软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(11):4008−4026 [doi: 10.13328/j.cnki.jos.006339]

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

基于源代码扩展信息的细粒度缺陷定位方法*

李晓卓[1,2], 卿笃军[1,2], 贺也平[1,2,3], 马恒太[1,2]

[1](基础软件国家工程研究中心(中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](计算机科学国家重点实验室(中国科学院 软件研究所), 北京 100190)

通信作者: 贺也平, E-mail: yeping@iscas.ac.cn; 马恒太, E-mail: hengtai@iscas.ac.cn

摘 要: 基于信息检索的缺陷定位技术, 利用跨语言的语义相似性构造检索模型, 通过缺陷报告定位源代码错误, 具有方法直观、通用性强的特点. 但是由于传统基于信息检索的缺陷定位方法将代码作为纯文本进行处理, 只利用了源代码的词汇语义信息, 导致在细粒度缺陷定位中面临候选代码语义匮乏产生的准确性低的问题, 其结果有用性还有待改进. 通过分析程序演化场景下代码改动与缺陷产生间的关系, 提出一种基于源代码扩展信息的细粒度缺陷定位方法, 以代码词汇语义显性信息及代码执行隐性信息共同丰富源代码语义实现细粒度缺陷定位. 利用定位候选点的语义相关上下文丰富代码量, 以代码执行中间形式的结构语义实现细粒度代码的可区分, 同时以自然语言语义指导基于注意力机制的代码语言表征生成, 实现细粒度代码与自然语言间的语义映射, 从而实现细粒度缺陷定位方法 FlowLocator. 实验分析结果表明: 与经典的 IR 缺陷定位方法相比, 该方法定位准确性在 Top-*N* 排名、平均准确率及平均倒数排名上都有显著提高.

关键词: 缺陷定位; 演化程序; 信息检索; 语义信息; 伪孪生网络

中图法分类号: TP311

中文引用格式: 李晓卓, 卿笃军, 贺也平, 马恒太. 基于源代码扩展信息的细粒度缺陷定位方法. 软件学报, 2022, 33(11): 4008−4026. http://www.jos.org.cn/1000-9825/6339.htm

英文引用格式: Li XZ, Qing DJ, He YP, Ma HT. Fine-grained Bug Location Method Based on Source Code Extension Information. Ruan Jian Xue Bao/Journal of Software, 2022, 33(11): 4008−4026 (in Chinese). http://www.jos.org.cn/1000-9825/6339.htm

Fine-grained Bug Location Method Based on Source Code Extension Information

LI Xiao-Zhuo[1,2], QING Du-Jun[1,2], HE Ye-Ping[1,2,3], MA Heng-Tai[1,2]

[1](National Engineering Research Center of Fundamental Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: Bug location based on information retrieval (IR) uses cross language semantic similarity to construct a retrieval model to locate source code errors through bug report. However, the traditional method of bug location based on IR treats the code as pure text and only uses the lexical semantic information of source code, which leads to the problem of low accuracy caused by the lack of candidate code semantics in fine-grained bug location, and the usefulness of the results needs to be improved. By analyzing the relationship between code change and bug generation in the scenario of program evolution, this study proposes a fine-grained bug location method based on source code extension information, the explicit semantic information of code vocabulary and implicit information of code execution are used to enrich source code semantics to realize fine-grained bug location. Based on the location candidate points, the semantic context is used to enrich the code quantity, and the structural semantics of code execution intermediate language is used to realize fine-grained code

* 基金项目: 核高基国家科技重大专项(2014ZX01029101); 中国科学院战略性先导科技专项(XDA-Y01-01)
收稿时间: 2020-09-30; 修改时间: 2021-01-06, 2021-03-02; 采用时间: 2021-03-17; jos 在线出版时间: 2021-04-20

distinguishability. Meanwhile, natural language semantics is used to guide the generation of code language representation based on attention mechanism, the semantic mapping between fine-grained code and natural language is implemented to implement fine-grained bug location method FlowLocator. The experimental results show that compared with the classical IR bug location method, the location accuracy of this method is significantly improved in the Top-*N* rank, mean average precision (MAP) and mean reciprocal rank (MRR).

Key words: bug location; evolution program; information retrieval; semantic information; pseudo-siamese network

在软件项目开发中, 程序员需要用大量的时间进行缺陷定位、修复及验证工作. 软件缺陷即为计算机软件或程序中存在的某种破坏正常运行能力的问题, 导致软件产品在某种程度上不能满足用户的需求[1]. 多数缺陷是由于源代码编码错误导致的功能缺陷, 通常在缺陷报告(bug report, BR)中对现象及产生场景进行描述说明[2].

在软件演化过程中, 缺陷定位和修复是最耗费时间及精力的工作之一[2].Nguyen 等人[3]在 2011 年首次提出将信息检索(information retrieval, IR)中的特征分析技术应用到缺陷定位领域中, 并通过构造 LDA (latent dirichlet allocation)主题模型定位缺陷. 与基于程序频谱的动态缺陷定位、基于程序规约及基于符号执行的静态缺陷定位研究相比, 基于 IR 的缺陷定位方法应用限制少、方法便捷、可迁移性强, 逐渐在缺陷定位领域兴起[4−8]. 基于 IR 的缺陷定位研究利用语义相似性将缺陷报告中自然语言及源代码中代码语言的语义相对应, 基于缺陷报告语义在源代码中进行错误位置的检索. 该类方法的有效性与两种语言的语义信息量有关, 代码信息量越大, 则可区分性越强, 语义映射过程中产生误报的可能性越低.

目前的 IR 缺陷定位研究通过代码相关产物(注释、代码变更说明等)扩充代码信息量, 或深入挖掘代码自身信息提高缺陷定位准确性, 如利用系统定义 API 信息[4]、程序注释[5]等相关内容扩充代码词汇语义, 或采用词嵌入技术(word embedding)挖掘词项(token)间的深层次关系[8]提高定位准确性. 但目前的研究基于程序命名规则(例如驼峰命名)的可理解性, 简单地将缺陷报告中自然语言及源代码中代码语言间的语义差异理解为词汇语义差异, 将代码语言作为纯文本进行语义提取, 利用代码中的人工命名词汇与缺陷报告中自然语言之间的语义相似性进行缺陷定位, 导致细粒度缺陷定位中代码语义的缺乏, 代码语义信息量不足以做到语义可区分, 因此目前基于 IR 的缺陷定位研究多数停留在程序文件粒度级[1,9]. 同时, 该类定位结果(基于代码文件、类等粒度)基于完美缺陷理解假设[1], 脱离了缺陷产生的环境及场景, 导致结果不被程序修复人员理解且难以真正应用[9,10]. 目前的 IR 缺陷定位研究从定位粒度及形式两方面受到结果有用性质疑[11−16]: 一方面, 定位粒度太粗, 不足以支持后续缺陷修复方法或工具的应用; 另一方面, 仅仅给出可能的缺陷位置, 缺少与缺陷位置相关信息, 与缺陷来源及相关代码缺乏关联, 使得修复人员难以对缺陷进行理解及修复.

开展细粒度 IR 缺陷定位研究主要面临以下问题: 首先, 随着代码定位粒度的细化, 直接表现为定位候选代码的代码量减少, 传统特征分析方法下, 代码直接包含的词汇信息量大大减小, 例如代码中包含可理解性词汇量的减少, 进而难以做到细粒度代码间的语义可区分, 导致 IR 缺陷定位语义映射过程的不准确, 则定位结果准确率降低而误报率增高; 第二, 随着代码定位粒度的细化, 定位候选代码的可扩充代码语义大大减少, 简单地采用传统附加信息扩充代码语义的方式难以有效地扩展代码信息, 进而无法做到定位候选代码间的语义可区分; 第三, 传统的细粒度缺陷定位指缺陷定位形式以代码块为单位进行细化, 即由文件、类、方法体到代码语句行, 这种定位方法的有效性基于完美缺陷理解假设, 无法为程序员提供与程序修复相关的辅助信息.

针对第 1 个问题, 本文利用代码关联性扩充代码词汇语义显性信息, 即直观上代码包含的明确信息, 使得后续的特征分析具有更多的显性信息来源, 实现代码语义信息量的直接扩充. 目前的 IR 缺陷定位方法基于文件、类、方法体简单地按代码序扩展定位代码范围, 包含了大量无关程序错误点的冗余信息, 同时受特征分析方法的限制, 在该类扩展方法下, 难以做到缺陷定位粒度的再细化. 分析发现: 代码的语义表现方式与自然语言有很大的区别[17], 实体种类更多, 代码语言的组成不仅存在人工定义的变量名、函数名等标识符, 还具有表达代码执行性质的操作符等, 且相关分析表明代码元素间的组成形式与代码语义直接相关[18], 即与程序错误关联的代码对应了缺陷现象自然语言描述语义. 本文提出了一种基于缺陷定位候选点, 以静态分析方法模拟动态执行路径, 以关联代码扩充候选点代码量的方法. 与传统按代码序扩充代码量相比, 缺陷候选代码

为文件等粒度的简单方法, 本文以获取缺陷相关信息的方式, 有针对性地扩充代码量, 有效减少了定位候选代码行数, 去除了程序错误无关信息, 实现了细粒度的缺陷定位. 本文研究在代码演化场景下的缺陷定位问题, 利用版本间代码改动(diff code)与缺陷位置间的关系确定代码定位候选点, 因此上述缺陷定位候选点假设是合理的.

针对第2个问题, 本文在丰富了代码词汇语义显性信息的基础上, 通过深入挖掘代码执行隐性信息, 即代码执行过程中存在的与代码功能语义相关的信息, 实现代码信息量的深层次扩充, 从而使细粒度候选代码语义可区分. 目前的 IR 缺陷定位方法采用代码相关产物扩充词汇语义信息, 在细粒度的代码中难以实现有效的信息挖掘; 同时, 将代码作为纯文本进行处理会使部分代码语义缺失. 分析发现: 代码的语义不仅包含表面关键词语义以及注释等代码相关信息语义, 还包含隐含的代码执行结构语义. 由于功能需求直接影响代码编写结构, 基于抽象语法树(abstract syntax tree, AST)的代码语言结构语义与功能描述自然语言语义具有相关性[19]. 分析代码元素间组成结构, 是解决细粒度缺陷定位中语义匮乏问题的有效方式. 代码执行中间语言的结构语义信息丰富了代码信息量, 使得在细粒度代码上进一步实现了语义可区分. 本文提出了一种基于硬注意力机制[20]提取代码执行中间形式结构信息进行语义扩展的方法, 有效地提高了细粒度代码语义的可区分性, 在代码粒度细化的基础上提高了缺陷定位的准确性.

