括:(1) 梳理了 IRBL 中数据预处理的过程和信息检索通用方法;(2)对 IRBL 技术中利用的数据特征进行了详细的分类和总结;(3)总结了技术评估中使用的性能评估指标;(4) 归纳出了 IRBL 技术的关键问题; (5) 最后展望了 IRBL 技术的未来发展.

关键词: 缺陷报告定位;信息检索;软件工程;

中图法分类号: TP311

Information Retrieval-based Bug Localization: A Literature Review

ZHANG Yun[1], LIU Jia-Kun[2], XIA Xin[3], WU Ming-Hui[1], YAN Hui[1]

[1](School of Computer & Computing Science, Zhejiang University City College, Hangzhou 310015, China)

[2](College of Computer Science and Technology, Zhejiang University, Hangzhou 310007, China)

[3](Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia)

Abstract: Bug localization is one of the most active domains in software engineering. Most of the bugs are submitted to bug tracker systems, e.g., Bugzilla and Jira. Because of the large number of the submitted bug reports, it is difficult for developers to resolve these defects in time. Therefore, an automatic tool to help developers to identify bug related files is needed. Many bug localization technologies have been proposed by researchers. Taking advantages of the text nature of bug report, information retrieval technologies are adopted to solve bug localization problems. Due to the low computing cost and the applicability to various programming languages, information Retrieval-based Bug Localization (IRBL) technologies become hot spots in bug localization and acquire a series of achievements. However, challenges still exist in data preprocessing, similarity calculation and engineering application. Therefore, we summarize current IRBL technologies. The contributions of this paper are : (1) We summarize the data preprocess methods and general information retrieval algorithms. (2) We conclude and classify the feature categories. (3) We conclude the performance measures. (4) We highlight the current problems in IRBL technologies. (5)We outlook the trends of IRBL technologies.

Key words: Bug report localization, Information retrieval, Software engineering

* 基金项目: 2019 年度高层次留学回国人员在杭创新项目(多元融合的缺陷定位技术研究)
收稿时间: 2020-02-16; 修改时间: 2020-04-07; 采用时间: 2020-05-07; jos 在线出版时间: 2020-05-26

1 引言

在软件开发过程中,由于软件代码的日益复杂,软件缺陷通常无法避免.根据 IEEE 标准定义,软件缺陷是指软件产品中存在的,使产品无法满足软件需求及其规格要求,需要修复的瑕疵、问题[1].软件缺陷影响软件产品应用与发展,同时造成大量经济损失.例如,美国国家标准与技术研究院(National Institute of Standards and Technology,简称 NIST) 的一项技术报告指出,软件缺陷每年对美国造成的经济损失高达 600 亿美元[2]. 因此,缺陷修复在软件开发中占据重要地位.实证研究表明,缺陷修复耗费成本占到软件开发所有成本的 50%－70%[3].

为对缺陷修复过程进行管理,缺陷跟踪系统(Bug Tracking System) 如 Bugzilla 等,被广泛应用到现代软件系统[4].在缺陷修复过程中,软件用户或者测试人员在发现缺陷后通过缺陷跟踪系统提交缺陷报告(Bug Report),然后负责缺陷分派的开发者将可重现缺陷报告分派给缺陷修复开发者.现代软件系统由于规模庞大,受到大量软件缺陷影响.Fan 等人统计发现,Mozilla 项目的 Bugzilla 缺陷跟踪系统平均每天收到约 300 个缺陷报告[5].因此,开发者缺陷修复任务非常繁重.

为修复缺陷,开发者需要根据缺陷报告描述,确定该缺陷的源代码位置,即软件缺陷定位[6].现代大型软件系统中通常有成千上万个源代码文件,而修复缺陷可能仅需修改一个或几个源文件.实证研究表明,84%－93%的缺陷被定位到 1 到 2 个源文件[7].因此,手动进行缺陷定位对开发者来说耗时费力.同时,手动缺陷定位严重依赖于开发者的经验,对经验不足的开发者非常困难.软件工程领域已有基于动态程序分析与静态程序分析的缺陷定位技术帮助开发者定位缺陷.动态缺陷定位技术是指通过分析程序运行时执行行为(如运行出错的测试用例)判断缺陷位置[8],目前一种主流研究思路是基于程序频谱的缺陷定位,分析测试用例的执行行为和运行结果,重点对被测程序的内在结构进行分析,此类方法可以较细粒度的确定缺陷语句在被测程序内的可能位置,但需要消耗较多程序运行时间成本和资源成本,且测试用例的数量和质量对缺陷定位性能影响较大,关系到是否能有效获取程序内部复杂信息.因此,动态缺陷定位方法有助于开发人员进行软件代码调试,帮助缺陷修复.静态缺陷定位技术不需要执行测试用例,利用源代码静态分析(如函数调用关系、数据依赖关系、类型约束等),对比一系列编程规则(如程序语言语法、编程规范),判断缺陷位置[10].由于编程规则与程序语言相关,该缺陷定位技术受程序语言限制,定位粒度比动态定位方法粗糙,通常定位到文件和方法级别的粒度.但由于无需执行测试用例,静态定位方法拥有执行成本低、使用简单等优势,适用于快速帮助开发人员定位缺陷可疑位置[83].

当前静态缺陷定位研究的一类主流方法是基于信息检索的软件缺陷定位(Information Retrieval-based Bug Localization,简称 IRBL) 技术,也是本文重点关注的研究问题.其目的在于利用缺陷报告内容,半自动或全自动的确定相关源代码文件或函数,从而降低开发成本,提高开发效率.信息检索(Information Retrieval,简称 IR)是指给定查询,从文档库中获取与该查询相关文档的过程.在 IRBL 技术中,一个新提交的缺陷报告被看作为一次查询,软件系统所有源文件或函数被看作为文档库,从而将缺陷定位规约为信息检索问题.其基本思路是对文档库中每个文档(文件或函数),利用一个或多个特征(如词袋特征 Bag-of-words) 量化表示该文档与给定缺陷报告,计算二者相关度分数(如文本相似度) 并整合这些相关度分数,最终对文档库所有文档按照与缺陷报告相关度高低排序.相比已有缺陷定位技术,应用 IR 技术进行缺陷定位具有以下优势:

(1) 与基于动态程序分析的缺陷定位相比,IRBL 技术计算成本更低[18][24].IRBL 技术无需运行程序获取程序执行行为,而是通过分析缺陷报告文字描述,找到该缺陷所在位置.

(2) 与传统基于静态程序分析的缺陷定位相比,IRBL 技术对程序语言更具普遍性.同时,IRBL 技术针对性更强,能够针对给定缺陷报告,充分利用缺陷报告提供的信息,分析该缺陷所在代码位置.

然而,IRBL 技术也存在着作为静态缺陷定位方法的不足,目前 IRBL 技术只能定位到方法粒度,无法定位到具体代码行,而且缺陷定位准确度受缺陷报告质量影响较大,缺陷报告由人工整理的缺陷信息组成,缺陷提出者的专业水平高低导致缺陷报告质量参差不齐,而如何精准的将缺陷信息从缺陷报告中提取并加以利

用也是 IRBL 技术的难点之一.

基于上述优劣势,IRBL 技术得到软件工程研究者广泛关注,产生了一批 IRBL 研究成果.其中 Lukins 等人于 2008 年 ICPC 论文首次提出直接基于 IR 方法的缺陷定位技术[9].IRBL 技术具备计算成本低,普遍性和针对性更强的优势.IR 领域前沿研究(如查询重建技术) 支持了 IRBL 技术性能进一步提升.同时.IRBL 技术还可与其他缺陷定位技术、缺陷预测技术结合提升性能.因此,该技术在近年来成为缺陷定位领域的热点.在大量工作中,研究者针对 IRBL 技术的缺陷报告与源文件相关度分数计算与整合以及技术评估方面提出有价值的理论、技术与观点,推动 IRBL 技术进步,但同时也产生了一定分歧.大量实证研究工作对 IRBL 技术的定位粒度、可信性、可用性、可重复性、可扩展性和可解释性等方面进行讨论,提出 IRBL 技术面临的新挑战.例如,Kochhar 等人发现 IRBL 技术只能对已有明确代码位置信息的缺陷报告实现准确定位,对于这类缺陷报告,开发者也同样能进行准确快速定位,因此作者对 IRBL 技术实际可用性提出质疑[10].

尽管 IRBL 技术研究者提出大量有价值的理论与技术,但缺乏对现有理论与技术的统一与整合尚无针对目前 IRBL 技术研究工作的梳理与归纳.鉴于此,本文拟针对基于信息检索缺陷定位技术的研究进展,从该技术使用的软件数据源、信息检索方法、相关度计算与整合以及技术评估等角度进行梳理、归纳和总结.同时,本文总结了当前 IRBL 技术存在的关键难点和未来发展方向.

文献选取方式.本文采用以下流程完成对文献进行索引与选取.

- 首先定义文献选取标准:该文献针对 IRBL 技术中的相关度计算与整合以及技术评估等方面提出新理论、新技术,或者该文献对 IRBL 技术相关理论技术提供实证研究支持或者对其应用进行实证研究讨论;
- 此外,该文献应公开发表在期刊、会议、技术报告或书籍中.

依据以上标准,本文通过以下 3 个步骤对文献进行检索和筛选.

(1) 本文文献搜索主要通过 ACM Digtal Library,IEEE Xplore Digital Library,Springer Link Online Library 以及 Google Scholar 等.论文检索的关键字包括 Information Retrieval Based Bug Localization,IR-Based Bug Localization,Text Retrieval Based Bug Localization,Query Bug Localization,IR-Based Fault Localization 等,同时在标题、摘要、关键词和索引中进行检索;

(2) 本文对软件工程领域的主要期刊与会议进行在线搜索,具体包括 TOSEM、TSE、EMSE、ASEJ、JSEP、IST、ASC、ICSE、FSE、ASE、ISSTA、ICSM/ICSME、MSR、ICPC、WCRE、APSEC、QRS 等;

(3) 本文基于上述步骤所获取的文献集合,对文献逐一查看,从文献的参考文献中进一步筛选出与基于信息检索的缺陷定位理论相关的文献.

通过以上文献索引与选取,共计 60 篇文献纳入本文后续文献总结中.其中有 45 篇为 IRBL 技术的相关度计算与整合以及技术评估方法等提供了新理论和新思路.另外 15 篇通过实证研究从 IRBL 技术可用性、可重复性以及其面临挑战等方面展开探讨.图 1 展示了本文所总结文献分布情况,其中包含 MSR 论文 9 篇、TSE 论文 7 篇、ICSM/ICSME 论文 7 篇、ASE 论文 6 篇、ISSTA 论文 5 篇、ICPC 论文 5 篇、ICSE 论文 4 篇、FSE 论文 4 篇、WCRE 论文 3 篇、EMSE 论文 3 篇、IST 论文 3 篇、ASC 论文 1 篇、ISSRE 论文 1 篇、JSEP 期刊 1 篇以及 QRS 会议 1 篇.图 2 展示了本文所总结文献发表年份的分布情况.

