软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(2):247−276 [doi: 10.13328/j.cnki.jos.006130]

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方法综述*

李政亮[1], 陈 翔[1,2], 蒋智威[1], 顾 庆[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](南通大学 信息科学技术学院,江苏 南通 226019)

通讯作者: 顾庆, 陈翔, E-mail: guq@nju.edu.cn, xchencs@ntu.edu.cn

摘 要: 基于信息检索的软件缺陷定位方法是当前软件缺陷定位领域中的一个研究热点.该方法主要分析缺陷报告文本和程序模块代码,通过计算缺陷报告和程序模块间的相似度,选取与缺陷报告相似度最高的若干程序模块,将其推荐给开发人员.对近些年国内外研究人员在该综述主题上取得的成果进行了系统的梳理和总结.首先,给出研究框架并阐述影响方法性能的 3 个重要因素——数据源、检索模型和场景应用;其次,依次对这 3 个影响因素的已有研究成果进行总结;然后,总结基于信息检索的软件缺陷定位研究中常用的性能评测指标和评测数据集;最后总结全文,并对未来值得关注的研究方向进行展望.

关键词: 软件维护;软件缺陷定位;信息检索;缺陷报告;程序模块

中图法分类号: TP311

中文引用格式: 李政亮,陈翔,蒋智威,顾庆.基于信息检索的软件缺陷定位方法综述.软件学报,2021,32(2):247−276. http://www.jos.org.cn/1000-9825/6130.htm

英文引用格式: Li ZL, Chen X, Jiang ZW, Gu Q. Survey on information retrieval-based software bug localization methods. Ruan Jian Xue Bao/Journal of Software, 2021,32(2):247−276 (in Chinese). http://www.jos.org.cn/1000-9825/6130.htm

Survey on Information Retrieval-based Software Bug Localization Methods

LI Zheng-Liang[1], CHEN Xiang[1,2], JIANG Zhi-Wei[1], GU Qing[1]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](School of Information Science and Technology, Nantong University, Nantong 226019, China)

Abstract: Information retrieval-based software bug localization is an active research topic in the domain of software fault localization. It first analyzes the contents of the bug reports and program modules. Then it calculates the similarity between the bug reports and program modules. Finally, it recommends the most similar program modules to developers when given a bug report. This paper presents a systematic survey of existing research achievements of the domestic and international researchers in recent years. First, a research framework is proposed and three key factors (i.e., data sources, retrieval model, and application scenario), which may influence the performance of bug localization methods are identified. Next, existing research achievements in these three key factors are discussed in sequence. Then, the performance evaluation measures and datasets commonly used in information retrieval-based bug localization are summarized. Finally, conclusions of this study are drawn and a perspective of the future work in this research area is discussed.

Key words: software maintenance; software bug localization; information retrieval; bug report; program module

在软件开发过程中,软件缺陷(software bug)难以避免.对需求理解的偏差、不合理的软件开发过程、抑或开

* 基金项目: 国家自然科学基金(61972192, 61202006, 61906085, 41972111); 第 2 次青藏高原综合科学考察研究项目(2019QZKK0204); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2019B14, KFKT2018B17)

Foundation item: National Natural Science Foundation of China (61972192, 61202006, 61906085, 41972111); Second Tibetan Plateau Scientific Expedition and Research Program (2019QZKK0204); Open Project of State Key Laboratory for Novel Software Technology at Nanjing University (KFKT2019B14, KFKT2018B17)

收稿时间: 2020-01-16; 修改时间: 2020-04-05, 2020-07-05; 采用时间: 2020-08-07; jos 在线出版时间: 2020-09-10

发人员的疏忽,均有可能在软件项目内引入软件缺陷.这些含有缺陷的软件产品在部署后,可能会产生无法预测的行为或结果,甚至会对人们的生产和生活方式产生重大影响.当检测到软件内存在缺陷时,开发人员需要借助软件调试定位并修复缺陷.传统的软件调试方法主要借助手工调试,在可疑代码处设置程序断点,随后重复执行失败的测试用例,并观察程序断点处的变量取值.不断缩小可疑代码的搜索范围,直至定位到缺陷代码.这种方法完全通过人工的方式完成,存在缺陷定位代价高、对缺陷报告信息利用不充分、费时费力等问题.因此,研究人员针对自动化软件缺陷定位方法展开了深入研究,试图在减轻开发人员负担的同时,提高缺陷定位的效率与精度.

已有的缺陷定位方法按照是否需要执行测试用例,可分为两类:动态缺陷定位方法和静态缺陷定位方法[1].其中:动态缺陷定位方法通过搜集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,以确定缺陷语句在被测程序内的可能位置;静态缺陷定位方法通过分析缺陷报告和程序模块的文本内容并提取特征,基于特定模型确定可疑的程序模块,并将其推荐给开发人员以辅助定位.动态缺陷定位方法基于测试用例的覆盖信息定位缺陷,通常可获得比静态缺陷定位方法更好的性能[2].然而在软件维护阶段,开发人员有时并不能获得足够的测试用例[3],仅能搜集到缺陷报告的信息.静态缺陷定位方法由于不需要搜集测试用例的执行信息,不会受到程序规模和编程语言等因素的影响,具有缺陷定位代价低和方法可扩展性强等特点,也颇受关注.其中,基于信息检索的缺陷定位(information retrieval-based bug localization,简称 IRBL)是目前静态缺陷定位研究中的热点,也是本综述重点关注的研究问题.

IRBL 可视为概念定位(concern localization)[4]或特征定位(feature localization)[5]的一个特例.当需要定位的概念或特征都为缺陷时,概念定位或特征定位又被称为缺陷定位.IRBL 输入为一个缺陷报告,输出为所有与该缺陷报告所述缺陷相关的程序模块.该问题在研究时面临缺陷报告和程序模块词汇不匹配和表示方式差异较大等问题,大部分研究人员从文本挖掘角度设计解决方案.为了对该研究问题进行系统的分析、总结和比较,我们首先在 IEEE Explore、ACM Digital Library、Elsevier、Springer、CNKI 等学术论文数据库中进行检索,检索时采用的英文关键词包括 bug localization、fault localization、information retrieval、bug localisation 等;然后,通过人工审查的方式移除掉与综述主题无关的论文,并通过查阅相关论文的参考文献和相关研究人员发表的论文列表来进一步识别出遗漏的论文;最终,我们筛选出与该研究问题直接相关的高质量论文共 87 篇(截止到 2019 年 12 月).

我们首先对每年发表的相关论文的累计总数进行统计,结果如图 1 所示.不难看出,每年发表的论文数整体上呈上升趋势.在 2013 年之前,每年发表的论文数较少(介于 0 至 4 篇).但在 2013 年之后,随着公开的数据集和共享的实验代码的不断增多,针对 IRBL 方法的研究逐渐成为静态软件缺陷定位领域内的一个主流研究方向.在最近 3 年,每年发表的论文总数平均可以达到 13 篇以上.

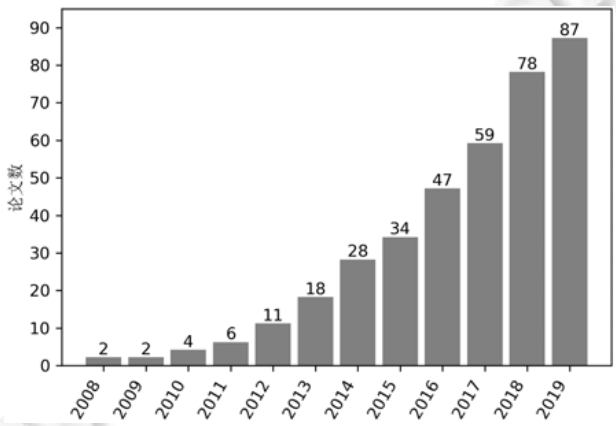

Fig.1 Statistics of annual cumulative number of papers on IRBL

图 1 IRBL 每年累计发表论文数的统计

随后,我们基于论文发表源对相关论文进行分类,并按照论文数从大到小进行排序,最终结果如表 1 所示(该表仅列出 CCF 推荐列表中论文数大于 1 的会议和期刊),其中,在软件工程领域权威期刊或会议上发表的论文数总结如下:ICSE 会议(5 篇)、ESEC/FSE 会议(4 篇)、ISSTA 会议(3 篇)、ISSRE 会议(2 篇)、TSE 期刊(5 篇)等.基于已搜索到的相关论文,我们对 IRBL 方法的已有研究工作进行了系统总结,并提出一个研究框架,从 3 个影响因素出发,对不同类型的方法进行系统的分析和比较,同时对常用的性能评测指标和评测数据集进行归纳.

Table 1 Statistics of the publication sources of papers on IRBL

表 1 针对 IRBL 论文的发表源统计

发表源类型	发表源简称	发表源全称	评级	论文数
会议	ICSME	Int'l Conf. on Software Maintenance and Evolution	B	10
	ASE	Int'l Conf. on Automated Software Engineering	A	6
	ICSE	Int'l Conf. on Software Engineering	A	5
	FSE	ACM SIGSOFT Symp. on the Foundations of Software Engineering	A	4
	ICPC	IEEE Int'l Conf. on Program Comprehension	B	4
	MSR	Mining Software Repositories	C	4
	ISSTA	Int'l Symp. on Software Testing and Analysis	A	3
	APSEC	Asia-Pacific Software Engineering Conf.	C	3
	IJCAI	Int'l Joint Conf. on Artificial Intelligence	A	2
	ISSRE	Int'l Symp. on Software Reliability Engineering	B	2
期刊	IST	Information and Software Technology	B	7
	TSE	IEEE Trans. on Software Engineering	A	5
	JSEP	Journal of Software: Evolution and Process	B	2
	ESEM	Empirical Software Engineering	B	2

本文第 1 节总结 IRBL 方法的研究框架,并识别出 3 个重要的影响因素(数据源、检索模型和场景应用).第 2 节~第 4 节分别对这 3 个影响因素进行详细的分析与总结.第 5 节介绍常用的性能评测指标.第 6 节介绍常用的评测数据集.最后总结全文并对未来可能值得关注的研究问题进行展望.

1 研究框架

IRBL 方法针对一个缺陷报告,尝试定位到可能包含缺陷的程序模块(其模块粒度根据实际定位需求可以设置为文件、类或方法等).其输入为缺陷报告和所有候选程序模块,通过建立程序模块语料库、建立索引、构建查询、检索和排序,推荐给开发人员一个有序列表(在该有序列表内,程序模块按照含有缺陷的可能性从高到低进行排序).

在缺陷定位过程中,一个典型的缺陷报告如图 2(a)所示,该缺陷报告来自软件项目 AspectJ.缺陷报告一般包含标题(title)、描述(description)、评论(comment)、产品(product)、组件(component)、版本号(version)和报告提交时间等信息,其对应的缺陷程序模块是 LazyTransfer.java 和 RTFTransfer.java 文件(如图 2(b)所示).缺陷报告中的标题总结缺陷问题,描述给出缺陷的症状以及触发缺陷的行为等,评论则包含评论者的反馈信息等.这些文本通常涉及缺陷代码中的词或片段,提供缺陷定位所需的大部分信息.然而总存在一些低质量的缺陷报告,其描述内容不清晰,甚至不包含评论内容.对于图 2(a)中缺陷报告而言,在评论中涉及到了缺陷程序模块的类名 RTFTransfer,这对缺陷的定位和验证起重要作用,在没有评论时,将无法提供足够的信息来进行缺陷定位,此时,这类缺陷报告需要使用其他额外信息来定位缺陷程序模块.在基于各类缺陷报告进行缺陷定位时,其典型研究框架如图 3 所示.

图 3 的上半部分总结 IRBL 的过程,该过程主要包括两个阶段:模型构建阶段和模型应用阶段.具体来说,模型构建阶段包括两个步骤.

(1) 挖掘缺陷跟踪系统、版本控制系统、API 文档等数据源,从中搜集缺陷报告、程序模块和提交日志等信息,建立缺陷报告与程序模块的关联,构建出缺陷定位数据集;同时,IRBL 的输入也由所搜集的数据源信息决定.

(2) 设计出衡量缺陷报告与程序模块相关度的特征,分析程序模块含有缺陷的概率,借助这些特征构建出

缺陷定位模型,即 IRBL 模型.

(a) 一个缺陷报告样例 (b) 缺陷程序模块的部分代码

Fig.2 A bug report and a corresponding buggy program module in AspectJ

图 2 AspecJ 项目中的某个缺陷报告和对应的缺陷程序模块

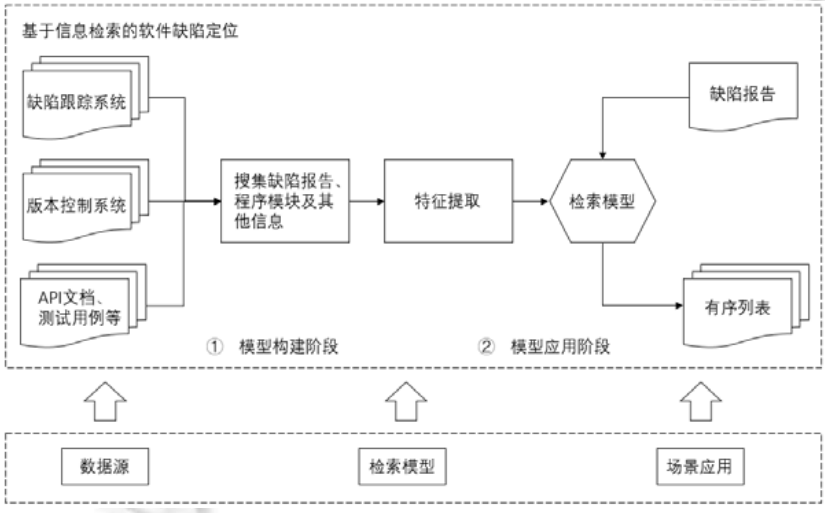

Fig.3 Research framework of IRBL

图 3 IRBL 的研究框架

在模型应用阶段,当处理一个新的缺陷报告时,基于前一阶段流程提取特征,并应用训练好的缺陷定位模型完成对该缺陷报告的定位,将缺陷程序模块的有序列表推荐给开发人员.

根据缺陷定位应用场景的不同需求,其缺陷定位粒度和结果不同,需要的输入信息、特征提取方式和检索过程都会稍有不同,最后输出不同的有序列表.

通过分析上述软件缺陷定位过程,我们识别出影响缺陷定位性能的 3 个重要因素(如图 3 的下半部分所示).

(1) 数据源.对开发人员而言,除可获得缺陷报告信息和程序模块代码之外,还可挖掘出软件的版本历史信息和程序模块的变更信息等,IRBL 方法的输入决定了在缺陷定位过程中所能包含信息的上限,从更多信息角度构建出与缺陷报告所述软件缺陷存在强相关性的特征,这是构建高质量缺陷定位模型的关键.

(2) 检索模型.IRBL 检索模型主要基于词法和语义角度进行检索,因此,本文将已有的检索模型方法分为两类:基于文本相似度的方法和基于语义相似度的方法.

(3) 场景应用.针对不同场景,不同开发人员的定位需求不同,因此对缺陷定位粒度和返回结果的要求也各不相同.本文将缺陷定位的应用场景按其定位粒度划分为 4 类:基于文件的缺陷定位、基于函数的缺陷定位、基于语句的缺陷定位和基于代码变更(code change)的缺陷定位.

2 数据源的分析

如何从缺陷跟踪系统和版本控制系统中挖掘出与缺陷报告所述软件缺陷存在强相关性的信息,是构建高质量缺陷定位模型的关键.早期的研究工作主要集中于分析缺陷报告的内容,重点关注缺陷报告的总结和描述内容,挖掘程序模块与缺陷报告之间的相关性.但存在部分缺陷报告本身只包含少量信息的情况,导致缺陷定位性能降低.近些年来,许多研究工作开始集中于从其他信息源角度挖掘与缺陷程序模块相关的信息,并设计出合适的特征,这些特征主要从缺陷报告与程序模块的相关性以及程序模块本身的缺陷概率两个方面来衡量,不同的输入信息也是围绕这两个方面进行考虑.本节将重点从这个角度对已有研究工作进行系统的总结,首先不同的数据源可划分为 3 类:(1) 缺陷报告和程序模块;(2) 项目元数据;(3) 测试用例的执行信息.

