文件信息增强的方法级软件缺陷定位

薄莉莉[1,2],朱 程[1],李 斌[1],孙小兵[1]
(1. 扬州大学信息工程学院,江苏扬州 225127;
2. 高安全系统的软件开发与验证技术工业和信息化部重点实验室,江苏南京 211106)

摘 要: 软件开发与维护中会产生大量缺陷报告,根据缺陷报告准确定位到缺陷代码的位置是极具挑战性的.目前大多数工作在文件粒度定位缺陷,虽然少量工作定位在方法粒度,但定位准确度较低.本文提出一个文件信息增强的方法级软件缺陷定位技术FMBL(a File information enhanced Method-level Bug Localization technology),考虑方法与文件之间的从属关系以增强缺陷定位准确性.通过综合考虑代码与缺陷报告的词汇相似度、语义相似度和代码长度度量它们之间的相关性.在六个开源软件项目上开展实验以评估FMBL的有效性.结果表明,本文方法在六个项目上的平均Accuracy@10、MAP(Mean Average Precision)和MRR(Mean Reciprocal Rank)值分别达到0.436、0.223、0.296.与现有方法BugLocator、BLIA(Bug Localization using Integrated Analysis)、BugPecker相比,本文方法在MAP指标上分别提升153.1%、209.1%、22.8%.

关键词: 软件维护;缺陷定位;词汇相似度;语义相似度;文件信息;方法粒度
基金项目: 国家自然科学基金(No.61872312, No.61972335, No.62002309);南京航空航天大学科研基地创新(理工类)项目(No.NJ2020022);扬州大学"高端人才支持计划"(No.2019);江苏省"六大人才高峰"高层次人才项目(No.RJFW-053);江苏省"333"工程中青年科学技术带头人项目;扬州大学畜牧学学科特区学科交叉课题支持项目(No.yzuxk202015)
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112(2023)03-0613-09
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20210878

Method-Level Software Bug Localization Enhanced with File Information

BO Li-li[1,2], ZHU Cheng[1], LI Bin[1], SUN Xiao-bing[1]
(1. *School of Information Engineering*, *Yangzhou University*, *Yangzhou*, *Jiangsu* 225127, *China*;
2. *Key Laboratory of Safety-Critical Software* (*Nanjing University of Aeronautics and Astronautics*), *Ministry of Industry and Information Technology*, *Nanjing*, *Jiangsu* 211106, *China*)

Abstract: A large number of bug reports are generated during software development and maintenance. Finding the location of the bug code based on the bug report is a challenging task. Most of the bug localization works focus on file granularity. Even though few works locate bugs in method granularity, the accuracy is relatively low. In this paper, we proposed FMBL (a File information enhanced Method-level Bug Localization technology), a method-level software bug localization approach enhanced with file information. It considers the affiliation between methods and files to enhance the accuracy of bug localization. Lexical and semantic similarity between code and bug reports, together with code size are used to measure the correlation of methods and files. We conduct experiments on six open source software projects to evaluate the effectiveness of FMBL. The results show that, with FMBL, the average Accuracy@10, MAP (Mean Average Precision), and MRR (Mean Reciprocal Rank) on the six projects can reach 0.436, 0.223 and 0.296, respectively. Compared with BugLocator, BLIA (Bug Localization using Integrated Analysis) and BugPecker, our approach improves by 153.1%, 209.1% and 22.8% in MAP, respectively.

Key words: software maintenance; bug localization; lexical similarity; semantic similarity; file information; method granularity
Foundation Item(s): National Natural Science Foundation of China (No.61872312, No.61972335, No.62002309);

收稿日期:2021-07-09;修回日期:2021-12-20;责任编辑:梅志强

Innovation (Science and Engineering) Project of Scientific Research Base of Nanjing University of Aeronautics and Astronautics (No.NJ2020022); Yangzhou University Top-level Talents Support Program (2019); Six Talent Peaks Project in Jiangsu Province (No.RJFW-053); Jiangsu "333" Project; Yangzhou University Interdisciplinary Research Foundation for Animal Husbandry Discipline of Targeted Support (No.yzuxk202015)

1 引言

软件缺陷是软件产品中导致软件执行不正确或不符合需求规范的异常错误. 软件缺陷定位是指找出导致软件某些功能失败或执行异常或者不符合规范的缺陷位置的过程. 对于包含数百个甚至数千个源文件的大型项目,手动进行缺陷定位是一项艰巨而费时的工作. 因此如何自动定位缺陷位置,以应对不断增长的缺陷数量和代码规模,从而提高软件缺陷修复效率和软件质量,成为研究人员近年来关注的热点之一.

近十年来,研究人员将信息检索技术应用于缺陷定位. 使用信息检索模型计算缺陷报告中的文本与源代码单元之间的相关度,并根据此相关度来对代码单元进行排名,将排名靠前的源文件返回给开发人员. 这类方法使用简单,成本低廉,适用于帮助开发者快速找到缺陷报告所描述的问题的相关位置[1].

然而,由于使用自然语言编写的缺陷报告和使用编程语言编写的源代码通常包含不同的术语,造成源代码和缺陷报告之间的词汇鸿沟[2],限制了基于信息检索的缺陷定位模型的性能. 为了解决这个问题,近年来,研究学者致力于基于深度学习技术进行缺陷定位,通过将缺陷报告和源代码嵌入相同的语义空间以缩小自然语言和编程语言之间的语义鸿沟.

