计算机科学
COMPUTER SCIENCE

基于词汇的源代码克隆检测技术综述

刘春玲，戚旭衍，唐永鹤，孙雪凯，李晴浩，张雨

引用本文

刘春玲，戚旭衍，唐永鹤，孙雪凯，李晴浩，张雨. 基于词汇的源代码克隆检测技术综述[J]. 计算机科学, 2024, 51(6): 12-22.

LIU Chunling, QI Xuyan, TANG Yonghe, SUN Xuekai, LI Qinghao, ZHANG Yu. Summary of Token-based Source Code Clone Detection Techniques [J]. Computer Science, 2024, 51(6): 12-22.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230400117

基于词汇的源代码克隆检测技术综述

刘春玲　戚旭衍　唐永鹤　孙雪凯　李晴浩　张　雨
信息工程大学网络空间安全学院　郑州 450001
(lcl_507@163.com)

摘　要　代码克隆指在软件开发过程中对源代码复用、修改、重构产生的文本相似或结构相似的代码。代码克隆对提升软件开发效率、节约开发成本有积极作用,但也会引起 Bug 传播,并对软件的稳定性、可维护性产生负面影响。代码克隆检测在剽窃检测、漏洞检测、版权侵权等领域具有重要的研究意义和应用价值。基于词汇的克隆检测技术能快速检测 1-3 型克隆,能扩展到其他编程语言,已被广泛应用于大规模克隆检测任务中。文中对近 5 年基于词汇的克隆检测技术的研究现状进行了梳理,根据相似性算法中的基本计算粒度将其分为 4 类,并对 10 余个技术特征进行了分析和总结,讨论其局限性及面临的挑战,最后结合新技术的发展提出了基于词汇的克隆检测技术未来可能的研究方向。

关键词:软件安全;源代码克隆检测;代码表征;深度学习
中图分类号　TP311

Summary of Token-based Source Code Clone Detection Techniques

LIU Chunling,QI Xuyan,TANG Yonghe,SUN Xuekai,LI Qinghao and ZHANG Yu
School of Network and Cybersecurity,Information Engineering University,Zhengzhou 450001,China

Abstract　Code cloning refers to the generation of similar or identical code during software development due to the reuse,modification,and refactoring of source code. Code cloning has a positive impact on improving software development efficiency and reducing development costs,but it can also do harm to the development and maintenance of software system,including but not limited to the decline of stability,and propagation of software defects. Clone detection techniques for source code have important research and application value in plagiarism detection,vulnerability detection,copyright infringement,and other fields. Although some excellent detection tools and techniques have emerged,there are still challenges in detecting syntactic and semantic clones on a large scale and in an effective manner. Among them,lexical-based clone detection technology can quickly detect type 1-3 clones and can be extended to other programming languages and large-scale projects,therefore it is commonly used for clone detection in large-scale databases. This paper reviews the research status of lexical-based clone detection technology in the past decade,analyzes and summarizes 16 selected literature from 10 characteristics,and finally proposes possible research directions for lexical-based clone detection technology in the future in light of new technological developments.

Keywords　Software security,Source code clone detection,Code representation,Deep learning

1　引言

代码克隆指项目内或项目间文本相似或功能相似的源代码片段。大量研究指出克隆代码在大型软件系统中普遍存在,约占 7%～23%[1],其中 Linux 存在 22.3% 的克隆代码[2]。代码克隆在一定程度上可以加快软件开发速度、提升代码质量,但如果含有 Bug 的代码被复用,可能引起 Bug 传播,产生一定的负面影响[3-5]。

在软件安全领域,代码开源也为攻击者恶意投毒提供了土壤,开源漏洞随之呈爆发式增长,高危漏洞占比呈现增加趋势,造成的危害和影响范围呈扩大态势,如 2021 年 12 月爆光的 Log4j 漏洞,由于包含漏洞的 Log4j 组件被广泛应用于大量其他业务系统开发,该漏洞影响 90% 以上基于 java 开发的应用平台,对互联网安全环境造成了核弹级的严重冲击。

鉴于代码克隆带来的不利影响,为了快速、准确地定位代码克隆,工业界和学术界对代码克隆进行了大量研究,研究指出,1 型和 2 型克隆已经能达到很好的查全率和查准率,3 型和 4 型克隆检测性能有待进一步提高。有研究表明,3 型克隆的稳定性更差[6]并且更有可能引入和繁衍 Bug[7],因此提高 3 型克隆检测的检测性能有重要的研究价值和现实需求。