通过以下例子进一步说明代码执行隐性信息的有效性. 由于 Linux 内核升级, 程序代码随之进行演化. 图 1 和图 2 分别为 Intel 网卡驱动 e1000 及 Freescale 驱动随着内核演化在前向移植过程中产生的缺陷, 两者均会导致程序构建错误, 缺陷现象相似, 即缺陷自然语言描述语义相似. 从源代码角度分析, 两个缺陷调用同一系统定义 API, API 调用参数来源及处理过程相同. 不仅如此, 触发缺陷的程序执行路径的 AST 图结构相似, 即触发缺陷的测试用例执行路径语义相似且缺陷来源相同, 均是由于 Linux 内核代码调用导致, 调用方式及处理参数相同. 因此, 分析出代码执行中间语言结构语义信息与缺陷现象描述语义信息间具有较高的相关性.

```
1   /linux-3.5/drivers/net/ethernet/intel/e1000/e1000_main.c
2   static void e1000_receive_skb(struct e1000_adapter *adapter, u8 status,
3   __le16 vlan, struct sk_buff *skb)
4   {skb→protocol=eth_type_trans(skb,adapter→netdev); if (status &
5   E1000_RXD_STAT_VP) {
6   u16 vid=le16_to_cpu(vlan) & E1000_RXD_SPC_VLAN_MASK;
7   −__vlan_hwaccel_put_tag(skb,vid);
8   +__vlan_hwaccel_put_tag(skb,htons(ETH_P_8021Q),vid);   //fix this bug
9   }
10  napi_gro_receive(&adapter→napi, skb); }
```

图 1 e1000 驱动前向移植产生的缺陷

```
1   diff—gita/drivers/net/ethernet/freescale/gianfar.c
2   b/drivers/net/ethernet/freescale/gianfar.c
3   index 5155544..2375a01 100644
4   −−− a/drivers/net/ethernet/freescale/gianfar.c
5   +++ b/drivers/net/ethernet/freescale/gianfar.c
6   @@ −2731,7 +2731,7 @@ static void gfar_process_frame(struct
7   net_device*dev, struct sk_buff*skb, if (dev→features &
8   NETIF_F_HW_VLAN_CTAG_RX &&
9   fcb→flags & RXFCB_VLN)
10  −__vlan_hwaccel_put_tag(skb,fcb→vlctl);
11  +__vlan_hwaccel_put_tag(skb,htons(ETH_P_8021Q),fcb→vlctl);
```

图 2 Freescale 驱动前向移植产生的缺陷

针对第 3 个问题, 通过分析程序修复人员的修复过程及习惯, 研究发现: 在演化程序中, 程序员关注版本间代码改动需求与缺陷现象间的关联, 并且多数采用断点调试及单步执行的方式[21], 观察执行结果与程序现象, 利用代码间的依赖关系定位缺陷位置. 本文研究软件系统演化过程中的细粒度缺陷定位问题, 对于这一代码开发过程中的常见场景, 以控制依赖关系模拟触发缺陷的动态执行结果, 基于缺陷位置获取与缺陷现象语义相关的上下文, 刻画缺陷来源, 打破了细粒度缺陷定位中文件、类、方法体等按代码序实现定位粒度细

化的传统概念, 在定位粒度细化的基础上, 以控制依赖相关的代码段作为定位结果, 为缺陷修复提供了从缺陷引入到向下传播的过程; 同时, 以控制依赖关系关联了与缺陷位置相关的代码模块, 有效地辅助后续的程序修复工作.

本文提出一种在程序演化过程中, 结合缺陷产生场景缩小定位代码候选范围、基于缺陷位置语义扩充代码词汇语义显性信息量、结合代码执行中间形式以结构语义丰富代码执行隐性信息量的细粒度缺陷定位方法, 提高细粒度缺陷定位的准确性. 本文在分析缺陷场景特点的基础上, 以版本间的 diff code 作为定位原点, 提出以静态分析模拟动态执行路径的方式扩充代码量, 利用与其具有控制依赖关系的上下文, 以双向切片的方式构建 diff-bug-link(DBL), 以具有控制依赖关系的代码段作为定位结果, 将定位粒度由“点”转向了具有程序修复辅助信息的“流”, 结合提交(commit)信息细粒度定位缺陷. 同时, 本文结合源代码各元素间的影响关系提取代码结构语义, 通过硬注意力机制关注代码中词项的组成关联, 基于代码结构实现词项间的关联方式构建, 丰富代码执行隐性信息, 从而完成细粒度代码间的语义可区分. 本文采用语义执导的方式解决跨语言间的语义差异, 利用不同的方式对两种语言的语义进行提取, 并利用伪孪生网络(pseudo-siamese network, PSN)[22]实现监督学习, 利用缺陷报告中自然语言语义修正代码语言语义表征生成, 完成细粒度缺陷定位中的语义映射. 在衡量两种语言的语义差异时, 本文将定位问题转换为信息检索问题, 实现在语义层面上通过缺陷报告细粒度定位源代码错误位置的目的, 最终形成 FlowLocator 定位方法.

本文的方法在包含 Jena, Mahout 等 JAVA 项目的数据集上进行实验, 我们的实验结果证实 FlowLocator 是一种更为准确且可用性更高的方法. 综上所述, 本文的贡献主要有以下几点.

- 本文提出了一种细粒度缺陷定位中代码词汇语义显性信息扩充方法, 利用基于语义相似性改进后的蒙特卡洛算法以静态分析的方式模拟动态执行路径, 将具有控制依赖关系的定位候选点语义相关代码作为缺陷定位候选代码, 实现以缺陷定位原点为基础的定位点有效代码量的扩充. 通过这种代码扩充后, 实验结果表明: 缺陷定位候选代码粒度小于传统 IR 缺陷定位的文件级粒度的代码行数一个数量级, 并且本文方法具有更高的准确性.
- 本文提出了一种细粒度缺陷定位中代码执行隐性信息扩充方法, 在丰富代码词汇语义显性信息的基础上, 通过代码执行中间形式的结构信息扩充细粒度代码语义, 利用硬注意力机制基于 AST 解析实现代码中结构语义的提取, 构造 Attention Tree-GRU 模型, 提高了细粒度候选代码语义的区分度. 本文关注自然语言与代码语言间语义的表达差异, 利用 PSN 框架以缺陷报告中的自然语言语义修正源代码语义表征生成, 达到弥补自然语言与代码语言间语义差异的目的, 提高了跨语言语义映射的准确性.
- 本文实现了一个细粒度缺陷定位工具, 该工具通过控制依赖关系, 更有效地帮助程序员理解软件演化过程中缺陷的来源及传播过程, 提高人工缺陷修复的效率. 通过实验验证: 在定位准确性及可用性上, 本方法优于传统的 IR 缺陷定位方法.

本文第 1 节介绍细粒度缺陷定位研究相关工作.第 2 节说明方法动机、背景、思路及具体细节.第 3 节说明验证评估工作,包括验证问题、测评对象、测评指标、实验流程以及关键参数设置.第 4 节对实验评估结果进行详细分析及总结.最后总结全文,并对下一步研究工作进行展望.

1 相关工作

细粒度的缺陷定位问题, 一直是缺陷分析研究领域的重点问题[23]. 基于 IR 的缺陷定位方法在长期积累的缺陷报告中, 通过特征分析或查询重构等方法, 实现缺陷报告中自然语言与源程序中代码语言的语义映射, 通过语义相似性, 实现缺陷定位的目的. 该类研究关注方法的实用性及有用性[9], 即定位粒度细化及定位结果的可用. 本文主要关注细粒度缺陷定位如何在保证定位效率及准确率的同时, 给出一种具有修复辅助信息的定位结果. 因此, 本节从缺陷细粒度定位研究(研究对象相关)及代码分析方法(研究方法相关)两部分, 对已有工作进行分析.

1.1 基于信息检索的缺陷定位

缺陷报告是缺陷定位工作的起点及重要信息来源[23]. 近年对于 IR 缺陷定位的研究由于深度学习研究领域的兴起逐渐被研究人员关注[1]. 大量研究利用缺陷报告中的信息及源代码信息, 以堆栈跟踪分析、描述信息提取及源代码结构信息表示、程序依赖关系研究等方法解决跨语言的语义差异[4−8,24]. 同时, 因为该方法具有应用面广、时间成本低[25,26]等优点, 基于 IR 的缺陷定位研究逐渐成为缺陷定位领域中的主流研究方法. 为了解决语义匮乏问题, 一方面, 部分研究利用缺陷副产物扩充代码语义. Zhou 等人[27]提出了 BugLocator 方法, 使用修正的向量空间模型(rVSM)与已修复缺陷报告的相似度得分(SimiScore), 针对新的待修复缺陷报告的相关源程序文件进行排序. Ye 等人[4]通过挖掘代码片段中的自然语言信息弥补词汇空缺, 解决基于 IR 缺陷定位中词项不平衡问题. Lam 等人[5]使用深度学习技术, 提出了一种将深度神经网络(DNN)与 rVSM 相结合的方法. rVSM 收集缺陷报告和源文件之间文本相似性特征, DNN 用于学习缺陷报告中的术语, 并将它们与源文件及文档中出现频率较高的代码标记或术语联系起来, 利用多模型共同解决词汇语义差异问题. 另一方面, 部分研究深入挖掘细粒度代码词汇间的深层语义, 达到扩充代码信息量的目的. Xiao[28]利用翻译技术解决词汇语义稀少情况下的语义差异问题, 通过构造代码语言与自然语言间的 IR 模型, 实现缺陷定位的目的. 但该类研究将语义差异归结为简单的词汇差异, 受到方法自身的限制, 难以进行粒度再细化, 并且结果可用性受到质疑. 本文不仅关注代码词汇显性信息, 同时深入挖掘代码执行隐性信息, 在扩充代码信息量的基础上分析语义差异问题, 以语义监督的方式进行跨语言的语义映射. 本文不仅定位结果代码行数远远小于传统 IR 文件级缺陷定位方法, 同时, 显著提高了缺陷定位准确性.