从缺陷跟踪系统中可挖掘出缺陷报告信息,从版本控制系统(例如 Git、CSV 或 SVN 等)中可挖掘出程序模块代码和项目元数据(即版本历史信息和代码变更信息等).测试用例的执行信息通过对项目运行多个测试用例生成并获取.缺陷报告和程序模块作为 IRBL 的主体,是必不可少的.在已公开数据集中,大部分数据集仅提供缺陷报告和程序模块代码,项目元数据和测试用例的执行信息需额外获取.

2.1 缺陷报告和程序模块

在 IRBL 研究的早期阶段,大部分研究工作仅通过分析缺陷报告和程序模块代码来定位可能包含缺陷的程序模块.在以缺陷报告和程序模块作为输入时,如图 2 中所示,缺陷报告包含总结、描述、元数据(版本、平台、操作系统、缺陷严重程度和汇报者)和评论,程序模块包含注释、标识符和字符串字面值等信息.大部分 IRBL 方法并不会使用全部的信息,通常被使用到的是缺陷报告中的总结和描述以及程序模块中的标识符和注释.对于缺陷报告和程序模块的文本内容,首先要进行预处理.除此之外,会根据缺陷报告中是否包含特殊文本(如堆栈跟踪(stack trace)以及文件名等)进行特殊处理.该小节首先介绍预处理过程,然后从缺陷报告与程序模块相关性以及程序模块缺陷概率这两个角度对已有方法进行总结.

2.1.1 预处理过程

原始的缺陷报告和程序模块文本具有很多噪音词汇及数据.早期的 IRBL 方法就已经开始针对缺陷报告和程序模块的文本执行数据预处理,以达到减少噪音和提高缺陷定位性能的目的.数据预处理过程主要包括文本标准化(text normalization)、停用词(stopword)移除以及词干提取(stemming)这 3 步.

- 文本标准化:首先从文档中移除特殊字符和标点符号等单词,随后将文档切分成不连续的单词,对于源代码中的标识符,还需依据驼峰式分割法(camel case splitting)来进一步切分为更小的单词.例如,标识符“processFile”可以被切分为单词“process”和“file”.

- 停用词移除:针对标准化的文档,借助英文中的常用停用词进行移除.停用词是指在各个文档中频繁出现,但在区分不同文档时作用极小的单词,如“a”“on”等单词.对程序模块而言,需移除程序语言相关的停用词,如“int”“private”等.
- 词干提取:将单词转换成对应的词干形式.例如,单词“localized”“localization”“localize”和“locally”都可还原为词干“local”.

许多研究人员同时采用这 3 步预处理过程,这些过程对减小缺陷报告和程序模块的词汇差异性有显著帮助.Thomas 等人[6]和 Tantithamthavorn 等人[7]在探索分类器配置对缺陷定位性能影响时,对 3 步预处理过程进行排列组合,并最终形成 8 种预处理方式.实验结果表明:对所有的实验项目和检索模型而言,同时使用 3 步预处理过程的检索结果相较于使用其他排列组合方式的结果始终是最优的.Hill 等人[8]则对词干提取在缺陷定位(或概念定位)中的影响进行了研究,比较了 5 种词干提取器(stemmer):Porter、Snowball、Kstem、Paice 和 Mstem.结果表明:不同的词干提取方法对长缺陷报告的影响较小,对短缺陷报告的影响则较大.其原因为:短缺陷报告的文本组织更加接近自然语言,而长缺陷报告的文本则通常会包含代码片段.

2.1.2 缺陷报告与程序模块相关性

缺陷报告与程序模块的相关性度量主要从文本信息、堆栈跟踪、相似缺陷报告这 3 个角度出发.

(1) 文本信息

这类信息的常见输入是缺陷报告的总结和描述以及程序模块的文本内容,可采用文本检索方法直接计算出缺陷报告和程序模块之间的相关性,也可基于语义模型计算出缺陷报告和程序模块的相关性.

早期,Lukins 等人[9,10]从指定粒度的程序模块(可以是包、类或方法等)提取代码中的注释和标识符以形成文档集,缺陷报告仅使用描述部分的内容,通过 LDA(latent dirichlet allocation)检索模型查询与缺陷报告相关联的程序模块.Nguyen 等人[11]则使用缺陷报告的总结和描述内容作为查询,提取程序模块的注释和标识符形成对应的文档集.他们改进了主题模型 LDA,分别对缺陷报告和程序模块应用主题模型,并且训练集中缺陷报告的主题还受其对应的缺陷程序模块的主题分布参数影响,即它们共享相似的主题分布参数,训练好模型后,将其应用到新的缺陷报告上.Lal 等人[12]则考虑从字符级别的 *n*-gram 模型出发进行信息检索,不局限于单词.从字符角度考虑,具有可匹配拼写错误的单词、可匹配缩写词等优点,因此具有更高的鲁棒性.该方法使用的输入为缺陷报告的标题和描述以及程序模块的文件名和文件路径名,两两组合计算相似度,*n*-gram 模型的字符设置长度从 3 增长到 10.结果表明:字符级别的 *n*-gram 方法是有效的,且描述与文件名的组合相较于其他 3 个组合具有更高的预测性能.Zhang 等人[13]发现,缺陷报告中的元数据(组件和版本信息等)被多数 IRBL 方法忽略,因此提出一个修正的主题模型 L2SS+,并考虑了缺陷报告元数据.该方法假设缺陷报告的不同元数据项有不同的主题分布,将不同的主题分布累加起来作为缺陷报告的主题分布,然后计算缺陷报告和程序模块的关联度.实验结果表明,使用元数据信息能有效提高缺陷定位性能.Khatiwada 等人[14]从信息理论的角度分析和建立缺陷报告和程序模块之间的关联,通过词与词之间的联系捕捉文档间的语义信息.他们采用点间互信息 PMI(pointwise mutual information)和规范化 google 距离 NGD(normalized google distance).其中,PMI 度量两个词间的信息重叠或统计依赖程度,NGD 基于信息距离和 Kolmogorov 复杂度来衡量词之间的语义相关度.随后,该方法基于 PMI 或 NGD 将词之间的关联拓展到文档之间的关联.实验结果表明,该方法要优于 VSM 和 LSI 等通用文本模型.

使用文本信息来进行缺陷定位,依赖于缺陷报告和程序模块之间的共享词汇.Moreno 等人[15]对此进行了分析,他们发现:缺陷报告和软件系统中的大部分类文件都共享部分词汇,平均而言,与打补丁的类(即缺陷类)有更多共享词.缺陷类各部位按照与缺陷报告共享词的数量从大到小排序的结果为:注释(26.98%)、函数调用(20.28%)和类型名(12.69%)等,而各部位共享词的比率从大到小排序的结果为:类名(53.57%)、字符串面值(21.40%)和类型名(20.69%)等.他们还发现,用共享词的数量定义相关性有时比一些文本检索方法定位更准确.可以看出:在这类基于文本信息进行检索的 IRBL 方法中,缺陷报告的质量会直接影响到缺陷定位的性能.

(2) 相似缺陷报告

在缺陷跟踪系统中,存在重复或相似的缺陷报告[16].有研究人员假设:如果两个缺陷报告的文本相似度很

高,则这两个缺陷报告很可能对应相同的缺陷程序模块.因此,可从已修复的缺陷报告中搜索出与当前缺陷报告相似的缺陷报告.相似缺陷报告评分作为增强缺陷报告与缺陷程序模块相关性的评分,被多个方法使用.

Davies 等人[17]基于缺陷报告间的相似度,增强缺陷定位缺陷性能.他们为程序模块中的每个函数训练一个二分类决策树,其输入是基于缺陷报告描述生成的词向量,类标签表示缺陷报告是否与函数相关.当新的缺陷报告被提交后,每个分类器按顺序来预测标签.Zhou 等人[18]基于相似缺陷报告的假设,对相似缺陷报告的缺陷程序模块给一个基于缺陷报告间相似度的评分,并且考虑到程序模块的代码长度问题,采用了修正的 VSM(vector space model),相较于直接利用 VSM 和 LDA 等文本检索模型的计算方法,缺陷定位性能有了显著提升.

(3) 堆栈跟踪

缺陷报告的质量有高有低,有时缺陷报告中会包含程序执行失败后的堆栈跟踪信息.当缺陷报告包含堆栈跟踪信息时,开发人员首先检查堆栈跟踪中涉及到的程序模块来进行缺陷定位.Schroter 等人[19]发现,几乎 60%的带堆栈跟踪的缺陷报告都可以通过修改堆栈跟踪记录中的函数来修复.其中,大概有 40%的缺陷函数可以在堆栈跟踪的第 1 帧中发现,90%的缺陷函数可以在前 10 帧中发现.通过使用堆栈跟踪这一额外的输入信息,可以有效提高缺陷定位的性能,这类信息通常也被用来作为计算缺陷报告和程序模块相关性的补充.

Moreno 等人[20]提出了 Lobster 方法,结合文本相似度和堆栈跟踪相似度来定位缺陷.方法分析缺陷报告的堆栈跟踪信息,并定义出结构相似度,根据堆栈跟踪与指定程序模块的距离来衡量.他们假设该程序模块若与堆栈跟踪中的程序模块越接近,其包含缺陷的概率越大.其距离计算公式如下.

$$dist(stackTrace,c_j)=\min(\forall_{d\in stackTrace}shortPath(d,c_j)\cup\forall_{d\in stackTrace}shortPath(c_j,d)) \tag{1}$$

$$Score_{ST}(r_i,c_j)=1-\frac{\min(dist(stackTrace,c_j),\lambda)}{\lambda} \tag{2}$$

其中,$shortestPath(d,c_j)$表示程序模块 d 与程序模块 c_j 的最短距离,$dist(stackTrace,c_j)$衡量程序模块 c_j 与堆栈跟踪中所有程序模块的最短距离,$Score_{ST}(r_i,c_j)$衡量缺陷报告 r_i 中的堆栈跟踪与程序模块 c_j 的相关性评分.

Wong 等人[21]采取了和文献[20]类似的操作,将文本相似度和堆栈跟踪相似度相结合.他们发现了当前缺陷定位方法尚未解决的两个问题:大文件的噪音问题以及堆栈跟踪信息未使用的问题.他们针对性地将程序模块代码进行等长分割,比较缺陷报告和分割后的代码文本相似度以过滤噪音;而对堆栈跟踪信息的利用,则是通过对堆栈跟踪中出现的所有程序模块赋予一个缺陷模块增强评分,其计算公式如下.

$$Score(r_i,c_j)=\begin{cases}\dfrac{1}{rank_{c_j}}, & \text{if } c_j\in D_{trace} \text{ and } rank_{c_j}<10\\ 0.1, & \text{if } c_j\in D_{trace} \text{ and } rank_{c_j}\geqslant 10 \text{ or } c_j\in D_{import}\\ 0, & \text{otherwise}\end{cases} \tag{3}$$

其中,$rank_{c_j}$ 表示程序模块 c_j 在堆栈跟踪中的排名,D_{trace} 表示堆栈跟踪中的程序模块形成的集合,D_{import} 表示堆栈跟踪中的程序模块所调用的程序模块形成的集合.该方法对堆栈跟踪中的程序模块以及其程序模块所调用的模块依据其排名给出增强评分.

2.1.3 程序模块缺陷概率

程序模块自身因其代码规模或复杂度不同,其内部含有缺陷的概率也不同[22].有研究人员考虑从缺陷程序模块特性出发,从程序模块中挖掘出更多信息.

(1) 程序模块调用关系

Ye 等人[23]认为,复杂的代码比简单代码更容易引入缺陷.他们从句法特征的角度衡量代码的复杂度:(1) 程序模块复杂度随着它使用的类的数目的增多而增加;(2) 程序模块复杂度不仅依赖于它依赖文件的数目,也取决于每个依赖文件的复杂度;(3) 程序模块感知复杂度随使用次数增多而降低.基于这些观察,Ye 等人基于程序模块调用关系构建一个程序模块依赖图,使用 PageRank 和 HITS(hyperlink-induced topic search)等基于连接查找关键信息节点的算法分析复杂缺陷程序模块,以定位缺陷.Zhou 等人[24]从词性和程序模块调用的角度来增强缺陷定位的性能.他们分析缺陷报告词汇的词性,提高名词的权重;同时,考虑缺陷报告中的类名,将类名对应的

文件定义为命中文件,选择相似度评分最高的命中文件,提高其调用文件的评分.Chen 等人[25]没有直接利用程序模块间的调用关系,他们基于缺陷报告将程序模块建模成一个社交网络,进行隐连接分析.该方法基于假设:如果一个缺陷报告与两个程序模块相关,则这两个程序模块存在一个共引用关系(co-cited relationship),可构建隐式双向连接.通过分析所有历史缺陷报告可以构建出社交网络图,该方法基于改进的 PageRank 算法可估算每个程序模块的评分.当遇到一个新的缺陷报告时,计算缺陷报告与历史缺陷报告的相似度,由其对应的缺陷程序模块评分得出.

(2) 代码异味(code smell)

代码异味指那些在源代码中很可能引发深层问题的特征,如出现重复代码、过长函数、过大的类、过长参数列表等情况.代码异味检测将源代码作为输入,生成包含代码异味的程序模块列表及其异味类型或严重程度.

Takahashi 等人[26]对使用代码异味来提高缺陷定位性能进行了初步研究,他们将缺陷报告和程序模块的文本相似度以及代码异味严重程度进行线性拟合来定位缺陷.实验结果表明,代码异味在函数和类级别定位上都能显著提高缺陷定位的性能.

2.2 项目元数据

除了从缺陷报告和程序模块角度进行挖掘,开发人员可从版本控制系统中获取软件代码的变更历史信息,这类信息主要输入包括从 Git,CVS,SVN 等版本控制系统中抽取的项目元数据(metadata).其中,项目元数据指从版本控制系统中可直接获取的项目文件修改信息、历史变更记录或提交日志(commit log)等.通过分析日志等信息,研究人员可以获得程序模块的历史修改次数、程序模块的缺陷修复次数、程序模块上一次缺陷修复时间、一次缺陷修复提交中被修改的所有文件等信息.经研究人员观察发现:如果一个文件在最近的版本中被修改或实现某个新增功能,它有很大的概率会引入缺陷.这类信息从程序模块缺陷概率的角度提供额外信息.

Sisman 等人[27]首次使用版本信息来估计程序模块的先验缺陷概率,用于缺陷定位.他们基于缺陷预测的已有研究工作,认为文件修改历史和文件缺陷历史是比较好的文件潜在缺陷预测特征.因此,他们定义了修改历史先验(modification history based prior,简称 MHbP)和缺陷历史先验(defect history based prior,简称 DHbP)两个指标,其计算公式如下.

$$P_{MHbP}(f \mid C_k, S_k) = \frac{\sum_{i=1}^{k} I_m(f, s_i)}{\sum_{f' \in C_k} \sum_{i=1}^{k} I_m(f', s_i)} \tag{4}$$

$$P_{DHbP}(f \mid C_k, S_k) = \frac{\sum_{i=1}^{k} I_b(f, s_i)}{\sum_{f' \in C_k} \sum_{i=1}^{k} I_b(f', s_i)} \tag{5}$$

其中,

- C_k 表示第 k 次变更后的程序模块集合.
- S_k 表示第 k 次变更修改的程序模块集合.
- 函数 $I_m(f,s_i)$指文件 f 是否在第 s_i 次变更中被修改:被修改则为 1,否则为 0.
- $P_{MHbP}(f|C_k,S_k)$统计的是文件 f 的修改总次数占所有文件修改次数的比率.
- $P_{DHbP}(f|C_k,S_k)$与之类似,统计的是该文件的缺陷修复次数占所有文件缺陷修复次数的比率.