此外,大多数软件缺陷定位技术定位粒度在文件级别,尽管减少了大量寻找缺陷位置的时间,但缺陷修复人员在文件中寻找缺陷的具体位置仍需付出很大的努力. 目前,少量研究人员开始致力于方法级别的缺陷定位研究,但方法粒度上的定位准确率较低. 主要原因有:(1)在软件项目中方法数量远大于文件数量使得检索更加困难;(2)和文件相比方法内容少导致区分度低. 本文提出一个新的方法粒度上的软件缺陷定位技术,通过加入文件级别信息增强方法级别软件缺陷定位效果. 传统的方法粒度上的缺陷定位技术把方法作为单独考虑对象,割裂了方法和文件之间的从属关系,丢失了文件级别的缺陷信息. 本文从代码的三种不同角度(词汇、语义、长度)计算缺陷报告和代码的相关性,在数据集上的实验结果表明,综合三种得分可以得到更好的效果,文件级信息可以显著提升方法级缺陷定位的准确度. 本文的贡献如下:

(1) 提出一种新的缺陷定位方法 FMBL(File information enhanced Method-level Bug Localization technology),结合语义相似度、词汇相似度和代码长度计算代码的可疑度,将文件得分和方法得分相加作为方法的最终得分,提高方法级缺陷定位准确性.

(2) 构建了一个文件级和方法级均适用的缺陷定位数据集并进行了综合实验,与最新技术进行比较,评估提出的缺陷定位方法. 和 BugLocator[3]、BLIA(Bug Localization using Integrated Analysis)[4]、BugPecker[5]相比,FMBL 在 MAP(Mean Average Precision)上分别提升 153.1%、209.1%、22.8%.

2 相关工作

2.1 基于信息检索的缺陷定位

Zhou 等[3]提出的缺陷定位方法 BugLocator,使用向量空间模型和文件长度计算缺陷报告和源代码的文本相似度,并结合相似历史缺陷对文件进行排序. 除了缺陷报告和源代码的文本相似度,缺陷的其他复杂特征[6]也可以被用来提高缺陷定位准确度. Sisman 等[7]将版本历史信息嵌入到基于信息检索的缺陷定位中. Wang 等[8]提出新发现的缺陷往往是由于最近的提交引入,因此丢弃了老的版本历史信息. Youm 等[4]在他们的方法中使用到堆栈踪迹特征. Saha 等[9]考虑到代码结构特征,将缺陷报告分为标题和描述两个部分,将代码分为类名、方法名、变量名和注释四个部分后和缺陷报告组合,分别计算每个组合的文本相似度得分,然后将所有组合得分加权相加作为文件的最终得分. 结果表明,合理利用代码的结构化信息可以提升缺陷定位的准确性.

2.2 基于深度学习的缺陷定位

自然语言和程序语言之间存在词汇鸿沟,单纯使用信息检索的缺陷定位技术达到瓶颈,深度学习被应用来解决这一问题. Huo 等[10]提出 NP-CNN((Natural language and Programming language Convolutional Neural Network)将卷积神经网络应用于源代码文本特征提取. 和文本不同,代码往往拥有良好的结构. Wang 等[11]提出一种多维度卷积神经网络模型 MD-CNN(Multi-Dimension Convolutional Neural Network). 首先从待定位的源代码文件、API(Application Programming Interface)库和历史缺陷报告中识别并提取五个维度的特征(文本词汇相似度、相似历史缺陷、代码修复历史、类名相似度、结构信息),然后将其输入 CNN(Convolutional Neural Network)以学习特征与缺陷位置之间的非

线性关系.

2.3 细粒度的缺陷定位

基于缺陷报告的缺陷定位方法大多基于文件级别,基于方法级的缺陷定位较少. 部分基于频谱的缺陷定位技术,如FLUCCS[12]、PRINCE(PRecise machINe-learning-based fault loCalization tEchnique)[13]、DeepFL(Deep Fault Localization)[14]等也在方法级别定位缺陷,但这些工作需要执行测试用例,属于动态的缺陷定位技术,而我们的工作属于基于缺陷报告的静态缺陷定位技术.

Zhang等[15]提出了一种方法级的细粒度缺陷定位. 他们考虑一种方法扩充的方式,对每个方法,计算该方法与其他所有方法的相似度并求其平均值,将大于平均值的方法的向量表示扩充进该方法中. 以方法扩充的方式解决源代码中方法长度短引起的表示稀疏问题. 在BLIA 1.5中,Youm等[4]将缺陷定位粒度从文件级别拓展至方法级别,在获得文件级别的可疑文件排序列表的基础上,对前10的文件中的方法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天内被修改的方法并计算方法变更得分,两者加权求和得到方法的可疑度得分. BugPecker是Cao等[5]提出的方法级基于缺陷修复图的定位方法. 该方法尽可能地捕捉历史修复的细节以构建缺陷修复图,然后根据缺陷修复图扩展方法内容并计算相似度进行排序.

和文件级相比,方法级细粒度的缺陷定位更有利于缺陷修复人员直接定位到缺陷位置,但由于已有的工作未考虑文件和方法的从属关系,仅尝试扩展方法内容或在更高可疑性的文件中无差别的寻找方法,导致定位效果偏低. 本文将文件信息融入到方法级缺陷定位中以提高方法级缺陷定位准确性.