研究基于词汇的克隆检测技术以提高 3 型克隆检测性能出于 3 方面的考虑:

到稿日期:2023-04-17　返修日期:2023-09-27
基金项目:河南省重点研发计划(221111210300)
This work was supported by the Key R&D Program of Henan Province(221111210300).
通信作者:唐永鹤(tyh_983@126.com)

1）虽然基于树和图的克隆检测技术能检测到3型克隆，但需要把源代码抽象到更高级的层次，这种抽象需要健壮的语法分析器或解析器。然而，目前还没有能解析多种语言的高鲁棒性解析器。

2）相比基于树的检测算法，基于词汇的检测技术速度快，可以通过各种启发式方法加快速度，能方便地扩展到多种语言，具有良好的可扩展性，并可以实施各种转换，消除程序员编码风格差异引起的误判（如编码风格自由的编程语言的换行重定位），检测到具有不同换行符或不同标识符名称的克隆。

3）虽然基于抽象语法树的技术能检测到更多的3型克隆，但不同的代码片段可能生成完全匹配的抽象语法树，从而产生误报[8]。

因此，研究基于词汇的克隆检测技术有理论基础和现实需求。

本文从软件安全的角度研究代码克隆检测，第1章介绍了研究基于词汇的克隆检测技术的意义；第2章介绍了代码克隆的相关术语；第3章介绍了基于词汇的克隆检测技术的一般处理流程；第4章对基于词汇的克隆检测技术进行了系统分析；第5章结合当前技术发展趋势，展望了基于词汇的克隆检测技术的研究方向；最后总结全文并展望未来。

2 代码克隆相关术语

2.1 基本概念

1）代码片段。代码片段由连续的代码行组成，用三元组〈文件名、开始行号、结束行号〉表示，可以是函数、方法、代码块等。

2）克隆关系/克隆对。克隆关系指两个代码片段在文本、语法或者语义上相似。如果两个代码片段存在克隆关系，则称它们是克隆对。克隆关系满足自反性和对称性。

3）克隆类。克隆对的集合称为克隆类。在实际应用中，通过检测和分析克隆类，可以帮助开发人员发现和解决软件系统中的克隆问题，以提高软件质量和可维护性。

2.2 克隆检测常用评价指标

精确率（Accuracy）又称查准率，指检测到的阳性克隆中真阳性所占的比例。

召回率（Recall）又称查全率，指检测到的阳性克隆在目标体系中所有真实克隆的比例。精确率和召回率越高，性能越好。但事实上这两个指标通常会出现矛盾的情况，例如只检测到一个阳性且是真的，此时精确度为100%，但召回率却很低，因此一般综合考虑这两个指标，其中最常见的指标就是F1-Score，即精确率和召回率的调和平均值。F1-Score越高，说明克隆检测方法的性能越好。

2.3 代码克隆分类

克隆分类：源代码克隆按照源代码之间的相似程度分为文本克隆和功能克隆[9]。

文本克隆指在语法上相似，按照相似性程度可进一步分为3类。

1型：精确克隆，又称文本相似，指两个代码片段除了空格、布局和注释等非可执行语句外完全相同，其相似度为100%。

2型：重命名克隆，又称词汇相似，指两个代码片段结构相似，可执行语句中只有变量名、函数名、类型名或常量不同，即存在变量名、函数名、类型名或常量的重命名，其相似度为90%。

3型：近似克隆（Near-Miss Clone）或重组克隆，又称语法相似，指在2型的基础上，存在语法相似的语句删除、插入、修改，其相似度为60%～80%。语法相似性结构和阈值由克隆算法提前设定。阈值有多种设定方法，如绝对值法和相对值法。绝对值法直接给定两个相比较代码片段中的不相同语句的数量，如两条语句不同即为克隆；相对值法是给定一个和代码长度相关的值，30%的代码存在不同即为克隆。编辑距离、余弦距离和欧氏距离常被用来衡量相似度。