Tantithamthavorn 等人[28]使用联合变更历史来增强 BugLocator 方法[18]的缺陷定位性能.其中,联合变更是指在开发人员的一次代码提交中存在多个程序模块被修改变更的现象.他们利用一个假设:如果一个缺陷模块被修复,那些与缺陷模块同时被修改的程序模块也需要被修复.该方法为所有程序模块创建一个交互矩阵,记录之前程序模块被一起修改的次数,在将交互矩阵标准化后,选取与 BugLocator 排序结果中前 k 个程序模块关联的并且大于某个阈值的程序模块作为缺陷程序模块,同时推荐给开发人员.

Wang 等人[29]同样利用缺陷预测领域的研究成果,对程序模块依据与之相关的缺陷修复提交的个数进行排序(即程序模块缺陷修复历史).经验表明:过早的缺陷修复提交因时间原因对最新的缺陷定位影响不大,甚至会降低缺陷定位性能.他们因此修改 Google 开发人员提出的缺陷预测算法,搜集包含“fix”或“bug”的提交,并限制

这些提交在缺陷报告提交时间的 k 天以内;随后赋予与提交相关的程序模块一个怀疑率评分,其计算公式如下.

$$Score_V(c_i,k,FS)=\sum_{c'\in FS\wedge c_i\in c'}\frac{1}{1+e^{12(1-((k-t_{c'})/k))}} \tag{6}$$

其中,FS 为缺陷相关提交的集合,$t_{c'}$为提交 c'和输入缺陷报告提交时间的相隔天数.

2.3 测试用例的执行信息

开发人员可执行多个测试用例,分析测试用例的覆盖信息和执行结果,搜集频谱(spectrum)信息和切片(slicing)信息等.其中:频谱信息描述失败测试用例和成功测试用例执行的覆盖比例,可计算每个执行类或函数的可疑评分;程序切片描述影响某个程序点状态的所有类或函数,在测试用例失败时,只有对失败点有影响的表达式可能包含缺陷.这些测试用例的执行信息常用于动态缺陷定位[30],可衡量程序模块的缺陷概率.

Le 等人[31]利用频谱信息和文本信息,提出一种多模态方法,并定义了 3 类特征:基于本文的特征、基于频谱的特征和基于可疑词的特征.具体来说:文本特征考虑缺陷报告和程序模块的文本相似度,频谱特征仅考虑使用基于频谱的缺陷定位技术来计算相关度,基于可疑词的特征则根据单词出现在执行失败的函数中的次数占单词出现在所有函数中次数的比例计算.这 3 类特征通过一个线性整合器进行拟合,并采用逻辑回归进行优化.Hoang 等人[32]在 Le 等人[31]工作的基础上提出了基于网络聚类的多模态缺陷定位方法,采用联合优化的方式.该方法同时考虑上述 3 类特征以及缺陷报告之间和程序模块之间构建的关系图,损失函数由相似缺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.

Dao 等人[33]则研究了频谱信息、切片信息和覆盖信息对 IRBL 方法的影响,他们比较了 4 类结合方式:覆盖信息与 IRBL 相结合、切片信息与 IRBL 相结合、覆盖信息和频谱信息与 IRBL 相结合、切片信息和频谱信息与 IRBL 相结合.覆盖信息和切片信息可用来优化搜索空间,过滤掉失败测试用例没有执行到的程序实体;频谱信息可用来计算每个实体的怀疑率评分,并对排序进行调整.实验结果表明:通过覆盖信息和切片信息来优化排序列表,能有效提高缺陷定位准确率,频谱信息能更进一步提高函数级别的缺陷定位性能.

2.4 不同数据源的分析

表 2 对 IRBL 方法使用的数据源信息进行了统计,根据该表我们发现:缺陷报告和程序模块作为 IRBL 方法的必要输入,对每个 IRBL 方法都不可或缺,因此涉及的论文数最多.其他数据源涉及的论文数则明显偏少.研究人员主要从程序模块和缺陷报告的相关性考虑,挖掘多种信息,仅有少数方法考虑从程序模块缺陷概率的角度出发.在缺陷报告和程序模块相关性度量中,从文献[18,21]的参数设置比例我们发现:对 IRBL 方法而言,基于文本信息计算的相关度的权重显著大于基于相似缺陷报告和堆栈跟踪计算的相关度的权重,基于文本信息计算的相关度起决定作用,后两者起着优化作用.将缺陷报告与程序模块相关性和程序模块缺陷概率相比时,缺陷报告与程序模块相关性起主要作用,程序模块缺陷概率起次要作用[34].为此,有许多研究人员直接考虑从缺陷报告和程序模块的文本内容出发,关注于提升缺陷报告和缺陷程序模块的关联度,着重解决缺陷报告和程序模块的词汇不匹配问题.

Table 2 Statistics of data sources

表 2 数据源统计

数据来源	特征分类	作用	论文数
缺陷报告与程序模块	缺陷报告和程序模块相关性	衡量缺陷报告和程序模块的关联度	87
程序模块	程序模块缺陷概率	根据代码复杂度等因素衡量程序模块的缺陷概率	5
项目元数据	历史变更信息	根据文件修改及修复信息衡量程序模块的缺陷概率	15
测试用例执行数据	测试用例执行信息	根据测试用例执行情况衡量程序模块的缺陷概率	2

3 检索模型

为了解决缺陷报告和缺陷程序模块的不匹配问题,IRBL 检索模型从基于词法的模型发展到基于语义的模型,参数优化方式从简单的启发式方法发展到基于学习的优化方法.IRBL 检索模型发展依时间层次递进,可分

为基于文本相似度的方法和基于语义相似度的方法,下面将依次介绍这些方法.

3.1 基于文本相似度的方法

3.1.1 基于通用检索模型的方法

这类方法主要依赖缺陷报告和程序模块间词的相似度,将缺陷报告视为查询,将程序模块视为语料库,进行文本检索.经典的文本检索模型都可应用于 IRBL.目前,在 IRBL 中被使用的经典模型有向量空间模型 VSM、主题模型和概率检索模型等.

(1) 向量空间模型(VSM)

VSM 基于词袋模型,将缺陷报告和程序模块表示为对应的向量,根据 TF(term frequency)和 IDF(inverse document frequency)计算方式的不同可有多个变种,随后通过计算向量间的余弦相似度来衡量缺陷报告和程序模块间的相似性.

Wang 等人[35]根据 5 种 TF 计算方式和 3 种 IDF 计算方式形成 15 种 VSM 变种,他们将 15 种 VSM 相似度评分通过线性相加进行组合建模,随后使用遗传算法来自动优化参数取值.实验结果表明:通过组合优化的方式,该方法的缺陷定位性能要显著优于标准的 VSM 模型.Gore 等人[36]将 N-Gram 模型与 VSM 相结合,提出了混合检索模型.常用的 VSM 模型只考虑单个词,词之间互相独立(即词袋模型),词组信息缺失.该方法同时采用一元模型(unigram model)和二元模型(bigram model),并分别基于 rVSM[18]计算两种相似度,最后加权求和.与传统方法相比,该混合模型缺陷定位性能有显著提升.

(2) 主题模型

IRBL 中使用的主题模型有潜在语义分析(latent semantic analysis,简称 LSA 或 latent semantic index,简称 LSI)和隐含狄利克雷分布 LDA:LSA 采用奇异值分解(singular value decomposition)将 VSM 模型所表示的向量降维,形成一个降维后的 K 维向量表示,随后通过计算余弦相似度来衡量缺陷报告和程序模块间的相似性;LDA 假设缺陷报告和程序模块都是由某些主题生成,因此缺陷报告和程序模块的相似度可基于主题分布来计算.

Lukins 等人[9,10]提出了基于 LDA 的缺陷定位方法,并与 LSI 模型进行比较,结果表明,LDA 模型的定位性能更好.Wang 等人[37]发现:(1) 有监督模型能有效利用现存的修复历史信息;(2) 缺陷报告中某些单词多次出现在相关缺陷模块中(即词共现现象);(3) 大的程序模块倾向于含有更多的缺陷(即长文件现象).基于上述发现,他们提出一种有监督主题模型 STMLOCATOR.该方法改进了有监督主题模型 LLDA[38]方法,对词共现现象和长文件现象进行建模.单词 w 可由缺陷报告和程序模块中的共现词或主题-词分布生成,他们设置主题数为程序模块的个数,引入一个非对称的 Dirichlet prior 来表示选择不同主题的不同概率.当一个文件规模比较大的时候,其被选中的概率更高.

(3) 概率检索模型

IRBL 中使用的概率模型包括 SUM(smoothed unigram model)和 BM25[39].

- SUM 将每个程序模块和缺陷报告表示为一个$|V|$维的概率向量,可通过 KL 散度计算出程序模块和缺陷报告的分布差异,并以此作为相似度指标;也可用程序模块和缺陷报告的生成概率 $P(\vec{w}_r \mid \vec{w}_c)$ 表示,其中,$\vec{w}_r$ 和 $\vec{w}_c$ 分别是缺陷报告和程序模块的向量表示.
- BM25 基于缺陷报告和程序模块的生成概率衡量其相似性.

Saha 等人[40]提出了 BLUiR 方法,将缺陷报告分成总结(summary)和描述这两个部分,将程序模块分成类名、函数名、变量名和注释这 4 个部分,基于已训练的 BM25 模型,将其表示为特征向量,然后分别计算缺陷报告和程序模块各部分文本相似度,并累加作为最终的文本相似度评分.

(4) 模型比较

Rao 等人[41]首次分析了文本检索模型对缺陷定位性能的影响,比较了 VSM、LSA、LDA、SUM 和 CBDM 等 5 种通用文本模型以及它们的组合,实验通过缺陷报告的描述文本信息定位缺陷程序模块.实证结果表明:与更复杂的模型(例如 LDA、LSA、CBDM 等)相比,简单的模型(例如 SUM、VSM)在基于检索的缺陷定位上更加有效.

Wang 等人[4]同样分析了文本检索模型对概念定位的影响,他们比较了 VSM、SUM、NMF、LSI、pLSI 和 LDA 等多个检索模型,得到了与 Rao 等人相似的实验结论,即,简单的 IR 模型(例如 VSM、SUM)比复杂的模型(例如 LDA,LSI 等)定位效果更好.

Thomas 等人[6]探索了不同分类器及其组合对缺陷定位性能的影响,他们比较了 VSM、LSI 和 LDA 等 3 种模型,将缺陷报告表示分为仅考虑标题、仅考虑描述和同时考虑标题与描述这 3 类,将程序模块表示分为仅考虑标识符、仅考虑注释、同时考虑标识符和注释以及与程序模块相关的历史缺陷报告集合等 6 类,预处理过程也将分割、词干提取、停止词移除这 3 个过程通过排列组合方式形成 8 种,然后分别对 VSM 的词权重与相似度计算规则、LSI 的词权重、主题数与相似度计算规则、LDA 的迭代数、主题数等参数设置进行多种设置,最后形成 3 172 种基于 IR 的分类器,用于缺陷定位.实验结果表明:基于 IR 的分类器其配置对缺陷定位性能有很大影响,其中最好的 IRBL 分类器配置是使用缺陷报告和程序模块的全部内容,执行 3 步预处理过程,采用基于 TF- IDF 的 VSM 模型并计算余弦相似度.随后,Thomas 等人还对分类器组合进行探讨,他们将分类器基于缺陷报告和程序模块的表示方式分为 4 个分类器子空间,保证每类分类器分类错误方式尽可能不同,同时采用两个基本策略:在每个子空间中选取最好的分类器或随机选择分类器,将选择的分类器采用线性拟合的方式进行定位.实验结果表明,分类器组合在几乎所有情况下都能够提高缺陷定位性能.

Tantithamthavorn 等人[7]基于 Thomas 等人[6]的工作,着重分析了分类器的配置对函数级别缺陷定位的性能和成本(effort)的影响,成本指找到缺陷程序模块所需检查代码行数的代价.他们采用相同的配置,比较了 3 172 个分类器.他们做出以下观察:(1) 分类器配置的选择对性能和成本影响较大,Top *K* Rank 性能的影响介于 0.44%~36%之间,需要的成本介于 4395LOC~50000LOC 之间;(2) 给出相似 Top *K* Rank 性能的分类器,需要的成本也不同;(3) VSM 模型在函数级别缺陷定位获得最好的 Top *K* Rank 性能和最少的成本代价;(4) 与程序模块表示相关的配置对 Top *K* Rank 性能和成本代价的影响最大;(5) 在函数级别缺陷定位上最有效的配置同样可用于文件级别缺陷定位上.

通过分析相关文献,我们发现:在经典检索模型中,VSM、SUM 等简单 IR 模型往往能表现更好的性能,VSM 模型是所有经典模型中被使用次数最多的模型.IR 模型配置的选择对缺陷定位性能影响较大,需要针对不同数据集做不同选择.

3.1.2 优化方法

1) 基于输入源拓展的方法

缺陷报告包含的自然语言与程序模块包含的程序语言存在较大差异,因此将文本检索模型应用于缺陷定位有时效果并不理想.研究人员尝试基于领域知识提取特征,以提高检索结果.这类方法基于领域知识(如将文件分解成函数、使用库组件的 API 描述、分析缺陷修复历史和代码变更历史等),设计出合适的特征来更为合理地计算出缺陷报告与程序模块之间的相似度.

Zhou 等人[18]发现:(1) 代码规模大的文件,其含有缺陷的可能性更高;(2) 如果两个缺陷报告的文本相似度很高,则这两个缺陷报告很可能对应相同的缺陷模块.他们据此提出了 BugLocator 方法,在计算 VSM 评分时将程序模块的词数加入考虑,用于修正 VSM 评分;同时,从已修复的缺陷报告中搜索与当前缺陷报告相似的缺陷报告,给对应的缺陷程序模块赋予一个评分.最终,基于 VSM 评分和相似缺陷报告评分来定位缺陷.

Rahman 等人[42]进一步考虑了程序模块代码结构和其缺陷修复次数,将缺陷报告中疑似类名或方法名的文本识别出,并对这些类或包含这些方法的类赋予额外的优先因子β,同时,根据程序模块缺陷修复次数对缺陷报告和程序模块的相关度评分进行微调.最后,对相关度评分和优先因子β进行线性拟合.

Youm 等人[43]基于上述的文本信息、堆栈跟踪信息、结构信息和代码变更信息求得对应的评分 $StructVsmScore(r,c)$,$SimiBugScore(r,c)$,$StraceScore(r,c)$以及从代码变更历史信息计算的评分 $CommScore(r,c,k)$,最后凭经验对 4 项缺陷报告和程序模块评分进行加权.

Uneno 等人[44]提出了 CombBL 模型,从 4 个方面计算缺陷报告和程序模块的相似度:基于 word2vec 的相关度评分、基于 BugLocator[18]方法计算的文本相似度评分、基于缺陷修复历史的评分和基于组件域的评分.其中:

基于 word2vec 的相关度评分是将缺陷报告和程序模块以其词向量的均值作为表示向量,计算余弦相似度;基于组件域的评分是以预处理后的缺陷报告中的每个词对程序模块全路径名进行松散搜索,成功则对应评分加 1.最后对 4 种评分进行线性拟合,并根据经验设置参数.

Dilshener 等人[45,46]从文本信息和堆栈跟踪信息的角度出发,分析词位置信息后发现,在缺陷报告总结中的第 1、第 2、倒数第 2 和倒数第 1 个词很可能与对应的缺陷程序模块相关;分析堆栈跟踪后发现,缺陷程序模块通常出现在堆栈跟踪的前 5 个文件中.他们对此采取的方法是,根据缺陷报告中的词在程序模块中出现的位置不同赋予不同的评分,例如,出现在文件名中的词赋予更高的评分.