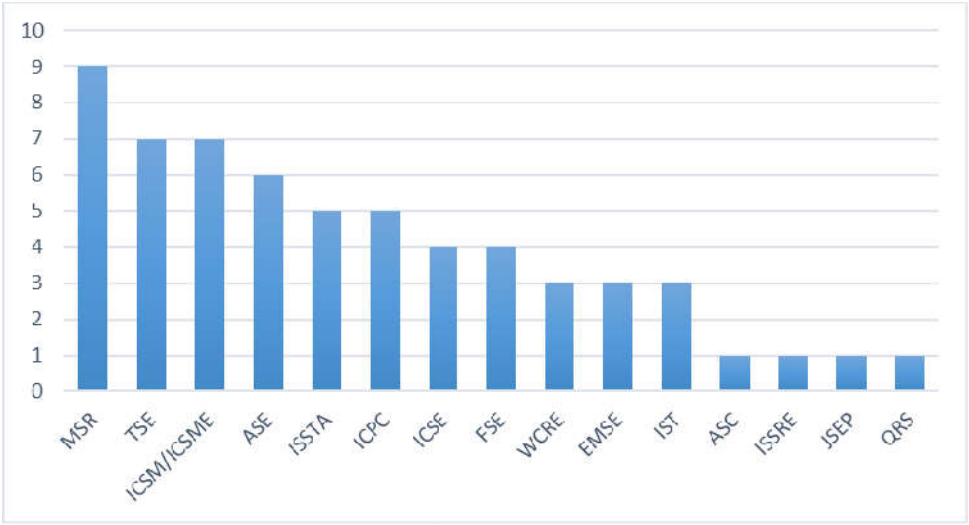

图 1 文献分布

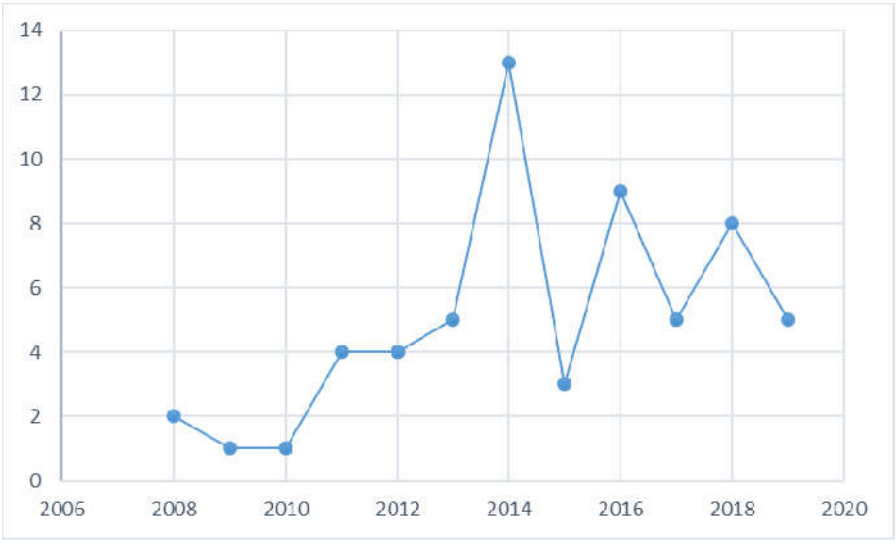

图 2 文献发表年份统计

图 3 展示了本文的综述过程. 首先确定文献搜索的关键词以及文件搜索的期刊和会议范围、时间范围, 进行初步检索. 而后对检索文献进行人工筛查, 剔除与信息检索的缺陷定位无关的文献. 随后, 对所选取的文献进行归纳总结和分类, 从 IRBL 新技术新理论和实证研究两大类展开深入探讨. 对于提出 IRBL 新技术新理论的文献, 本文根据研究中利用的多种数据源对文献进行分类总结, 归纳出以下几类数据源: 当前缺陷报告、缺陷跟踪仓库、版本控制仓库、堆栈跟踪信息和其他软件数据, 对于不同子类的文献进行系统分析和比较, 同时总结了此类文献常用的算法性能评价指标. 通过对 IRBL 技术实证研究相关的文献归纳总结, 分析了 IRBL 技术目前存在的数据噪音处理、多源特征处理、技术重现和工程实践等关键问题和挑战.

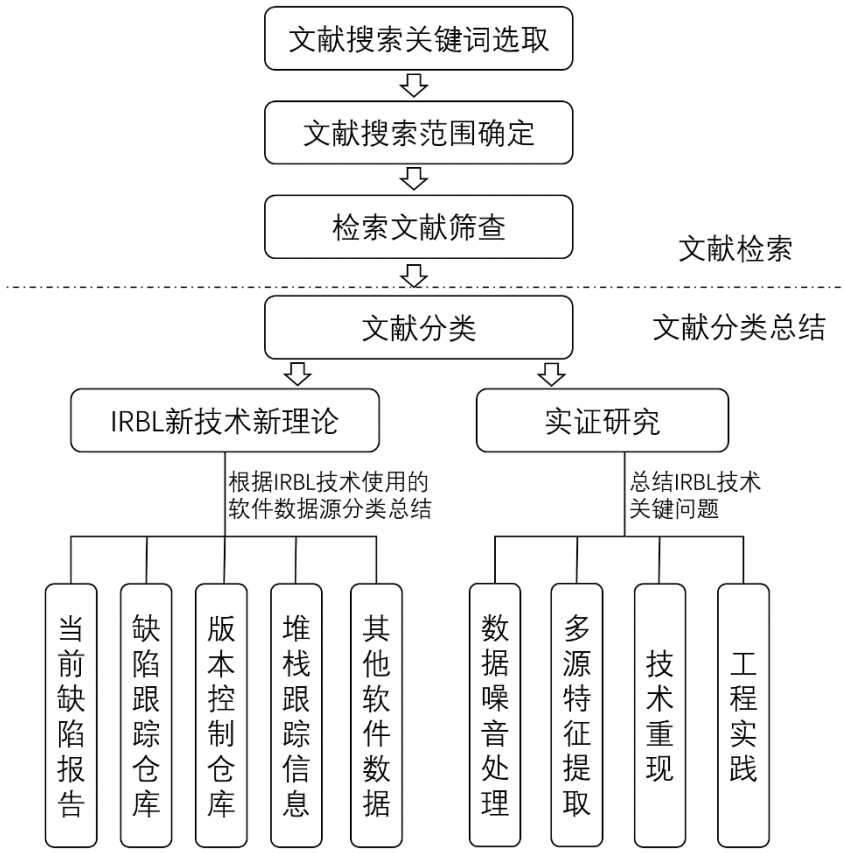

图 3 综述框架

本文整体结构:本文第二节介绍缺陷定位的例子,然后引出 IRBL 的通用步骤,对 IRBL 技术中的数据预处理的通用方法进行梳理.第三节介绍了通用的信息检索技术.第四节从不同数据源出发,对 IRBL 新技术新理论研究进展进行系统分析和比较.第五节介绍了用于评估 IRBL 技术性能的常用评价指标.第六节总结目前 IRBL 技术研究面临的关键问题,包括数据噪音处理、多源数据特征提取、技术重现以及工程实践等方面.第七节展望 IRBL 技术未来发展趋势.最后在第八章对本文进行了总结.

2 缺陷定位概况

本章介绍关于基于信息检索的缺陷定位技术的预备知识.具体而言,本文在 2.1 通过展示缺陷定位的例子,来进一步帮助读者理解缺陷定位问题;在 2.2 节中,描述了基于信息检索的缺陷定位方法的通用步骤;在 2.3 节梳理了数据预处理的一般方法.

2.1 缺陷定位例子

图 4 显示了 Eclipse 的一份真实缺陷报告（包括缺陷摘要和描述）.收到此报告后,开发人员需要在上万个 Eclipse 源文件中找到相关文件来修复此缺陷.实际上,该缺陷报告包含很多的关键字,比如 JUnit,iteration,loop,test,suite 等,都与单元测试有关.在 Eclipse 中,有一个名为 HyadesTestRunner.java 的源代码文件,它也包含很多相似的关键字.图 4 显示了缺陷报告和源代码之间的良好匹配.

然而,在软件开发的过程中,开发人员经常收到这样的缺陷报告:缺陷报告描述的语言和程序源码中的语言并不是完全相同,汇报缺陷报告的人员使用了和源码中不同的词汇描述了他们所遇到的问题,因而开发人员不得不跨越两种描述之间的鸿沟来解决问题.例如,图 5 显示了另一份缺陷报告及其相关的 Eclipse 源代码文件.这个缺陷报告包含如下关键字:console,corrupt,IAC,string,echo 等.在检查了版本控制系统之后,开发人员发现要通过更改源文件 SocketListener.c 来修复此缺陷.但是,SocketListener.c 代码中不包含任何缺陷报告中含有的关键字.相反,代码中包含 bytesRead,process,size,connection,offset 这样的关键字.在这种情况下,仅仅考虑文本的相似性是不够的,还要考虑语义相似性.在分布假设下[13],缺陷报告中的关键字及其相关的源代码文件中的关键字在语义上是相关的.一些方法,如主题建模[14] 和词嵌入[15],可以被用来度量语义相似度.这些方法为解决这些问题提供了可能的解决方案.

Bug ID: 184845
Summary: **JUnit** runner emits incorrect **iteration** count for **loop**s.
Description: **JUnit** runner emits incorrect **iteration** count for **loop**s.
Steps to reproduce:
1) Create a **JUnit test suite** with a **loop** contains > 1 **iteration**s.
2) Run the **test suite**.
3) Open the **Test** Log view and note the value of the **Iteration**s field at the top of the Common Properties pane when selecting the **Loop** event in the Events tree of the Events tab is 0.
Build: TPTP-4.4.0-200704231237

src-common-runner/org/eclipse/hyades/test/common/**junit**/HyadesTestRunner.java

```
// loop id from test is also the id of the loop in the test suite.
        loopEvent.setOwnerId(hyadesTest.getId());
        loopEvent.setParentId(parentID);
        if (hyadesTest instanceof HyadesTestSuite)
            loopEvent.setIterations(((HyadesTestSuite) hyadesTest).getLoopCount());
        writeEvent(loopEvent);
```

图 4 缺陷报告（ID:184845）及其 Eclipse 项目中的相关代码

Bug ID: 234568
Summary: Input from **console** in WB is **corrupted**
Description: 1. Start profiling application with default values using **IAC**
2. Input **string** to the **console** and watch **echo**
output looks different than expected

org.eclipse.tptp.platform.agentcontroller/src-native-new/src/transport/socketTL/SocketListener.c

```
   bytesRead += offset;
   if ((bytesRead + offset) < sizeof(tptp_basic_msg_header_t))
   // We can pass the DATA_CHANNEL data through processRecdMsgs at any size, as it will
always be written
   // in its entirety, once we get into processRecdMsgs. (bug 234596)
   if ((bytesRead + offset) < sizeof(tptp_basic_msg_header_t) && !(pRdb->connectionType ==
DATA_CHANNEL))
   /* too small to determine the payload length. Go read some more */
      offset += bytesRead ;
```

图 5 缺陷报告（ID:234568）及其 Eclipse 源码中的相关代码

2.2 缺陷定位通用步骤

图 6 展示了 IRBL 技术的一般过程.IRBL 技术的输入包括新提交的缺陷报告、源代码文档(源文件或函数)以及各种软件开发仓库(包括但不限于版本控制仓库、缺陷跟踪仓库).IRBL 技术主要包括预处理、相关度分数计算、相关度分数整合三个阶段.其预处理是指数据集的构建、语料库的创建以及索引和查询的构造的过程.相关度计算是指依据对缺陷报告与源代码文档预处理后的特征表示,对每个源代码文档,计算该文档与缺陷报告相关程度的过程.相关度计算过程可能产生多个相似度分数.相关度分数整合是指对每个源代码文档计算出的多个相似度分数进行整合,以求得其与缺陷报告的整体相关度分数,从而最终得到对源代码文档的排序结果.然后开发者可以从排序的开头逐个检查文件.直到找到和缺陷相关的文件.这样,与缺陷报告相关的文件就可以很快被定位到.显然,缺陷定位的目标是尽量提高缺陷文件的排名.