1.2 修复建议

修复建议的研究是调试研究中关联缺陷定位及修复的研究内容[1]. 部分修复建议研究利用缺陷报告文本相似性对源文件进行排名, 即基于相似度计算对相关代码文件进行排序, 并给出合理的解释信息或辅助修复信息. 例如在驱动升级过程中, 根据代码变化信息推荐示例程序. Thung 等人[29]通过遍历内核代码提交, 挖掘历史提交数据中后向移植代码变化信息, 构造驱动后向移植信息推荐系统. Rodriguez 等人[30]通过兼容库屏蔽底层内核变化, 自动分析待移植驱动针对兼容库所需的适配性修改信息, 利用辅助信息手工构造适配兼容库补丁, 达到移植驱动的目的. Lawall 等人[31]发现: 驱动移植中产生的缺陷需要结合多个 commit 提取辅助修复信息, 结合编译和源代码信息构造查询条件, 利用 PQL(patch query language)查询内核 commit 获得驱动移植所需的辅助信息. 修复建议研究在改动代码实例推荐及改动信息推荐等领域有较大进展, 但建议形式及有效性验证方法还具有很大争议, 推荐信息可用性需要深入分析. 本文与修复建议研究不同, 以程序错误相关语义代码扩充定位候选点代码量, 有针对性地扩充代码信息量, 达到提高跨语言语义映射准确性的目的. 在细粒度定位缺陷的基础上, 利用具有控制依赖关系的代码段, 为后续的缺陷修复提供辅助信息.

1.3 深度学习技术用于代码特征分析

在基于 IR 的缺陷定位研究中, 深度学习技术是目前研究中较多使用的方法[9,10]. 深度学习技术用于缺陷研究领域的代码特征分析. 在缺陷预测方面, 利用数据聚类的思想对代码改动或特定代码段、组件等判断存在缺陷的概率及趋势[32,33]. 在缺陷检测方面, 2018 年首次利用深度学习方法进行了漏洞检测研究[34]: Li 等人提出, 他们将深度学习技术与缺陷检测研究相结合, 对于 CWE-119 及 CWE-399 漏洞实现了基于深度学习技术的自动检测. 近年来, 随着程序分析领域的兴起, 深度学习已经应用在 API 参数推荐[35]、代码补全[36,37]、程序自动生成[38−40]及误用检测[41]等领域. 深度学习方法在特征建模、特征学习等方面, 为缺陷分析领域中研究数据内在规律和外部表示形式提供了技术支持.

2 基于源代码扩展信息的细粒度缺陷定位方法

本节介绍基于源代码扩展信息的细粒度缺陷定位方法 FlowLocator 的研究动机及背景, 对研究框架进行描述, 对技术路线中的关键难点及解决方法进行介绍.

2.1 研究动机及背景

细粒度缺陷定位中存在的 3 个主要问题可以归结为, 在保证准确高效检索的同时增加结果有用性.分析发现: 在一般化的场景下, 定位粒度细化首先导致的是检索数量骤增带来的效率问题, 软件系统中存在的方法体较文件数量通常多 10 倍到数十倍, 语句行数较方法体数量也是如此, 代码粒度细化会加大代码处理工作量, 进而影响算法的效率. 目前的软件系统由于需求变更、应用环境改变及程序修复等原因, 导致项目不断迭代[42]. 在代码演化场景中, 缺陷的产生往往与软件项目的迭代相关[43,44]. 相关分析表明: 缺陷来源于版本间的差异代码, 代码改动影响分析(change impact analysis, CIA)[43,45−47]及人工修复规律均说明: 缺陷位置与程序演化过程间的差异代码有关, 两者的位置一致或具有极强的控制依赖关系. 同时, 与缺陷位置相关的代码改动依据功能需求具有语义相关性, 缺陷的引入及演化过程与 BR 的时间域紧密相关[19]. 因此, 以版本间的 diff code 作为定位原点, 根据 diff code 与 bug 间的关系缩小代码检索范围, 一定程度上减少了由于定位粒度减小而导致的候选集数量骤增问题.

同时, 在确定 diff code 为定位原点后, 独立的语句行很难做到语义可区分, 利用传统的特征分析方法构建一个具有良好性能的检索模型十分困难. 一方面, 通过分析发现, 缺陷点关联代码语义与缺陷描述语义相关, 因此本文以双向切片的方式获取缺陷定位原点关联的上下文, 通过增加代码量的方式扩展代码词汇语义显性信息. 通过这种代码扩充后的缺陷定位候选代码远小于传统 IR 缺陷定位到文件级粒度的代码行数. 另一方面, 通过深入挖掘代码结构信息扩展代码执行隐性信息, 分析发现: 代码执行中间形式的结构语义与缺陷描述语义间具有相关性, 关注代码元素间的关系可以达到扩充信息量的目的. 因此, 本文利用中间语言抽象语法树形式对代码进行解析, 在获取代码中词汇表达语义的同时, 提取 AST 节点间的影响关系作为模型输入的一部分, 通过改动门控循环神经网络(gate recurrent unit, GRU)节点间的连接方式及计算策略, 实现多节点间结构信息的关联, 以硬注意力的方式改动网络结构, 达到更准确地处理代码结构信息的目的, 提高细粒度缺陷定位的准确性.

最后, 通过分析发现, 程序修复过程中关注缺陷产生逻辑. 本文以基于语义的控制依赖关系构造定位形式, 通过扩充信息量的方式分析缺陷位置与缺陷现象间的关系, 定位结果涵盖了与缺陷相关的代码组件, 帮助程序员在不产生新缺陷的基础上修复缺陷. 同时, 对于多位置缺陷具有解释错误点间逻辑关系的作用, 为缺陷的完整修复提供了依据.

2.2 方法框架

FlowLocator 方法的框架如图 3 所示, 该方法包含 3 个阶段: 词汇语义显性信息扩充阶段、代码执行隐性信息扩充阶段及语义映射模型构建阶段.

- 词汇语义显性信息扩充: 本文依据代码语义表现方式扩充缺陷定位候选点代码量, 利用基于蒙特卡洛的双向切片方法实现定位数据的信息扩充, 利用 BR 语义指导后向切片(backward slicing)的路径选择. 在此基础上, 对于多位置改动情况, 对多位置点进行合并并进行数据去冗余, 结合多个 DBL 生成 DBL Forest.
- 代码执行隐性信息扩充: 为了实现细粒度代码结构语义的提取, 本文基于 AST 对代码执行中间形式进行解析, 通过遍历节点获取关联关系, 以硬注意力机制实现代码中各元素间关系的语义提取, 构造 Attention Tree-GRU, 通过改动 GRU 底层数据传输方式提取代码结构信息, 达到扩充代码语义的目的.
- 语义映射模型构建: 为了实现代码语言与缺陷报告中自然语言间的语义映射, 构造检索模型. 本文利用 PSN 框架分别处理缺陷报告的自然语言及源代码的代码语言, 利用 BERT 模型提取自然语言语义, 以缺陷报告自然语言语义监督代码语义表征生成, 从而减少语言差异带来的语义鸿沟问题, 在准确提取语义的同时实现跨语言的语义映射.

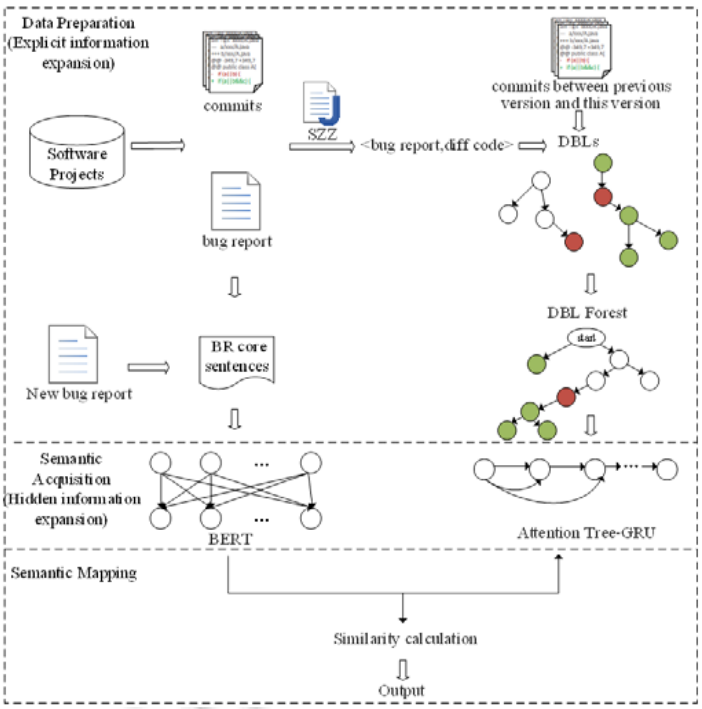

图 3 基于源代码扩展信息的细粒度缺陷定位方法框架

2.2.1 词汇语义显性信息扩充

1) git 版本回溯和多版本下的系统依赖图构建

静态切片算法的执行依赖于系统依赖图(system dependence graph, SDG)的构造, SDG 是描述系统中文件之间关联关系和函数调用关系的一种大型的树型语法组织结构[45]. 在数据集构建过程中, 需要获取不同 diff code 所在的源代码环境进行 git 版本回溯. 但目前, 真实项目中的 commit 次数很大, 例如, hadoop 项目已经有 23 410 次 commit, 迭代版本众多. 如果每个版本都要基于源代码生成对应的 SDG, 并进行预构建, 会占用极大的内存, 导致程序无法运行. 本文提出一种基于版本回溯的动态源代码获取方法, 构建多版本下的 SDG. 相较于“用时获取再构建”, 本文方法可以保持较高的构建速度.