Swe 等人[47]从文本结构、堆栈跟踪和相似缺陷报告这 3 个角度计算缺陷报告和程序模块的相似度.其中,文本结构仅考虑程序模块的结构信息,通过抽象语法树将程序模块分为类名、方法名、变量名,分别与缺陷报告计算文本相似度.该方法最后通过对 3 种评分进行线性拟合来定位缺陷.

Barbosa 等人[48]结合软件特征对缺陷报告和程序模块学习新的特征表示:在第 1 阶段,通过语言模型学习软件工程领域文本数据的词嵌入表示;第 2 阶段,通过一个由缺陷报告、程序模块、代码指标等形成的异构网络学习一个基于网络的特征表示.通过新的特征表示,可计算缺陷报告和程序模块的相似度.

Rath 等人[49,50]分析了缺陷报告中的结构化信息对 IRBL 的影响,他们将缺陷报告文本分为自然语言、源代码和堆栈跟踪这 3 类,并形成 7 种组合进行实验.结果表明:源代码文本通常能提高缺陷定位性能,堆栈跟踪则有好有坏.

2) 基于优化函数的方法

IRBL 中,不同的特征有不同的意义,并且对多个软件项目有不同的影响.凭经验设置参数不具有通用性,通过参数学习方法可自动学出合适的参数.这类方法将缺陷定位问题转换成基于参数学习的排序问题,即根据历史缺陷报告修复信息对当前程序模块与给定缺陷报告的相关性进行优化排序.该方法需定义一个优化函数或排序函数.

Ye 等人[51]提出一种基于学习排序(learning to rank)的方法,他们利用领域知识,结合代码中使用的库函数 API 描述、缺陷修复历史和代码变更历史,设计了 6 类特征:表面词汇相似度(surface lexical similarity)、API 增强词汇相似度(API-enriched lexical similarity)、协同过滤评分(collaborative filtering score)、缺陷修复近因(bug-fixing recency)、缺陷修复频率(bug-fixing frequency).在标准化后,使用一个排序模型进行训练,其评分函数 $f(r,c)$ 定义如下.

$$f(r,c)=w^T\Phi(r,c)=\sum_{i=1}^{k}w_i\times\phi_i(r,c) \tag{7}$$

其中,ϕ_i 表示缺陷报告和程序模块的第 i 类特征.Ye 等人[23]随后额外使用了结构信息和文件依赖信息,他们将缺陷报告分为总结和描述,将程序模块分为类名、函数名、变量名和注释,然后两两计算相似度;同时,根据文件依赖信息定义包含入度和出度的局部特征、通过 PageRank 计算的复杂度评分以及通过 HITS(hyperlink-induced topic search)计算的 Hubs 评分和 Authorities 评分;之后,采用排序模型进行训练优化.学习参数 w 等价于解决以下优化问题.

$$\left.\begin{aligned}&\text{minimize}:\\&J(w,\xi)=\frac{1}{2}\|w\|^2+C\sum\xi_{rpn}\\&\text{subject to}:\\&w^T\Phi(r,p)-w^T\Phi(r,n)\geqslant 1-\xi_{rpn},\xi_{rpn}\geqslant 0,\forall r\in R,p\in P(r),n\in N(r)\end{aligned}\right\} \tag{8}$$

其中,w 是权重参数矩阵,$P(r)$表示与缺陷报告 r 相关的程序模块,$N(r)$表示与缺陷报告无关的程序模块.

Zhao 等人[52]探讨了学习排序算法在缺陷定位上的成本代价问题,即找到缺陷程序模块所需检查代码行数的代价,并定义了 3 个衡量代价的指标:Effort@k,MAE 和 MFE.他们复现了 Ye 等人的缺陷定位方法[51],并与 VSM、Usual Suspects 方法进行了比较.实验结果表明:学习排序算法在成本敏感指标上要显著优于 Usual

Suspects,但在 Effort@k 上要差于 VSM 算法.

Shi 等人[53]比较了学习排序算法在混合缺陷定位中的性能,他们将不仅考虑了文本相似度特征、还利用其他特征(如版本历史、程序模块结构、动态分析等特征)的缺陷定位方法定义为混合缺陷定位.在总结前人工作的基础上,他们将输入特征分为 7 类:文本信息、堆栈跟踪、版本历史、依赖图、文本结构、相似缺陷报告、执行信息,并对特征进行不同的归一化操作:linear 归一化、sum 归一化、Z-score 归一化以及不进行任何操作.Shi 等人之后对 8 种学习排序方法的性能进行比较,这 8 种方法分别为 MART(multiple additive regression trees)、RankNet、RankBoost、AdaRank、Coordinate Ascent、LambdaMART、ListNET 和 Random Forests.实验结果表明,没有进行归一化处理的 coordinate ascent 算法是所以学习排序算法中性能最好的.

Wang 等人[34]从版本历史、相似缺陷报告、结构信息、堆栈跟踪和报告人信息的角度出发,计算了 5 种特征,其综合评分为 5 种特征的加权求和.该方法使用遗传算法对权重进行调参,优化目标为使历史已修复缺陷报告的定位性能达到最高,即,使评价指标 MAP 和 MRR 总和达到最高.优化目标具体如下.

$$ObjFunc=e^{(MAP+MRR)} \tag{9}$$

其中,MAP 和 MRR 为 IRBL 性能评估时常用的评测指标.

Gharibi 等人[54]设计了 5 类特征:token 匹配评分、VSM 相似度评分、堆栈跟踪评分、语义相似度评分和基于已修复缺陷报告的评分.其中,

- token 匹配评分是统计缺陷报告和程序模块指定部分的 token 匹配数:第 1 阶段检查总结中是否包含程序模块名,第 2 阶段则检查总结和描述中的词性 token 与类名、方法名、注释中的 token 是否匹配.
- 基于已修复缺陷报告的评分则使用多标签分类算法 One-Vs-The-Rest 对缺陷报告进行分类.

该方法基于目标函数 $ObjFunc=MRR+MAP$ 对权重参数进行优化.

Almhana 等人[55]提出一种多目标优化方法,在最小化推荐的程序模块数的同时,最大化提出方案的相关性.他们采用算法 NSGA-II 进行搜索,候选集为一个向量表示,每一维表示推荐给缺陷报告的类,推荐类的顺序对应它们在向量中的位置.其适应值函数为

$$f=\frac{LS+HS}{2} \tag{10}$$

其中,LS 表示词法相似度,由缺陷报告描述与程序模块的相似度以及缺陷报告描述与推荐的程序模块中函数 API 说明的相似度组成;HS 表示基于历史的相似度,由程序模块的历史修复次数度量、最近的修改时间度量和历史缺陷报告的推荐一致性指标决定.

(3) 基于查询重构的方法

这类方法基于缺陷报告、程序模块、堆栈跟踪信息或者补丁等中的关键词来尝试构建新的查询,并进行检索.查询重构方法基于不同的策略可进一步细分为查询扩充(query expansion)、查询替换(query replacement)、词选择(term selection)、查询缩减(query reduction)[56].

Sisman 等人[57]基于程序模块中描述相同概念或功能的词距离比较接近的观察,提出一个新的查询重构框架,根据程序模块中的词与缺陷报告中的词距离,构建一个新的查询.该方法用初始查询(即缺陷报告)检索程序模块语料库,并从检索结果中挑选出与初始查询中词距离最接近的词,将其加入到初始查询中,最后重新进行检索.实验结果表明:该方法能够有效地提高缺陷定位性能,并且效果超过两个经典的 QR 方法 Rocchio's Formula[58]和 Relevance Model[59].Sisman 等人[60]在之后的工作中进一步识别缺陷报告中的堆栈跟踪信息或补丁,从这些结构组件中提取词来构建最终的查询.Chaparro 等人[61]发现:在大多数情况下,缺陷报告变成低质量查询的原因是非相关词的存在(即噪音),因此猜想,将低质量查询删减成只描述观测行为(observed behavior,简称 OB)的词可以提高 IRBL 的性能.其中,观测行为指当前软件的错误行为.观测行为的识别主要通过人工识别,采用词选择策略选择 OB 词进行删减.实验结果表明:当缺陷报告形成低质量查询时,将缺陷报告词删减为观测行为 OB 相关的词,能有效提高 IRBL 的性能.Rahman 等人[62]提出了基于上下文感知的查询重构方法,他们根据缺陷报告是否包含堆栈跟踪信息和是否包含程序元素,将缺陷报告分为 3 类:包含堆栈跟踪的缺陷报告(即有噪

声的缺陷报告)BR_{ST}、包含程序元素的缺陷报告(即富余的缺陷报告)BR_{PE}、只有自然语言的缺陷报告(即贫乏的缺陷报告)BR_{NL}.针对包含不同信息的缺陷报告,Rahman 等人进行不同的处理:对于 BR_{ST},将堆栈跟踪转换为跟踪图(trace graph),节点为类名或方法名,边表示节点间的依赖关系;对于 BR_{PE},基于词共现(word co-occurrence)和词性依赖(pos dependency)构建两个文本图(text graph);对于 BR_{NL},搜集返回的前 k 个程序模块,抽取其中的方法名或变量名等,并构建源词图(source term graph).随后采用 PageRank 方法进行词权重赋值并挖掘关键词,基于权重较大的前 k 个关键词构建新的查询.

缺陷报告的质量(即文本内容)有高有低,有些缺陷报告中包含堆栈跟踪或文件名等信息,对查询重构和缺陷定位有显著影响.Mills 等人[63]探讨了该类缺陷定位中存在的潜在偏见,比较了带提示(包含代码片段和方法名等)的缺陷报告和不带提示的缺陷报告在通过遗传算法进行近优查询时的定位结果.实验结果表明:在使用缺陷报告的全部文本作查询时,尤其在没有定位提示(即类名或方法名等)时,基于文本检索的缺陷定位性能很差.但存在一个最优查询,使其缺陷定位性能相比于默认查询有较大的性能提升.该结论在没有定位提示的缺陷报告中依然成立.他们认为,最优查询的规划在 IRBL 中值得进一步研究.

3.1.3 基于文本相似度的方法间的对比

如表 3 所示,该表统计基于文本相似度的方法所使用的软件特征(仅列出 CCF 列表中 B 类以上的论文),•表示文献使用了该列所对应的软件特征.在基于输入源拓展的方法中,研究人员使用的软件特征逐渐增多.通过经典模型与软件特征相结合,IRBL 的缺陷定位性能得到了有效提高.基于优化函数的方法在利用多个软件特征的基础上,定义一个优化目标.相较于基于输入源拓展的方法,基于优化函数的方法可有效学习模型参数.基于查询重构的方法在不利用额外输入源的基础上,提高了 IRBL 的缺陷定位性能.从总体缺陷定位效果上比较,目前,基于优化函数的方法取得相对较好的性能.

Table 3 Usage statistics of the features used by text similarity methods

表 3 基于文本相似度的方法的特征使用情况统计

类别	参考文献	文本信息	相似缺陷报告	堆栈跟踪	依赖关系	代码异味	版本历史	执行信息
通用检索模型	[6,7,9−11,37,40,64,65]	•	–	–	–	–	–	–
基于输入源拓展的方法	[17,18]	•	•	–	–	–	–	–
	[54,66,67]	•	•	•	–	–	•	–
	[29,35]	•	•	–	–	–	•	–
	[21]	•	•	•	–	–	–	–
	[20,46]	•	–	•	–	–	–	–
	[31]	•	–	–	–	–	–	•
	[68]	•	–	–	–	–	•	–
	[26]	•	–	–	–	•	–	–
基于优化函数的方法	[34,51]	•	•	•	–	–	•	–
	[55]	•	•	–	–	–	•	–
	[23]	•	•	•	•	–	•	–
	[32]	•	•	–	–	–	–	•
基于查询重构的方法	[61,62,69]	•	–	–	–	–	–	–
	[60]	•	–	•	–	–	–	–

3.2 基于语义相似度的方法

自然语言和程序语言存在较大的差异,因此造成缺陷报告与程序模块间也存在较大的差异性.为解决词汇不匹配问题,有研究人员从语义角度出发,通过提取文本语义特征来计算语义相似度.这类方法主要构建多层神经网络,通过深度学习提取特征并计算.常用的可选模型有 CNN(convolutional neural network),DNN(deep neural network)或 LSTM(long short-term memory)等.

Huo 等人[70]借助 CNN 来捕获程序模块中代码的结构信息和语义信息,该方法的输入为缺陷报告和程序模块,主要结构包括语言内特征提取层和跨语言特征融合层:在语言内特征提取层,分别对缺陷报告和程序模块提取词法信息和程序结构信息;在跨语言特征融合层,将两个向量拼接为一个统一的特征向量.之后,通过一个全连接层分类,其目标函数如下.

$$\min_{w}\sum_{i,j}[\cos t_n L(z_i^r, z_j^c, y_{ij}; w)(1-y_{ij}) + \cos t_p L(z_i^r, z_j^c, y_{ij}; w)(1+y_{ij})] + \lambda \| w \|^2 \tag{11}$$

其中, z_i^r, z_j^c 是缺陷报告和程序模块的向量表示;y_{ij} 表示程序模块与缺陷报告是否相关:相关为 1,否则为−1;w 为网络结构的参数.Huo 等人[71]随后在文献[70]工作的基础上,进一步捕捉程序模块的顺序特性,他们认为:代码的顺序不同会表现不同的作用,携带额外的语义信息.该方法更改语言内特征提取层中对程序语言的特征提取方式,通过 LSTM 来抽取程序语言的顺序特性,增强特征表示性.

Lam 等人[72,73]使用 DNN 解决缺陷报告和程序模块的词不匹配问题,并与基于软件特征的方法相结合,提出了 HyLoc 方法.HyLoc 方法使用 rVSM 收集缺陷报告和程序模块的文本相似度特征;根据项目元数据获取缺陷修复频率、类名相似度等特征;通过 DNN 网络学习缺陷报告和程序模块的标识符、类名、函数名、注释以及 API 描述的关联;最后,通过一个 DNN 特征融合层学习各类特征的权重参数.

Xiao 等人[74]充分利用缺陷报告和程序模块中的语义信息,并将 CNN 和缺陷修复历史信息相结合,提出了基于增强 CNN 的 DeepLocator.该方法将预处理后的缺陷报告和程序模块用词嵌入向量表示,以缺陷报告和程序模块中最大句子长度为单位,填充每个句子到同一维度,作为增强 CNN 的输入.增强 CNN 包含 1 个卷积层、1 个最大池化层和 1 个 dropout 层,在代价函数中引入缺陷修复近因和缺陷修复频率,具体如下.

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} cost_i \tag{12}$$

$$cost_i = -\sum_{j=1}^{T} t_{ij}\log(y_{ij}) - \omega_1 re_i - \omega_2 fr_i \tag{13}$$

其中,N 是 batch 大小,T 为程序模块的数目.$cost_i$ 为缺陷报告 i 的代价函数,t_{ij} 为缺陷报告 i 关于程序模块 j 的真实值,y_{ij} 是缺陷报告 i 关于程序模块 j 的输出概率值,ω_1 和 ω_2 是缺陷修复近因和缺陷修复频率权重调节参数.Xiao 等人[75]随后改进了文献[74]的模型,从语句角度出发,将语句表示为一个向量,对缺陷报告和程序模块采用两种不同的词嵌入技术:Sent2Vec 和加权平均嵌入(weighted average word2vec).缺陷报告的描述通过 Sent2Vec 可将其中的每个句子转化为一个向量,这个策略在降低内存占用和计算代价的同时,获得与 word2vec 相似或更好的性能,程序模块通过加权平均嵌入可获得每个代码语句的嵌入表示.随后,通过 CNN 对缺陷报告和程序模块的向量表示进行特征检测和提取,统一两个向量的维度以便平行拼接,其最终代价函数与文献[74]中相同.

Xiao 等人[76]使用词嵌入将缺陷报告和程序模块转换为向量,保存语义信息,然后考虑到缺陷报告和程序模块的不同特性,对缺陷报告使用多 filter 的 CNN 网络提取特征和降维,对程序模块使用多粒度扫描的随机森林,最后通过两个 CRF(completely-random tree forests)和两个随机森林形成的级联森林来提取缺陷报告和程序模块的深层特征并分类.