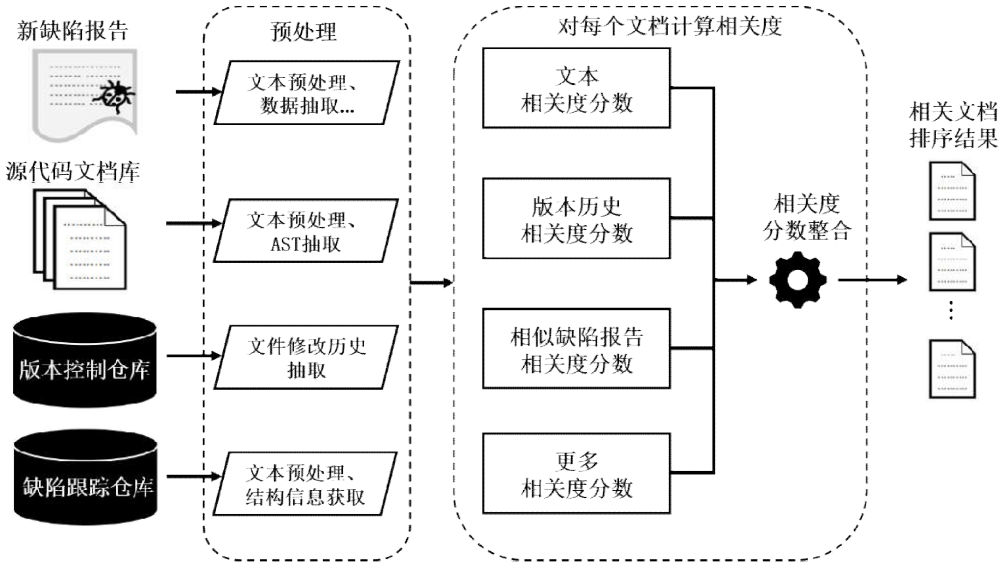

图 6 基于信息检索的缺陷定位技术一般研究框架

2.3 数据预处理的通用方法

在**数据集的构建**过程中,为了评估 IRBL 技术的性能,研究人员需要收集已经修复的缺陷及其对应的源代码文件.研究人员通过以下启发式规则来建立缺陷报告和源代码文件之间的链接:

(1) 扫描所有提交信息,检查是否有按照“issue618”,“bug1111”等规则标记的提交记录.

(2) 每个提交记录修改的文件即为改缺陷报告对应的源代码文件.

收集缺陷报告极其相应的源代码文件后,需要进行**语料库创建**.为此,需要对每个源代码文件进行词法分析,并创建一个词法符号的向量.一些词汇,比如保留字(如 int,double,char 等),分隔符,运算符在所有程序中都是一致的,应该被剔除.英语停用词(如 he(他),she(她),the(这))也应被剔除.很多在程序中定义的变量实际上是单词的串联.例如,变量 TypeDeclaration 包含两个单词:“type”和“declaration”;变量 isCommitable 由两个词组成:“is”和“Commitable”.这些复合符号应被分隔成单独的符号.许多符号具有相同的词根形式.例如,“reads”,“reading”和“read”有着相同的词根“read”.我们使用 Porter 词干算法[16]将单词缩减为其词根形式.

创建语料库后,语料库中的所有文件都被**建立索引**.通过使用这些索引,研究人员就可以快速定位到包含被查询词的文件,再根据它们的相关性对文件进行排名.

缺陷定位将整个缺陷报告作为查询的内容,来**构造查询语句**.为此,需要从缺陷报告的标题和描述中提取单词,删除停用词,将每个单词转化为词根形式,并使用它来搜索已建好索引的源代码语料库中的相关文件.

3 信息检索通用方法

由于缺陷报告的文本特性,许多信息检索的通用方法在 IRBL 技术中得到了应用.研究人员不仅利用这些方法进行缺陷报告和源代码文件的相似度计算,还利用这些算法来提取其他数据源的特征,例如相似缺陷报告等.在本节中,本文分别介绍了三种信息检索的通用基本方法,即向量空间模型、主题模型和词嵌入, 它们是构建缺陷定位工具的基础;在 3.4 节介绍了相似性度量方法.

3.1 向量空间模型

向量空间模型(VSM)是 IRBL 中一种常用的向量表示算法,许多研究人员都利用 VSM 模型来表征缺陷报告和源代码[17-23].在 VSM 中,每个文档都被表示为一个向量.向量中的每个值表示每个词在文档中的权重.一种分配权重的常用方法,是结合使用词频(Term Frequency) 和逆文档频率(Inverse Document Frequency) 的概念$(tf-idf)$.标准$tf-idf$方案按以下公式给文档d中的词t赋予权重:

$$weight(t,d)=tf(t,d)\times idf(t,D)$$

其中, $tf(t,d)$是指词t在文本d中的频率,它表明了该词对该文本的重要性,$idf(t,D)$是指词t在语料库D中的逆文本频率,它刻画了词本身的稀有程度:一个越稀有的词,在区分文本时就越有用.为了给稀有的词更高的权重,逆文本频率等于该词文档频率(df)倒数的对数值.

给定查询语句和语料库,VSM 首先将查询语句和语料库中的每个文档转化成词袋（Bags of words）.然后它对词袋中的每个词计算$tf-idf$权重,并转化为向量.

3.2 主题建模

隐含狄利克雷分布（Latent Dirichlet Allocation,简称 LDA）是一种主题建模的常用算法.它是一种词袋模型,认为一篇文档可以包含多个主题,文档中的每个词都是由某个主题生成的,LDA 给出文档属于每个主题的概率分布,同时给出每个主题上词的概率分布.许多研究人员都利用 LDA 来表示缺陷报告和源代码[17][24][25].

LDA 的输入为:作为训练集的文本语料库和一系列的参数,包括主题的数量(K)等.在训练阶段,对于每个文档s,将得到一个具有K个元素、每个元素对应着一个主题的主题概率向量$\vec{s}$.$\vec{s}$中每个元素的值是 0 到 1 之间的实数,表示s中的单词属于对应主题的比例.训练之后,LDA 可以对新文档m预测其主题概率向量$\vec{m}$.这样,将文档m中的词映射为主题概率向量$\vec{m}$,它包含了文档对应的每个主题的概率.训练后,LDA 可以预测新文档中每个词的主题.对于新文档m,假设有K个主题,基于每个词对应的主题概率值,使用如下公式计算主题向量 $\vec{z_m}$:

$$\vec{z_m}=<t_1,t_2,\cdots,t_k>,where$$

$$t_i=\frac{\#words\ assigned\ in\ the\ i^{th}\ topic\ in\ m}{\#word\ in\ m}$$

通过这种方式,将文档m中的术语映射到主题向量中,该主题向量包含了文档中出现每个主题的概率.

3.3 词嵌入

词的分布式是一种深度学习方法,表示假设出现在相似上下文中的词往往具有相似的含义[13],因而被许多研究人员用来弥合出现在缺陷报告和源代码文件的词汇和代码符号差异[26][27].因此,单个单词不再被视为唯一的符号,而是映射到密集的实值低维向量空间.每个维度代表该词的潜在语义或句法特征.语义上相似的单词在嵌入空间中距离很接近.

词嵌入通常使用基于神经网络的语言模型来学习得到.Mikolov 等[29,30]提出了一种流行的词向量（word2vec）模型,即连续的 skip-gram 模型.它可以学习中心词（即w_i）的嵌入,中心词的嵌入可以在$2k+1$个单词的上下文窗口中预测周围的单词.skip-gram 模型的目标函数是最大化以中心词为条件的上下文词对数概率之和.模型的输出是单词的“字典”,里面对应着每个单词的向量表示.给定文档（如缺陷报告或源代码文件）,通过对文档文本内容查找“字典”得到词嵌入向量并连接,可以将其文本内容转换为向量.

3.4 相似度计算

确定性模型如 VSM、词嵌入等模型,对文档和查询的规范化表示通常使用余弦相似性度量:

$$cos(q,d)=\frac{\sum_{t\in(q\cup d)}weight(t,q)\times weight(t,d)}{\sqrt{\sum_{t\in q}weight(t,q)}\times\sqrt{\sum_{t\in d}weight(t,d)}}$$

其中, q和d分别表示查询语句和文本.计算完语料库中每个文本对应的相似度分数后,据此进行排序.最后,输出排名前k个的文本.

在 IRBL 技术中,研究人员发现,经典 VSM 在排序时对待不同文本不是一视同仁的:越短的文本更容易排名靠前,越长的文本通常具有较低的相似性,因而很难排名靠前.然而在缺陷定位算法中,较大的源文件往往更容易包含缺陷.为此他们提出了修订的向量空间模型（rVSM）[11].在 rVSM 中,他们考虑了文档长度来优化用于缺陷定位的经典 VSM 模型的相似度.具体而言,他们使用:

$$rVSM\ Score(q,d) = g(\#\ term) \times cos(q,d)$$

修订后的向量空间模型是 IRBL 技术中常用的相似度计算方法.研究人员经常使用 rVSM 来衡量缺陷报告和源码之间的相似性[31][32][33].

关于概率模型如 UM、LDA 和 BM25 等模型,研究人员一般利用似然概率和 KL 散度来度量相似性.

似然概率:$\overrightarrow{w_m}$表示第m个文档的向量表示,并且$\overrightarrow{w_q}$是查询的向量表示,则似然概率由$P(\overrightarrow{w_q}|\overrightarrow{w_m})$表示.

$$sim(\overrightarrow{w_q},\overrightarrow{w_m}) = P(\overrightarrow{w_q}|\overrightarrow{w_m})$$

KL 散度:衡量两个概率分布的差异.对于 LDA,如果$\overrightarrow{\theta_m}$和$\overrightarrow{\theta_q}$分别代表文档和查询的多项参数的分布,相似性度量为$-KL(\overrightarrow{\theta_m},\overrightarrow{\theta_q})$.分歧越高,相似性越低.该相似性度量可以用以下形式表示:

$$sim(\overrightarrow{w_q},\overrightarrow{w_m}) = -KL(\overrightarrow{\theta_m},\overrightarrow{\theta_q})$$

4 IRBL 技术研究进展

在 IRBL 技术中,除了当前缺陷报告和代码文件,研究人员利用了软件开发中的多种数据源来协助进行缺陷定位.该步骤是 IRBL 技术的关键步骤.本文分别从不同的数据源对这些 IRBL 算法进行总结.具体而言,研究人员利用以下几种数据特征:当前缺陷报告、缺陷跟踪仓库、版本控制仓库、堆栈跟踪信息、其他软件开发数据.

在下文将介绍这些 IRBL 算法和其中使用的主要信息检索的技术.

4.1 当前缺陷报告

这一类 IRBL 算法从当前缺陷报告和当前版本的源代码文件中提取有效信息.具体而言,研究人员利用了 1）代码的文本特征和 2）代码的结构信息进行分析.为了表征当前版本的文件信息,其中很多算法利用历史数据进行训练.