3 方法级软件缺陷定位

图1所示为本文提出的软件缺陷定位方法FMBL的总体流程. 首先构建一个文件粒度和方法粒度均适用的缺陷定位数据集,然后在对样本数据进行预处理后分别从语义相似度、词汇相似度和代码长度三个方面评估缺陷报告和文件/方法的相关性. 由于文件与方法之间存在从属关系,缺陷文件中的方法和其他文件中的方法相比含有缺陷的可能性更高,我们将方法得分和其所在文件的得分相加作为方法的最终得分.

3.1 数据集构建

Ye等[16]提供了一个文件粒度上的缺陷定位数据集,它包含了六个Java开源项目的缺陷报告和修复提交. 由于没有给出缺陷报告对应的缺陷方法,我们需要在修复缺陷的提交中标记缺陷方法. 此外一个项目有多次提交,在提交中会增加,修改或删除一些文件以修复缺陷[17]. 因此不同的缺陷报告对应的项目版本不同,对于每一个缺陷我们切换出它被修复的前一个版本.

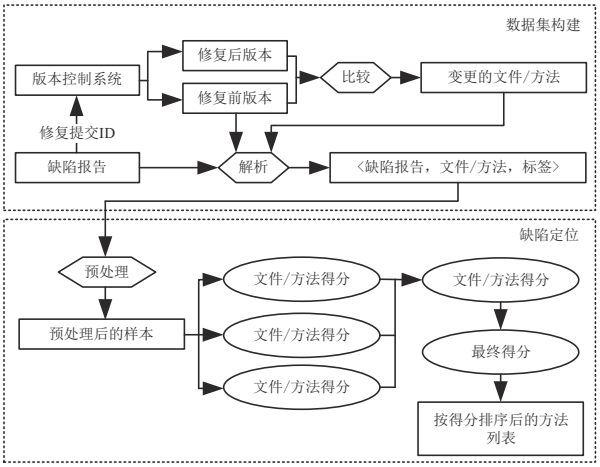

图1　FMBL总体流程

数据集构建的方法如算法1. 算法输入是缺陷报告集合和版本控制库,输出是一个缺陷定位数据集. 首先从缺陷报告中获取缺陷ID(Identity Document)、摘要、描述及修复提交ID(第4行);然后根据修复提交ID在版本控制系统中获取修复前的版本和修复后的版本(第6~7行),getVersion函数实现了这一功能,并返回了对应版本以及其中变更的文件与方法(第17~47行);最后使用上述信息创建缺陷实体并保存在数据集中(第9行).

通过这种方式,我们构建了一个同时适用于文件和方法级别的缺陷定位数据集,该数据集最大程度保留了项目的原始数据.

算法1 数据集构建算法

输入: 缺陷报告集合BR, 版本控制库gitRepo

输出: 缺陷数据集dataset

1. dataset ← ∅
2. FOR br_i∈BR DO
3. // 获取缺陷ID、摘要、描述及修复提交ID
4. Bid, sum, desc, fcid ← getBugInfo(br_i)
5. // 获取修复前版本VBF和修复后版本VAF
6. VBF, cfs, cms ← getVersion(LV, lcid, fcid)
7. VAF, cfs, cms ← getVersion(VBF, fcid^, fcid)
8. // 保存缺陷信息到数据集中
9. bug_i ← Bug(bid, sum, desc, fcid, VBF, VAF, cfs, cms)
10. dataset.add(bug_i)

```
    // 链式处理所有缺陷报告
    LV ← VAF, lcid ← fcid
  END FOR
  RETURN dataset
  // 获取缺陷所在的版本
  FUNCTION getVersion(LV, lcid, cid)
    // 检出对应版本文件
    checkout(cid, gitRepo)
    IF LV为空 THEN    // 初始化第一个版本
      FOR javaFile ∈ projectFiles DO
        file ← parse(javaFile)    // 解析文件
        version.add(file)
      END FOR
    ELSE
      version ← LV    // 建立链接
      // 获取版本删除、修改、增加的文件
      Dfs, mfs, afs ← diff(lcid, cid, gitRepo)
      FOR df ∈ dfs DO    // 处理删除的文件
        cms.add(df.m)
        cfs.add(df)
        version.remove(df)
      END FOR
      FOR mf ∈ mfs DO    // 处理修改的文件
      file ← parse(mf)
        cms.add(getChangedMethods(version.mf, file))
        cfs.add(version.mf)
        version.remove(version.mf)
        version.add(file)
      END FOR
      FOR af ∈ afs DO    // 处理增加的文件
        file ← parse(af)
        version.add(file)
      END FOR
    END IF
    RETURN, version, cfs, cms
  END FUNCTION
```

3.2 词汇相似度

词汇相似度是度量两个文档之间相似性的有效方式，两个文档间共同的词汇越多表明它们越相似. 由于缺陷报告中可能存在和缺陷代码相同的词汇，使用词汇相似度可以有效找到缺陷相关代码. 我们使用词频-逆文档频率(Term Frequency-Inverse Document Frequency, TF-IDF)[18]表示文档d中术语t的权重，其计算方式如式(1)、(2). 其中f_{td}表示术语t在文档d中的出现频次，n_t表示所有文档中出现术语t的文档数量，#docs指所有文档的数量.

$$\mathrm{tf}(t,\ d)=\log f_{\mathrm{td}}+1,\quad \mathrm{idf}\left(t\right)=\log\left(\frac{\#\mathrm{docs}}{n_t}\right) \tag{1}$$

$$\mathrm{weight}_{t\epsilon d}=\mathrm{tf}(t,d)\times \mathrm{id}(t) \tag{2}$$

在缺陷定位中文档d表示缺陷报告或代码，t表示缺陷报告或代码中的单词.