缺陷报告和程序模块具有不同的表达方式.Xiao 等人[77]首次将机器翻译技术用于解决缺陷报告和程序模块的词不匹配问题.该方法的训练实例为缺陷报告和对应缺陷程序模块解析的抽象语法树节点形成的文本对以及从 API 文档中提取的类、函数、字段的注释与其声明、函数调用、类实例形成的文本对.该方法通过一个基于注意力机制的 RNN 编码器和 RNN 解码器训练模型.当给定一个新的缺陷报告,该方法基于缺陷报告和程序模块的概率评分来定位缺陷程序模块.

现有的基于深度学习的缺陷定位技术大多集中于新型网络结构的设计,而对网络损失函数的设计缺乏应有的关注.已有的工作大多在输出层使用 sigmoid 函数作为激活函数,采用交叉熵损失进行训练.这种损失函数对距离分类边界较远的异常点比较敏感,限制了缺陷定位性能的提升.针对这个问题,解铮等人[78]提出一种代价敏感的间隔分布优化损失函数,将间隔分布优化问题通过代价敏感的方式扩展至非对称情况.该方法通过一层卷积层和一层池化层对缺陷报告提取特征,通过 2 个卷积层和 2 个池化层对程序模块提取特征,然后拼接两个特征表示,并通过全连接层学习一个统一的特征映射,最后通过训练好的模型定位缺陷程序模块.

3.3 两类方法对比

从定位模式分析,基于文本相似度的方法主要考虑缺陷报告和程序模块的共有词匹配,尤其缺陷报告中的程序实体名与程序模块中的程序实体名的匹配.Wang 等人[79]早期以基于文本相似度的方法为基准方法评价

IRBL 技术的有效性,他们发现:缺陷报告中的程序实体名通常对 IRBL 的性能起正面作用,缺陷报告中的堆栈跟踪信息和测试用例信息则并不总是有效的,因其会引入许多无关的程序模块.而如果仅利用基于自然语言描述的缺陷报告,IRBL 方法的缺陷定位性能通常并不理想.基于文本相似度的方法对由自然语言和程序语言差异造成的词不匹配问题并不能很好地予以解决,从查询优化的角度来试识别并过滤掉缺陷报告中的冗余和无效文本以及从软件项目中挖掘与缺陷程序模块相关的特征信息等措施,是从其他角度补充缺陷定位所需的信息,提升定位准确率.基于语义相似度的方法主要挖掘缺陷报告和程序模块的内在语义,从词义[76]、语义[76]、程序结构[70]和语句顺序[71]等多个角度考虑,并通过学习 API 文档、项目文档等获取额外知识[77,80],以缓解缺陷报告和程序模块的词不匹配问题.Polisetty 等人[81]在分析基于深度学习的缺陷定位方法在软件开发中的可用性时,与部分基于文本相似度的方法进行了比较,并从缩小词汇差异、方法有效性和方法普适性等角度进行分析研究.结果表明:基于深度学习的方法,其缺陷定位性能要优于基于传统检索模型的方法,并可有效缓解词不匹配问题,但也需要更多的训练时间和硬件资源.

从定位粒度的角度分析,基于文本相似度的方法根据选择方法的不同,可定位到文件、函数、语句和代码变更等级别;而基于语义相似度的方法因模型训练因素,一般都定位到文件级别.受限于应用场景和开发人员的实际需求,这两类方法各有优缺点.

4 场景应用

根据不同的场景需求,开发人员对缺陷定位粒度和准确率的需求不同.基于 IRBL 的定位粒度,IRBL 方法输出级别可分为文件(类)级别、函数级别、语句级别和代码变更级别,下面将依次介绍.其中,文件和类级别因多数文件是单个文件包含单个类,文件和类的文本量级比较接近,对 IRBL 方法而言几乎没有区别,因此不作区分.

4.1 文件级别

目前,大多数 IRBL 方法都定位到文件级别.给定一个缺陷报告,将一个按照缺陷怀疑率从大到小排序的有序文件列表推荐给开发人员.文件级别缺陷定位因其代码文本包含足够多的信息,可将 IR 方法直接应用于文件级别的缺陷定位.

Klnc 等人[82]通过多级重排序算法来实现文件级别的缺陷定位:在第 1 级,为解决词的语义信息被忽视的问题,从程序模块中提取出去除注释的代码、类名、方法名和注释 4 类文档,分别计算与缺陷报告的相似度并调整权重;在第 2 级,在第 1 级的基础上保存每个缺陷报告对应的前 k 个程序模块形成新的文档集,丢弃不相关的程序模块,重新构建新的索引并查询,将两级的结果综合考虑.

陈理国等人[83]提出了基于高斯过程的缺陷定位方法.基于高斯过程的缺陷定位,其直观含义可以表述为:每个缺陷报告节点和程序模块节点对对应一个隐含函数值,隐含函数值完全决定了节点对之间是否有链接关系,且隐含函数值服从高斯分布,协方差函数与缺陷报告节点和源文件节点对应的向量相关.陈理国等人首先采用优先采集算法,使得样本集中的正负样本是均等的;然后学习缺陷报告和程序模块节点对隐含函数值的高斯分布,当隐含值大于指定阈值时,则预测值为 1,程序模块与缺陷报告相关.

Distante 等人[84]提出一个两阶段缺陷定位方法:第 1 阶段,针对每个程序模块训练一个多层感知机,对新来的缺陷报告预测该程序模块是否与缺陷报告相关,相关的程序模块集合即为缺陷程序模块集合;第 2 阶段,对与缺陷程序模块相关的多个缺陷报告表示求其平均表示,计算新的缺陷报告与相关缺陷报告平均表示的相似度,根据相似度结果排序选前 K 个候选缺陷程序模块,优化排序结果.

除以上常见方法外,在基于文件级别的缺陷定位中存在一类特殊的方法,在推荐有序缺陷列表前会执行一个可靠性判断.因在 IRBL 中,总存在一些缺陷报告缺乏足够的定位信息,造成 IRBL 方法定位效果不理想.如果将不理想的缺陷定位结果推荐给开发人员,反而会增加缺陷定位成本,并降低缺陷定位方法的可信度.针对这个问题,Kim 等人[85]提出一种两阶段推荐模型,判断缺陷报告是否含有足够的信息,只有在缺陷报告包含足够的信息时,才进一步基于缺陷报告的内容预测缺陷模块.他们从缺陷报告的总结、描述、元数据以及报告者的评论中抽取特征,缺陷报告可预测的标准是基于缺陷报告特征使用朴素贝叶斯模型推荐的前 k 个程序模块中,命中

至少一个缺陷程序模块,则该缺陷报告被认为是可预测的.基于历史缺陷报告训练一个二分类模型后,可判断新的缺陷报告是否可预测,然后对可预测的缺陷报告使用多分类模型推荐缺陷程序模块.Le 等人[86,87]则通过构建一个喻言(oracle)数据库,自动预测 IRBL 方法生成的排序列表是否有用.他们认为:一个排序列表是有用的,仅当缺陷程序模块出现在该排序列表的前 N 位.其考虑的特征分为 4 类:怀疑率的评分特征(含有 64 个特征)、报告的文本特征(含有 2 个特征)、主题模型特征(含有 8 个特征)和元数据特征(含有 14 个特征),在这 4 大类特征上分别预测有效性,并加权计算得到最终评分.当评分大于某个阈值时,则排序列表有效;反之则无效.

4.2 函数级别

类和文件级别对开发人员而言存在粒度过粗的问题,需要开发人员检查大量的代码.通常,缺陷的修复是位于某个函数代码内.给定一个缺陷报告,基于函数级别的缺陷定位方法可以将一个按照缺陷怀疑率从大到小排序的有序函数列表推荐给开发人员.然而函数级别的缺陷定位相比于文件级别的缺陷定位,因为粒度缩小,每个函数包含的代码文本和词汇数要少许多,这会导致传统的信息检索模型因为稀疏表示问题运行困难.

有研究人员从文本扩充的角度出发,增加函数相关词的数量.Nichols[88]提出一个函数级别的 IRBL 方法,依据缺陷报告的修复记录,将缺陷报告中的词加入到对应缺陷程序模块受影响的函数代码中,这样函数就额外包含与修复缺陷相关的缺陷文本信息.当一个新的缺陷报告到来时,用 LSI 模型检索增强后的函数文本.

张文等人[89]考虑另一种函数增强方法,对于每个函数,计算该函数与其他所有函数的相似度并求其平均值,将大于平均值的函数向量表示扩充进该函数中,其计算公式如下.

$$m_i^{(a)} = m_i + \alpha \sum_{k=1,k\neq i}^{K} \cos(m_i,m_k)m_i, \cos(m_i,m_k) > \theta_i \tag{14}$$

其中,m_i,m_k 分别表示函数 i 和函数 k 的向量表示,$cos(m_i,m_k)$表示函数 i 和函数 k 的余弦相似度,θ_i 表示函数 i 与其他函数的余弦相似度的平均值.之后,张文等人[90]改进了函数增强方法,从语义相似度、时间近似和调用依赖这 3 个角度衡量函数之间的相似程度,每个函数的向量表示由 k 个最相似函数的向量表示构成,其计算公式如下.

$$m_i^{(a)} = \sum_{k=1}^{K} ac_{ik} \times m_k \tag{15}$$

其中,m_k 表示函数 k 的向量表示,ac_{ik} 是从 3 个角度衡量函数 i 和函数 k 的相似度的加权和.针对新来的缺陷报告,可基于增强后的函数进行检索.

Youm 等人[67]将缺陷定位粒度从文件级别拓展至函数级别,在获得文件级别的可疑文件排序列表的基础上,对前 10 的文件中的函数计算其与缺陷报告的文本相似度,然后根据代码变更历史信息,找出报告提交的 k 天内被修改的函数并计算函数修改评分,两者加权计算可得函数的怀疑评分.

Rahman 等人[91,92]提出了函数级别的缺陷定位方法 MBuM,通过最小化搜索空间来提高定位准确度.他们将动态分析和静态分析相结合,首先根据程序模块执行路径获取函数调用顺序,排除不相关的函数;然后根据函数名对函数内容进行静态分析,基于缺陷报告检索函数内容.为了避免语义信息的损失,MBuM 还进行同义词匹配,如“terminate”,“stop”与“close”词同义.实验结果表明,MBuM 在大部分用例中将缺陷函数排在第一位.

Davies 等人[66]综合多源信息设计与函数强相关的特征,从文本信息的角度计算缺陷报告描述与函数的相似度;从缺陷报告的角度,用历史缺陷报告作为文档集,对每个函数训练一个二分类器,判断缺陷报告是否与函数相关,然后对新来的缺陷报告为每个函数做分类;同时统计每个函数曾包含的缺陷数以及在堆栈跟踪的排名的倒数;最后,基于这 4 类特征使用线性回归方法来定位缺陷.

针对函数级别缺陷定位,也有研究人员不作区分,将函数级别和文件级别陷定位统一对待.Chaparro[93]将缺陷报告的描述分为 5 个部分:标题、观测行为 OB、期望行为(expected behavior)、复现步骤和代码片段.这些文本识别都是通过人工识别,然后对这 5 个部分排列组合可形成 31 个不同的重构策略,其检索文档为文件、类或函数.实验结果表明:标题和观察行为相结合是最好的重构策略,并在大部分样例中都适用.基于缺陷报告描述的查询重构能有效去除报告中的噪音,提高缺陷定位性能.Hill 等人[94]考虑使用词的位置距离(positional proximity)来度量程序模块和缺陷报告的相似度,以整合词邻近和词序信息,捕捉其语义信息.他们直接比较了 8

种方法:顺序依赖马尔科夫随机场(MRF-SD)、DLM(Dirichlet language model)、基于语句距离的方法(STMT)、基于函数距离的方法(MTHD)、TF-IDF 方法、NL-Sig、NL-Body、SWUM(software word usage model).其中,NL-Sig 只使用从函数签名(method signatures)中提取的短语概念,NL-Body 使用从函数的方法调用中提取的短语概念,SWUM 则结合 NL-Sig 和应用于函数的 TF-IDF.实验结果表明:位置距离对函数级别的定位提升不如文件级别定位明显,其中,MRF 和其变种 DLM 在多个数据集上都更加有效.

4.3 语句级别

基于语句级别的缺陷定位因为文本内容过少,需设计与语句强相关的特征.Rahman 等人[95]提出了语句级别的缺陷定位方法 MSDG,采用动态分析和静态分析相结合的方式.首先跟踪程序模块执行过程,丢弃不相关的程序模块,将相关的函数文本保留并处理;之后进行静态分析,MSDG 最大化每个语句的有效信息,将该语句和影响该语句每个元素的前语句信息加入一起考虑,生成节点和前节点依赖图,每个语句节点包含该语句及其前语句的关键词,最后计算缺陷报告与语句节点的文本相似度.同样,为了增加语句的信息,MSDG 定位时考虑语句对应函数的相似度评分和语句相似度评分,计算公式如下.

$$Score(r_i,s_i)=M_S\times N_S \tag{16}$$

其中,M_S 和 N_S 分别表示包含语句 S 的函数的评分和语句 S 本身的相似度评分.

4.4 代码变更级别

从实际角度出发,向开发人员推荐与缺陷报告直接相关的程序模块列表并不能提供修复缺陷的上下文信息.列表中的程序模块排序独立,开发人员只可能知道缺陷在哪,不知道什么引起的缺陷.基于代码变更级别的缺陷定位方法定位到一次代码变更(即在一次提交中被修改的多行代码,也被称为变更块(change hunk)),可辅助开发人员理解缺陷,简化缺陷修复任务.

Wen 等人[68]认为,基于代码变更的缺陷定位有一些优势:首先,描述变更的日志包含代码变更目的、功能等重要信息;其次,变更块通常是小段小段的代码,是分割源代码文件的一个可选方案,避免了大文件的噪音问题;最后,利用变更历史可以增强缺陷定位性能.他们为此提出了 Locus 方法,使用缺陷报告、该缺陷报告创建前的所有变更块作为输入,每个变更块包含改变代码行、上下文行的内容和相应的提交日志,可创建自然语言语料库和代码实体语料库.Locus 针对这两个语料库分别进行自然语言查询和程序实体查询,即分别计算缺陷报告与自然语言语料库和代码实体语料库的相似度;之后,Locus 根据代码变更历史对代码变更或文件计算一个增强评分(即越近的代码修改越有可能引入缺陷),其可定位代码变更级别或文件级别.实验结果表明:相较于基于文件级别的缺陷定位,基于代码变更的方法始终是最优的.

Loyola 等人[96]从语法和代码变更依赖的角度对代码变更学习特征表示,提出一个基于代码变更的缺陷定位方法.该方法对缺陷报告中的词进行词嵌入表示和字符级别的嵌入表示,并通过双向 LSTM 生成一个上下文丰富的特征表示;对代码变更的语法表示通过双线性 LSTM 接收一系列词,并将最终输出的状态作为语法特征表示,对代码变更依赖则定义了一个代码变更谱图(code change genealogy),通过随机游走(random walk)获得节点的嵌入表示,作为变更依赖表示.该方法之后合并特征,并通过一个学习排序算法 RankNet 计算相关评分.

4.5 定位粒度分析

该节对不同定位粒度的 IRBL 方法进行总结.各定位级别所占论文比例如图 4 所示.文件级别的缺陷定位论文占据 IRBL 论文 3/4 的比例,是所有定位粒度中最多的.该定位粒度数据集获取容易、程序模块代码大小适当,被研究人员广泛研究,定位准确率较高.目前,文件级别缺陷定位的公开数据集和共享实验代码也是最多的.函数级别的缺陷定位数据集获取困难,程序模块代码行偏少,一般需要进行额外处理,缺陷定位准确率相较于文件级别缺陷定位偏低[67].语句级别的缺陷定位则必须获取程序的执行信息,动态分析和静态分析相结合.有研究表明[97]:开发人员更倾向于选择函数、语句和块级别的缺陷定位方法,文件级别的缺陷定位带来的高准确率并不能满足所有开发人员的需求,而且 IRBL 方法的可靠性也让众多开发人员难以接受.目前,IRBL 方法无论从缺陷定位准确率或是缺陷定位粒度,都有待提高.