我们跟踪多个版本之间的源代码差异, 以差异代码的时间戳为基点, 实现面对大数据集的 git 回滚, 在维护一个 SDG 的基础上, 使用版本之间的差异代码更新不同版本下的 SDG, 从而解决 DBL 获取过程中的多版本依赖图构建问题. 我们基于最新的代码提交版本构建 SDG. 当需要进行版本转换时, 通过快照技术获取文件的变化, 找到需要重构的代码, 从而快速更新 SDG. 具体来说, 我们建立倒排索引, 形成函数和文件的包含列表, 扫描程序系统, 通过 JavaParser[48]对 Java 程序进行解析, 得到相应的抽象语法树和上下文, 通过快照判断变化类型, 建立文件与函数之间的关系以及函数之间的关系, 并在新版本中快速更新 SDG.

本文在多版本 SDG 构造的基础上, 基于依赖跟踪和调用堆栈分析[49]进行静态双向切片. 但静态切片具有路径选择的问题, 模型输入数据直接影响缺陷定位的准确性.

2) 基于语义相似性的蒙特卡洛路径选择

本文以版本间的 diff code 作为定位原点, 采用双向切片的方式获取与定位原点关联的上下文. 而静态切片方法具有数据量大、冗余信息多的问题, 其中, 前向切片(forward slicing)由于代码运行具有方向性, 只有面对判断条件(if-else, while 等)或循环选择(switch 等)时, 面临路径选择问题. 但由于函数调用具有多路径多方式的特点, 静态分析中进行后向切片无法明确触发缺陷的执行路径. 因此, 后向切片具有路径选择问题, 在面对代码量较大的软件系统时, 难以判定缺陷来源路径, 路径节点间具有可选择性.

本文基于代码编写可理解性进行路径选择, 即采用节点语义与缺陷描述语义的相似性判断后向切片路径, 但在大型项目中路径难以全部枚举[49], 因此, 本文利用蒙特卡洛算法[50]实现静态分析下的路径选择, 以语义相似性实现具有控制依赖关系代码的获取, 解决静态切片中数据量过大的问题. 依据缺陷报告语义与以函数为粒度的 SDG 节点语义相似性计算, 选择后向切片路径, 通过基于语义相似性改进后的蒙特卡洛路径选择算法进行函数级别的路径判断, 利用语义关联及程序可理解性进行静态分析下的路径筛选.

由于全局路径通过 SDG 图已知, 我们将传统具有选择(selection)、扩张(expansion)、模拟(simulation)及回溯(back propagation)过程的蒙特卡洛算法变化为包括利用树的上限置信区间(upper confidence bounds for trees, UCT)选择、产生随机路径模拟及回溯变量更新过程, 逐层记录添加当前节点模拟路径后初始集合与 BR 文本语义相似性(通过 cosine similarity 计算), 同时, 利用公式(1)确定下一层头节点, 在初始集合的基础上添加新的被选择节点, 并循环该过程.

$$UCT=\frac{Q(v')}{N(v')}+c\sqrt{\frac{2\ln N(v)}{N(v')}} \tag{1}$$

其中, v 为父节点; v'为当前节点; $Q(v)$为 v 节点当前的相似度值, 即包含当前节点模拟路径的节点词汇集合与 BR 文本间的语义相似度; $N(v)$为当前节点的访问次数; c 为常量参数(根据经验值取 $\frac{1}{\sqrt{2}}$).

基于以上过程, 本文完成定位候选点的初步信息扩充, 对于一个缺陷报告对应程序错误点具有多个时, 采用 DBL Forest 算法, 通过虚拟节点关联 DBL, 达到去冗余及关联数据的目的.

3) DBL Forest 构建

DBL Forest 是依据调用关系及控制依赖关系对多个 DBL 的合并, 一方面, 对于具有多个错误点的缺陷进行数据关联; 另一方面, 通过去环操作减少后续模型输入中重复性的冗余数据. 本文在面对缺陷具有多个改动位置的情况时, 在蒙特卡洛路径选择的基础上, 对多个 DBL 进行基于 AST 的解析, 以虚拟节点作为头节点, 通过函数哈希值标定 DBL 在 SDG 中路径相同的位置, 获得 DBL Forest; 同时, 对新获取的 AST 依据依赖关系去环, 避免缺陷对应多个改动位置时产生重复的冗余数据, 保证在构造模型输入时, 自底向上进行有效的关系遍历(如图 4 所示).

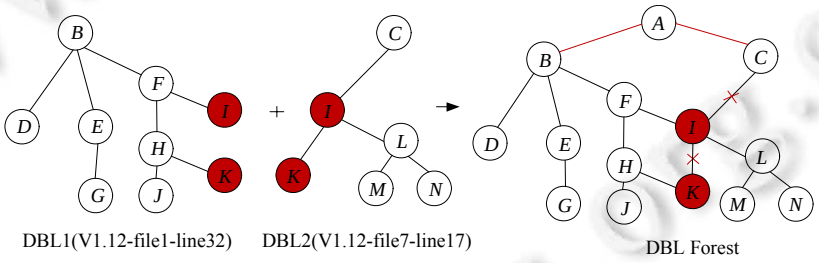

图 4 DBL Forest 构建流程图

2.2.2 代码执行隐性信息扩充

代码元素的组成结构与自然语言大不相同, 这些元素不是简单地以线性关系组合, 而是具有控制依赖或数据依赖关系. 传统的机器学习模型很难对树状结构的代码段进行有效的分析[51]. Tai 等人[51]指出: 在自然语言处理中获得较好效果的 GRU 模型[52]是一种具有复杂计算单元的递归神经网络, 它具有随时间变化保存序列信息的优越能力. 但是目前使用的基于 GRU 模型的底层结构是一个线性链(linear chain), 对于复杂的结构化语言处理缺乏一定的应用性[51]. 本文在 GRU 的基础上结合硬注意力算法[53], 通过改动数据获取方式及 GRU 节点间的数据传输关系形成 Attention Tree-GRU, 实现多代码行间元素的关联关系获取, 利用结构语义提高代码语义表征生成的准确性.

首先, 将 DBL(或 DBL Forest)利用 AST 中间语言形式进行解析, 由于在构造 DBL(或 DBL Forest)过程中基于多版本 SDG 对节点进行了 hash 赋值, 因此本文通过图遍历算法实现 AST 图的再编码; 同时, 在该过程中,

顺序获取各节点的父亲节点, 以⟨编码,父亲节点 hash⟩进行存储(虚拟节点编码总为−1). 全部编码完成后, 所有节点的hash与编码相对应, 将二元组中的hash值替换为编码值, 将该结果作为模型硬注意力机制的关系输入. 另外, 在 DBL(或 DBL Forest)构造过程中获取每个节点的词项, 即每个节点含有的自然语言词汇, 包括操作符、变量名、函数名、变量类型等, 作为模型的实际输入.

我们将代码解析成 AST 形式, 在数据处理过程中自下而上得到的子节点和父节点间的关系代表了代码的外部结构. 结合硬注意力机制, 改变GRU节点之间的数据关联, 形成Attention Tree-GRU模型. 该模型解决了GRU模型受严格的序列信息传播限制的问题, 允许GRU有选择地从多个单元获取信息, 更新记忆单元, 有效地应用于代码结构语义的提取.

在 DBL(或 DBL Forest)解析的一个 AST 中, 硬注意的应用使神经网络具有集中处理输入(或特征)子集的能力, 即在数据处理中选择特定的输入. 当处理第 j 个节点时, 使用位置变量 S_{jn} 表示相关子节点的位置, 如果 t 位置是用于提取特征的位置, 则设置为 1. 因此, 对于任意一个节点 j, 根据 S_{jn}, 确定从 1 到 N 的相关子节点, 将其中第 l 个子节点的隐藏状态(hidden state)写成 h_{jl}. Attention Tree-GRU 转移方程如下.

$$z_j=\sigma\left(W^{(z)}x_j+\sum_{l=1}^{N}U_l^{(z)}h_{jl}+b^{(z)}\right) \tag{2}$$

$$r_j=\sigma\left(W^{(r)}x_j+\sum_{l=1}^{N}U_l^{(r)}h_{jl}+b^{(r)}\right) \tag{3}$$

$$\tilde{h}_j=\tanh\left(r_j\odot\sum_{l=1}^{N}U_l^{(\tilde{h})}h_{jl}+W^{(\tilde{h})}x_j+b^{(\tilde{h})}\right) \tag{4}$$

$$h_j=(1-z_j)\odot\tilde{h}_j+\sum_{l=1}^{N}z_j\odot h_{jl} \tag{5}$$

其中, h_j 为隐藏层状态, x_j 为该节点的输入, z_j, r_j, $\tilde{h}_j$ 分别为更新门(update gate)、复位门(forget gate)和新记忆单元(new memory cell), W, U 为隐藏层中对于上一层的隐含状态和输入权重, b 为偏置, σ为 sigmod 函数.

通过在 GRU 模型中引入硬注意力机制, 本文以 AST 节点间关系改变 GRU 的网络关联策略, 根据关联关系输入获取对应节点的数据. 父节点多为数据间的操作信息(例如操作符等), 子节点包涵被操作元素, 父节点的处理需要子节点的数据支持, 因此, 基于父节点与子节点间的关系, 通过改动 GRU 数据间的传输方向及连接结构, 本文实现了代码结构的表征学习, 为后续语义映射提供更加准确的数据支持.

2.2.3 语义映射模型构建

基于跨语言的语义差异, 本文分别利用 Attention Tree-GRU 获取代码语义, 利用 BERT[54]获取 BR 自然语言语义, 并基于 PSN 框架利用缺陷描述语义指导代码语义生成, 方法框架如图 5 所示, 利用语义监督的方式解决跨语言的语义鸿沟问题, 在 IR 过程中提高细粒度缺陷定位的准确性.