4型：功能相似，又称语义相似，指两个代码片段语法不同，但能实现相同的功能。两个代码段功能相似但没有明确的定义，比文本克隆检测难度更大。BigCloneEval[10]把3型和4型又细化为4类：VST3（Very-Strongly Type-3），相似度[0.9，1]；ST3（Strongly Type-3），相似度[0.7，0.9]；MT3（Moderately Type-3），相似度[0.5，0.7]；WT3/4（Weakly Type-3 or Type-4），相似度[0.0，0.5]。

克隆相似度从1型到4型逐渐降低，检测难度逐渐加大，对于3型最小语法相似没有共识，特别是4型没有严格的定义。

大差异克隆（Large-Gap Clones）：大量插入或删除语句集中在同一位置的克隆，并且代码块的行数比小于等于0.7[11]。注：A和B是两个3型克隆代码块的行数（$A<B$），且$A/B\leqslant 0.7$。

大方差克隆（Large-Variance Clones）：大量插入或删除语句分散在多个位置，大方差克隆在3型克隆中更常见[12]。

3 基于词汇的克隆检测的处理流程

词汇又称符号或token，是编译器能理解的最小预处理单元。基于词汇的源代码克隆检测技术通过比较代码中的标识符、常量、运算符等词法来识别相似的代码片段，该技术通常根据词汇序列或频率检测相似，已被广泛应用在源代码克隆检测和漏洞检测任务中，如能检测2型克隆的CCFinder[13]、检测3型克隆的通用克隆检测技术SourcererCC[14]、检测3型克隆的未修补漏洞检测工具ReDeBug[15]、大差异克隆和相关Bug发现的检测工具CCAligner[11]、大方差检测工具LV-Mapper[12]。其一般处理流程包括图1所示的4个阶段。

1）预处理。根据任务需求，删除空格、注释等无意义的代码片段，然后确定克隆检测粒度，如文件、函数或代码块等，最后利用词法分析器或解析器按照确定的克隆检测粒度将源代码划分为符号序列。

2）代码表征。该步骤进一步对符号序列进行表征，然后确定基本计算单元。通过基于词汇的克隆检测技术研究发现，这类技术的基本计算单元可以分为4类，包括n-gram、token-by-token、词袋和行，并且该基本比较单元会影响下一步骤相似性比较算法的选择。

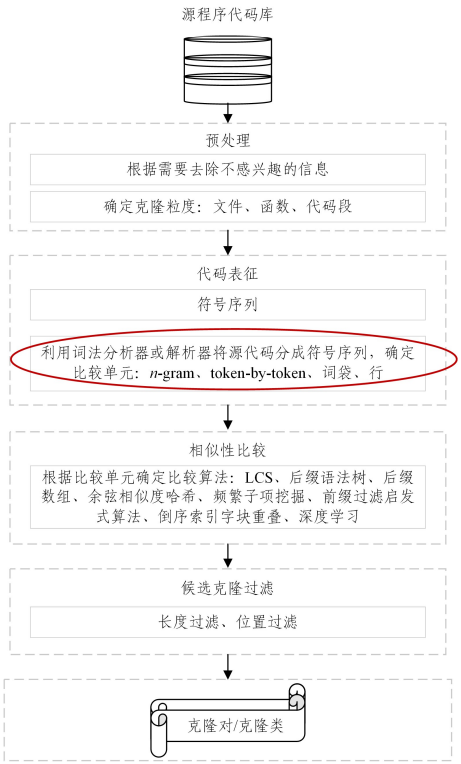


图 1 基于词汇的代码克隆检测的一般流程

Fig. 1 General process of code cloning detection based on token

3)相似性比较。在这一步骤,每一个代码片段都会与其他代码片段进行对比来找到代码的克隆,比对的结果将以克隆对列表的方式呈现。其中,相似比较的算法很大程度上由源代码表征方式决定,而比较算法会影响检测技术的性能,如时间复杂度、精确率和召回率等。

4)候选克隆过滤。可选步骤,过滤掉误报的代码克隆,过滤方法包括长度过滤和位置过滤启发式算法等。

虽然已有文献对源代码相似性检测技术和工具进行了系统的介绍和比较[3,16-22],研究了各技术的检测性能及最优参数、相似性阈值设置和各个工具适用的应用场景,也有学者对机器学习、深度学习技术在代码克隆检测中的应用进行了综述[23-24],研究该领域的最新趋势和最新技术。因为基本比较单元一定程度上决定了可供选择的相似性比较算法,而算法会影响检测技术的性能,因此本文提出根据相似性算法中选择的基本比较单元把基于词汇的克隆检测技术分为 4 类,即 *n*-gram、token-by-token、词袋和行,并进行了系统的梳理,研究了相似性算法基本比较单元对克隆检测性能的影响,提出了基于词汇的克隆检测技术的局限性和未来可能的研究方向。