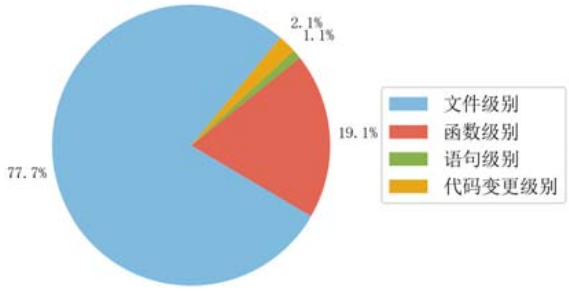

Fig.4 Statistics of the number of papers with different granularity levels for IRBL

图 4 IRBL 不同定位粒度方法的论文数统计

5 性能评测指标分析

为了方便后续研究工作更好地评估所提的 IRBL 方法,本文总结研究人员在 IRBL 中所使用的评测指标. IRBL 可视为一个经典的信息检索问题,因此可用信息检索领域的常用评测指标进行评估(注意:与缺陷报告相关的程序模块可能不止 1 个).我们假定有 Q 个缺陷报告需要被查询,将已有研究工作中常用的模型性能评测指标总结如下.

(1) MAP(mean average precision):该指标返回的是所有缺陷报告的平均查准率的均值.平均查准率是对一个检索结果在不同召回率点上的查准率进行平均,其计算公式为

$$P(j)=\frac{\text{number of positive instances in top } j \text{ positions}}{j} \tag{17}$$

$$AvgP_i=\sum_{j=1}^{N}\frac{P(j)\times pos(j)}{\text{number of positive instances}} \tag{18}$$

$$MAP=\frac{1}{Q}\sum_{i=1}^{Q}AvgP_i \tag{19}$$

其中,

- $pos(j)$表示排名为 j 的实例是否与缺陷报告相关:相关为 1,否则为 0.
- $P(j)$表示检索在 j 位置时的查准率.
- $AvgP_i$ 表示查询 i 的平均检索查准率.
- MAP 表示 Q 个查询的平均查准率的均值.

(2) Top K Rank:该指标在有些文献中又被称为 Accuracy@k[75]、Hit@k[62]、Prediction Accuracy[25]、Recall at Top N[33]等,其返回正确定位的缺陷报告占所有缺陷报告的比例.正确定位的缺陷报告指在推荐的前 k(k=1,5,10)个程序模块中至少包含一个缺陷程序模块,在有些文献中,这个指标返回的是正确定位的缺陷报告的个数,这里,我们把该指标统一为比例.

(3) MRR(mean reciprocal rank):该指标返回的是一系列查询的倒数排名的平均.一个查询的倒数排名指第 1 个相关文档的排名的倒数,即缺陷程序模块排名的倒数,其计算公式为

$$MRR=\frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i} \tag{20}$$

其中,$rank_i$ 表示第 1 个相关文件的排名.

(4) $P@t$:该指标返回的是在给定排序 t 为止检索的平均查准率.在指定 t 值时,有些文献直接以查准率(precision)表示 $P@t$.其计算公式为

$$P(t)=\frac{\text{number of positive instances in top } t \text{ positions}}{t} \tag{21}$$

$$P@t=\frac{\sum_{i=1}^{Q}P(t)}{Q} \tag{22}$$

其中,$P(t)$计算推荐 t 个程序模块时的查准率.

(5) $R@t$:该指标在一些文献中又被称为 Average Coverage Ratio[28],其返回的是在给定排序 t 为止检索的平均查全率.在指定 t 值时,有些文献直接以查全率(recall)表示 $R@t$.其计算公式为

$$R(t)=\frac{\text{number of positive instances in top } t \text{ positions}}{\text{number of all the positive instances}} \tag{23}$$

$$R@t=\frac{\sum_{i=1}^{Q}R(t)}{Q} \tag{24}$$

其中,$R(t)$计算推荐 t 个程序模块时的查全率.

(6) First Relevant Rank:该指标返回的是排序列表中第 1 个缺陷程序模块的排名.

(7) f-measure:该指标返回的是查准率和查全率的调和平均数,可以对查准率和查全率这两个指标进行有效的平衡,其计算公式为

$$f\text{-}measure=\frac{2\times precision\times recall}{precision+recall} \tag{25}$$

(8) SCORE:该指标返回的是不需要被开发人员调查的程序模块的比例,其计算公式为

$$SCORE=1-\frac{\max(p_1,p_2,...,p_m)}{N} \tag{26}$$

其中,m 表示缺陷程序模块的个数,p_m 表示第 m 个缺陷程序模块的排名,N 表示所有程序模块的个数.

(9) Rank of Retrieved Files:该指标返回的是第 1 个相关文件在指定范围的缺陷报告个数.通常有 $rank$=1,2≤$rank$≤5,6≤$rank$≤10 和 $rank$>10 的范围.

(10) Avg.Rank:该指标返回的是所有缺陷模块的平均排名.其计算公式为

$$Avg.Rank=\frac{\sum_{f\in F_B\cap F_R}rank_f}{|F_B\cap F_R|} \tag{27}$$

其中,F_B 表示缺陷报告的真实修复程序模块,F_R 表示推荐的缺陷程序模块,$rank_f$ 表示程序模块 f 的排名.

(11) AFH@n(average first hit at top-n):该指标返回的是排序列表中第 1 个缺陷程序模块的平均排名,衡量开发人员的工作量,其计算公式为

$$\delta_n(rank_i)=\begin{cases}rank_i, & rank_i\leqslant n\\ n, & rank_i>n\end{cases} \tag{28}$$

$$AFH@n=\frac{\sum_{i=1}^{Q}\delta_n(rank_i)}{Q} \tag{29}$$

其中,$rank_i$ 表示排序列表中第 1 个缺陷模块的排名.

上述指标都是从程序模块的角度考虑,Zhao 等人[52]首次从代价敏感(cost-effective)的角度分析,基于代码审查工作新定义 3 个缺陷定位性能指标:Effort@k、MAE(mean average effort)、MFE(mean first effort),具体如下.

(1) Effort@t:该指标返回的是成功定位相关缺陷程序模块所花费的平均代码审查成本.其计算公式为

$$Effort@k=\frac{\sum_{i=1}^{Q}\sum_{j=1}^{k}LOC(f_{ij})}{|R_s|} \tag{30}$$

其中,f_{ij} 表示缺陷报告 i 的排序列表中第 j 个程序模块,LOC 函数表示程序模块的代码行数,R_s 表示成功定位的缺陷报告数.

(2) MAE:该指标返回的是定位缺陷报告的所有缺陷程序模块所花费的平均成本的均值,是指标 MAP 的成本代价变种.其计算公式为

$$Avg(p)=\sum_{j=1}^{p}\frac{LOC(f_{ij})}{p} \tag{31}$$

$$AvgP_i=\sum_{j=1}^{N}\frac{Avg(j)\times pos(j)}{\text{number of positive instances}} \tag{32}$$

$$MAE=\frac{1}{Q}\sum_{i=1}^{Q}AvgP_i \tag{33}$$

(3) MFE:该指标返回的是定位缺陷报告的第 1 个缺陷程序模块所花费的成本的均值.其计算公式为

$$MFE=\frac{1}{Q}\sum_{i=1}^{Q}\sum_{j=1}^{rank_i}LOC(f_{ij}) \tag{34}$$

其中,$rank_i$ 为缺陷报告 i 第 1 个缺陷程序模块的排名.

本文对已有 IRBL 研究中常用的性能评测指标的累计使用次数进行了统计,并从大到小进行排序;同时给出了每个指标的首次使用时间(包括对应的参考文献),最终结果见表 4.从表中不难看出:经常使用的评测指标是 MAP、Top *K* Rank 和 MRR,这些指标使用率远超其他指标.除此之外,一些研究人员从成本收益角度来评估模型的缺陷定位性能,主要考虑的指标包括 Effort@*k*、MAE 和 MFE.

Table 4 Statistics of performance evaluation measures

表 4 性能评测指标统计

序号	指标	累计使用次数	首次使用时间	序号	指标	累计使用次数	首次使用时间
1	MAP	66	2011[41]	9	SCORE	2	2011[41]
2	Top *K* rank	59	2008[25]	10	Avg.Rank	1	2013[85]
3	MRR	50	2012[18]	11	Effort@*k*	1	2015[52]
4	*R*@*t*	14	2012[27]	12	MAE	1	2015[52]
5	*P*@*t*	13	2012[27]	13	MFE	1	2015[52]
6	First relevant rank	5	2008[9]	14	Rank of retrieved files	1	2011[41]
7	AUC	4	2016[70]	15	AFH@*n*	1	2018[37]
8	*f*-measure	3	2014[86]	–	–	–	–

6 评测数据集总结

在 IRBL 中,因开源软件项目数据获取相对容易,研究人员一般使用开源软件项目进行科学研究.这一节,本文对 IRBL 中常被使用的评测数据集其累计使用次数和首次使用时间进行总结,方便后续研究人员进行实验对比.其最终结果见表 5(表 5 仅列出累计使用次数超过 3 次的评测数据集).

Table 5 Usage statistics of evaluation datasets

表 5 评测数据集的使用情况统计

序号	数据集名称	累计使用次数	首次使用时间	序号	数据集名称	累计使用次数	首次使用时间
1	AspectJ	48	2011[41]	12	Mozilla	7	2010[10]
2	SWT	41	2012[18]	13	Ant	7	2013[17]
3	Eclipse	29	2010[10]	14	Jodatime	7	2013[17]
4	JDT	22	2013[6]	15	Rhino	6	2010[10,88]
5	Zxing	19	2012[18]	16	Derby	5	2014[20]
6	Tomcat	19	2014[51]	17	Eclipse platform	5	2016[70]
7	Birt	13	2014[51]	18	Pig	5	2014[20]
8	Eclipse UI	11	2010[10]	19	Maven	4	2014[20]
9	ArgoUML	9	2008[25]	20	JBOSS	4	2012[70]
10	PDE	8	2016[70]	21	APACHE	4	2012[70]
11	Lucene	8	2014[99]	22	Jedit	4	2012[17]

IRBL 中的缺陷定位数据集需要搜集缺陷报告和软件代码,进行链接关系分析,建立缺陷报告和软件代码的关联,具体过程如图 5 所示.缺陷报告通常使用爬虫方法从缺陷跟踪系统中抓取,再人工过滤掉不属于缺陷的报告,缺陷管理工具主要为 Bugzilla 和 JIRA 等.软件代码因为大部分用于研究的软件项目是开源的,可从网上直接下载源代码,其软件代码管理工具主要为 Git、CVS 和 SVN 等.链接关系分析一般采用 Dallmeier 等人[98]提出的

启发式方法,有显式的提交编号(commit id)则检查对应的提交记录,其对应的代码变更视为一个缺陷修复变更;同时,也从提交日志中检索与缺陷报告 ID 相关的提交,确定对应变更与缺陷修复是否相关.

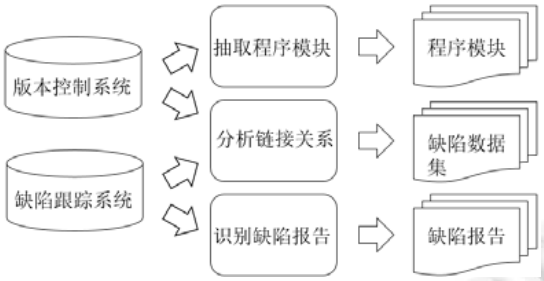

Fig.5 Collection process of bug localization datasets

图 5 缺陷定位数据集的收集过程

早期由于缺陷定位数据集并未共享,研究人员主要对开源软件项目的少量缺陷报告进行用例分析[51],造成相关研究工作难以重现,相关论文数较少.从 2012 年开始,随着 Eclipse、SWT、AspectJ 和 Zxing 缺陷定位数据集的公开,对 IRBL 问题的研究开始逐步升温.在 2014 年,Ye 等人[51]共享了 Birt、Tomcat 等新的缺陷定位数据集,Wang 等人[100]还提供了一个便于研究人员比较和复现缺陷定位技术的实验平台 BOAT.这些公开数据集及共享实验代码的增多,使得 IRBL 问题成为当前软件缺陷定位领域的一个研究热点,并产生了很多高质量的研究成果.接下来,本文将对这些数据集进行简要介绍.

有研究表明:Bugzilla 中的缺陷报告与修复代码是弱关联的;而 JIRA 中的缺陷报告与修复代码则关联明显,主要因为 JIRA 提供了附件帮助关联问题(issue)和版本控制系统中的提交[101].本文对由 JIRA 管理的数据集和其他数据集分别进行介绍.

在上述数据集中,由 JIRA 管理的数据集有 Lucene、JBOSS、APACHE、Derby 和 Pig 等.其中,

- Lucene 是一个开放源代码的全文检索引擎工具包.在缺陷定位研究中使用的 Lucene 版本和报告数不尽相同,有的研究中包含 2 443 个缺陷报告,有的则只包含 30 个缺陷报告.
- JBOSS 是一个基于 J2EE 的开放源代码的应用服务器,Lal 等人[12]抽取了 569 个缺陷报告,Lee 等人[102]则对 JBoss 中不同项目(例如 ENTESB、JBMETA、ELY、SWARM、WFARQ 等)分别抽取缺陷报告,构建多个数据集,缺陷报告从 1 个到 984 个不等.
- APACHE 是一个 Web 服务器软件,目前在 Web 服务器软件中全世界使用率第一,Lal 等人[12]抽取了 637 个缺陷报告.
- Derby 是一个完全使用 java 编写的数据库系统.
- Pig 是一种数据流语言,Moreno 等人[20]在研究中各抽取了 2 个版本、累计 129 和 133 个缺陷报告进行实验.

在缺陷定位研究中,这些数据集存在多个版本,研究人员基于实际需要选择特定版本内的缺陷报告作为实验对象.

对于其他数据集,多数软件项目都是使用 Bugzilla 作为缺陷跟踪系统,少部分使用 Github 管理缺陷报告或问题报告.其中,AspectJ、SWT、Eclipse、JDT、Birt、PDE 和 Eclipse Platform 等都来自开源软件项目 Eclipse,Tomcat 则是 Jakarta 项目中的一个核心项目,这些软件项目都是由 Bugzilla 管理的.Zxing 是一个开放源码的,用 Java 实现的多种格式的 1D/2D 条码图像处理库,其缺陷管理工具是 Github,包含 20 个缺陷报告.

从以上统计可发现:即使是相同的软件项目,由于不同研究人员的需求不同,使用的缺陷报告也不尽相同.同一个缺陷报告,根据研究人员定位要求的不同,缺陷报告关联的缺陷程序模块粒度也会不同.以后随着公开数据集的增多,对 IRBL 方法的普适性和通用性研究可在更多数据集中验证.

7 总 结

目前,基于信息检索的缺陷定位虽然已经取得了一定的研究进展,但缺陷定位的准确率和可靠度还有待提高,这也是当前软件缺陷定位研究成果难以成功应用到企业软件质量保障流程的主要阻碍之一.虽然有一些缺陷定位工具[103]被实现出来给开发人员使用,但距离被多数开发人员认可还存在一定的距离.对基于信息检索的缺陷定位的研究具有丰富的理论研究价值和工业界应用前景,我们认为,IRBL 还存在很多值得国内研究人员关注的研究问题.本文将依次从数据源、检索模型、场景应用这 3 个维度对未来研究工作进行展望.

(1) 针对数据源的研究展望

通过对评测数据集采集过程的分析,我们发现在挖掘软件历史仓库的过程中,对程序模块进行类型标记时可能会产生噪音.这些噪音的存在会影响缺陷定位模型的构建,并对已有实证研究结论的有效性产生严重影响.