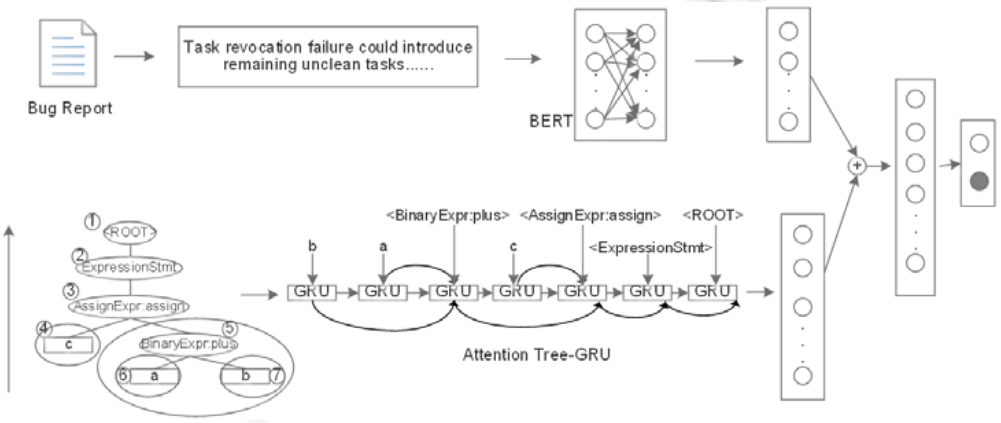

图 5 基于 PSN 框架的检索模型

本文利用PSN框架, 以师生网络(teacher-student network)的形式解决跨语言的语义鸿沟问题, 代码语义的生成以缺陷报告中自然语言语义进行指导, 减少两者间的语义差异. 在应用 PSN 的过程中, 利用全连接层(fully connected layer)对生成的自然语言与代码语言语义表征进行结构一致化, 将两者的输出维度进行同化, 为后续 loss 的计算准备输入, 实现跨模式(cross-modal)的监督学习.

1) Loss 函数设计

学生网络(Attention Tree-GRU)$S(\cdot)$的训练目标是: 最小化其与教师网络(BERT)$T(\cdot)$在输入为 C_j 及 I_j 下, embedding 表征输出 $S(C_j)$与 $T(I_j)$的差异, 即

$$\min L(T(I_j),S(C_j)) \tag{6}$$

因此,我们将损失函数定为二元交叉熵损失, 如公式(7)所示.

$$L(T(I_j),S(C_j))=-\frac{1}{N}\sum_{j=1}^{N}[S^u(C_j)\log T^u(I_j)+(1-S^u(C_j))\log(1-T^u(I_j))] \tag{7}$$

其中, N 为样本数量, $S^u(C_j)$及 $T^u(I_j)$表示维度同化后的表征输出.

为了避免过拟合问题, 使用 Dropout[55]随机丢弃 GRU 单元部分输入, 通过最小化损失函数优化训练模型.

2) 相似度计算策略

在测试阶段, 本文将 IR 过程转化为每个候选代码的二元分类问题, 并以可疑度进行排序. 在给定这些表征输出的情况下, 基于 softmax 分类器预测目标类, 该分类器考虑$(S^u(C_j),T^u(I_j))$对之间的距离(distance)及角度(angle). 策略如公式(8)–(12)所示.

$$h_x=S^u(C_j)\odot T^u(I_j) \tag{8}$$

$$h_l=|S^u(C_j)-T^u(I_j)| \tag{9}$$

$$h_s=\sigma(W^{(x)}h_x+W^{(l)}h_l+b^{(h)}) \tag{10}$$

$$\hat{p}_\theta(y\mid\{x\}_j)=softmax(W^{(p)}h_s+b^{(p)}) \tag{11}$$

$$\hat{y}_j=\arg\max_y \hat{p}_\theta(y\mid\{x\}_j) \tag{12}$$

其中, h_x, h_l为组合向量, h_s为激活后的隐藏层, W为权重矩阵, b为偏置, x, y为数据点对应标签, $\hat{y}_j$ 为预测标签.

3 实验设计

本节提出验证问题, 说明实验数据集的来源及构建方法; 同时, 根据目标期望设计实验验证策略, 分析实验结果, 并对方法优势及限制进行分析.

3.1 验证问题

本文依据预期目标设计以下实验期望, 即评估如下 4 个问题.

- RQ1: 与经典的基于信息检索的缺陷定位方法相比, FlowLocator 方法是否提高定位准确性?

该问题评估在相同数据集下, FlowLocator 与 Lam 等人[5]提出的经典的基于 IR 缺陷定位方法的结果准确性差异. 本文采用 IR 缺陷定位领域经典的准确性评价指标[4−8,56], 即 Top-N 排名(Top-N rank)、平均准确率(mean average precision, MAP)和平均倒数排名(mean reciprocal rank, MRR)对方法准确性进行评估.

- RQ2: FlowLocator 方法中, 扩充代码词汇显性信息对于定位结果准确性的影响如何?

FlowLocator 采用基于语义的双向切片方法扩充代码词汇显性信息, 相较于传统基于 IR 缺陷定位研究中按照代码序扩充显性信息方式[5,28,57], 是否具有更高的定位准确性. 其中, 准确性评价指标与 RQ1 相同.

- RQ3: FlowLocator 方法中, 扩充代码执行隐性信息对定位结果准确性的影响如何?

FlowLocator 方法通过注意力机制构建模型输入间的关联, 基于 Attention Tree-GRU 模型, 采用代码执行中间形式的结构信息扩充代码语义信息量, 相较于不处理词项间隐性关系单独应用 GRU 模型[52]及只处理代码标识符信息, 是否提高定位准确性. 其中, 准确性评价指标与 RQ1 相同.

- RQ4: FlowLocator 方法的语义映射策略对于定位结果准确性影响如何?

FlowLocator 采用基于 PSN 框架的语义映射策略实现定位, 相较于经典基于 IR 的缺陷定位方法[5,28,57]采用孪生网络(siamese neural network)等语义映射策略, FlowLocator 定位结果是否具有更高的准确性. 其中, 准确性评价指标与 RQ1 相同.

3.2 数据集构建

本文在真实项目演化场景中进行实验, 为了验证 FlowLocator 方法在实际软件缺陷定位中的有效性, 以包括 Hbase, Flink, Mahout 等 7 个开源项目组成的数据集下, 通过爬取项目中的缺陷报告和源代码变更(gitlog)等信息开展实验. 数据集的构建方法如下所述.

(1) 获取源代码文件: 从开源项目代码库中获取实验所需的源代码文件, 本文利用 SVN 工具及 git 工具获取源代码数据.

(2) 爬取缺陷报告内容: 本文分别从缺陷跟踪系统 JIRA 上获取 7 个软件项目对应的缺陷报告. 首先, 通过缺陷状态(status)、类型(type)及重要程度(priority)筛选系统中的缺陷报告, 获取已解决且主要的软件错误报告, 在此基础上获取缺陷报告的总结(summary)、描述(description)和标题(title)这 3 个字段的内容, 作为缺陷的自然语言描述内容作后续处理. 同时, 为了保证实验的可重复性和可验证性, 本文通过 SZZ 算法[58]从以上缺陷报告中选出缺陷变更记录可追踪性良好的缺陷报告作为实验数据, 即通过缺陷编号(bug number)能够准确地对应步骤(1)中的源代码修改记录的缺陷报告. 在此基础上, 将存在字段空白的缺陷报告筛除, 对于过长的缺陷报告, 本文通过 rVSM 获取 top-3 的关键核心句.

(3) 建立缺陷相关代码信息数据集: 首先, 本文爬取步骤(1)所收集的源代码文件中的 log 日志, 从 log 日志中获取 bug number 对应的 commit 版本号. 之后, 从 log 日志中筛选出该 bug number 前所有版本(按修改时间由近到远排序), 利用 SZZ 算法[58]获得对应 diff code(即缺陷位置), 进行人工验证后, 将其作为后续生成 DBL 的初始点. 同时, 根据爬取到的 BR 获取修复信息, 包括修复 commit 版本、diff code 位置及改动信息, 作为修复具有该 bug number 缺陷所导致的代码变化. 之后, 利用第 2.2.1 节所述方法生成 DBL Forest(或 DBL). 将 bug number、DBL Forest(或 DBL)、修改的代码行建立三元组, 组建数据集. 在候选集中, 选取缺陷报告提交时间戳前一定时间域的改动代码作为候选(本文选择前 200 个更改). 表 1 展示了本文收集到的包含 7 个开源项目的数据集信息.

表 1 数据集信息

软件系统	文件数量	方法体数量	代码语句行数量	可执行文件平均代码行数	DBL (or DBL Forest)平均代码行数	DBL (or DBL Forest)平均词项数量	缺陷报告数量	缺陷报告时间
Flink	13 522	71 865	1 478 240	228.011	12.601	223.544	5 147	2010/12–2020/02
Hbase	5 071	59 771	1 093 178	264.472	11.273	238.053	9 971	2007/04–2020/02
Jena	13 973	58 540	939 413	134.104	8.573	121.344	885	2012/05–2020/02
Kafka	3 481	24 473	404 339	247.234	12.934	218.024	3 871	2011/08–2020/02
Lucene-solr	12 507	79 058	1 883 020	221.763	12.852	343.651	3 642	2001/09–2020/02
Mahout	2 080	8 392	180 412	245.971	17.322	432.421	775	2008/01–2020/02
Pig	2 456	17 105	400 735	224.881	12.713	230.984	2 577	2009/03–2019/07

3.3 比较对象及评价指标

3.3.1 比较对象

Lam[5]采用 rVSM 及 DNN 模型, 基于词汇间的语义相似性, 利用多模型, 以文件为单位定位缺陷. Xiao[28]基于目前取得突破进展的机器翻译中的编码-解码架构(encoder-decoder), 利用 LSTM 模型解决语义差异问题, 实现缺陷定位. Desh[57]利用 Bi-LSTM 模型处理缺陷数据, 以孪生网络的形式实现语义映射, 用于缺陷查重及定位等工作. 本文与以上方法进行比较, 记为 Lam[5], Xiao[28]和 Desh[57].

3.3.2 评价指标

本文利用 Top-*N* 排名、平均准确率和平均倒数排名这 3 个 IR 缺陷定位领域的经典评价指标进行实验结

果比较, 以此来评价 FlowLocator 方法的优劣.

1) Top-N 排名

表示缺陷报告对应做出变更的代码出现在返回结果前 N(N 为自然数)位数量的比率. 若前 N 个查询结果包含至少一个目标代码, 即认为缺陷被准确定位. 前 N 排名度量值越大, 说明缺陷定位方法的定位性能越好.