4 基于词汇的克隆检测技术的系统分析

本章对基于词法的克隆检测技术进行了系统的文献综述,从 IEEE, ACM, Springer 和 Elsevier 等流行文献库中搜索并筛选出近 5 年和经典的基于词法的克隆检测研究文献 17 篇,其中 TSE, ICSE, S&P 有 6 篇, ICST 和 JSS 等 4 篇。表 1 分类列出了这些文献,列举了文献的发表年份、在文章中的引用和题目,文章围绕这些文献展开研究。

表 1 筛选的文献

Table 1 Identified primary works

基本计算粒度	发表时间	题目
n-gram	2017/[25]	CCFinderSW: Clone detection tool with flexible multilingual tokenization
	2021/[26]	NIl: large-scale detection of large-variance clones
token-by-token	2002/[13]	CCFinder: A multilinguistic token-based code clone detection system for large scale source code
	2006/[27]	CP-Miner: Finding copy-paste and related bugs in large-scale software code
	2017/[28]	CClearner: A deep learning-based clone detection approach
	2017/[29]	A technique to detect multi-grained code clones
	2019/[30]	VCIPR: vulnerable code is identifiable when a patch is released(hacker's perspective)
	2020/[31]	SAGA: efficient and large-scale detection of near-miss clones with GPU acceleration
词袋	2016/[14]	Sourcerercc: Scaling code clone detection to big-code
	2017/[32]	Fast and flexible large-scale clone detection with CloneWorks
	2018/[33]	Scalable code clone detection and search based on adaptive prefix filtering
	2021/[34]	Multi-threshold token-based code clone detection
	2022/[35]	MSCCD: grammar pluggable clone detection based on ANTLR parser generation
	2023/[36]	Ccstokener: Fast yet accurate code clone detection with semantic token[J]. Journal of Systems and Software
行	2012/[15]	ReDeBug: finding unpatched code clones in entire os distributions
	2018/[11]	CCAligner: a token based large-gap clone detector
	2020/[12]	Lvmapper: A large-variance clone detector using sequencing alignment approach

4.1 结构分析

本节根据编程语言、克隆类型、数据集、开源性、克隆粒度等技术特性对文献进行分析。

4.1.1 编程语言和克隆类型

本节讨论筛选文献所涉及的编程语言和克隆类型,如表 2 所列,图 2 显示了其百分比。图 2 表明,研究者关注的编程语言主要有 C, C++, C# 和 Java,其中,71% 的技术支持 Java 和 C 编程语言克隆检测,17% 的技术支持 C#,其余 12% 支持 COBOL。在单一编程语言和多种编程语言的分析中,有 40% 的技术只支持单一编程语言,只有 24% 的检测技术与语言无关,多语言克隆检测技术需要进一步研究。