对于缺陷报告和代码，计算缺陷报告和代码的余弦相似度作为其词汇相似度的度量. 这种方式考虑了缺陷报告和代码中的共现词及词的权重，可以有效衡量两者之间的词汇相似度，其计算方法如式(3).

$$\mathrm{Lexical}=\cos\left(\boldsymbol{V}_r,\boldsymbol{V}_c\right) \tag{3}$$

其中，$\boldsymbol{V}_r$、$\boldsymbol{V}_c$分别由缺陷报告和代码中单词的权重构成.

3.3 语义相似度

对于一个缺陷报告和项目中的一个文件或方法，我们参考Huo等的CNN代码特征提取过程[10]，如图2所示. 在CNN特征提取阶段，为了保留代码单行语句的完整语义信息，分别使用两次卷积池化以提取代码行内特征和代码行间特征. 在一段文本被送入CNN前，需要将其嵌入到一个向量空间中. 我们使用Glove[19]作为词嵌入模型，它可以学习单词的全局向量表示. 我们将实验用到的6个软件项目的所有版本的文件和缺陷报告作为Glove的训练语料库，对于每个单词Glove学习到一个k维的单词向量.

本文将缺陷报告视为单行文本，使用一次卷积池化进行特征提取. 特征提取后将缺陷报告特征向量和代码特征向量拼接送入全连接层中做二分类任务，将结果通过Softmax函数生成缺陷报告和代码的语义相似度得分. 如果使用En简要表示上述过程，则缺陷报告r和代码c之间的语义相似度可以表示为式(4).

$$\mathrm{Semantic}=\mathrm{En}\left(r,\ c\right) \tag{4}$$

由于和缺陷报告相关的文件/方法一般有几个或十几个，和缺陷报告不相关的文件/方法有几千甚至上万个，因此误分类一个相关的代码单元比误分类一个不相关的代码单元带来的后果更加严重. 我们设计了一个在训练时确定的类别权重以让模型正确判别缺陷报告和代码的相关性. 我们使用随机梯度下降算法作为模型的优化算法，交叉熵损失函数作为训练的损失函数. 损失函数形式化定义如式(5)、(6)：

$$L(x,\ \mathrm{class})=\mathrm{weight}_{\mathrm{class}}\left(-x_{\mathrm{class}}+\log\left(\sum_j \mathrm{e}^{\mathrm{x_j}}\right)\right) \tag{5}$$

$$\mathrm{loss}=\frac{\sum_{i=1}^{N}L(x_i,\ \mathrm{class}_i)}{\sum_{i=1}^{N}\mathrm{weight}_{\mathrm{class}_i}} \tag{6}$$

其中，N是样本数量，x是预测结果，class是真实值，

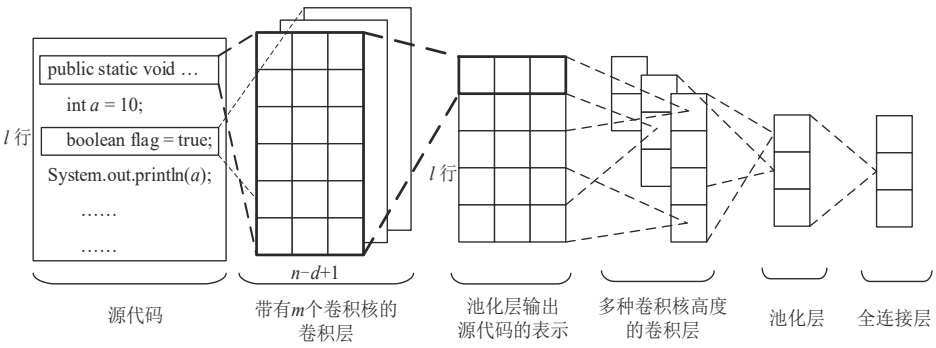

图2　CNN代码特征提取过程

weight是不同类别的权重. 式(5)计算单个样本附加权重的损失,式(6)计算所有样本的总损失.

3.4　代码长度

除了词汇相似度和语义相似度,代码的一些其他特征也可以帮助缺陷定位. 例如,越长的代码含有缺陷的可能性越高[3]. 因此,本文使用代码长度帮助评估代码含有缺陷的可能性,其计算方式如式(7).

$$\text{LengthScore} = \arctan\left(\frac{\text{len}}{\text{len}_{\text{median}}}\right) \tag{7}$$

其中,len是代码行数,$\text{len}_{\text{median}}$是所有检索的代码行数中位数.

3.5　组合得分及排序

本文从代码的三种不同角度评估其含有缺陷的可能性,分别是词汇相似度、语义相似度和代码长度. 使用它们的乘积作为代码含有缺陷的可能性评分. 代码得分计算方式如式(8).

$$\text{Score} = \text{Lexical} \times \text{Sementic} \times \text{LengthScore} \tag{8}$$

在我们的方法中,代码可以是整个文件也可以是文件中的一个方法. 对应的,代码长度即是文件长度或方法长度. 由于文件和方法的长度有明显差异,在计算语义相似度时,我们调整模型输入数据的大小、数量以适应不同粒度的特征.