4.1.1 代码的文本特征

这一类 IRBL 利用缺陷报告的文本型特征,将源代码文件也视为单纯的文本文件,运用通用的信息检索方法进行缺陷的定位.在最初的 IRBL 技术中,研究人员仅利用缺陷报告和源代码文件进行定位[17][24][35][9].例如,Lukins 等人使用基于隐狄利克雷分配（LDA）的静态技术来自动化定位缺陷[24],他们首先基于源代码文件构建的语料库生成 LDA 模型.LDA 模型的输出是词主题概率分布和主题-文档分布,然后使用缺陷报告作为查询语句来测量缺陷报告与生成的 LDA 模型之间的相似性.Rao 等人[17]的工作将一系列通用的文本信息检索方法:向量空间模型（VSM）,潜语义分析模型（LSA）,一元模型（Unigram Model）,隐狄利克雷分配模型（LDA）,以及基于簇的文档模型（CBDM）,应用在缺陷定位问题上并进行了组合,用以平滑不同信息检索算法所带来的噪音.他们在 iBUGS[36]上进行了实验,iBUGS 是一个基准的缺陷定位数据集,包含 75 千行代码和大量的缺陷.结果表明,LDA,LSA 和 CBDM 等复杂模型的性能比 SUM 和 VSM 等简单模型差.例如,在具有 LDA 和 LM 的线性组合模型中,随着 LDA 的权重增加,得到更差的性能.

Thomas 等人[35]研究了 VSM,LSI 和 LDA 的不同配置对缺陷定位的影响,发现无论是在单一模型还是在组合模型上配置都很重要.Zhang 等人[37]将代码抽象到不同层次的隐狄利克雷分配模型（LDA）,并结合多

个不同抽象层次的 LDA 模型和向量空间模型（VSM）进行缺陷定位,同时尝试了多种数据融合方法以得到最优的组合模型.Chaparro 等人[38]关注于查询优化策略,提取缺陷报告中有用的文本,删除无关的噪音,以提高检索缺陷相关的代码的性能.Zhang 等人[39]同样研究查询优化策略,针对代码方法粒度的缺陷定位时,部分方法过于简短,他们使用上下文方法中的元素来扩充当前方法,随后通过相似度值排序定位缺陷.Almhana 等人提出了一种基于多目标优化的缺陷定位算法[40].他们认为,推荐数量巨大的可能有缺陷的类让用户进行检查是耗时的,因而他们期望他们的算法不仅能最大化基于语义和基于历史的相似性来找到最有可能有缺陷的类,并最小化推荐类数量.他们利用历史的缺陷报告文本训练多目标优化算法,实验结果表明,他们提出的方法在前 10 预测精度上得分达到 0.87.

后来的 IRBL 工作中, 也都会使用以上方法或其组合来对当前缺陷报告和代码文件进行相似度的计算.

4.1.2 代码的结构信息

研究人员发现,源代码文件中不仅有丰富的文本信息,还有代码特有的结构信息,例如,代码注释、类名、方法名和变量名等.在 IRBL 中利用这些结构信息可以有效的区分不同的代码文件,带来精度上的提升.

Bread 等人将 LDA 模型和源代码中的结构信息结合起来进行缺陷定位[41].在 LDA 模型中,他们使用与 Lukins 等人提出的相同的技术[24].他们通过调用图分析和本地类/继承分析,以确定其他方法/类是否受到缺陷的影响.

Sisman 和 Kak 等人同样利用了代码的结构信息[19].他们利用**代码的空间结构**搜索缺陷报告相关的代码实体当作附加词来进行**查询重构**.他们认为低质量的缺陷报告难以可靠地定位期望的软件组件:缺陷报告中虽然包含了与信息检索任务相关的关键字,但同时也可能会包含和用户查找内容无关的术语;此外, 虽然编程语言都有各自的命名规范,但是开发人员仍有可能根据业务需要随意的缩写或者拼接变量名.

虽然 Sisman 等人利用了代码的空间结构,并且通过查询重构的方式提升了 IRBL 技术的性能.但是研究人员更期望一种通用的算法,能够利用代码的结构信息度量缺陷报告和源代码文件的相似程度.Saha 等人提出的工具 BLUiR 中,分别利用了代码文件中的四个代码结构的字段（类, 方法, 变量和注释）,同时将缺陷报告分成标题和正文两个部分,分别与代码的不同类别字段进行匹配,利用 VSM 算法计算相似度,然后将多个相似度值进行带权重的整合 [18].后来的研究人员利用 BLUiR 算法,整合其他数据特征进行相关度分数的计算[42][43][44][45][20][46].Dilshener 等人[47]在此基础上又考虑了文件名的重要性.

另外,Takahashi 等人[48]还注意到了**代码异味**对缺陷定位的影响.他们认为具有代码异味的模块更易于变化和更容易有缺陷,因而将代码异味的严重等级与基于信息检索的缺陷定位的文本相似性相结合,不仅能够识别出与缺陷报告相似的模块,更能指出更有可能包含缺陷的模块.

4.2 缺陷跟踪仓库

这一类方法利用缺陷跟踪仓库中已修复的相似缺陷报告来进行协同过滤.研究人员认为,相似的缺陷报告通常和相似的缺陷文件相关联[11].因此,对于新报告,检查已修复的类似缺陷及其相关文件会给 IRBL 算法带来的更加良好的效果.

Zhou 等人[11]首先利用了相似缺陷报告.他们基于 VSM 找到相似的缺陷报告并获得相似性分数.然后根据这些相似的缺陷报告找到对应的修复的源码文件.经典 VSM 在排序时对待不同文本不是一视同仁的:越短的文本更容易排名靠前,越长的文本通常具有较低的相似性,因而很难排名靠前.然而在缺陷定位算法中,较大的源文件往往更容易包含缺陷.考虑到经典 VSM 方法在缺陷定位问题上的不足,他们提出了修订的向量空间模型（rVSM）,在 rVSM 中,他们考虑了文档长度来优化用于缺陷定位的经典 VSM 模型.源码文件和新的缺陷报告的相似性分数由和相应源码文件相关的相似缺陷报告的相似性分数求和得出.许多利用相似缺陷报告的后来的研究工作都利用了这一算法[42][44][27][49][20][26][43].

$$SimiScore = \sum_{All\ S_i\ that\ connected\ to\ F_j} (Similarity(B, S_i)/n_i)$$

Davies 等人[50]利用缺陷报告的相似性来提高基于信息检索的缺陷定位技术的准确性.他们认为和同一个有缺陷的方法代码相关的缺陷报告是相似的.他们的方法有两个组件:缺陷描述组件和源代码组件.在缺陷描述组件中,他们为每一个方法代码训练一个基于 C4.5 的决策树,来判定该方法代码是否和某一缺陷相关.为了处理正负样本的不平衡性（即过少的相关的缺陷报告和过多的无关缺陷报告）,他们在每次训练时,训练集中都保留相关的缺陷报告和 5%的无关缺陷报告.在源代码组件中,他们使用 TF-IDF 来度量缺陷报告和源代码中每个方法之间的相似性.最后将这两个组件相结合:当给定一个新的缺陷报告时,首先根据缺陷描述组件中内置的分类器的结果（即相关或不相关）对方法进行排序,然后由源代码组件根据计算出的相似性得分对这些方法进行排序.

Gay 等人[51]提出基于信息检索和相关性反馈机制的缺陷定位技术,通过计算源代码和缺陷报告的相似度得到排序列表,并通过相关性反馈机制不断优化查询关键字并更新排序列表,最终定位到缺陷相关源代码.BugScout[25]是一个基于主题模型的算法, 它使用先前缺陷报告与其相关缺陷文件之间的关联来预测要修复新缺陷的位置.BugScout 的假设是,缺陷报告和相应的源代码,在文本内容上共享某些技术方面,这些类似的技术方面可用于给新的缺陷报告找到相关源代码文件.BugScout 分别通过从开发人员的角度对源文件, 以及从缺陷报告的角度对缺陷报告进行 LDA 建模, 来确定主题向量、主题比例向量、进行单词的选择, 然后来确定缺陷报告和源代码之间的关联.针对当前项目缺陷数据不充足的情况,Huo 等人[52]提出了基于深度迁移学习的跨项目缺陷定位方法,利用其它项目的历史缺陷数据训练模型来定位当前项目的缺陷.

4.3 版本控制仓库

研究人员从多个角度利用版本控制仓库中的信息.许多研究人员认为,如果一个文件多次修改以修复同一个缺陷或实现同一个功能,那么这个文件就更有可能是有缺陷的.Sisman 等人[53]使用贝叶斯估计文件有缺陷的可能性.其中先验概率来自版本控制仓库的历史记录,可能性来自信息检索算法,后验概率是最终的相关性得分.Sisman 等人[19] 此后又尝试了几种归一化来权衡信息检索技术和历史数据的分数以获得最佳结果.Ye 等人[20]统计了每个文件用于修复缺陷的次数.这个方法在他们后来的工作中也得到了使用[26].

Wang 等人[42]则额外考虑了文件的先前修改/实现时间.如果一个方法刚刚在最近的版本中实现或修改,它很有可能导致缺陷.这是因为缺陷通常是从刚刚实现的新功能或刚刚修改的最新缺陷中创建的.他们使用以下等式为每个源文件f分配可疑性分数:

$$Score^H(f,k,R) = \sum_{c \in R \wedge f \in c} \frac{1}{1 + e^{12\left(1-((k-t_c)/k)\right)}}$$

其中参数k为超参数可以凭经验设置.默认情况下,将k设置为 15.R指的是相关提交的集合,t_c是提交c和缺陷报告之间间隔的天数.许多后来的算法利用该版本历史组件考虑版本信息[43][44][33][49].然后他们集成该版本历史组件,缺陷相似性组件和结构组件这三个组件,提出了 AmaLgam 以定位有缺陷的源代码文件.在版本历史组件中,他们假设最近更改的源代码文件很有可能在将来出错,因此他们根据文件更改时间和缺陷提交的时间差,为每个源代码文件分配一个分数;在缺陷相似性组件中,给定一个新的缺陷报告,他们根据缺陷报告与历史缺陷报告的相似性,寻找相关的源代码文件;在结构组件中,它们应用 Saha 等人提出的 BLUiR[18],测量缺陷报告与结构信息（如类名、方法名、变量名和注释）之间的相似性.最后并将这四种类型的结构信息组合在一起.

Tantithamthavorn 等[54]通过考虑共变历史改进了 BugLocator 方法.他们认为,当修复一个有缺陷的文件时,那些曾经和这个有缺陷的文件一起更改过的文件也应该被修复.为此,他们创建了文件的共变矩阵, 用以记录历史中两个文件一起修改的次数,使用特征缩放对矩阵中的值进行归一化,如果共变值高于阈值并且共同更改的文件位于前 N 列表,则该文件被添加到前 N 列表.

Shi 等人[55]结合了历史版本数据、代码结构信息、动态分析等一系列特征,调研了 8 种排序学习算法(Learning to Rank)在缺陷定位任务上的性能,实验结果表明无需标准化的协调上升算法(coordinate ascent algorithms without normalization)是较适用于缺陷定位的排序学习算法.Xiao 等人[56]结合了缺陷修复时效和频率数据和文本语义信息,提出了基于卷积神经网络的缺陷定位技术.