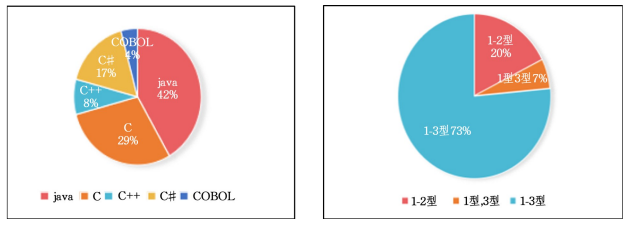
表 2 克隆检测语言和类型

Table 2 Clone detection languages and types

工具/技术/作者	数据集	检测语言	参数配置	检测粒度	检测类型
CCFinder	JDK、操作体统 FreeBSD、NetBSD 和 Linux	C，C++，Java，COBOL	N/A	文件	1 型 2 型
CP-Miner	Linux，FreeBSD，Apache Web server，PostgreSQL	C，C++	冲突阈值 0.6	代码块	1 型 2 型
ReDeBug	发行的 OS 版本	编程语言无关	相似阈值 1	文件	1 型 3 型
Sourcerercc	BigCloneBench Mutation and Injection	语言无关，目前实现 Java，C 和 C#	最小 6 行或 50 个词汇，相似性阈值 0.7	函数	1 型 2 型 3 型
CCFinderSW	Rosetta Code https://github.com/acmeism/RosettaCode.Data	Java，C，C#	阈值 0.7	代码块、函数、文件	1 型 2 型
CClearner	BigCloneBench DeepLearning4j	C	DNN，隐藏层 2，迭代次数 300，输出层克隆的可能性 0.98	函数	1 型 2 型 3 型
Decrescendo	Apache http://svn.apache.org/repos/asf/	Java	最小克隆长度 50 标记，匹配的词汇数/不匹配词汇数最大为 0.3	文件、函数、代码块	文件和函数粒度： 1 型 2 型 代码块粒度： 1 型 2 型 3 型
CloneWorks	BigCloneBench	Java，C，C#	最小词汇是 10 行，相似性阈值设置为 0.7	代码块、函数和文件	1 型 2 型 3 型
Nishi	IJaDataset 2.0	Java	相似性阈值为 40%～80%	函数	1 型 2 型 3 型
CCAligner	BigCloneBench The Mutation and Injection Framework	C，Java，C#	滑动窗口：6 行 非对称相似性阈值 0.7	函数	1 型 2 型 3 型
VCIPR	Linux kernel 等开源软件 已发布的 1 000 多个 CVE 漏洞源代码	编程语言无关	特征提取 精确匹配	函数	1 型 2 型 3 型
SAGA	BigCloneBench，Mutation and Injection	Java	最小克隆长度 50 标记，最小候选克隆长度 20 个标记，相似性阈值 0.6	函数和代码块	1 型 2 型 3 型
Lvmapper	Linux，BigCloneBench	C，Java，C#	滑动窗口：3 行动态分段相似性阈值	函数	1 型 2 型 3 型
NIl	Ant，Maven JDK6，Apache Commons Mutation Framewor BigCloneBench	Java	过滤阈值 0.1，验证阈值 0.7，n-gram 中 $N=6$	函数	1 型 2 型 3 型
Golubev	BigCloneBench	Java	代码块的大小研究最佳参数配置	代码块	1 型 2 型 3 型
CCStokener	BigCloneBench	Java，C	过滤参数 β 为 0.5，θ 为 0.4，相似性阈值 η 为 0.65	函数	1 型 2 型 3 型
MSCCD	BigCloneBench	编程语言无关	最小令牌 50，最小 6 行，阈值 0.7	函数	1 型 2 型 3 型

综上所述，随着软件开发的多样化，克隆检测面临多语言、跨语言和跨框架的挑战，如语言和框架之间的差异和克隆代码的表征方式。

关于克隆检测类型，80%的技术能检测 3 型克隆，这一分析表明提升 3 型克隆检测能力值得更深入研究，可以考虑利用语法信息来提高克隆检测精度和效率，例如直接在语法树上进行比较。另外，图 2 显示有 7%的技术无法检测 2 型克隆，原因在于 ReDeBug 没有对变量名或数据类型进行规范化，因此不能检测变量重命名或数据类型更改引起的 2 型克隆。



(a)编程语言百分比 (b)克隆类型百分比

图 2 编程语言和克隆类型的分析

Fig. 2 Analysis of programming languages and clone types

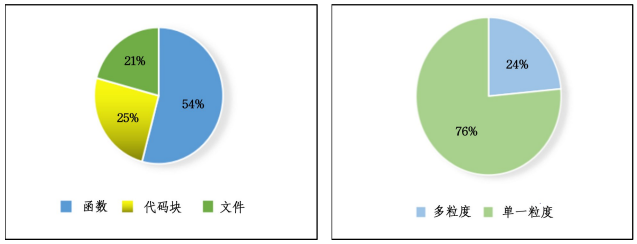
4.1.2 基准数据集

由表2可见，在源码克隆检测中，研究者主要使用BigCloneBench[37]和Mutation and Injection Framework[38]两个基准数据集，其中BigCloneBench是公认的真实数据集，它由25000个Java开源系统组成，共有8375313个人工验证的4类克隆，尽管其召回率、执行时间和可扩展性可以通过BigCloneEval自动评估，但由于它是人工验证的真实数据集，克隆类型中难免存在错标和漏标情况。Mutation and Injection Framework是人工合成基准，它能精准无偏差地测量人工合成的每种克隆类型的召回率。理想情况下，这两个基准数据集互补使用，为测试工具提供综合性能。其他数据集包括JDK，Linux，PostgreSQL，Apache Web server和Linux kernel等开源软件，主要用来进行漏洞检测。