现有的方法级缺陷定位工作把方法作为单独的考虑对象,没有考虑文件级的信息,割裂了方法和文件的从属关系. 如果一些方法有相似的内容但是却存在于不同的文件,这些方法级缺陷定位技术很难将这些方法区分开来,导致缺陷定位准确性较低. 考虑到方法与文件的从属关系有利于增强方法的区分度,如式(9)所示,我们将方法得分与其所在的文件得分相加作为该方法的最终得分. 其中,文件得分和方法得分在其各自粒度上计算得出.

$$\text{MethodFinalScore} = \text{MethodScore} + \text{FileScore} \tag{9}$$

最后,根据得分排序得到可疑缺陷文件/方法列表,排名靠前的文件/方法含有缺陷的可能性更大. 这种组合得分的方式可以从文件和方法两个层次上考虑缺陷和代码的相关性,以定量(非定性)的方式度量代码和缺陷报告的相关性.

4　实验

4.1　实验数据集

我们在Ye等[16]提供的一个文件级的缺陷定位数据集基础上,通过3.1节所述方法,为每个缺陷报告对应的版本建立索引并标记缺陷文件和缺陷方法,可以同时实现文件粒度和方法粒度上的缺陷定位. 数据集基本信息如表1所示. 第2列表示缺陷报告的数量. 第3、4列表示缺陷所在的项目版本的平均文件和方法数量. 第5、6列表示平均每个缺陷所涉及的文件和方法数量. 可以看出,平均每个缺陷报告对应的缺陷文件有2到3个,对应的缺陷方法有4~9个. 每个版本的项目平均文件数量和平均方法数量在几千到几万不等.

表1　数据集基本信息

项目	#报告	#平均文件	#平均方法	#平均缺陷文件	#平均缺陷方法
AspectJ	593	4 457.5	31 237.5	3.7	9.3
Eclipse	4 194	3 040.2	27 223.0	2.5	8.4
Birt	4 168	8 460.2	76 587.7	3.0	9.3
JDT	6 259	10 360.7	51 315.0	2.5	8.3
Tomcat	1 054	1 584.1	19 195.4	2.6	7.0
SWT	3 596	1 925.5	32 156.0	2.0	4.1

由于在软件演化过程中,时间跨度过长的版本之间差异过大,对于每个项目,将缺陷按时间顺序每600个划分一次子数据集. 在每个子数据集中,前80%的缺陷用于训练,后20%的缺陷用于测试.

4.2　基线方法

本文选择BugLocator[3]、BLIA[4]和BugPecker[5]作为基线方法,评估本文提出的缺陷定位方法的有效性. BugLocator是文件粒度的缺陷定位技术,本文将方法视为文件输入得到BugLocator的方法级定位结果.

4.3　评估指标

为了评估本文提出的缺陷定位方法,和多数缺陷

定位方法[3~10]相同，本文选择前N预测精度（Accuracy@N）、平均精度均值（Mean Average Precision，MAP）和平均倒数排名（Mean Reciprocal Rank，MRR）作为评估指标.

前N预测精度表示在前N（N=1，5，10）个返回结果中包含缺陷代码的缺陷数量占全部缺陷数量的比例. 给定一个缺陷报告，如果结果列表中的前N个结果至少包含一个缺陷代码，就认为成功定位到缺陷.

一份缺陷报告常常对应多段代码，每段代码在缺陷定位返回的结果列表中都有对应的排名. 平均精度（Average Precision，AvgP）由式（10）定义，度量一份缺陷报告的平均定位准确率.

$$\mathrm{AvgP}=\frac{1}{|C_b|}\sum_{k=1}^{|C_b|}\frac{k}{\mathrm{rank}_k} \tag{10}$$

其中C_b表示缺陷代码集合，$|C_b|$表示缺陷代码数量，rank_k表示第k段缺陷代码在结果列表中的排名. 平均精度均值度量全部缺陷的定位准确率，计算公式如式（11）：

$$\mathrm{MAP}=\sum_{i=1}^{|R|}\frac{\mathrm{AvgP}_i}{|R|} \tag{11}$$

其中，R表示所有缺陷报告集合，$|R|$表示所有缺陷报告数量，AvgP_i表示第i个缺陷报告的定位准确率.

平均倒数排名表示相关代码的位置倒数的平均值，计算公式如式（12）.

$$\mathrm{MRR}=\frac{1}{|R|}\sum_{i=1}^{|R|}\frac{1}{\mathrm{rank}_i} \tag{12}$$

其中，R为缺陷报告的集合，$|R|$表示缺陷报告的数目，rank_i表示定位出的与第i个缺陷报告相关的代码最靠前排名.

4.4 实验设置

本文实验环境为Intel® Xeon® Silver 4214R CPU、64 GB内存和Tesla T4 GPU. 我们将文件最大行数设置为400，方法最大行数设置为30，每行最大词数设置为20，GloVe词向量嵌入维度设置为128. 为每份缺陷报告在项目中随机挑选20个文件、100个方法作为负样本进行训练. CNN卷积核高度设置为3、4、5，卷积核数量设置为100.

5 实验结果及分析

为了评估我们的方法，我们设计了三个研究问题.

研究问题1 FMBL的定位效果如何，能否优于其他方法粒度缺陷定位技术？

研究问题2 加入文件信息对方法粒度缺陷定位改进效果如何？

研究问题3 本文从代码的三种不同角度计算缺陷报告和代码的相关性，各个角度得分对最终结果影响如何？

5.1 研究问题1

为了回答研究问题1，我们在六个项目上评估了我们的方法，实验结果如表2所示.

FMBL在六个项目上的平均Accuracy@10、MAP、MRR分别为0.436、0.223、0.296，最高Accuracy@10达到了0.559并且六个项目中的三个Accuracy@10都在0.5以上，表明FMBL能够有效地定位到缺陷方法. FMBL在Birt项目上的Accuracy@10仅有0.216. 因为Birt有更多的文件和方法，数量是AspectJ的两倍.