- 首先,将缺陷报告从问题报告中识别出时就可能产生噪音.Herzig 等人[104]指出:在问题跟踪系统中大量问题报告被错误分类,接近 1/3 被标记为缺陷的问题报告不是真的缺陷报告.他们指出,这种偏差严重影响了对基于历史先验缺陷的缺陷预测研究.
- 其次,将版本控制系统中的修改日志与缺陷跟踪系统中的缺陷报告建立链接关系时容易产生噪音.例如:通过在提交日志中搜索特定关键词(例如 fixed 或 bug)和缺陷 ID 号(例#53322)可建立一种链接关系,随后,基于该链接关系将代码修改涉及的程序模块标记为缺陷模块,与缺陷报告对应起来.但并不是所有修改的程序模块都是有缺陷的,有些程序模块变更是一些非必要变更.

除了数据集的 Ground Truth 的可靠性问题,数据集中的缺陷报告质量也存在巨大差异,对 IRBL 有显著影响.Kochhar 等人[99]就此分析了 IRBL 中存在的潜在偏见,实验结果表明:缺陷报告错误分类的影响是可以忽略不计的;而因建立链接关系造成的不准确的 Ground Truth 对缺陷定位结果的影响也不大;显著地影响缺陷定位结果的是那些已定位的缺陷报告.已定位的缺陷报告指那些文本描述中已经指定包含缺陷程序模块文本的报告,这些已定位报告不需要额外的缺陷定位方法.实验表明:那些已定位或部分定位的缺陷报告比那些非已定位的缺陷报告准确率高得多,对缺陷定位性能影响很大.

针对数据集的质量问题,研究人员可以在收集数据集时选择由 JIRA 管理的软件项目,加强缺陷报告和修复提交的关联,以提高缺陷数据集的质量.针对缺陷报告的质量问题,一方面,在用户提交缺陷报告时对缺陷报告的质量进行把关,有研究人员考虑在用户提交缺陷报告时检测缺失信息并给出反馈[61];另一方面,针对低质量的缺陷报告,可通过查询重构等方式来提高缺陷定位性能.Lawrie 等人[69]分析了基于遗传算法的查询重构方法,发现该方法的优化目标主要基于查询性能指标,因此存在一定的作弊问题.他们采用了基于自动总结的方法,并分析了该方法在不作弊的情况下是否可以进行高质量的查询重构.结果表明,该方法的效果还有待提高.虽然上述方法对数据集质量进行了初步的探索,但如何进一步提升缺陷报告质量以提高最终的缺陷定位效果,仍是一个亟待解决的问题.

(2) 针对检索模型的研究展望

目前,针对检索模型的研究存在多个问题.

- 首先,检索模型的缺陷定位准确性和实际应用的需求存在较大差距,仍有较大的提升空间.研究人员可引入软件缺陷预测等领域的知识,设计与缺陷报告和缺陷程序模块相关的软件特征,提高缺陷定位准确率;同时,IRBL 作为一个检索问题,可进一步引入信息检索领域内的最新研究成果,充分利用来自 Stack Overflow 的众包知识、API 文档和软件项目文档等来改进模型,辅助解决缺陷报告和缺陷程序模块的词不匹配问题.
- 其次,目前的检索模型很少探讨训练数据不足的问题.当开发人员团队开发一个新的软件项目时,新项目会存在训练数据不足的问题,这需要利用其他软件项目的训练数据来建立模型.但不同项目的训练数据因项目的管理方式、人员构成、采用的编程语言不同,会存在较大的数据分布差异.因此,如何借助迁移学习方法来缓解这种数据分布差异问题,是一个研究难点.Huo 等人[105]在这个方面进行了尝试,首次提出了跨项目缺陷定位方法,通过一个深度迁移学习模型,解决目标项目训练数据不足的问题.该

方法主要包含迁移特征提取层和项目指定预测层:在迁移特征提取层,通过 CNN 共享权重参数的方式将知识从原项目迁移到目标项目;在项目指定预测层,通过两个全连接层共同训练原项目和目标项目.

- 最后,随着软件快速频繁地演化,如果每次演化后都需要重新训练检索模型,则存在过程重复并且训练费时费力等问题,因此,模型的增量更新是一种有效的解决方案.对此,Rao 等人[64]提出一个增量更新框架,使用每次软件演化中的所有代码变更信息来持续地更新索引和模型,对 VSM 模型和 SUM 提出索引更新方法和增量更新规则.实验结果表明,该方法仅需额外花费少量时间即可获得同样的检索准确率.Rao 等人[106]随后又比较了两种基于 LSI 的增量更新方法 iLSI 和 iSVD,随着项目的演化,可以增量更新模型的参数.

在软件项目的不同阶段会遇到不同的问题,采用合适的 IRBL 方法以适应软件开发过程、保障软件质量,是一个值得探索的方向.

(3) 针对场景应用的研究展望

目前,IRBL 方法大多基于 Java 语言开发的软件工程项目,对其他类型语言的软件项目研究较少.面向对象语言和面向过程语言对编程风格和习惯有很大影响,其对缺陷定位造成的影响很值得研究.为此,Saha 等人[65]探索了 IRBL 在 C 语言上的有效性.他们拓展了 BLUiR 方法用于 C 语言的缺陷定位,实验结果表明:在 C 语言软件项目上,基于 IR 的缺陷定位方法一样有效;不过,部分软件特征如结构信息等,对 C 语言软件项目的益处并不大.

Garnier 等人[107]对先前工作中数据集存在的问题进行了改进,在 20 个 C#软件项目上,评估 BLUiR、BLUiR+和 AmaLgam 方法的有效性:首先是版本选择问题,实验中找到缺陷报告创建前的提交版本,确保每个缺陷报告都定位到其对应的版本;然后是缺陷报告选择问题,排除已定位缺陷报告的出现;最后是程序模块选择问题,移除用于测试的程序模块.实验结果表明:基于 IR 的缺陷定位方法在 C#软件项目上依然有效,并且将更多的程序构造(如字符串文字、接口、枚举类型等)加入结构化信息检索的考虑之中,能有效提高缺陷定位性能.除基于 C 和 C#开发的项目外,IRBL 方法在基于其他编程语言开发的软件项目上的缺陷定位性能仍需要进一步探讨.

除了从横向维度进行拓展,模型也可从纵向维度进行考虑.当前,文件级别的缺陷定位数据集是最多的,函数和代码变更级别的缺陷定位数据集较少.因此,研究人员可以继续挖掘、搜集和共享更多细粒度的数据集,分析 IRBL 方法在相关数据集上的有效性[79],并通过提高细粒度缺陷定位的性能,以满足开发人员的实际定位需求,促进软件缺陷定位的研究成果应用到工业界中.

若能对上述研究挑战提出有效的解决方案,将使得软件缺陷定位更好地应用于项目开发和维护过程,并有助于快速定位缺陷程序模块,减轻开发人员负担,最终提高软件产品的质量.

References:

[1] 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 with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[2] Wong WE, Gao R, Li Y, Abreu R, Wotawa F. A survey on software fault localization. IEEE Trans. on Software Engineering, 2016,42(8):707−740. [doi: 10.1109/TSE.2016.2521368]

[3] Kochhar PS, Bissyande TF, Lo D, Jiang L. An empirical study of adoption of software testing in open source projects. In: Proc. of the Int'l Conf. on Quality Software. IEEE, 2013. 103−112. [doi: 10.1109/QSIC.2013.57]

[4] Wang S, Lo D, Xing Z, Jiang L. Concern localization using information retrieval: An empirical study on linux kernel. In: Proc. of the Working Conf. on Reverse Engineering (WCRE). IEEE, 2011. 92−96. [doi: 10.1109/WCRE.2011.72]

[5] Dit B, Revelle M, Gethers M, Poshyvanyk D. Feature location in source code: A taxonomy and survey. Journal of Software: Evolution and Process, 2013,25(1):53−95. [doi: 10.1002/smr.567]

[6] Thomas SW, Nagappan M, Blostein D, Hassan AE. The impact of classifier configuration and classifier combination on bug localization. IEEE Trans. on Software Engineering, 2013,39(10):1427−1443. [doi: 10.1109/TSE.2013.27]

[7] Tantithamthavorn C, Lemma Abebe S, Hassan AE, Ihara A, Matsumoto K. The impact of IR-based classifier configuration on the performance and the effort of method-level bug localization. Information and Software Technology, 2018,102:160−174. [doi: 10.1016/j.infsof.2018.06.001]

[8] Hill E, Rao S, Kak A. On the use of stemming for concern location and bug localization in java. In: Proc. of the Int'l Working Conf. on Source Code Analysis and Manipulation. IEEE, 2012. 184−193. [doi: 10.1109/SCAM.2012.29]

[9] Lukins SK, Kraft NA, Etzkorn LH. Source code retrieval for bug localization using latent dirichlet allocation. In: Proc. of the Working Conf. on Reverse Engineering. IEEE, 2008. 155−164. [doi: 10.1109/WCRE.2008.33]

[10] Lukins SK, Kraft NA, Etzkorn LH. Bug localization using latent dirichlet allocation. Information and Software Technology, 2010, 52(9):972−990. [doi: 10.1016/j.infsof.2010.04.002]

[11] Nguyen AT, Nguyen TT, Al-Kofahi J, Nguyen HV, Nguyen TN. A topic-based approach for narrowing the search space of buggy files from a bug report. In: Proc. of the Int'l Conf. on Automated Software Engineering (ASE 2011). IEEE, 2011. 263−272. [doi: 10.1109/ASE.2011.6100062]

[12] Lal S, Sureka A. A static technique for fault localization using character *n*-gram based information retrieval model. In: Proc. of the India Software Engineering Conf. ACM, 2012. 109−118. [doi: 10.1145/2134254.2134274]

[13] Zhang X, Yao Y, Wang Y, Xu F, Lu J. Exploring metadata in bug reports for bug localization. In: Proc. of the Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2017. 328−337. [doi: 10.1109/APSEC.2017.39]

[14] Khatiwada S, Tushev M, Mahmoud A. Just enough semantics: An information theoretic approach for IR-based software bug localization. Information and Software Technology, 2018,93:45−57. [doi: 10.1016/j.infsof.2017.08.012]

[15] Moreno L, Bandara W, Haiduc S, Marcus A. On the relationship between the vocabulary of bug reports and source code. In: Proc. of the Int'l Conf. on Software Maintenance. IEEE, 2013. 452−455. [doi: 10.1109/ICSM.2013.70]

[16] Chaparro O. Improving bug reporting, duplicate detection, and localization. In: Proc. of the Int'l Conf. on Software Engineering Companion (ICSE-C). IEEE, 2017. 421−424. [doi: 10.1109/ICSE-C.2017.27]

[17] Davies S, Roper M, Wood M. Using bug report similarity to enhance bug localisation. In: Proc. of the Working Conf. on Reverse Engineering. IEEE, 2012. 125−134. [doi: 10.1109/WCRE.2012.22]

[18] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the Int'l Conf. on Software Engineering (ICSE). IEEE, 2012. 14−24. [doi: 10.1109/ICSE.2012.6227210]

[19] Schröter A, Bettenburg N, Premraj R. Do stack traces help developers fix bugs? In: Proc. of the Working Conf. on Mining Software Repositories (MSR 2010). IEEE, 2010. 118−121. [doi: 10.1109/MSR.2010.5463280]

[20] Moreno L, Treadway JJ, Marcus A, Shen W. On the use of stack traces to improve text retrieval-based bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 151−160. [doi: 10.1109/ICSME.2014.37]

[21] Wong CP, Xiong Y, Zhang H, Hao D, Zhang L, Mei H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 181−190. [doi: 10.1109/ICSME.2014.40]

[22] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016, 27(1):1−25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[23] Ye X, Bunescu R, Liu C. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Trans. on Software Engineering, 2016,42(4):379−402. [doi: 10.1109/TSE.2015.2479232]

[24] Zhou Y, Tong Y, Chen T, Han J. Augmenting bug localization with part-of-speech and invocation. Int'l Journal of Software Engineering and Knowledge Engineering, 2017,27(6):925−949. [doi: 10.1142/S0218194017500346]

[25] Chen IX, Yang CZ, Lu TK, Jaygarl H. Implicit social network model for predicting and tracking the location of faults. In: Proc. of the Annual IEEE Int'l Computer Software and Applications Conf. (COMPSAC). IEEE, 2008. 136−143. [doi: 10.1109/COMPSAC.2008.162]

[26] Takahashi A, Sae-Lim N, Hayashi S, Saeki M. A preliminary study on using code smells to improve bug localization. In: Proc. of the Int'l Conf. on Program Comprehension (ICPC 2018). ACM, 2018. 324−327. [doi: 10.1145/3196321.3196361]

[27] Sisman B, Kak AC. Incorporating version histories in information retrieval based bug localization. In: Proc. of the Working Conf. on Mining Software Repositories (MSR). IEEE, 2012. 50−59. [doi: 10.1109/MSR.2012.6224299]

[28] Tantithamthavorn C, Ihara A, Matsumoto KI. Using co-change histories to improve bug localization performance. In: Proc. of the ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. IEEE, 2013. 543−548. [doi: 10.1109/SNPD.2013.92]

[29] Wang S, Lo D. Version history, similar report, and structure: Putting them together for improved bug localization. In: Proc. of the Int'l Conf. on Program Comprehension (ICPC 2014). ACM, 2014. 53−63. [doi: 10.1145/2597008.2597148]

[30] Reis S, Abreu R, D'Amorim M. Demystifying the combination of dynamic slicing and spectrum-based fault localization. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2019. 4760−4766. [doi: 10.24963/ijcai.2019/661]

[31] Le TDB, Oentaryo RJ, Lo D. Information retrieval and spectrum based bug localization: better together. In: Proc. of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, 2015. 579−590. [doi: 10.1145/2786805.2786880]

[32] Hoang TVD, Oentaryo RJ, Le TDB, Lo D. Network-clustered multi-modal bug localization. IEEE Trans. on Software Engineering, 2019,45(10):1002−1023. [doi: 10.1109/TSE.2018.2810892]

[33] Dao T, Zhang L, Meng N. How does execution information help with information-retrieval based bug localization? In: Proc. of the Int'l Conf. on Program Comprehension (ICPC). IEEE, 2017. 241−250. [doi: 10.1109/ICPC.2017.29]

[34] Wang S, Lo D. AmaLgam+: Composing rich information sources for accurate bug localization. Journal of Software: Evolution and Process, 2016,28(10):921−942. [doi: 10.1002/smr.1801]

[35] Wang S, Lo D, Lawall J. Compositional vector space models for improved bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 171−180. [doi: 10.1109/ICSME.2014.39]

[36] Gore A, Choubey SD, Gangrade K. Improved bug localization technique using hybrid information retrieval model. In: Proc. of the Int'l Conf. on Distributed Computing and Internet Technology. 2016.127−131. [doi: 10.1007/978-3-319-28034-9_16]

[37] Wang Y, Yao Y, Tong H, Huo X, Li M, Xu F, Lu J. Bug localization via supervised topic modeling. In: Proc. of the Int'l Conf. on Data Mining (ICDM). IEEE, 2018. 607−616. [doi: 10.1109/ICDM.2018.00076]

[38] Ramage D, Hall D, Nallapati R, Manning CD. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In: Proc. of the Conf. on Empirical Methods in Natural Language Processing. 2009. 248−256.