2) 平均准确率值(MAP)

表示对所有缺陷报告进行源代码定位后准确率的平均值. MAP 值反映了缺陷定位方法在全部缺陷上准确定位源代码的精度, 其中, 对于单个缺陷定位的平均精度(表示为 $AvgP$)如公式(13)所示.

$$AvgP=\frac{1}{|R|}\sum_{k=1}^{|R|}\frac{k}{rank_k} \tag{13}$$

其中, R 表示一次缺陷定位下排名中正确定位代码的集合, $|R|$表示正确定位的源代码个数, $rank_k$ 表示第 k 个正确定位的源代码位置的排名. 对于所有缺陷报告的 MAP 如公式(14)所示.

$$MAP=\sum_{j=1}^{|Q|}\frac{AvgP_j}{|Q|} \tag{14}$$

其中, Q 为缺陷报告的集合, $|Q|$表示 Q 中缺陷报告的数目, $AvgP_j$ 表示第 j 个缺陷报告的定位平均精度值.

3) 平均倒数排名(MRR)

表示正确定位的源代码位置排序倒数的平均值, MRR 数值与方法的准确性正相关. MRR 计算如公式(15)所示.

$$MRR=\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{15}$$

其中, Q 为缺陷报告的集合, $|Q|$表示 Q 中缺陷报告的数目, $rank_i$ 表示定第 i 个缺陷报告与其对应的源代码定位位置最高排名.

3.3.3 参数设定

本文实验将数据集中每个 Apache 项目的 10%数据作为测试集, 学习率(learning rate)设定为 0.01, 轮数(epoch)设定为 10, 迭代(iteration)设定为 100, dropout 比率设定为 0.1.

本文实验在以下配置的台式机上运行: 操作系统 Ubuntu 16.04; GPU: GeForce GTX 1080 Ti cuda9.0.

4 实验结果及分析

4.1 RQ1的实验结果及分析

本节验证在同一数据集下, FlowLocator 方法与第 3.3 节中提到的经典的基于 IR 缺陷定位方法[5,28,57]相比, 在信息获取与语义映射策略差异下是否提高定位结果准确性. 本文在包含 Hbase 等软件项目的数据集上, 通过实验给出了包含 MAP 等准确性评价指标(如第 3.3 节所述)的实验结果, 如表 2 和图 6 所示.

与 Lam[5]相比, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.110−0.361 (相对提高 182.4%−554.8%); 在 MRR 上, FlowLocator 相对提高 188.6%; 在 MAP 上, FlowLocator 相对提高 285.0%. 由此可见, FlowLocator 方法的准确性高于 Lam[5]的方法. 其中, Lam[5]方法处理代码标识符及注释信息, 在计算跨语言语义相似性的同时, 引入缺陷报告间语义相似性及代码改动频度分数. 方法基于 rVSM 模型计算缺陷描述及代码词项间的文本相似性, rVSM 属于策略较为简单的信息检索模型, 把对输入数据的处理简化为向量空间中的向量运算, 并且以空间上的相似度表示语义的相似度, 忽视上下文语境对定位准确性的影响, 定位准确率低; 而 FlowLocator 方法在细粒度代码中不仅获取标识符信息, 同时获取基于 AST 解析的隐性信息, 将自然语言与代码语言分别差异化处理, 采用具有保存序列信息能力的 GRU 模型及注意力机制, 关注上下文对节点数据的影响, 利用语义指导的形式提高缺陷定位准确性, 因此具有更好的定位效果.

表 2 验证问题 1 的 MAP 及 MRR 实验结果

Model	1	2	3	4	5	10	15	20	MRR	MAP
FlowLocator	0.141	0.172	0.203	0.234	0.234	0.438	0.453	0.547	0.202	0.154
Lam[5]	0.031	0.031	0.031	0.047	0.047	0.094	0.109	0.188	0.070	0.040
Xiao[28]	0.016	0.031	0.063	0.063	0.063	0.125	0.156	0.203	0.053	0.054
Desh[57]	0.063	0.109	0.125	0.203	0.219	0.250	0.391	0.423	0.124	0.110

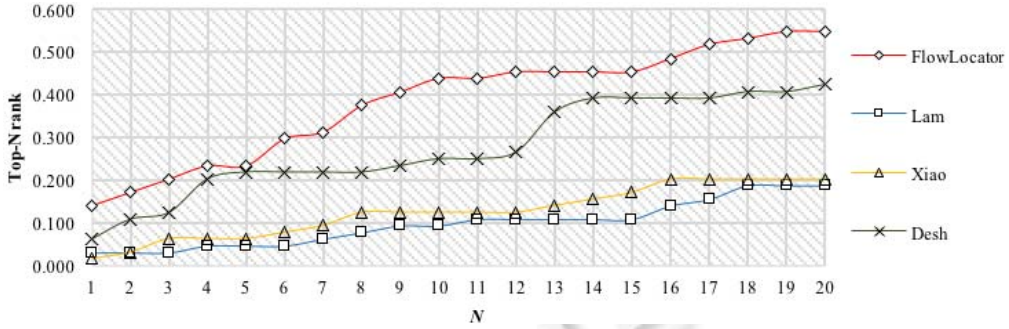

图 6 验证问题 1 的前 N 排名实验结果

与 Xiao[28]相比, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.125−0.344(相对提高 138.4%−781.3%); 在 MRR 上, FlowLocator 相对提高 281.1%; 在 MAP 上, FlowLocator 相对提高 185.2%. 由此可见, FlowLocator 方法的准确性高于 Xiao[28]的方法. 其中, Xiao[28]方法处理缺陷报告中的关键词, 利用历史缺陷报告及 API 文档构建词汇表, 基于 LSTM 模型及编码-解码框架定位缺陷. LSTM 适用于处理较长输入数据的情况, 对于长期依赖有较好的处理效果. 而 GRU 模型是 LSTM 的一种变体, 逻辑计算更加简便, 数据量较少时, 模型收敛得更为迅速, 更适合于数据量小的数据集. 由于本数据集构建过程中数据进行了逐层筛选, 数据量相对少, FlowLocator 的定位准确性高于基于 LSTM 的方法. 并且与 FlowLocator 相比, Xiao[28]依赖于数据预处理过程的全面性, 同时, 方法忽略了代码结构信息, 对于输入数据元素间的关系构造缺乏指导信息, 因此定位准确性低于 FlowLocator.

与 Desh[57]相比, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.015−0.188(相对提高 6.8%−123.8%); 在 MRR 上, FlowLocator 相对提高 62.9%; 在 MAP 上, FlowLocator 相对提高 40.0%. 由此可见, FlowLocator 方法的准确性高于 Desh[57]的方法. 其中, Desh[57]的方法以缺陷报告关键词及代码 AST 信息作为输入, 基于 Bi-LSTM 模型处理自然语言及代码语言. Bi-LSTM 由前向 LSTM 与后向 LSTM 叠加在一起组成, 输出由两个 LSTM 的隐藏层的状态共同决定. 与 Lam[5]和 Xiao[28]相比, Desh[57]考虑了代码附加信息对于定位结果的影响, 因此定位准确率有所提高. 但与 FlowLocator 相比, Desh[57]虽然获取了代码 AST 信息, 但对于节点元素间的关系处理不具有初始指向性, 以孪生网络采用一致化的策略处理代码语言及自然语言, 忽视了两者的语义表达方式差异, 定位准确性低于 FlowLocator.

综上所述, FlowLocator 方法相较于目前 IR 缺陷定位研究中效果较好或较为公认的方法, 在以上 3 个指标评估下准确性更高. 同时, 以上实验数据表明: 相较于传统 IR 缺陷定位的文件级粒度(本文实验数据集中平均可执行文件代码行数为 223.777 行), 本文实现了在具有平均 12.610 行的代码段上进行细粒度缺陷定位工作, 并且与目前应用较多的 IR 缺陷定位方法相比取得了更好的效果.

4.2 RQ2的实验结果及分析

为了评估本文扩充代码词汇显性信息方法对于定位结果准确性的影响, 本节采用基于语义双向切片的 FlowLocator 方法与基于传统代码序号扩充的 FlowLocator 方法[5,28,57]进行定位结果准确性评估. 本文在包含 Hbase 等软件项目的数据集上进行实验验证, 采用差异化的显性信息扩充方式, 在保证代码行数一致的基础上进行实验, 给出了包含 MAP 等准确性评价指标(如第 3.3 节所述)的实验结果, 见表 3 和表 4.

表 3 验证问题 2 的前 N 排名实验结果

Model	1	2	3	4	5	6	7	8	9	10
FlowLocator	0.192	0.269	0.317	0.377	0.401	0.441	0.479	0.501	0.510	0.522
FlowLocator-代码序	0.014	0.027	0.057	0.059	0.064	0.068	0.074	0.090	0.119	0.119
Model	11	12	13	14	15	16	17	18	19	20
FlowLocator	0.534	0.540	0.549	0.560	0.581	0.587	0.590	0.601	0.610	0.621
FlowLocator-代码序	0.120	0.122	0.134	0.141	0.144	0.153	0.158	0.170	0.189	0.199

表 4 验证问题 2 的 MAP 及 MRR 实验结果

Model	MRR	MAP
FlowLocator	0.314	0.211
FlowLocator-代码序	0.051	0.049

从实验结果来看, FlowLocator 采用基于语义的双向切片方式扩充代码显性信息, 在相同的代码行数下(20 行), 相比于传统利用代码序号的扩充方式, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.178−0.437 (相对提高 212.1%−1271.4%); 在 MRR 上, FlowLocator 相对提高 515.7%; 在 MAP 上, FlowLocator 相对提高 330.6%. FlowLocator 通过 AST 解析的形式获取模型输入, 在代码序号的扩充方式下部分代码行参数与定位起始点无关, 导致信息量减小. 而在基于语义的双向切片扩充方式下, 通过硬注意力机制, 基于 AST 解析结果获取词项间的节点关系, FlowLocator 重构了模型中节点间的关联方式, 根据定位起始点获取相关定位信息, 有针对性地扩充信息量, 相较于传统按代码序的扩充方式, 在定位准确性上具有更好的效果.

4.3 RQ3的实验结果及分析

为了评估扩充代码执行隐性信息对定位准确性的影响, 本节分别采用单独应用 GRU 模型[52]及只获取代码标识符信息的方式与 FlowLocator 进行对比实验. 本文在同一显性信息扩充方式下, 以包含 Hbase 等软件项目的数据集上进行验证, 并给出了包含 MAP 等准确性评价指标(如第 3.3 节所述)的实验结果, 见表 5 和表 6.