表2 FMBL的实验结果

项目	Acc@1	Acc@5	Acc@10	MAP	MRR
AspectJ	0.256	0.378	0.456	0.273	0.324
Birt	0.081	0.162	0.216	0.074	0.125
Eclipse	0.144	0.308	0.365	0.160	0.223
JDT	0.340	0.433	0.515	0.323	0.396
Tomcat	0.294	0.461	0.559	0.258	0.374
SWT	0.243	0.414	0.505	0.251	0.335
平均	0.226	0.359	0.436	0.223	0.296

本文在AspectJ、Tomcat、SWT（Standard Widget Toolkit）三个项目上和基线方法对比. 实验中使用的数据均是已经完成缺陷定位和修复的历史缺陷数据，这部分数据有较多的评论信息（一般为测试、开发人员针对该缺陷进行沟通的过程信息）. 而新提交的缺陷一般不包含评论信息. 因此，为了能够客观公正地评估论文提出方法对新提交缺陷进行定位的有效性，我们的实验数据没有使用评论信息. 实验结果如表3所示. 和基线方法相比，FMBL在所有度量指标上均有显著的提升. 基线方法均将方法粒度的代码作为单独考虑的对象. 一方面，一个缺陷往往涉及多个方法，单个方法和缺陷报告之间不会有很高的相似度. 另一方面，项目中方法数量极多，仅使用缺陷报告和方法的相似度对方法进行排名难以达到很好的区分度. BLIA在排名前十的文件中寻找缺陷方法，虽然缩小了检索范围，但方法粒度定位结果严重依赖文件粒度定位结果. 此外该方法忽略了文件之间差异，同等地看待各个文件中的方法，丢失了文件信息. 本文通过组合文件和方法的得分加以改进. BugPecker关注缺陷的修复历史以扩展方法的内容，但抛弃了缺陷报告和代码最直接的词汇相似度，导致最终的定位效果较差.

从实验结果看，FMBL的定位效果优于基线方法. 三个项目上的平均Accuracy@1、Accuracy@5、Accuracy@10分别达到0.264、0.417、0.506，MAP、MRR分别达到0.26、0.34. 和BugPecker相比，FMBL的平均Accuracy@1、Accuracy@5、Accuracy@10分别提高34.4%、

表3　FMBL与基线方法的比较结果

项目	方法	Acc@1	Acc@5	Acc@10	MAP	MRR
AspectJ	BugLocator	0.062	0.147	0.197	0.095	0.125
	BLIA1.5	0.033	0.108	0.211	0.074	0.076
	BugPecker	0.229	0.270	0.354	0.263	0.291
	FMBL	**0.256**	**0.378**	**0.456**	**0.273**	**0.324**
Tomcat	BugLocator	0.098	0.225	0.324	0.126	0.174
	BLIA1.5	0.042	0.186	0.297	0.104	0.112
	BugPecker	0.122	0.137	0.158	0.121	0.143
	FMBL	**0.294**	**0.461**	**0.559**	**0.258**	**0.374**
SWT	BugLocator	0.065	0.137	0.173	0.088	0.112
	BLIA1.5	0.050	0.125	0.166	0.075	0.087
	BugPecker	0.239	0.282	0.369	**0.253**	0.267
	FMBL	**0.243**	**0.414**	**0.505**	0.251	**0.335**

81.8%、72.5%,MAP和MRR分别提高22.8%、47.4%.

5.2　研究问题2

为了回答研究问题2,本文实验评估了仅使用方法得分、仅使用文件得分和组合两者得分的缺陷定位方法,表4给出了定位结果. 其中,仅使用文件得分的缺陷定位是指在得到文件粒度的得分后,按方法所在文件的得分为方法排序. 从实验结果中可以看到,有的项目仅依靠文件得分就有很好的定位效果,如JDT项目的定位结果中仅使用文件得分的Accuracy@1值为0.34. 但有的项目却不能仅依靠文件得分,如Tomcat的定位结果中仅依靠文件得分的Accuracy@1仅0.02.

表4　使用不同粒度信息的定位结果

项目名称	使用信息	Acc@1	Acc@5	Acc@10	MAP	MRR
AspectJ	方法	0.176	0.275	0.352	0.157	0.233
	文件	0.144	0.200	0.222	0.166	0.182
	方法+文件	**0.256**	**0.378**	**0.456**	**0.273**	**0.324**
Birt	方法	0.054	0.126	0.198	0.043	0.101
	文件	0.045	0.072	0.090	0.042	0.062
	方法+文件	**0.081**	**0.162**	**0.216**	**0.074**	**0.125**
Eclipse	方法	0.125	0.269	**0.385**	0.106	0.203
	文件	0.135	0.183	0.231	0.159	0.168
	方法+文件	**0.144**	**0.308**	0.365	**0.160**	**0.223**
JDT	方法	0.175	0.309	0.392	0.136	0.248
	文件	0.340	0.351	0.371	0.315	0.351
	方法+文件	**0.340**	**0.433**	**0.515**	**0.323**	**0.396**
Tomcat	方法	0.206	0.363	0.441	0.218	0.280
	文件	0.020	0.108	0.167	0.043	0.055
	方法+文件	**0.294**	**0.461**	**0.559**	**0.258**	**0.374**
SWT	方法	0.072	0.135	0.234	0.080	0.120
	文件	0.162	0.198	0.252	0.161	0.194
	方法+文件	**0.243**	**0.414**	**0.505**	**0.251**	**0.335**

在AspectJ项目中不考虑文件得分,仅依靠方法和缺陷报告的相似度进行排序,其定位结果的Accuracy@10仅0.352,添加方法所在的文件得分后提升至0.456,提升了29.5%. 同时,其余各项指标也均有显著提高. 因此,在六个项目上的实验研究结果表明,仅依靠文件信息或方法信息都不能有效定位缺陷,组合文件信息和方法信息可以有效改善最终的定位结果.