4.4 堆栈跟踪信息

有时,缺陷报告包含失败任务堆栈跟踪信息,如图 7 所示.堆栈跟踪信息显示了在崩溃之前执行的指令序列[57].它也可以被认为是可能与缺陷相关的源代码文件的排序列表.使用堆栈跟踪信息的原因是堆栈跟踪信息中具有更高排名的文件更容易出错.当软件崩溃并抛出堆栈跟踪信息时,开发人员进行缺陷定位的第一个活动是检查堆栈跟踪信息中的实体.Schroter 等得出结论,通过修改堆栈跟踪信息中的方法,可以修复近 60%的带有堆栈跟踪信息的缺陷报告[58].作者指出,在堆栈轨迹的第一帧中可以找到 40%的修改方法,并且在堆栈轨迹的前 10 帧中可以找到几乎 90%的修改方法.Wong 等人[31]将堆栈跟踪信息中的排名前十位的方法实体的排名的倒数作为堆栈跟踪信息的相关性分数.将其他相关的方法实体和堆栈跟踪信息中的其他方法实体的相关性分数置为 0.1.许多利用堆栈跟踪信息的算法都利用了这一算法[49][44][47].

```
org.eclipse.swt.SWTException: Unable to load graphics library [Cairo is
required] (java.lang.UnsatisfiedLinkError:
/hp410/eclipse_3_1_0/configuration/org.eclipse.osgi/bundles/76/1/.cp/libswt-cairo-gtk-3138.so:
libcairo.so.1: cannot open shared object file: No such file or directory)
        at org.eclipse.swt.SWT.error(SWT.java:2942)
        at org.eclipse.swt.graphics.Device.checkCairo(Device.java:154)
        at org.eclipse.swt.graphics.GC.initCairo(GC.java:2268)
        at org.eclipse.swt.graphics.GC.setAntialias(GC.java:2475)
        at org.eclipse.draw2d.SWTGraphics.reconcileHints(SWTGraphics.java:795)
        at org.eclipse.draw2d.SWTGraphics.checkGC(SWTGraphics.java:282)
        at org.eclipse.draw2d.SWTGraphics.checkPaint(SWTGraphics.java:292)
        at org.eclipse.draw2d.SWTGraphics.drawPolyline(SWTGraphics.java:438)
        …
        at org.eclipse.swt.graphics.Device.checkCairo(Device.java:152)
        … 54 more
```

图 7 Eclipse 中#110370 缺陷报告,其中包含了堆栈跟踪信息

Wu 等人[59]提出了 CrashLocator,一种使用软件崩溃报告中的崩溃堆栈信息来定位故障功能的方法.通过使用静态调用图中相关的函数来扩展崩溃堆栈,推断出可能导致崩溃的失败执行跟踪.然后,在可能导致崩溃的失败执行跟踪中计算每个函数的可疑性,根据函数的可疑程度对其进行排名,并建议开发人员进行进一步筛查.

Moreno 等人[21]基于依赖图计算每一个源代码文件实体与堆栈跟踪中的实体之间的距离,这一方法结合堆栈跟踪信息和代码的结构信息.距离越近,当前源实体就越可疑.

除此之外,研究人员还利用堆栈跟踪信息来减少 IRBL 算法的搜索空间.这是因为如果一个程序元素没有出现在出现缺陷的堆栈信息中,那么就不可能是缺陷.Dao 等人[60]利用了缺陷报告中提交的堆栈跟踪信息,有效地减少搜索空间.

4.5 其他软件数据

除了缺陷报告和项目源码外,很多算法还考虑了其他的软件开发相关数据资源.许多基于信息检索的缺陷定位算法结合了对这些数据资源的利用和处理,来提高准确性.

(1) **报告者信息:**Wang 等人[43]提出了 AmaLgam+,基于 AmaLgam 考虑了堆栈跟踪和报告者信息.他们认

为同一个报告者可能会报告来自相同/类似软件组件的问题.

(2) **API 库的描述:**Ye 等人利用一系列领域内的知识来对缺陷报告相关的文件进行排名[20].他们利用代码中使用的 API 的库的描述来弥合缺陷报告和源码之间的语义代沟.同时考虑了历史缺陷报告的相似性、类名相似性、缺陷修复时间和和缺陷修复频率对当前缺陷报告进行定位的贡献.在后来的工作中,Ye 等人[12]通过词嵌入模型,将自然语言和代码片段都作为词向量,投影到共享的表示空间中,来弥补词汇之间的差距.他们利用 API 文档、教程和参考文档来训练词嵌入模型,然后使用该模型来估计文档之间的语义相似性.他们将词嵌入模型整合到他们的排序学习方法（LR）[20]中,并发现使用词嵌入来获取额外的语义相似性特征,可以帮助提高他们 LR 排名方法的性能.

(3) **项目需求信息:**Rath 等人[46]提出一种新方法 TraceScore,在使用缺陷报告、源代码信息和缺陷历史信息的基础上增加了项目需求信息和显式依赖项跟踪链接来进一步缩小缺陷报告和源代码间的语义差距.

(4) **提交信息:**Wen 等[61]提出 LOCUS,从软件变化提交日志中提取代码实体.它不仅可以找到有缺陷的源代码文件,还可以找到缺陷是如何产生的.Locus 从软件变化中检索信息,并且它从缺陷报告、提交日志和源代码文件中,使用余弦相似性来测量缺陷报告和源代码之间的相似性,从缺陷报告中提取形似代码的词汇,并将每个词与代码实体中的符号进行比较.

(5) **系统跟踪日志:**Zhou 等人[62]提出 MEPFL 方法,通过从系统跟踪日志中提取的一系列特征,在跟踪级别和微服务级别上训练预测模型.该预测模型可用于在生产环境中预测潜在缺陷,缺陷的微服务和运行时捕获的跟踪实例的故障类型.

(6) **测试用例:**Le 等[22]提出了一种多元自适应的缺陷定位方法 AML,利用测试用例生成程序频谱,并与信息检索方法结合生成复合模型,提高缺陷定位精度. AML 由三个模块 AML^{Text},$AML^{Spectra}$ 和 $AML^{SuspWord}$ 组成,AML^{Text} 运用 IRBL 算法处理处理缺陷报告中的文本信息,输出源代码中每个方法与缺陷报告的相似度值,$AML^{Spectra}$ 模块通过测试用例生成程序频谱,根据频谱同样会给每个方法计算一个可疑度分值,$AML^{SuspWord}$ 同时处理缺陷报告和程序频谱数据,并计算单词的可疑度分值用以给方法排序.最后 AML 将三个模块的输出通过限行整合器整合,并采用逻辑回归进行优化.实验结果表明 IRBL 算法和基于程序频谱的缺陷定位算法结合可有效的提高缺陷定位准确度.Hoang 等人[27]在 AML 方法的基础上提出基于网络聚类的多元缺陷定位方法,同时考虑上述三个模块特征以及构建缺陷报告和程序模块之间的关系图,并进行联合优化,损失函数由相似缺陷报告和相似程序模块的聚类损失、缺陷定位的误差损失组成.

4.6 相关度分数整合

近年来,研究人员通常组合上述数据特征以提升 IRBL 技术的性能.例如,Wang 等人[42]在 BLUiR[18]和 BugLocator[11]的基础上,结合了版本控制仓库中的历史提交记录,组成了 AmaLgam 工具.AmaLgam 的效果比 BLUiR 和 BugLocator 的效果都要好.同样的,新的数据特征组件:报告者信息和堆栈执行信息也被利用用来构建 AmaLgam+,并取得了更好的效果.

为此,数据特征相应的分数通常与预设的权重值线性组合.例如,线性组合两个特征分数x和y以求出可疑文件的最终相关度.与报告r相关联的文件f的最终得分计算如下:

$$(f,r)=\alpha x+(1-\alpha)y$$

为了确定线性组合的系数,一般α的设置为从 0 到 1 变化.然后,用不同的值比较缺陷定位性能.如果有三个具有两个线性权重α和β,则首先优化α值,令$\beta=0$以避免第三个特征的影响.然后固定β优化α值.

除此之外,Ye 等人[20]和[45]通过 SVMRank 算法来学习权重组合.Wang 等人[44]使用遗传算法来学习权重组合.Zhang 等人[37]使用了多种数据融合方法整合相似度分值,如 CombMNZ、CombSUM、Borda Count 等.

Wang 等人在 AmaLgam 的基础上随后还提出了一种启发式近似最优组合模型[43].他们发现,尽管已经存

在很多基于信息检索的缺陷预测技术,但是 VSM 模型及其变体仍然比其他 9 种模型表现的更好.基于此,他们组合各种 VSM 变体,通过遗传算法、利用历史的缺陷报告文本来求解可能的 VSM 组合,并将其组成建模为优化问题.在 AspectJ,Eclipse 和 SWT 的实验表明,复合模型将前 5 预测平均精度、MAP 和 MRR 分别提高了 18%,21%和 11%.Wang 等还将复合模型整合到 AmaLgam 中,结果表明它将 AmaLgam 的前 5 预测平均精度、MAP 和 MRR 分数分别提高了 8%,14%和 7%.

4.7 其他应用

Thung 等人将缺陷定位技术集成到流行的 Bugzilla 缺陷跟踪系统中[63].以前的文献中提出了许多缺陷定位方法,其中一些已作为可公开下载的原型发布.遗憾的是,这些现有技术和原型未集成到缺陷跟踪系统中.这使得从业者很难将缺陷定位技术整合到他们的日常软件开发和调试中.为解决这样的问题,Thung 等人开发了 BugLocalizer,一个实现了 BugLocator 的 Bugzilla 扩展.开发人员可以在 Bugzilla 中安装 BugLocalizer 插件,并使用其 BugLocator 功能.给定一个新的缺陷报告,BugLocalizer 将返回最有可能出错文件的排序列表.

Wang 等人提出一个实验平台,允许研究人员可比较地和可重复地评估缺陷定位技术[64].他们发布了一个可扩展的 Web 应用程序 BOAT 包含数千个真实的缺陷源代码文件及其缺陷报告.使用 BOAT,研究人员可以上传他们自己缺陷定位技术的可执行文件,并测试他们的算法对于 BOAT 中缺陷报告的有效性.他们还可以将自己缺陷定位技术的与其他研究人员上传的方法进行比较,对比其有效性.BOAT 预装了几个缺陷定位技术,包括一个简单的基于 VSM 的方法和 BugLocator[11].

Le 等提出了一种技术,可以预测基于信息检索的缺陷定位技术对特定缺陷报告的有效性[65].他们发现基于信息检索的缺陷定位技术有时对许多缺陷报告的定位表现非常糟糕,实际的缺陷文件在返回的排序列表中排名非常低.在这种情况下,对于开发人员来说,传统的调试方法比基于信息检索的自动化缺陷定位技术更好.如果这种情况经常发生,开发人员会对这种自动调试工具失去信心,并可能放弃这些工具,例如[66][67]所述.为了缓解现有基于信息检索的缺陷定位技术的可靠性问题,Le 等人构建了一个预报系统 APRILE(Automated PRediction of Ir-based bug Localization's Effectiveness),可以预测基于信息检索的缺陷定位技术为特定缺陷报告生成的排序列表是否可信.APRILE 将缺陷定位实例(即缺陷定位技术在缺陷报告上的应用)建模为一组特征.一共 4 组特征被纳入考量:从基于信息检索的缺陷定位工具返回文件的可疑度分数中提取的特征,从缺陷报告的单词中提取的特征,从缺陷报告学习的主题模型中提取的特征,以及从缺陷报告的元数据中提取的特征(比如缺陷报告的优先级,缺陷报告的严重性等).然后基于这些特征,通过支持向量机(SVM)构建一个模型,可以区分有效和无效的缺陷定位实例.在实验中,作者使用 APRILE 来预测 BugLocator[11],BLUiR[18]和 AmaLgam[42]在三个流行的 Java 项目(即 AspectJ,Eclipse 和 SWT)的三千多个缺陷报告上的有效性.实验结果表明,APRILE 可以准确识别有效和无效的缺陷定位实例,它的 F-measure 值接近 70%.