[39] Robertson S, Walker S, Beaulieu M. Experimentation as a way of life: Okapi at TREC. Information Processing and Management, 2000,36(1):95−108. [doi: 10.1016/S0306-4573(99)00046-1]

[40] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2013. 345−355. [doi: 10.1109/ASE.2013.6693093]

[41] Rao S, Kak A. Retrieval from software libraries for bug localization. In: Proc. of the Working Conf. on Mining Software Repositories. ACM, 2011. 43−52. [doi: 10.1145/1985441.1985451]

[42] Rahman S, Ganguly KK, Sakib K. An improved bug localization using structured information retrieval and version history. In: Proc. of the Int'l Conf. on Computer and Information Technology (ICCIT). IEEE, 2015. 190−195. [doi: 10.1109/ICCITechn.2015.7488066]

[43] Youm KC, Ahn J, Kim J, Lee E. Bug localization based on code change histories and bug reports. In: Proc. of the Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2015. 190−197. [doi: 10.1109/APSEC.2015.23]

[44] Uneno Y, Mizuno O, Choi EH. Using a distributed representation of words in localizing relevant files for bug reports. In: Proc. of the Int'l Conf. on Software Quality, Reliability and Security (QRS). IEEE, 2016. 183−190. [doi: 10.1109/QRS.2016.30]

[45] Dilshener T, Wermelinger M, Yu Y. Locating bugs without looking back. In: Proc. of the Int'l Workshop on Mining Software Repositories. ACM, 2016. 286−290. [doi: 10.1145/2901739.2901775]

[46] Dilshener T, Wermelinger M, Yu Y. Locating bugs without looking back. Automated Software Engineering, 2018,25(3):383−434. [doi: 10.1007/s10515-017-0226-1]

[47] Swe KEE, Oo HM. Bug localization approach using source code structure with different structure fields. In: Proc. of the Int'l Conf. on Software Engineering Research, Management and Applications (SERA). IEEE, 2018. 159−164. [doi: 10.1109/SERA.2018.8477206]

[48] Barbosa JR, Marcacini RM, Britto R, Soares F, Rezende S, Vincenzi AMR, Delamaro ME. BULNER: Bug localization with word embeddings and network regularization. In: Proc. of the Anais do VII Workshop on Software Visualization, Evolution and Maintenance (VEM). SBC, 2019. 21−28. [doi: 10.5753/vem.2019.7580]

[49] Rath M, Mader P. Influence of structured information in bug report descriptions on IR-based bug localization. In: Proc. of the Euromicro Conf. on Software Engineering and Advanced Applications (SEAA). IEEE, 2018. 26−32. [doi: 10.1109/SEAA.2018.00014]

[50] Rath M, Mäder P. Structured information in bug report descriptions—Influence on IR-based bug localization and developers. Software Quality Journal, 2019,27(3):1315−1337. [doi: 10.1007/s11219-019-09445-6]

[51] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014. 689−699. [doi: 10.1145/2635868.2635874]

[52] Zhao F, Tang Y, Yang Y, Lu H, Zhou Y, Xu B. Is learning-to-rank cost-effective in recommending relevant files for bug localization? In: Proc. of the Int'l Conf. on Software Quality, Reliability and Security. IEEE, 2015. 298−303. [doi: 10.1109/QRS.2015.49]

[53] Shi Z, Keung J, Bennin KE, Zhang X. Comparing learning to rank techniques in hybrid bug localization. Applied Soft Computing, 2018,62:636−648. [doi: 10.1016/j.asoc.2017.10.048]

[54] Gharibi R, Rasekh AH, Sadreddini MH, Fakhrahmad SM. Leveraging textual properties of bug reports to localize relevant source files. Information Processing and Management, 2018,54(6):1058−1076. [doi: 10.1016/j.ipm.2018.07.004]

[55] Almhana R, Mkaouer W, Kessentini M, Ouni A. Recommending relevant classes for bug reports using multi-objective search. In: Proc. of the Int'l Conf. on Automated Software Engineering. ACM, 2016. 286−295. [doi: 10.1145/2970276.2970344]

[56] Rahman MM, Roy C. Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2018. 473−484. [doi: 10.1109/ICSME.2018.00057]

[57] Sisman B, Kak AC. Assisting code search with automatic query reformulation for bug localization. In: Proc. of the Working Conf. on Mining Software Repositories (MSR). IEEE, 2013. 309−318. [doi: 10.1109/MSR.2013.6624044]

[58] Rocchio JJ. Relevance feedback in information retrieval. In: Proc. of the SMART Retrieval System: Experiments in Automatic Document Processing. 1971.

[59] Lavrenko V, Croft WB. Relevance based language models. In: Proc. of the Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM, 2001. 120−127. [doi: 10.1145/383952.383972]

[60] Sisman B, Akbar SA, Kak AC. Exploiting spatial code proximity and order for improved source code retrieval for bug localization. Journal of Software: Evolution and Process, 2017,29(1):1−24. [doi: 10.1002/smr.1805]

[61] Chaparro O, Florez JM, Marcus A. Using observed behavior to reformulate queries during text retrieval-based bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2017. 376−387. [doi: 10.1109/ICSME.2017.100]

[62] Rahman MM, Roy CK. Improving IR-based bug localization with context-aware query reformulation. In: Proc. of the ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2018. 621−632. [doi: 10.1145/3236024.3236065]

[63] Mills C, Pantiuchina J, Parra E, Bavota G, Haiduc S. Are bug reports enough for text retrieval-based bug localization? In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2018. 381−392. [doi: 10.1109/ICSME.2018.00046]

[64] Rao S, Medeiros H, Kak A. An incremental update framework for efficient retrieval from software libraries for bug localization. In: Proc. of the Working Conf. on Reverse Engineering (WCRE). IEEE, 2013. 62−71. [doi: 10.1109/WCRE.2013.6671281]

[65] Saha RK, Lawall J, Khurshid S, Perry DE. On the effectiveness of information retrieval based bug localization for c programs. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 161−170. [doi: 10.1109/ICSME.2014.38]

[66] Davies S, Roper M. Bug localisation through diverse sources of information. In: Proc. of the Int'l Symp. on Software Reliability Engineering Workshops (ISSREW). IEEE, 2013. 126−131. [doi: 10.1109/ISSREW.2013.6688891]

[67] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017,82:177−192. [doi: 10.1016/j.infsof.2016.11.002]

[68] Wen M, Wu R, Cheung SC. Locus: Locating bugs from software changes. In: Proc. of the Int'l Conf. on Automated Software Engineering. ACM, 2016. 262−273. [doi: 10.1145/2970276.2970359]

[69] Lawrie D, Binkley D. On the value of bug reports for retrieval-based bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2018. 524−528. [doi: 10.1109/ICSME.2018.00048]

[70] Huo X, Li M, Zhou ZH. Learning unified features from natural and programming languages for locating buggy source code. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2016. 1606−1612.

[71] Huo X, Li M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2017. 1909−1915. [doi: 10.24963/ijcai.2017/265]

[72] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports. In: Proc. of the Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2015. 476−481. [doi: 10.1109/ASE.2015.73]

[73] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. of the Int'l Conf. on Program Comprehension (ICPC). IEEE, 2017. 218−229. [doi: 10.1109/ICPC.2017.24]

[74] Xiao Y, Keung J, Mi Q, Bennin KE. Improving bug localization with an enhanced convolutional neural network. In: Proc. of the Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2017. 338−347. [doi: 10.1109/APSEC.2017.40]

[75] Xiao Y, Keung J, Bennin KE, Mi Q. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 2019,105:17−29. [doi: 10.1016/j.infsof.2018.08.002]

[76] Xiao Y, Keung J, Mi Q, Bennin KE. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In: Proc. of the Int'l Conf. on Evaluation and Assessment in Software Engineering 2018 (EASE 2018). ACM, 2018. 101−111. [doi: 10.1145/3210459.3210469]

[77] Xiao Y, Keung J, Bennin KE, Mi Q. Machine translation-based bug localization technique for bridging lexical gap. Information and Software Technology, 2018,99:58−61. [doi: 10.1016/j.infsof.2018.03.003]

[78] Xie Z, Li M. Cost-sensitive margin distribution optimization for software bug localization. Ruan Jian Xue Bao/Journal of Software, 2017,28(11):3072−3079 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5345.htm [doi: 10.13328/j.cnki.jos.005345]

[79] Wang Q, Parnin C, Orso A. Evaluating the usefulness of IR-based fault localization techniques. In: Proc. of the Int'l Symp. on Software Testing and Analysis. ACM, 2015. 1−11. [doi: 10.1145/2771783.2771797]

[80] Ye X, Shen H, Ma X, Bunescu R, Liu C. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proc. of the Int'l Conf. on Software Engineering. ACM, 2016. 404−415. [doi: 10.1145/2884781.2884862]

[81] Polisetty S, Miranskyy A, Başar A. On usefulness of the deep-learning-based bug localization models to practitioners. In: Proc. of the Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. ACM, 2019. 16−25. [doi: 10.1145/3345629.3345632]

[82] Kılınç D, Yücalar F, Borandağ E, Aslan E. Multi-level reranking approach for bug localization. Expert Systems, 2016,33(3):286−294. [doi: 10.1111/exsy.12150]

[83] Chen LG, Liu C. Bug localization method based on gaussian processes. Ruan Jian Xue Bao/Journal of Software, 2014,25(6):1169−1179 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4430.htm [doi: 10.13328/j.cnki.jos.004430]

[84] Distante D, Faralli S. A two-phase bug localization approach based on multi-layer perceptrons and distributional features. In: Proc. of the Int'l Conf. on Computational Science and Its Applications. Springer Int'l Publishing, 2019. 518−532. [doi: 10.1007/978-3-030-24289-3_38]

[85] Kim D, Tao Y, Kim S, Zeller A. Where should we fix this bug? A two-phase recommendation model. IEEE Trans. on Software Engineering, 2013,39(11):1597−1610. [doi: 10.1109/TSE.2013.24]

[86] Le TDB, Thung F, Lo D. Predicting effectiveness of IR-based bug localization techniques. In: Proc. of the Int'l Symp. on Software Reliability Engineering. IEEE, 2014. 335−345. [doi: 10.1109/ISSRE.2014.39]

[87] Le TDB, Thung F, Lo D. Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. Empirical Software Engineering, Empirical Software Engineering, 2017,22(4):2237−2279. [doi: 10.1007/s10664-016-9484-y]

[88] Nichols BD. Augmented bug localization using past bug information. In: Proc. of the Annual Southeast Regional Conf. ACM, 2010. 1−6. [doi: 10.1145/1900008.1900090]

[89] Zhang W, Li ZQ, Du YH, Yang Y. MethodLocator: A fine-grained bug location approach with the method level. Ruan Jian Xue Bao/Journal of Software, 2019,30(2):195−210 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/35565.htm [doi: 10.13328/j.cnki.jos.005565]

[90] Zhang W, Li Z, Wang Q, Li J. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Information and Software Technology, 2019,110:121−135. [doi: 10.1016/j.infsof.2019.03.001]

[91] Rahman S, Sakib K. An appropriate method ranking approach for localizing bugs using minimized search space. In: Proc. of the Int'l Conf. on Evaluation of Novel Software Approaches to Software Engineering. SCITEPRESS (Science and and Technology Publications), 2016. 303−309. [doi: 10.5220/0005896403030309]

[92] Rahman S, Rahman MM, Sakib K. An improved method level bug localization approach using minimized code space. In: Proc. of the Int'l Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE). 2016. 179−200. [doi: 10.1007/978-3-319-56390-9_9]

[93] Chaparro O, Florez JM, Marcus A. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empirical Software Engineering, Empirical Software Engineering, 2019,24(5):2947−3007. [doi: 10.1016/j.infsof.2016.11.002]

[94] Hill E, Sisman B, Kak A. On the use of positional proximity in IR-based feature location. In: Proc. of the Int'l Conf. on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE, 2014. 318−322. [doi: 10.1007/s10664-018-9672-z]

[95] Rahman S, Rahman MM, Sakib K. A statement level bug localization technique using statement dependency graph. In: Proc. of the Int'l Conf. on Evaluation of Novel Approaches to Software Engineering. SCITEPRESS (Science and Technology Publications), 2017. 171−178. [doi: 10.5220/0006261901710178]

[96] Loyola P, Gajananan K, Satoh F. Bug localization by learning to rank and represent bug inducing changes. In: Proc. of the Int'l Conf. on Information and Knowledge Management. ACM, 2018. 657−665. [doi: 10.1145/3269206.3271811]

[97] Kochhar PS, Xia X, Lo D, Li S. Practitioners' expectations on automated fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. ACM, 2016. 165−176. [doi: 10.1145/2931037.2931051]

[98] Dallmeier V, Zimmermann T. Extraction of bug localization benchmarks from history. In: Proc. of the Int'l Conf. on Automated Software Engineering (ASE 2007). ACM, 2007. 433−436. [doi: 10.1145/1321631.1321702]

[99] Kochhar PS, Tian Y, Lo D. Potential biases in bug localization: Do they matter? In: Proc. of the Int'l Conf. on Automated software engineering. ACM, 2014. 803−814. [doi: 10.1145/2642937.2642997]

[100] Wang X, Zhou B, Lo D, Xia X, Wang X, Kochhar PS, Tian Y, Yang X, Li S, Sun J. BOAT: An experimental platform for researchers to comparatively and reproducibly evaluate bug localization techniques. In: Companion Proc. of the Int'l Conf. on Software Engineering. ACM, 2014. 572−575. [doi: 10.1145/2591062.2591066]

[101] Wu R, Zhang H, Kim S, Cheung SC. ReLink: Recovering links between bugs and changes. In: Proc. of the ACM SIGSOFT Symp. and 13th European Conf. on Foundations of Software Engineering (SIGSOFT/FSE 2011). ACM, 2011. 15−25. [doi: 10.1145/2025113.2025120]

[102] Lee J, Kim D, Bissyandé TF, Jung W, Le Traon Y. Bench4BL: Reproducibility study on the performance of IR-based bug localization. In: Proc. of the ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2018. 61−72. [doi: 10.1145/3213846.3213856]

[103] Thung F, Le TDB, Kochhar PS, Lo D. BugLocalizer: Integrated tool support for bug localization. In: Proc. of the ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE 2014). ACM, 2014. 767−770. [doi: 10.1145/2635868.2661678]

[104] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the Int'l Conf. on Software Engineering (ICSE). IEEE, 2013. 392−401. [doi: 10.1109/ICSE.2013.6606585]

[105] Huo X, Thung F, Li M, Lo D, Shi ST. Deep transfer bug localization. IEEE Trans. on Software Engineering, 2019. Early Access. [doi: 10.1109/TSE.2019.2920771]

[106] Rao S, Medeiros H, Kak A. Comparing incremental latent semantic analysis algorithms for efficient retrieval from software libraries for bug localization. ACM SIGSOFT Software Engineering Notes, 2015,40(1):1−8. [doi: 10.1145/2693208.2693222]

[107] Garnier M, Garcia A. On the evaluation of structured information retrieval-based bug localization on 20 C# projects. In: Proc. of the Brazilian Symp. on Software Engineering. ACM, 2016. 123−132. [doi: 10.1145/2973839.2973853]

附中文参考文献:

[1] 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[22] 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[78] 解铮,黎铭.基于代价敏感间隔分布优化的软件缺陷定位.软件学报,2017,28(11):3072−3079. http://www.jos.org.cn/1000-9825/5345.htm [doi: 10.13328/j.cnki.jos.005345]

[83] 陈理国,刘超.基于高斯过程的缺陷定位方法.软件学报,2014,25(6):1169−1179. http://www.jos.org.cn/1000-9825/4430.htm [doi: 10.13328/j.cnki.jos.004430]

[89] 张文,李自强,杜宇航,杨叶.MethodLocator:一种方法级别的细粒度软件缺陷定位方法.软件学报,2019,30(2):195−210. http://www.jos.org.cn/1000-9825/5565.htm [doi: 10.13328/j.cnki.jos.005565]

李政亮(1993－),男,博士生,CCF 学生会员,主要研究领域为软件缺陷定位.

蒋智威(1988－),男,博士,助理研究员,CCF 专业会员,主要研究领域为自然语言处理.

陈翔(1980－),男,博士,副教授,CCF 高级会员,主要研究领域为软件缺陷预测,软件缺陷定位,回归测试,组合测试.

顾庆(1972－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件质量保障,分布式计算.