5.3　研究问题3

为了回答研究问题3,本文从完整模型中分别移除词汇相似度、语义相似度和代码长度来评估代码不同角度的信息对结果的影响,实验结果如表5所示. 其中no_ls、no_ss和no_cls分别表示从完整模型中移除词汇相似度得分、语义相似度得分和代码长度得分,all表示完整模型.

表5　代码不同角度的信息对最终结果的影响

项目名称	模型	ACC@1	ACC@5	ACC@10	MAP	MRR
AspectJ	no_ls	0.099	0.132	0.154	0.061	0.122
	no_ss	0.055	0.132	0.165	0.069	0.096
	no_cls	0.121	0.242	**0.363**	0.138	0.192
	all	**0.176**	**0.275**	0.352	**0.157**	**0.233**
Birt	no_ls	0.018	0.018	0.027	0.011	0.024
	no_ss	0.018	0.054	0.054	0.015	0.037
	no_cls	0.045	0.117	0.189	0.037	0.092
	all	**0.054**	**0.126**	**0.198**	**0.043**	**0.101**
Eclipse	no_ls	0.029	0.077	0.115	0.027	0.063
	no_ss	0.038	0.144	0.202	0.045	0.101
	no_cls	0. 115	0.24	0.356	0.102	0.193
	all	**0.125**	**0.269**	**0.385**	**0.106**	**0.203**
JDT	no_ls	0.031	0.041	0.082	0.014	0.045
	no_ss	0.021	0.062	0.124	0.033	0.055
	no_cls	0.175	0.309	0.392	0.136	0.247
	all	**0.175**	**0.309**	**0.392**	**0.136**	**0.248**
Tomcat	no_ls	0.029	0.059	0.108	0.035	0.047
	no_ss	0.157	0.284	0.402	0.162	0.231
	no_cls	0.136	0.331	0.439	0.197	0.274
	all	**0.206**	**0.363**	**0.441**	**0.218**	**0.28**
SWT	no_ls	0.036	0.045	0.099	0.048	0.05
	no_ss	0.054	0.108	0.153	0.068	0.088
	no_cls	0.072	0.135	0.225	0.075	0.12
	all	**0.072**	**0.135**	**0.234**	**0.08**	**0.12**

在移除词汇相似度后,所有项目的定位结果都出现了不同程度的下降,和其他信息相比,词汇相似度对定位结果贡献最大. 移除语义相似度后,文件级别定位结果中,在五个项目的定位效果出现了下降,仅Eclipse项目的定位效果增加. 这是因为在Eclipse中,用于计

算语义相似度的深度学习模型未能达到很好的效果，过差的语义相似度计算模型导致了定位结果的降低. 期望我们的方法能够在充分训练的情况下对所有项目有普遍适用性，我们对于所有项目都设置了相同的超参数进行训练. 虽然项目之间的差异性导致了训练结果的波动，但是总体看来在大多数项目中语义相似度对最终结果是有提升的. 在SWT项目中，语义相似度发挥了重要的作用，在移除语义相似度后Accuracy@1的定位结果从0.517下降到0.125，降幅75.8%. 在方法级别的定位结果中，语义相似度均能增强最终结果.

相较于和词汇相似度和语义相似度，代码长度对定位的最终结果影响较小. 如式(7)所示，本文使用arctan函数来计算代码的长度得分，Zhou等[3]使用exp函数来计算代码的长度得分. 本文对比了不同长度表示函数和不考虑代码长度相比对方法级定位结果的影响，结果如表6所示. 其中len指代码行数.

表6 不同代码长度表示函数对定位结果的影响

长度函数	定位粒度	ACC@1	ACC@5	ACC@10	MAP	MRR
$\frac{1}{1+e^{-\frac{len-len_{min}}{len_{max}-len_{min}}}}$	文件	0.074 1	0.014 1	−0.000 7	0.038 5	0.034 3
	方法	−0.077 4	−0.035 6	−0.045 0	−0.058 9	−0.054 1
$\arctan(\frac{len}{len_{mean}})$	文件	**0.323 5**	0.025 6	-0.038 3	**0.095 3**	**0.114 2**
	方法	0.112 7	0.066 8	0.022 1	0.075 7	0.064 1
$\arctan(\frac{len}{len_{median}})$	文件	0.117 2	**0.069 5**	**0.013 0**	0.065 9	0.062 4
	方法	**0.148 3**	**0.066 8**	**0.027 8**	**0.081 1**	**0.073 4**

从实验结果来看，在方法粒度上使用exp长度表示函数不能得到很好的效果，尽管在文件粒度上exp函数是适用的. arctan_mean使用代码行数均值，和arctan_median相比，在文件级Accuracy@1上有更好的效果，但是也有降低总体定位结果的风险. 使用代码行数中位数则更加平稳，在文件级和方法级都能改善缺陷定位结果，对MAP和MRR的提升均在6%以上.