代码克隆检测的现有基准数据集偏向于源代码丰富且易于获得的流行语言，评估其他语言的代码克隆检测的召回率仍然很困难。

4.1.3 克隆检测粒度

克隆检测粒度包括文件、类、函数、代码块等，选择不同的粒度会影响克隆检测的效果和速度。一般来说，细粒度的克隆检测可以更准确地找到相似的代码，但是会增加检测复杂度和运行时间，在实际应用中需要在精度和效率之间做出取舍，但在克隆检测最小长度上具有共识，即6行或50个词汇。选择函数作为克隆检测粒度有理论和实验依据，原因在于函数是实现特定功能的完整单元，并且在编程实践中常被用来复用，例如VUDDY通过理论和实验证明了函数粒度能在漏洞检测中达到可扩展性和精确性的平衡；CP-Miner通过实验发现在函数粒度上存在11.3%～19.2%的克隆代码；Ishihara[39]在约13000个软件中发现存在约49%的函数级克隆；并且有两个基准集BigCloneBench和Mutation and Injection Framework支持函数级克隆检测粒度的综合检测性能验证。表2显示，54%技术支持函数级克隆检测，25%的文献能在更细粒度如代码块检测克隆。由此分析，在注重精确性场合有必要研究细粒度的克隆检测技术。在单一粒度和多粒度的检测中，图3显示约24%的克隆检测技术支持多粒度，为扩大急速的扩展性需要加大多粒度克隆检测技术的研究力度。



(a)克隆粒度百分比 (b)单一粒度和多粒度百分比

图3 文献研究的克隆粒度

Fig. 3 Analysis of clone granularity

表3列出了开源技术的链接，仅有约59%的文献公开了它们的源码。为了使后续研究者能够重现实验结果、公平对比技术间的检测性能以及进行更深入的性能优化等研究，更多的研究技术需要公开其源码。公开源码不仅可以促进技术的共享和交流，还可以提高技术的可重复性和可信度，推动克隆检测技术的发展。

表3 工具/开源链接

Table 3 Tools/open source links

时间/引用	工具/技术/作者	研究动机	是否开源	链接
2002/[13]	CCFinder	检测政府部门大型软件系统	是	https://github.com/gpoo/ccfinderx
2006/[27]	CP-Miner	寻找大规模软件代码中的代码克隆和相关Bug	否	
2012[15]	ReDeBug	检测未修补的代码克隆	是	http://security.ece.cmu.edu/redebug/
2016/[14]	Sourcerercc	大规模克隆检测	是	https://github.com/Mondego/SourcererCC/releases
2017/[25]	CCFinderSW	检测任何编程语言	是	https://github.com/YuichiSemura/CCFinderSW
2017/[28]	CClearner	最早利用深度学习的基于词汇的克隆检测	是	https://github.com/liuqingli/CCLearner
2017/[29]	Decrescendo	多粒度克隆检测	否	
2017/[32]	CloneWorks	可定制、快速、可扩展的项目间大规模克隆检测	是	https://github.com/jeffsvajlenko/CloneWorks
2018/[33]	Nishi	可扩展的3型克隆检测	否	
2018/[11]	CCAligner	检测大差异克隆	否	
2019/[30]	VCIPR	可扩展的未修补漏洞检测	否	
2020/[31]	SAGA	段粒度大规模近似克隆	是	https://github.com/FudanSELab/SAGA
2020/[12]	Lvmapper	检测大方差克隆	否	
2021/[26]	NIl	检测大方差克隆	是	https://github.com/kusumotolab/NIL
2021/[34]	Golubev	多阈值搜索，在不降低精度前提下提高召回率	否	
2022/[35]	MSCCD	语言无关的3型克隆检测	是	https://doi.org/10.5281/zenodo.5886550
2023/[36]	CCStokener	使用语义令牌快速而准确地检测3型代码克隆	是	https://github.com/CCStokener/CCStokener

4.2 克隆检测技术的性能分析

本节将基于词汇克隆检测技术的性能分为4类进行分析。

表4—表7列出了各文献的精确率和召回率，这些数据均来自各自的文献。如果文献中没有明确指出，则用“—”符号代替。少数文献(如ReDeBug)的精确性和召回性只评估了相对较少的具有安全漏洞的代码实例，而大多数文献采用了基准数据集，因此该分析相对公平。