Xia 等人[68]发现全球用户提交的缺陷报告可能不是开发人员熟悉的语言,而是另一种语言.现有的缺陷定位技术无法在这种情况下很好地工作,因为缺陷报告和源代码文件之间的常见词汇将接近于零.为了应对这一挑战,Xia 等提出构建含有多个在线翻译器的缺陷定位工具 CrosLocator,它可以将一种语言编写的缺陷报告转换为源代码文件标识符和注释所使用的目标语言.使用多个翻译器的原因是,每个翻译器可能都不完美,特别是缺陷报告中常常含有低质量文本.翻译后,每个缺陷报告都会有多个翻译版本.对每个版本,CrosLocator 应用 BugLocator[11]对源代码文件进行排名.这样,就会得到多个排名的源代码文件列表.然后将这些列表合并在一起以使用学习排名算法创建单个列表.

与上述专注于找到有缺陷源代码文件的研究不同,Wen 等提出 LOCUS,它不仅可以找到有缺陷的源代码文件,还可以找到缺陷是如何产生的[61].Locus 从软件变化中检索信息,并且它从缺陷报告、提交日志和源代码文件中,创建了两个语料库:即 NL(自然语言)语料库和 CE(代码实体)语料库.然后,Locus 基于在这两个语料库上构建三个模型进行推荐:即 NL 模型、CE 模型和增强模型.NL 模型使用余弦相似性来测量缺陷报

告和源代码之间的相似性.CE 模型从缺陷报告中提取形似代码的词汇,并将每个词与 CE 中的 token 进行比较.增强模型基于这样的假设,即最近提交的变更产生缺陷的可能性最大.最后,LOCUS 集成了这 3 个模型,得到了更好的性能.

4.8 IRBL在人工智能领域的研究

不仅在软件工程领域有很多 IRBL 研究,在人工智能领域也有学者进行过 IRBL 技术相关的研究.Nath 等人[79]提出的缺陷定位概率模型 TFLMs 可进行跨项目的缺陷定位,用包含多个历史项目的语料库进行模型训练,学习缺陷的重复模式、上下文关系.TFLMs 为每代码行的缺陷指示变量定义了联合概率分布,且能融合其他的信息源,如程序覆盖频谱相关指标.作者在四个 C 语言项目上进行实验,验证了 TFLMs 模型缺陷定位的有效性.

Huo 等人[80]提出全新的卷积神经网络 NP-CNN,结合语义信息和代码结构信息进行缺陷定位.NP-CNN 模型包含四个模块:输入层、语言内特征提取层、跨语言特征融合层和输出层.语言内特征提取层采用独立的卷积神经网络分别从自然语言和编程语言中提取特征,跨语言特征融合层采用全连接神经网络融合从缺陷报告和源代码文件中提取的中间特征,生成统一的特征表示形式,并用于确定源代码文件和缺陷报告间的相关性.在四个软件项目上的实验结果表明学习代码结构特征对于缺陷定位是有益的,NP-CNN 的性能优于当时其他缺陷定位方法.随后,Huo 等人在此基础上提出了缺陷定位性能更好的 LS-CNN 模型[81],增加了源代码语句的执行顺序特征,LS-CNN 模型结合了 CNN 算法和 LSTM 算法,CNN 算法用于抽取和学习语义特征和代码结构特征,LSTM 算法用于对源代码语句之间的顺序交互进行建模.实验结果显示引入代码执行顺序特征能够提升缺陷定位的效果.

Gupta 等人提出一种基于深度学习的技术 NeuralBugLocator[82],可以针对失败的测试来定位缺陷在程序中的位置,无需运行程序.NeuralBugLocator 技术的核心是一个新型的树卷积神经网络,它经过训练可以预测程序是否通过了给定的测试.为了定位缺陷,NeuralBugLocator 使用最先进的神经预测归因技术分析训练好的神经网络,查看程序的哪些行使其能够预测测试结果.实验结果表明,NeuralBugLocator 比当前两个基于程序频谱和一个基于语法差异的缺陷定位方法更为准确.

5 评价指标

在 IRBL 技术的研究中,为了验证和对比技术的效果,研究人员需要对提出的新的技术进行验证,用不同的评价指标对技术的性能进行量化评估.本节将详细介绍在 IRBL 技术中使用的评价指标.

5.1 前K预测精度

前 K 预测精度(Top K Accuracy),即缺陷相关文件在返回结果的排序前K（$K = 1,5,10$）个中的数量.给出缺陷报告,如果前K个查询结果包含至少一个应该修复该缺陷的文件,则认为该缺陷被成功定位.此度量值越高,缺陷定位技术性能越好.

5.2 平均倒数排名

平均倒数排名（MRR）是一个常用的信息检索技术评价指标[34].给定一个查询语句,其倒数排名是排序算法生成排序列表中第一个正确文档排名的倒数.MRR 是给定一组查询语句 BR,其所有正确文档倒数排名的平均值,计算公式如下:

$$MRR(R) = \frac{1}{|BR|}\sum_{b \in BR}\frac{1}{rank(b)}$$

在上述公式中,$rank(b)$表示,基于信息检索的缺陷定位技术下,第一个被正确找到的源文件的排名.在本文中,MRR 是给定一组缺陷报告 BR,在基于信息检索的缺陷定位算法的输出排名中,其所有第一个正确的源代码文件排名的倒数的平均值.

5.3 平均精度均值

平均精度均值（MAP）是衡量性能的标量,它已经被证明具有特别好的辨别力和稳定性来评估排序技术[34].与仅考虑第一个正确结果的前k预测精度和 MRR 不同,MAP 考虑所有正确的结果.对于单个查询, 其平均精度定义为,在每个相关的文档被检索到之前,前k个文档集获得的精度值的平均值.即对于单个缺陷报告,其平均精度定义为,在每个相关的文档被检索到之前,前k个源代码文件获得的精度值的平均值.它的计算方法如下:

$$AvgP(b) = \frac{\sum_{j=1}^{M} P(j) \times Rel(j)}{\#Relevant\ Source\ Code\ Files}$$

在上述公式中,M是排序列表中的候选源代码文件的数量,$Rel(j)$指位置j处的源代码文件是否相关（本文的例子中:是否是真实包含缺陷的源代码文件）,$P(j)$指截断位置j的精度,它的计算公式如下:

$$P(j) = \frac{\#relevant\ source\ code\ files\ in\ top\ j\ positions}{j}$$

然后,一组缺陷报告 BR 的 MAP 是 BR 中所有缺陷报告的平均精度的平均值:

$$MAP = \frac{\sum_{b \in BR} AvgP(b)}{|BR|}$$

在基于信息检索的缺陷定位中,为了修复缺陷,需要更改一系列源代码文件.使用 MAP 来测量基于信息检索的缺陷定位技术找到所有相关的源代码文件的平均性能.MAP 值越高,基于信息检索的缺陷定位技术性能越好.

在本文中,对于单个缺陷报告b,其平均精度定义为,在每个相关的文档被检索到之前,前k个源代码文件获得的精度值的平均值.

6 关键问题

虽然 IRBL 技术近年来获得了研究人员的大量关注,取得了诸多进展,但是仍然存在一些亟待解决的关键问题.本章从数据噪音处理、多源数据特征提取、技术重现和工程实践四个方面详述目前 IRBL 技术中存在的问题.

6.1 数据噪音处理

(1) **缺陷报告的误分类:**现有的 IRBL 技术利用缺陷跟踪系统中的已经修复的缺陷进行缺陷的定位.然而,最近有研究人员发现,存在很多缺陷跟踪系统中的缺陷报告反应的并不是缺陷,而是代码的重构、增强请求,或是文档的更改等等,这些不是缺陷的缺陷报告实际上是误分类的缺陷报告[69].使用误分类的缺陷报告进行缺陷的定位技术的训练,会对算法的性能产生影响.例如,Kochhar 等人发现测试用例相关的问题和维护任务相关的问题被标记为缺陷对 IRBL 技术的精度有显著的影响[10].

(2) **缺陷文件的误分类:**现有的 IRBL 技术在建立缺陷报告和有缺陷的文件的关联时,首先根据提交信息,通过启发式规则找到修复某一缺陷的提交记录;然后把这一提交路中的所有修改的文件视为这一缺陷对应的缺陷文件.然而,最近的研究发现在修复缺陷的提交中,很多文件的变更时非必需的,即不是用来修改缺陷的[70].将没有缺陷的文件视作有缺陷的文件,在训练 IRBL 算法时,会影响 IRBL 技术的性能.

(3) **缺陷报告的质量:**现有的 IRBL 技术将缺陷报告当作查询语句,然后对所有的源代码文件进行信息检索.然而,不是所有的缺陷报告都可以能够用来进行缺陷定位的[71][72][73].研究人员发现有些缺陷报告中已经指明哪些文件是有缺陷的,开发人员只需要根据缺陷报告进行修复即可,这一类的缺陷报告不应该被用于 IRBL 算法.缺陷报告中的冗余信息会成为 IRBL 算法的噪音,进而影响 IRBL 算法的效果.为此,有些研究人员将缺陷报告看作是单纯的查询语句,利用查询语句的重构技术来提升缺

陷报告的质量[74][19].Mills 等人[73]的实验结果表明缺陷报告中包含的词汇已足够制定有效查询所需的全部内容,使基于文本检索的缺陷定位成功,无需补充查询扩展,而改进基于文本检索的缺陷定位的下一步是如何能够制定接近最优的查询.还有些研究人员对缺陷报告是否能够用于 IRBL 进行判定,并告知只用 IRBL 工具的开发人员,以提升开发效率[75][76][71].

6.2 多源数据特征提取

现有的 IRBL 算法从多种软件开发文档中提取多种特征进行组合来提升 IRBL 算法的效果,包括源代码文件,缺陷报告,版本控制仓库,缺陷跟踪仓库,开发人员信息等等.然而这些信息的存在形式不同,IRBL 算法对不同形式信息源的数据信息进行充分的利用是一大技术难点.

此外,在 IRBL 算法中相关度的计算模型上存在不同的策略.对于相同的数据源,不同的研究人员采用不同的算法.例如,在如何提取文本中的信息时,有些研究人员赞成使用简单的 VSM 模型[11],有些研究人员则考虑了深度学习模型[32].不同的算法应用在不同的数据源,应该考虑不同数据源的特征,而关于那一种方法能够取得更好的效果仍没有结论.因而,对于不同数据源,应该使用何种算法来进行特征的提取,仍需要进一步的研究.