表 5 验证问题 3 的前 N 排名实验结果

Model	1	2	3	4	5	6	7	8	9	10
FlowLocator	0.141	0.172	0.203	0.234	0.234	0.297	0.313	0.375	0.406	0.438
FlowLocator-GRU	0.070	0.125	0.125	0.172	0.172	0.172	0.219	0.219	0.219	0.234
FlowLocator-标识符	0.047	0.047	0.053	0.091	0.091	0.104	0.112	0.136	0.155	0.160
Model	11	12	13	14	15	16	17	18	19	20
FlowLocator	0.438	0.453	0.453	0.453	0.453	0.484	0.517	0.531	0.547	0.547
FlowLocator-GRU	0.250	0.250	0.250	0.250	0.297	0.328	0.359	0.359	0.359	0.359
FlowLocator-标识符	0.175	0.175	0.183	0.183	0.192	0.204	0.223	0.230	0.237	0.237

表 6 验证问题 3 的 MAP 及 MRR 实验结果

Model	MRR	MAP
FlowLocator	0.202	0.154
FlowLocator-GRU	0.112	0.090
FlowLocator-标识符	0.080	0.067

从实验结果来看, FlowLocator 采用 Attention Tree-GRU 模型, 相比于单独使用 GRU 模型及仅处理代码标识符的 FlowLocator 方法, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.047−0.310 (相对提高 36.0%−283.0%); 在 MRR 上, FlowLocator 相对提高 80.4%及 152.5%; 在 MAP 上, FlowLocator 相对提高 71.1%及 129.9%. FlowLocator 相较于单独应用 GRU 处理 AST 信息, 利用硬注意力机制改动模型中各节点数据来源, 子节点数据流向父节点, 并在父节点中进行处理. FlowLocator 相较于只获取代码标识符信息, 在模型输入前, 基于 AST 解析获取代码执行中间形式数据, 增加代码语义信息量. 因此, FlowLocator 基于注意力机制处理代码执行中间形式结构信息, 相较于单独使用 GRU 模型或只处理代码标识符信息, 在定位准确性上具有更好的效果.

在代码量较小的情况下, 相较于以传统方式使用代码中的词汇语义显性信息, 获取代码执行中间形式结

构语义可以提高细粒度定位准确性. 因此, 在具体场景中深入挖掘代码深层次语义, 有助于提高基于 IR 缺陷定位方法在细粒度代码上应用的准确性.

4.4 RQ4的实验结果及分析

为了验证语义映射策略对定位准确性的影响, 本节分别采用经典的基于 IR 缺陷定位方法[5,28,57]中的语义映射策略, 在保证语义映射模型输入数据一致的情况下, 与 FlowLocator 方法进行对比实验. 本文在包含 Hbase 等软件项目的数据集上进行了实验验证, 并给出了包含 MAP 等准确性评价指标(如第 3.3 节所述)的实验结果, 见表 7 和表 8.

表 7 验证问题 4 的前 N 排名实验结果

Model	1	2	3	4	5	6	7	8	9	10
FlowLocator	0.141	0.172	0.203	0.234	0.234	0.297	0.313	0.375	0.406	0.438
rVSM	0.055	0.080	0.113	0.117	0.117	0.154	0.170	0.186	0.197	0.205
LSTM+Encoder-Decoder	0.039	0.043	0.049	0.053	0.053	0.069	0.071	0.087	0.093	0.105
Bi-LSTM+SNN	0.063	0.109	0.125	0.203	0.219	0.219	0.219	0.219	0.234	0.250
Model	11	12	13	14	15	16	17	18	19	20
FlowLocator	0.438	0.453	0.453	0.453	0.453	0.484	0.517	0.531	0.547	0.547
rVSM	0.205	0.224	0.224	0.224	0.224	0.261	0.269	0.270	0.274	0.279
LSTM+Encoder-Decoder	0.105	0.114	0.114	0.114	0.114	0.126	0.137	0.154	0.194	0.194
Bi-LSTM+SNN	0.250	0.266	0.359	0.391	0.391	0.391	0.391	0.406	0.406	0.423

表 8 验证问题 4 的 MAP 及 MRR 实验结果

Model	MRR	MAP
FlowLocator	0.202	0.154
rVSM	0.090	0.081
LSTM+Encoder-Decoder	0.074	0.063
Bi-LSTM+SNN	0.124	0.110

从实验结果来看, FlowLocator 采用基于 PSN 的语义映射策略, 相比于 rVSM 等映射策略, 在 Top-N rank (n 为 1,2,…,20)上, FlowLocator 提高定位准确度 0.015–0.380 (相对提高 6.8%–341.5%); 在 MRR 上, FlowLocator 相对提高 62.9%–173.0%; 在 MAP 上, FlowLocator 相对提高 40.0%–144.4%. FlowLocator 相较于 rVSM 映射策略, 差异化提取自然语言及代码语言语义, 并以自然语言语义指导代码语言表征生成. 与基于 LSTM 的 Encoder-Decoder 方法相比, FlowLocator 不需要进行缺陷数据预对应, 语义映射准确性不受预处理数据的影响. 相较于基于 Bi-LSTM 的 SNN 方法, FlowLocator 采用伪孪生网络, 将自然语言表征作为指导信息, 监督代码语言表征生成, 以差异化的表征生成技术处理自然语言及代码语言, 相较于采用一致化语义处理方法, 更有效地保留了语义信息, 提高了跨语言语义映射的准确性. 因此, FlowLocator 方法相较于 rVSM 等语义映射策略, 在定位准确性上具有更好的效果.

4.5 有效性影响因素分析

目前, FlowLocator 方法仍然具有一定限制.

- 方法有效性的外部威胁主要来自 commit 自身的复杂程度. 在版本迭代过程间, 存在版本间 diff code 数量较大的情况. 首先, 对 FlowLocator 的算法效率产生威胁, 密集改动使得 DBL Forest 构建需要考虑多个位置点的概率增加, 可能导致数据准备过程时间较长. 同时, 随着改动位置数量的增加,DBL (或 DBL Forest)间具有重复数据的可能性提高, 进而导致候选代码间显性数据的语义相似性增大, 对 FlowLocator 定位准确性产生的威胁随之增大.
- 方法有效性的内部威胁主要来自方法性能的限制. FlowLocator 方法基于 SZZ 算法构建数据集, SZZ 算法受到项目复杂度的影响, 且对于设备内存要求较高, 对于迭代时间较长的大型项目, 算法效率会降低.

对于以上问题, 未来可能采取的有效措施如下.

- 首先, 针对外部威胁. 对于候选量增大产生的效率威胁, commit 需进行预处理, 对于不规范提交及冗

余改动数据进行预筛除, 进而减少无关 commit 数量, 为后续 FlowLocator 的数据准备过程减少可疑改动位置数据, 缩小定位候选代码范围. 同时, 对于 DBL(或 DBL Forest)间显性数据获取语义相似性增大的威胁, 一方面, 未来可以在本文的基础上进一步修正基于语义相似性的切片算法, 或采用动静态分析结合的方式减少显性数据的重复性; 另一方面, 增大对隐性数据的挖掘, 利用可能的缺陷位置相关数据提高候选代码的可区分性, 进而提高细粒度代码间的语义差异性, 最终提高定位的准确性.

- 针对内部威胁, 对 SZZ 算法进行性能优化, 可以有效地提高 FlowLocator 的算法效率, 同时减少内存溢出的风险.

5 总结与展望

本文基于源代码扩展信息进行了细粒度的缺陷定位研究, 提出了一种演化场景下基于信息检索的细粒度缺陷定位方法, 并在定位粒度细化的基础上提高结果的准确性. 通过获取定位候选点关联代码, 以静态分析方法模拟测试用例动态执行路径, 达到丰富代码词汇语义显性信息的目的; 同时, 以代码执行中间形式结构语义扩展代码执行隐性信息, 实现细粒度代码间的语义可区分, 以差异化的形式处理缺陷报告中自然语言及源代码中代码语言语义, 以语义指导的方式构造自然语言及代码语言间的语义映射关系, 实现细粒度的缺陷定位.

本文通过源代码扩展信息实现了细粒度的缺陷定位, 脱离了完美缺陷理解假设下的缺陷分析方式, 增强了调试各过程间的联系, 实现了缺陷研究各过程间的信息互用. 未来, 在本工作的基础上, 从缺陷理解的角度对于缺陷调试各过程中结果的可用性及各过程间的关联性进行更深层次的研究.

References:

[1] Li XZ, He YP, Ma HT. Defect comprehension research: Present, problem and prospect. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 20−46 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5887.htm [doi: 10.13328/j.cnki.jos.005887]

[2] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011, 34(8): 1411−1422 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01411]

[3] Nguyen AT, Nguyen TT, SI-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. 2011. 263−272. [doi: 10.1109/ASE.2011.6100062]

[4] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE 2014). New York: ACM, 2014. 689−699.

[5] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports (N). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln, 2015. 476−481.

[6] Xiao Y, Keung J, Mi Q, Bennin KE. Improving bug localization with an enhanced convolutional neural network. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. (APSEC 2017). Nanjing, 2017. 338−347.

[7] Xiao Y, Keung J, Mi Q, Kwabena B. Bug localization with semantic and structural features using convolutional neural network and cascade Forest. In: Proc. of the 22nd Int'l Conf. on Evaluation and Assessment in Software Engineering. 2018. 101−111.

[8] 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 IEEE/ACM 38th Int'l Conf. on Software Engineering (ICSE 2016). Austin, 2016. 404−415. [doi: 10.1145/2884781.2884862]

[9] Guo ZQ, Zhou HC, Liu SR, Li YH, Chen L, Zhou YM, Xu BW. Information retrieval based bug localization: research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2020, 31(9): 2826−2854 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[10] Zhang Y, Liu JK, Xia X, Wu MH, Yan H. Research progress on software bug localization technology based on information retrieval. Ruan Jian Xue Bao/Journal of Software, 2020, 31(8): 2432−2452 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[11] 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, 2017, 22(4): 2237−2279.

[12] Pearson S, Campos J, Just R, *et al*. Evaluating and improving fault localization. In: Proc. of the IEEE/ACM 39th Int'l Conf. on Software Engineering (ICSE). IEEE Computer Society, 2017. 609−620. [doi: 10.1109/ICSE.2017.62]

[13] Le TDB, Lo D, Thung F. Should I follow this fault localization tool's output? Empirical Software Engineering, 2015, 20(5): 1237−1274.