6 结论

本文将文件信息添加到方法级软件缺陷定位中，考虑方法与文件的从属关系，有效增强方法级缺陷定位效果. 实验结果表明，结合代码各个角度的得分可以达到更优的效果. 未来计划进一步研究深度学习和知识图谱[20]在软件缺陷定位上的应用，旨在解决不同项目之间的差异对深度学习技术的影响，增强软件缺陷定位技术的适用性.

参考文献

[1] 郭肇强, 周慧聪, 刘释然, 等. 基于信息检索的缺陷定位: 问题、进展与挑战[J]. 软件学报, 2020, 31(9): 2826-2854.
GUO Z Q, ZHOU H C, LIU S R, et al. Information retrieval based bug localization: Research problem, progress, and challenges[J]. Journal of Software, 2020, 31(9): 2826-2854. (in Chinese)

[2] ZHU Z Y, LI Y, WANG Y, et al. A deep multimodal model for bug localization[J].Data Mining and Knowledge Discovery, 2021, 35(4): 1369-1392.

[3] ZHOU J, ZHANG H Y, LO D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports[C]//2012 34th International Conference on Software Engineering (ICSE). Zurich: IEEE, 2012: 14-24.

[4] YOUM K C. Improved bug localization based on code change histories and bug reports[J]. Information and Software Technology, 2017, 82: 177-192.

[5] CAO J M, YANG S L, JIANG W H, et al. BugPecker: locating faulty methods with deep learning on revision graphs[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. Melbourne: IEEE/ACM, 2020: 1214-1218.

[6] 李斌, 陈定山, 孙小兵, 等. 面向缺陷知识的多特征匹配搜索算法[J]. 电子学报, 2021, 49(4): 661-664.
LI B, CHEN D S, SUN X B, et al. Multi-feature matching search algorithm for bug knowledge[J]. Acta Electronica Sinica, 2021, 49(4): 661-664. (in Chinese)

[7] SISMAN B, KAK A C. Incorporating version histories in information retrieval based bug localization[C]//Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. Zurich: IEEE, 2012: 50-59.

[8] WANG S W, LO D. Version history, similar report, and structure: Putting them together for improved bug localization[C]//Proceedings of the 22nd International Conference on Program Comprehension. Hyderabad: ACM, 2014: 53-63.

[9] SAHA R K, LEASE M, KHURSHID S, et al. Improving bug localization using structured information retrieval[C]// 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE, 2013: 345-355.

[10] HUO X, LI M, ZHOU Z H. Learning unified features from natural and programming languages for locating buggy source code[C]//Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. New York: AAAI Press, 2016: 1606-1612.

[11] WANG B, XU L, YAN M, et al. Multi-dimension convolutional neural network for bug localization[J]. IEEE Transactions on Services Computing, 2022, 15(3): 1649-

1663.
[12] SOHN J, YOO S. Empirical evaluation of fault localisation using code and change metrics[J]. IEEE Transactions on Software Engineering, 2021, 47(8): 1605-1625.
[13] KIM Y, MUN S, YOO S, et al. Precise learn-to-rank fault localization using dynamic and static features of target programs[J]. ACM Transactions on Software Engineering and Methodology, 2019, 28(4): 1-34.
[14] LI X, LI W, ZHANG Y Q, et al. DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization[C]//ISSTA 2019: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. Beijing: ACM Press, 2019: 169-180.
[15] ZHANG W, LI Z Q, WANG Q, et al. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion[J]. Information and Software Technology, 2019, 110: 121-135.
[16] YE X, BUNESCU R, LIU C. Learning to rank relevant files for bug reports using domain knowledge[C]//FSE 2014: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong: Association for Computing Machinery 2014: 689-699.
[17] BO L L, ZHU X R, SUN X B, et al. Are similar bugs fixed with similar change operations? An empirical study [J]. Chinese Journal of Electronics, 2021, 30(1): 55-63.
[18] WORTH D. Introduction to Modern Information Retrieval [M]. 3rd ed. Kingston, Australian: Australian Academic and Research Libraries, 2010.
[19] PENNINGTON J, SOCHER R, MANNING C D. GloVe: Global vectors for word representation[J]. British Journal of Neurosurgery, 2017, 31(6): 682-687.
[20] 孙小兵, 王璐, 王经纬, 等. 基于知识图谱的bug问题探索性搜索方法[J]. 电子学报, 2018, 46(7): 1578-1583.
SUN X B, WANG L, WANG J W, et al. Construct knowledge graph for exploratory bug issue searching[J]. Acta Electronica Sinica, 2018, 46(7): 1578-1583. (in Chinese)

作者简介

薄莉莉　女，1989年10月出生于山东省泰安市．现为扬州大学信息工程学院讲师，从事软件测试方面的研究工作．
E-mail: lilibo@yzu.edu.cn

朱　程　男，1997年11月出生于江苏省扬州市，现为扬州大学信息工程学院硕士研究生，主要研究方向为软件缺陷定位．
E-mail: zhucheng0a@gmail.com

李　斌(通讯作者)　男，1965年12月出生于江苏省扬州市．现为扬州大学信息工程学院教授、博士生导师．主要研究方向为软件工程．在国内外发表学术论文150余篇．
E-mail: lb@yzu.edu.cn

孙小兵　男，1985年9月出生于江苏省泰州市．现为扬州大学信息工程学院教授．主要研究方向为智能化软件工程．在国内外发表学术论文80余篇．
E-mail: xbsun@yzu.edu.cn