6.3 技术重现

虽然缺陷定位算法被广泛研究,但是研究人员仍然质疑 IRBL 算法的有效性.这是因为,大多数的缺陷存在于多行或多个程序方法中[7].现有的研究并不能通过 IRBL 算法返回的可疑文件列表找到全部的有缺的文件,并且并不是所有的 IRBL 算法返回的可以文件列表中的文件都是和缺陷文件是相关的[41]. Lee 等人发现文献中丰富的方法与现实中开发人员采用的基于信息检索的缺陷定位技术截然不同[77].为此,他们对最先进的基于信息检索的缺陷定位技术的进行全面的再现分析,以展示当前方法的实际表现,为进一步推进该研究领域提供了更新的基准,显示了不同评估策略的性能变化.他们发现,虽然不同的 IRBL 方法表现出相似的性能分数,但并不是每个技术都能完全适应研究对象,测试用例不会降低性能,甚至可以改善定位结果,另外,他们还发现利用重复的缺陷报告可以提升基于信息检索的缺陷定位技术的性能.

6.4 工程实践

尽管目前存在相关的工作试图将 IRBL 技术引至工业界[63],然而在工业界进行大规模的推广仍存在以下挑战:

IRBL 算法是否可以有效的帮助开发人员进行缺陷定位.高质量的缺陷报告可以帮助开发者有效的进行缺陷定位,但并不一定可以有效地帮助 IRBL 算法实现更好的效果[72].开发人员使用缺陷报告中的实体名称在源码文件中搜索来进行缺陷定位.当开发人员无法从缺陷报告中获得足够的信息时,IRBL 算法返回的可疑文件列表才会对开发者有很大的帮助.例如,使用 BugLocator 生成的列表,参与者查找和关注缺陷文件所需的时间中位数比不使用 BugLocator 时要短,尽管 BugLocator 没有帮助参与者在缺陷文件中找到缺陷的方法.Le 等人[76]发现,基于信息检索的缺陷定位技术有时对许多缺陷报告的缺陷定位表现并不理想:对于有些报告,实际的缺陷文件在返回的排序列表中排名较低.在这种情况下,对于开发人员来说,传统的调试方法比基于信息检索的缺陷定位技术更好.如果这种情况经常发生,开发人员会对这种自动调试工具失去信心,并可能放弃这些工具.

现有的 IRBL 算法对于其他编程语言的项目的有效性还未可知.现有的 IRBL 算法针对同一种编程语言的项目(Java).然而在实际的开发过程中,开发者可能利用不同的语言完成不同的任务,或者在同一个项目中使用不同的编程语言.尽管研究人员已经研究了某些算法在某些语言上的适用性,同时也指出了处理不同语言是的挑战.Saha 等人研究了缺陷定位技术是否适用于 C 项目[78].他们的实证研究很好地补充了现有的研究;大多数 IR 缺陷定位技术都是对 Java 项目的评估,目前还不清楚它们是否同样适用于用包括 C 在内的其他常用语言编写的程序.作者创建了一个基准数据集,包含来自 5 个流行的 C 项目

（Phyton,GDB,WineHQ,GCC 和 Linux Kernel）,总计 7500 个缺陷报告,并重用现有数据集,其中包含来自 3 个流行的 Java 项目（SWT,AspectJ 和 Eclipse）的缺陷报告.然后,他们比较了被命名为 BLUiR[18] 的 IR 缺陷定位技术应用于 C 和 Java 项目的有效性.他们发现:IRBL 算法对于 C 项目和 Java 项目同样有效;将 IRBL 算法应用于 C 项目的主要挑战是需要处理预处理器指令和宏.然而对与其他语言,研究人员需要面对的挑战还未可知.

缺乏 IRBL 算法训练环境.IRBL 算法需要利用历史已修复的缺陷对算法进行训练.如果在实际的工程实践中,已修复的缺陷记录的与之对应的源码文件不够准确,或项目历史数据缺失,将对模型的训练带来巨大的挑战.

7 未来展望

针对上一章节所总结的关键问题,本章围绕数据集构建、复杂模型构建、工程实践和缺陷修复四个方面,展望 IRBL 缺陷定位研究的未来趋势.

7.1 构建高质量数据集,净化IRBL算法训练环境

实证研究表明,现有的 IRBL 算法所采用的数据集方法存在多种多样的噪音,降低数据集质量,导致 IRBL 模型训练环境不够准确.例如,缺陷的误分类[69],缺陷报告中已经指出缺陷文件[72]等.缺陷跟踪系统中有大量的缺陷报告被错误分类,Kocchar 等人发现误分类的数据对于 IRBL 算法的结果存在显著性影响.同时,数据集中的缺陷报告的质量也是参差不齐,部分已定位的缺陷报告的文本描述中已经指定包含缺陷程序模块,不需要额外的缺陷定位方法.实验表明 IRBL 方法对于那些已定位或部分定位的缺陷报告比那些未定位的缺陷报告准确率高得多,对评估 IRBL 算法缺陷定位性能影响很大.因而处理这些噪音数据,并且减少噪音数据对 IRBL 算法的影响,净化算法的训练环境,有利于构建更准确的 IRBL 模型,并进一步提高 IRBL 技术的有效性.

7.2 结合多维特征,构建复杂模型

如今大型软件项目开发和维护过程大多会利用多个系统进行.例如,利用缺陷跟踪系统、版本控制系统、代码审查系统等.现有的 IRBL 技术已经利用了这些系统中的多种特征来提高算法的有效性.例如,利用已经修复的相似缺陷[11],利用最近经常修改的文件[27]等.现有的 IRBL 技术简单的利用这些信息进行组合.然而,近年来,深度学习技术已经成为机器学习的热点的领域.Lam 等人[32]首先在 IRBL 技术中使用了 DNN.将目前深度学习领域大量研究与应用的技术,如卷积神经网络等应用到 IRBL 技术中,将有可能进一步提升 IRBL 的性能.如何有效的经济将这些技术应用到 IRBL 技术中需要未来工作的进一步分析与研究.此外,Le 等人[22]提出的一种多元自适应的缺陷定位方法 AML 能够将基于程序频谱的缺陷定位方法与 IRBL 技术融合在一起,生成更准确的复杂缺陷定位模型.由此想到,静态缺陷分析工具 Findbugs、缺陷预测方法等都可能对缺陷定位性能有所增益,如何将 IRBL 技术与利用测试用例的动态程序频谱缺陷定位技术、静态 Findbugs 工具、缺陷预测技术等进行多元融合,形成复合的缺陷定位模型,进一步提升缺陷定位的能力,将是缺陷定位领域的一个突破方向.

7.3 结合数据环境,提升模型实用性

现有的 IRBL 算法大多处于实验研究阶段,想要在工程实践中大量应用还存在诸多挑战.Wang 等人研究了基于信息检索的缺陷定位工具在现实场景中的有效性[72].他们将 BugLocator 应用于 4 个项目（SWT,JodaTime,ZXing 和 AspectJ）的缺陷报告,进行了分析研究和用户研究.他们发现当程序实体名称存在与否,BugLocator 的有效性存在统计上的显著差异.Lee 等人发现文献中丰富的方法与现实中开发人员采用的基于信息检索的缺陷定位技术截然不同[77].为此,他们对最先进的基于 IR 的缺陷定位技术的进行全面的再现分析.以展示当前方法的实际表现,为进一步推进该研究领域提供更新的基准,显示不同评估策略的性能变化.他们发现,虽然不同的 IRBL 方法表现出相似的性能分数,但并不是每个技术都能完全适应研究对象.如何

结合实际的数据环境,进一步提升 IRBL 模型在工程实践中的准确性和泛化性还需进一步的研究分析.

7.4 结合历史缺陷,指导缺陷修复

缺陷定位的一个热点应用领域是对下一步的缺陷修复提供指导信息.目前 IRBL 技术应用到缺陷修复还存在一些难点问题.首先,IRBL 技术的定位准确度还不够高,而且定位粒度较粗糙,只能定位到方法层面,开发人员只能按照给定的推荐结果列表逐一审查,仍然需要花费大量精力找到缺陷具体位置并修复.其次,人工提交的缺陷报告质量存在较大差异,在影响缺陷定位性能的同时也影响了开发人员对于缺陷的理解,造成的理解偏差进一步增大了缺陷修复的难度.未来的研究若能自动化生成缺陷报告,自动化整理缺陷相关信息,从而提高缺陷报告质量,将大大降低 IRBL 技术数据处理的难度,进一步提高 IRBL 技术定位的准确度和通用性,同时也能帮助开发人员快速理解缺陷,促进缺陷自动化修复的研究.另一方面,历史缺陷报告和修复记录蕴含着大量修复指导信息,根据代码的文本特征和结构特征搜索相似的历史缺陷代码,或者根据缺陷报告的文本特征和非文本特征搜索相似的历史缺陷报告,都能检索到相似缺陷,然后追溯修复该历史缺陷的软件变更,获取修复代码,进而综合当前缺陷的上下文信息和缺陷定位的结果,可推荐给开发人员备选缺陷修复代码列表,帮助他们理解并快速修复缺陷.

8 总结

近年来,IRBL 由于技术计算成本低,对程序语言更具有普适性,成为缺陷定位领域的研究热点.本文围绕 IRBL 技术的数据预处理、相关度计算与整合、模型评估等方面,梳理并总结了当前的研究进展和关键问题,并展望了未来的研究趋势.主要工作如下:(1)梳理了 IRBL 中数据预处理的过程和信息检索通用方法;(2)对 IRBL 技术中利用的数据特征进行了详细的分类和总结;(3)总结了技术评估中使用的性能评估指标;(4)归纳出了 IRBL 技术的关键问题;(5) 最后展望了 IRBL 技术的未来发展.

References:

[1] IEEE standard classification for software anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), pages 1–23, Jan 2010.

[2] Software errors cost u.s. economy $59.5 billion annually: Nist assesses technical needs of industry to improve software-testing.

[3] Thomas D. LaToza, Gina Venolia, and Rob DeLine. Maintaining mental models: a study of developer work habits. In ICSE '06: Proceedings of the 28th international conference on Software engineering, pages 492–501. ACM, May 2006.

[4] N. Serrano and I. Ciordia. Bugzilla, itracker, and other bug trackers. IEEE Software, 22(2):11–13, March 2005.

[5] Y. Fan, X. Xia, D. Lo, and A. E. Hassan. Chaff from the wheat: Characterizing and determining valid bug reports. IEEE Transactions on Software Engineering, pages 1–1, 2018.

[6] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. IEEE Transactions on Software Engineering, 42(8):707–740, Aug 2016.

[7] Lucia, F. Thung, D. Lo, and L. Jiang. Are faults localizable? In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pages 74–77, June 2012.

[8] M. Renieres and S. P. Reiss. Fault localization with nearest neighbor queries. In 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings., pages 30–39, Oct 2003.

[9] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. Bug localization using latent dirichlet allocation. Information and Software Technology, 52(9):972–990, 2010.

[10] Pavneet Singh Kochhar, Tien-Duy B Le, and David Lo. It's not a bug, it's a feature: does misclassification affect bug localization? In Proceedings of the 11th Working Conference on Mining Software Repositories, pages 296–299. ACM, 2014.

[11] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In 2012 34th International Conference on Software Engineering (ICSE), pages 14–24. IEEE, 2012.

[12] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 404–415, May 2016.

[13] Zellig S Harris. Distributional structure. Word, 10(2-3):146–162, 1954.

[14] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.

[15] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In Advances in neural information processing systems, pages 2177–2185, 2014.

[16] Martin F Porter. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.

[17] Shivani Rao and Avinash Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In Proceedings of the 8th Working Conference on Mining Software Repositories, pages 43–52. ACM, 2011.

[18] Ripon K Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E Perry. Improving bug localization using structured information retrieval. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 345–355. IEEE, 2013.