[14] Lucia, Thung F, Lo D, *et al*. Are faults localizable? In: Proc. of the Mining Software Repositories. IEEE, 2012. 74−77. [doi: 10.1109/MSR.2012.6224302]

[15] Parnin C, Orso A. Are automated debugging techniques actually helping programmers? In: Proc. of the Int'l Symp. on Software Testing & Analysis. ACM, 2011. 199−209. [doi: 10.1145/2001420.2001445]

[16] Motwani M, Sankaranarayanan S, Just R, *et al*. Do automated program repair techniques repair hard and important bugs? Empirical Software Engineering. 2018, 23(4): 2901−2947.

[17] Wei HH, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2017). 2017. 3034−3040. [doi: 10.24963/ijcai.2017/423]

[18] Duan X, Wu JZ, Ji SL, *et al*. VulSniper: Focus your attention to shoot fine-grained vulnerabilities. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence Main Track. 2019. 4665−4671.

[19] Alsulami B, Dauber E, Harang RE, Mancoridis S, Greenstadt R. Source code authorship attribution using long short-term memory based networks. In: Proc. of the 22nd European Symp. on Research in Computer Security. Oslo, 2017. 65−82.

[20] Xu K, Ba J, Kiros R, Cho K, Courville A, Salakhudinov R, Zemel R, Bengio Y. Show, attend and tell: Neural image caption generation with visual attention. In: Proc. of the 32nd Int'l Conf. on Machine Learning (PMLR 37). 2015. 2048−2057.

[21] Do LNQ, Krüger S, Hill P, Ali K, Bodden E. Debugging static analysis. IEEE Trans. on Software Engineering, 2020, 46(7): 697−709. [doi: 10.1109/TSE.2018.2868349]

[22] Zhao MM, Li TH, Alsheikh MA, Tian YL, Zhao H, Torralba A, Katabi D. Through-wall human pose estimation using radio signals. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). 2018. 7356−7365.

[23] Wang KC, Wang TT, Su XH, Ma PJ. Key scientific isues and state-art of automatic software fault localization. Chinese Journal of Computers, 2015, 38(11): 2262−2278 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2015.02262]

[24] Le TDB, Oentaryo RJ, Lo D. Information retrieval and spectrum based bug localization: better together. In: Proc. of the ESEC/SIGSOFT FSE. 2015. 579−590.

[25] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. of the IEEE/ACM 25th Int'l Conf. on Program Comprehension (ICPC). Buenos Aires, 2017. 218−229.

[26] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley, 2013. 345−355.

[27] 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 34th Int'l Conf. on Software Engineering (ICSE). Zurich, 2012. 14−24.

[28] 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.

[29] Thung F, Le XD, Lo D, Lawall J. Recommending code changes for automatic backporting of Linux device drivers. In: Proc. of the IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Raleigh, 2016. 222−232.

[30] Rodriguez LR, Lawall J. Increasing automation in the backporting of Linux drivers using Coccinelle. In: Proc. of the 11th European Dependable Computing Conf.—Dependability in Practice (EDCC). 2015. 132−143.

[31] Lawall J, Palinski D, Gnirke L, *et al*. Fast and precise retrieval of forward and back porting information for Linux device drivers. In: Proc. of the Usenix Conf. on Usenix Technical Conf. USENIX Association, 2017. 15−26.

[32] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Liang B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 80−109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[33] Allamanis M, Barr ET, Devanbu P, Sutton C. A survey of machine learning for big code and naturalness. ACM Computing Surveys, 2018, 51(4): Article 81.

[34] Zhen L, Zou DQ, Xu SH, Ou XY, Jin H, Wang SJ, Deng ZJ, Zhong YY. VulDeePecker: A deep learning-based system for vulnerability detection. In: Proc. of the Network and Distributed Systems Security (NDSS) Symp. San Diego, 2018. 1−15.

[35] Zhang C, *et al*. Automatic parameter recommendation for practical API usage. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). Zurich, 2012. 826−836.

[36] Raychev V, Vechev M, Yahav E. Code completion with statistical language models. ACM SIGPLAN Notices, 2014, 49(6): 419−428. [doi: 10.1145/2666356.2594321]

[37] Bhoopchand A, Rocktäschel T, Barr E, Riedel S. Learning python code suggestion with a sparse pointer network. arXiv:1611.08307, 2016.

[38] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Vol.1). 2016. 2073−2083. [doi: 10.18653/v1/p16-1195]

[39] Yin P, Neubig G. A syntactic neural model for general-purpose code generation. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics. Vancouver, 2017. 440−450.

[40] Rabinovich M, Stern M, Klein D. Abstract syntax networks for code generation and semantic parsing. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics Vancouver, 2017. 1139−1149.

[41] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs. In: Proc. of the Int'l Conf. on Learning Representations. 2018. 1−17.

[42] Kim S, Jr. Whitehead EJ, Zhang Y. Classifying software changes: Clean or buggy? IEEE Trans. on Software Engineering, 2008, 34(2): 181−196.

[43] Wu RX, Zhang HY, Kim SH, Cheung SC. ReLink: Recovering links between bugs and changes. In: Proc. of the SIGSOFT/FSE 2011 19th ACM SIGSOFT Symp. on Foundations of Software Engineering. 2011. 15−25.

[44] Yi Q, Yang Z, Liu J, Zhao C, Wang C. A synergistic analysis method for explaining failed regression tests. In: Proc. of the IEEE/ACM 37th IEEE Int'l Conf. on Software Engineering. Florence, 2015. 257−267.

[45] Thung F, Lo D, Jiang L. Automatic recovery of root causes from bug-fixing changes. In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE 2013). Koblenz, 2013. 92−101.

[46] Śliwerski J, Zimmermann T, Zeller A. When do changes induce fixes? In: Proc. of the Int'l Workshop on Mining Software Repositories (MSR 2005). New York: ACM, 2005. 1−5.

[47] Rungta N, Person S, Branchaud J. A change impact analysis to characterize evolving program behaviors. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance (ICSM). IEEE, 2012. 109−118.

[48] Roya H, Peter B. JavaParser: A fine-grain concept indexing tool for java problems. In: Proc. of the CEUR Workshop, Vol.1009. 2013. 60−63.

[49] Mark H, Robert H. An overview of program slicing. In: Proc. of the Software Focus, Vol.2. 2001. 85−92. [doi: 10.1002/swf.41]

[50] Silver D, Schrittwieser J, Simonyan K, *et al*. Mastering the game of Go without human knowledge. Nature, 2017, 550: 354−359.

[51] Tai KS, Socher R, Manning C, Christopher D. Improved semantic representations from tree-structured long short-term memory networks. In: Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Int'l Joint Conf. on Natural Language Processing. Beijing, 2015. 1556−1566.

[52] Cho K, van Merrienboer B, Gülçehre Ç, *et al*. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proc. of the EMNLP. 2014. 1724−1734.

[53] Elsayed G, Kornblith S, Le QV. Saccader: Improving accuracy of hard attention models for vision. In: Proc. of the Advances in Neural Information Processing Systems. 2019. 700−712.

[54] Devlin J, Chang MW, Lee K, *et al*. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the NAACL-HLT. 2019. 4171−4186.

[55] Chen R, Liu Y, Jia Z, Gao J. Isolating and understanding program errors using probabilistic dispute model. In: Proc. of the IEEE 37th Annual Computer Software and Applications Conf. Kyoto, 2013. 633−638. [doi: 10.1109/COMPSAC.2013.102]

[56] Zhang W, Li ZQ, Du YH, Yang Y. Fine-grained software bug location approach at method level. Ruan Jian Xue Bao/Journal of Software, 2019, 30(2): 195−210 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5565.htm [doi: 10.13328/j.cnki.jos.005565]

[57] Deshmukh J, Annervaz KM, Podder S, eds. Towards accurate duplicate bug retrieval using deep learning techniques. In: Proc. of the IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2017.

[58] Borg M, Svensson O, Berg K. SZZ unleashed: An open implementation of the SZZ algorithm—Featuring example usage in a study of just-in-time bug prediction for the jenkins project. In: Proc. of the 3rd ACM SIGSOFT Int'l Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE 2019). Tallinn, 2019. 7−12.

附中文参考文献:

[1] 李晓卓, 贺也平, 马恒太. 缺陷理解研究: 现状、问题与发展. 软件学报, 2020, 31(1): 20−46. http://www.jos.org.cn/1000-9825/5887.htm [doi: 10.13328/j.cnki.jos.005887]

[2] 虞凯, 林梦香. 自动化软件错误定位技术研究进展. 计算机学报, 2011, 34(8): 1411−1422. [doi: 10.3724/SP.J.1016.2011.01411]

[9] 郭肇强, 周慧聪, 刘释然, 李言辉, 陈林, 周毓明, 徐宝文. 基于信息检索的缺陷定位: 问题、进展与挑战. 软件学报, 2020, 31(9): 2826−2854. http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[10] 张芸, 刘佳琨, 夏鑫, 吴明晖, 颜晖. 基于信息检索的软件缺陷定位技术研究进展. 软件学报, 2020, 31(8): 2432−2452. http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[23] 王克朝, 王甜甜, 苏小红, 马培军. 软件错误自动定位关键科学问题及研究进展. 计算机学报, 2015, 38(11): 2262−2278. [doi: 10. 11897/SP.J.1016.2015.02262]

[32] 张健, 张超, 玄跻峰, 熊英飞, 王千祥, 梁彬, 李炼, 窦文生, 陈振邦, 陈立前, 蔡彦. 程序分析研究进展. 软件学报, 2019, 30(1): 80−109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[56] 张文, 李自强, 杜宇航, 杨叶. 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]

李晓卓(1992－), 女, 硕士, 主要研究领域为程序分析及缺陷定位.

贺也平(1962－), 男, 博士, 研究员, 博士生导师, 主要研究领域为系统安全, 隐私保护.

卿笃军(1993－), 男, 硕士, 主要研究领域为程序分析.

马恒太(1972－), 男, 博士, 副研究员, 主要研究领域为程序分析, 系统安全.