表 4 基于 n-gram 基本比较单元的克隆

Table 4 Cloning based on n-gram basic comparison unit

检测工具	精确率/%		召回率/%						检测算法	复杂度
	大方差	一般克隆	大方差	1 型	2 型	VST	ST	MT		
CCFinderSW	—	—	—	—	—	—	—	—	基于 n-gram 的二元倒序索引	—
NIL	87	94.0	100	99.9	96.6	93.5	67.1	10.6	利用 Hunt-Szymanski 算法的基于词汇的 LCS	$O(rlog\|A\|+\|B\|log\|B\|)$，$A$，$B$ 为词汇序列且 $\|A\|<=\|B\|$，r 为其共享词汇数

表 5 基于 token-by-token 基本比较单元的克隆

Table 5 Cloning based on token-by-token basic comparison unit

检测工具/时间	精确率/%	召回率/%					检测算法	复杂度
		1 型	2 型	VST	ST3	MT		
CCFinder	—	—	—	—	—	—	后缀语法树	$O(mn)$
CP-Miner	92	—	—	—	—	—	频繁子序列挖掘技术	$O(n^2)$
CCLearner	93	100	98	98	89	28	深度学习	采用两种过滤技术，复杂度小于 $O(n^2)$
Decrescendo	—	—	—	—	—	—	MD5 哈希算法 Smith Waterman 识别相似哈希序列	—
SAGA	99	100	100	95	60	10	利用 Data Parallel Prefix-Doubling (DPPD)构造后缀数组	$n*(n-1)/2$，数据分块加速
VCIPR	83(漏洞检测)	—	—	—	—	—	MD5 哈希	$O(nm)$

表 6 基于词袋基本比较单元的克隆

Table 6 Cloning based on bag-of-words basic comparison unit

检测工具/时间	精确率/%	召回率/%					检测算法	复杂度
		1 型	2 型	VST3	ST3	MT3		
SourcererCC	91.0	100.0	98.0	93.0	61.0	5	倒序索引，子块重叠，位置过滤	利用过滤策略低于 $O(n^2)$
CloneWorks	93.0	100.0	100.0	100.0	96.0	—	分区部分索引的子块过滤启发式算法	—
Nishi	91.0	97.0	77.0	60.0	26.0	16	前缀过滤启发式算法	利用过滤策略低于 $O(n^2)$
Golubev	—	99.5	96.4	92.1	59.6	—	根据代码长度的多阈值搜索	—
MSCCD	92.0	100.0	98.0	93.0	63.0	7	SPT 优化和关键词过滤，相似性和克隆检测算法同 SourcererCC	利用过滤策略低于 $O(n^2)$
CCStokener	90.2	100.0	99.0	98.0	92.0	53	全局 k-tokens 倒序索引，并利用过滤技术优化	$O(k*n^2)$

表 7 基于行基本比较单元的克隆

Table 7 Cloning based on row basic comparison units

检测工具/时间	F1/%	精确率/%			召回率/%						检测算法	复杂度
	大差异	大方差	大差异	一般克隆	1 型	2 型	VST3	ST3	MT3	WT		
ReDeBug	—	—	—	—	—	—	—	—	—	—	散列哈希，利用 bloom 过滤器	$O(n)$
CCAligner	86.5	—	85	80.0	100	99	97	70	10	—	MurmurHash 哈希	$O(n)$
LVMapper	—	88		88.5	100	99	98	82	19	0.3	MurmurHash 哈希	$O(n)$

4.2.1 基于 n-gram 基本比较单元的克隆检测技术

n-gram 指一个给定的字符串或符号序列中 n 个连续的符号，CCFinderSW[25] 和 NIL[26] 使用了这种方法。

CCFinderSW 是 CCFinderX 的扩展工具，利用三元组(序号，n-gram，hashcode 哈希值)表示唯一的 n-gram，然后合并哈希值的序号，建立按序号排序的二元组索引序列〈哈希值，序号序列〉，生成克隆类，最后把具有相同的哈希值的下一个序号开始的 n-gram 组成 $n+1$-gram 更大的克隆。该工具采用词汇分析机制，允许用户根据目标语言灵活地更改评论、标识符名称和关键字的语法。然而，由于 CCFinderSW 只支持词法分析更改，因此它无法检测到包含句法差异的 3 型克隆。