[19] Bunyamin Sisman and Avinash C Kak. Assisting code search with automatic query reformulation for bug localization. In Proceedings of the 10th Working Conference on Mining Software Repositories, pages 309–318. IEEE Press, 2013.

[20] Xin Ye, Razvan Bunescu, and Chang Liu. Learning to rank relevant files for bug reports using domain knowledge. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 689–699. ACM, 2014.

[21] Laura Moreno, John Joseph Treadway, Andrian Marcus, and Wuwei Shen. On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization. In 2014 IEEE International Conference on Software Maintenance and Evolution, pages 151–160, Victoria, BC, Canada, September 2014. IEEE.

[22] Tien-Duy B Le, Richard J Oentaryo, and David Lo. Information retrieval and spectrum based bug localization: better together. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pages 579–590. ACM, 2015.

[23] X. Ye, R. Bunescu, and C. Liu. Mapping bug reports to relevant files: A ranking model, afine-grained benchmark, and feature evaluation. IEEE Transactions on Software Engineer-ing, 42(4), 2016.

[24] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. Source code retrieval for bug localization using latent dirichlet allocation. In 2008 15th Working Conference on Reverse Engineering, pages 155–164. IEEE, 2008.

[25] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N Nguyen. A topic-based approach for narrowing the search space of buggy files from a bug report. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pages 263–272. IEEE Computer Society, 2011.

[26] Y. Uneno, O. Mizuno, and E. Choi. Using a distributed representation of words in localizing relevant files for bug reports. In 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), pages 183–190, Aug 2016.

[27] Hoang T, Oentaryo R J, Le T D B, et al. Network-clustered multi-modal bug localization[J]. IEEE Transactions on Software Engineering, 2018, 45(10): 1002-1023.

[28] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.

[31] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In 2014 IEEE International Conference on Software Maintenance and Evolution, pages 181–190. IEEE, 2014.

[32] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 476–481, Nov 2015.

[33] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. Bug localization with combination of deep learning and information retrieval. In Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on, pages 218–229. IEEE, 2017.

[34] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. Modern information retrieval, volume 463. ACM press New York, 1999.

[35] Stephen W. Thomas, Meiyappan Nagappan, Dorothea Blostein, and Ahmed E. Hassan. The impact of classifier configuration and classifier combination on bug localization. IEEE Transactions on Software Engineering, 39:1427–1443, 10 2013.

[36] Valentin Dallmeier and Thomas Zimmermann. Extraction of bug localization benchmarks from history. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pages 433–436. ACM, 2007.

[37] Yun Zhang, David Lo, Xin Xia, Giuseppe Scanniello, Tien-Duy B Le, and Jianling Sun. Fusing multi-abstraction vector space models for concern localization. Empirical Software Engineering, 23(4):2279–2322, 2018.

[38] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empirical Software Engineering, pages 1–61, 2019.

[39] Wen Zhang, Ziqiang Li, Qing Wang, and Juan Li. Finelocator: A novel approach to method-level fine-grained bug localization by query expansion. Information and Software Technology, 110:121–135, 2019.

[40] Rafi Almhana, Wiem Mkaouer, Marouane Kessentini, and Ali Ouni. Recommending relevant classes for bug reports using multi-objective search. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pages 286–295. ACM, 2016.

[41] Matthew Beard, Nicholas Kraft, Letha Etzkorn, and Stacy Lukins. Measuring the accuracy of information retrieval based bug localization techniques. In 2011 18th Working Conference on Reverse Engineering, pages 124–128. IEEE, 2011.

[42] Shaowei Wang and David Lo. Version history, similar report, and structure: Putting them together for improved bug localization. In Proceedings of the 22nd International Conference on Program Comprehension, pages 53–63. ACM, 2014.

[43] Shaowei Wang, David Lo, and Julia Lawall. Compositional vector space models for improved bug localization. In 2014 IEEE International Conference on Software Maintenance and Evolution, pages 171–180. IEEE, 2014.

[44] Shaowei Wang and David Lo. Amalgam+: Composing rich information sources for accurate bug localization: Composing rich information sources for accurate bug localization. Journal of Software: Evolution and Process, 28, 10 2016.

[45] X. Ye, R. Bunescu, and C. Liu. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Transactions on Software Engineering, 42(4), 2016.

[46] Michael Rath, David Lo, and Patrick Mäder. Analyzing requirements and traceability information to improve bug localization. Proceedings of the 15th International Conference on Mining Software Repositories. 2018.

[47] Tezcan Dilshener, Michel Wermelinger, and Yijun Yu. Locating bugs without looking back. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR’16, pages 286–290, New York, NY, USA, 2016. ACM.

[48] Takahashi Aoi, Sae-Lim Natthawute, Hayashi Shinpei, and Saeki Motoshi. A preliminary study on using code smells to improve bug localization. In Proceedings of the 26th Conference on Program Comprehension, ICPC, Gothenburg, Sweden, May 27-28, 2018, pages 324–327, 2018.

[49] Klaus Changsun Youm, June Ahn, and Eunseok Lee. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 82:177–192, 2017.

[50] Steven Davies, Marc Roper, and Murray Wood. Using bug report similarity to enhance bug localisation. In 2012 19th Working Conference on Reverse Engineering, pages 125–134. IEEE, 2012.

[51] Gregory Gay, Sonia Haiduc, Andrian Marcus, and Tim Menzies. On the use of relevance feedback in ir-based concept location. In 2009 IEEE International Conference on Software Maintenance, pages 351–360. IEEE, 2009.

[52] Xuan Huo, Ferdian Thung, Ming Li, David Lo, and Shu-Ting Shi. Deep transfer bug localization. IEEE Transactions on Software Engineering, 2019.

[53] Sisman, Bunyamin, and Avinash C. Kak. Incorporating version histories in information retrieval based bug localization. 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 2012.

[54] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. The Impact of Automated Parameter Optimization on Defect Prediction Models. IEEE Transactions on Software Engineering, pages 1–1, 2018.

[55] Zhendong Shi, Jacky Keung, Kwabena Ebo Bennin, and Xingjun Zhang. Comparing learning to rank techniques in hybrid bug localization. Applied Soft Computing, 62:636–648, 2018.

[56] Yan Xiao, Jacky Keung, Kwabena E Bennin, and Qing Mi. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 105:17–29, 2019.

[57] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Extracting structural information from bug reports. In Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR '08, pages 27–30, New York, NY, USA, 2008. ACM.

[58] A. Schroter, A. Schröter, N. Bettenburg, and R. Premraj. Do stack traces help developers fix bugs? In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pages 118–121, May 2010.

[59] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. Crashlocator: locating crashing faults based on crash stacks. In Proceedings of the 2014 International Symposium on Software Testing and Analysis, pages 204–214. ACM, 2014.

[60] Tung Dao, Lingming Zhang, and Meng Na. How does execution information help with information-retrieval based bug localization? In IEEE/ACM International Conference on Program Comprehension, 2017.

[61] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. Locus: Locating bugs from software changes. In 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 262–273. IEEE, 2016.

[62] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, pages 683–694. Association for Computing Machinery, Inc, 2019.

[63] Ferdian Thung, Tien-Duy B. Le, Pavneet Singh Kochhar, and David Lo. BugLocalizer: integrated tool support for bug localization. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014, pages 767–770, Hong Kong, China, 2014. ACM Press.

[64] Xinyu Wang, Bo Zhou, David Lo, Xin Xia, Xingen Wang, Pavneet Singh Kochhar, Yuan Tian, Xiaohu Yang, Shanping Li, and Jianling Sun. BOAT: an experimental platform for researchers to comparatively and reproducibly evaluate bug localization techniques. In Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014, pages 572–575, Hyderabad, India, 2014. ACM Press.

[65] Tien-Duy B Le, Ferdian Thung, and David Lo. Predicting effectiveness of ir-based bug localization techniques. In 2014 IEEE 25th International Symposium on Software Reliability Engineering, pages 335–345. IEEE, 2014.

[66] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. Practitioners' expectations on automated fault localization. In Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, pages 165–176, New York, NY, USA, 2016. ACM.

[67] Chris Parnin and Alessandro Orso. Are automated debugging techniques actually helping programmers? In Proceedings of the 2011 international symposium on software testing and analysis, pages 199–209. ACM, 2011.

[68] Xin Xia, David Lo, Xingen Wang, Chenyi Zhang, and Xinyu Wang. Cross-language bug localization. In Proceedings of the 22nd International Conference on Program Comprehension, pages 275–278. ACM, 2014.

[69] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In 2013 35th International Conference on Software Engineering (ICSE), pages 392–401, San Francisco, CA, USA, May 2013. IEEE.

[70] Pavneet Singh Kochhar, Yuan Tian, and David Lo. Potential biases in bug localization: Do they matter? In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pages 803–814. ACM, 2014.

[71] Dongsun Kim, Yida Tao, Sunghun Kim, and Andreas Zeller. Where should we fix this bug? a two-phase recommendation model. IEEE transactions on software Engineering, 39(11):1597–1610, 2013.

[72] Qianqian Wang, Chris Parnin, and Alessandro Orso. Evaluating the usefulness of IR based fault localization techniques. In Proceedings of the 2015 International Symposium on Software Testing and Analysis - ISSTA 2015, pages 1–11, Baltimore, MD, USA, 2015. ACM Press.

[73] Chris Mills, Jevgenija Pantiuchina, Esteban Parra, Gabriele Bavota, and Sonia Haiduc. Are bug reports enough for text retrieval-based bug localization? In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018.

[74] O. Chaparro, J. M. Florez, and A. Marcus. Using observed behavior to reformulate queries during text retrieval-based bug localization. In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 376–387, Sep. 2017.

[75] Tien-Duy B. Le, Ferdian Thung, and David Lo. Will this localization tool be effective for this bug? mitigating the impact of unreliability of information retrieval based bug localization tools. Empirical Software Engineering, 22(4):2237–2279, Aug 2017.

[76] Tien-Duy B. Le, Ferdian Thung, and David Lo. Predicting Effectiveness of IR-Based Bug Localization Techniques. In 2014 IEEE 25th International Symposium on Software Reliability Engineering, pages 335–345, Naples, Italy, November 2014. IEEE.

[77] Lee Jaekwon, Kim Dongsun, F. Bissyandé Tegawendé, Jung Woosung, and Le Traon Yves. Bench4bl: reproducibility study on the performance of ir-based bug localization. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018, pages 61–72, 2018.

[78] Ripon K Saha, Julia Lawall, Sarfraz Khurshid, and Dewayne E Perry. On the effectiveness of information retrieval based bug localization for c programs. In 2014 IEEE International Conference on Software Maintenance and Evolution, pages 161–170. IEEE, 2014.

[79] Nath A, Domingos P. Learning tractable probabilistic models for fault localization. Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[80] Huo X, Li M, Zhou Z H. Learning unified features from natural and programming languages for locating buggy source code. IJCAI. 2016: 1606-1612.

[81] Huo X, Li M. Enhancing the Unified Features to Locate Buggy Files by Exploiting the Sequential Nature of Source Code. IJCAI. 2017: 1909-1915.

[82] Gupta R, Kanade A, Shevade S. Neural Attribution for Semantic Bug-Localization in Student Programs. Advances in Neural Information Processing Systems. 2019: 11861-11871.

[83] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):390−412 (in Chinese). http://www.jos.org.cn/1000-9825/4708.htm