NIL 是实现大方差的可扩展开源克隆检测工具，其预处理阶段将源代码中的函数转换为词汇序列，不需要进行词法分析，另外为了增强技术的可扩展性，利用 n-gram 的反向分组的部分索引来减少函数间的比较次数，但由于没有进行标识符的规范化，在实现低误报率的同时也可能漏报 2 型克隆。在检测阶段，采用和 LVMapper 相同的处理过程，即定位、过滤、验证。NIL 在大方差克隆检测中的精确率和召回率高于 LVMapper 和 CCAligner，这个高精度来源于其不会因为连续赋值语句(如构造函数)和连续 if 语句的代码块而引起误报，并且不会因为共享少量连续行而引起漏报。在一般 1 型－3 型克隆检测中，NIL 的准确度与现有先进工具 LVMapper，CCAligner，SourcererCC 和 NiCad[40] 相当，并且精确度高于 LVMapper，CCAligner，具有高可扩展性(250 MLOC 的代码库约 7h40min)[26]。

基于 n-gram 的克隆检测技术的优点如下。

1)效率高:利用哈希表等数据结构,快速地计算出代码片段的 n-gram 特征,从而提高检测效率。

2)可扩展性强:通过调整 n-gram 的大小和阈值,适应不同的克隆检测需求,并且可以与其他克隆检测技术相结合,提高克隆检测的精度和效率。

3)易于实现:实现比较简单,只需要计算代码片段的 n-gram 特征,然后进行相似度比较,因此容易实现和应用。

4.2.2 基于 token-by-token 基本比较单元的克隆检测技术

token-by-token 指逐词对比的算法,对于两个待比较的源码,按照相同的方式分词并构建 token 序列,然后计算它们之间的 token 相似度。常用的计算方法包括余弦相似度、Jaccard 相似度等,其中 CCFinder 是一个经典工具。

CCFinder 将词法分析器产生的词序列进行规则转换,检测语法不同但含义相似的克隆,并过滤具有指定结构模式的代码克隆部分,时空复杂度较高。CCFinder 有较高的召回率,在对 FreeBSD 和 Linux 的检测中比基于行的检测技术多检测 23%的克隆,但准确率偏低,并且一次只能检测一种编程语言。

CP-Miner[27]在克隆检测中利用 CloSpan 增强算法,使用深度优先搜索生成包含所有闭合频繁子序列的候选频繁子序列集(要求代码块最少包含 30 个词汇、最多 30 条语句,对每个语句标识化后用 HashPJW 函数计算哈希值),然后从候选集中修剪非闭合子序列(标识符符号化引起的语法结构相似)以减少误报。在频繁序列的最大长度受常数约束时,最坏情况下计算复杂度为 $O(n^2)$。CP-Miner 由于能检测 3 型克隆,因此能检测到的克隆代码比 CCFinder 多 17%~52%,但是当复制粘贴时对代码有复杂修改时,CP-Miner 并不能检测出,并且有文献指出 CP-Miner 的假阳性率为 90%[6],如此高的假阳性限制了它在漏洞检测中的应用。

CCLearner[28]是最早利用深度学习的基于词汇的克隆检测方法。该方法分为训练和测试两个阶段。在训练阶段,使用 ANTLR 和 Eclipse ASTParser 对每种方法对(克隆对和非克隆对)进行解析,识别所有标记并分成 8 类(对一些特性类型标识符还使用了抽象语法树),并统计每种方法中这 8 类标记的频率,把方法表示为〈词汇、出现次数〉词汇频率列表,然后建立 8 维向量列表,通过向量度量两种方法的相似性(阈值为 0.5),利用 DeepLearning4j 数据集训练出二分分类器,并且删除 6 行以下的方法来减低噪音。在测试阶段,过滤掉 6 行以下的所有候选克隆后,再过滤掉候选克隆中较大方法的代码行数是另一个的 3 倍以上的候选克隆,当判断克隆的可能性大于等于 0.98 时则判定其为克隆。CCLearner 在 BigCloneBench 基准数据集上与 3 个经典工具(基于文本的 NiCad、基于词汇 SourcererCC、基于树的 Deckard[41])第一次进行了系统的研究,实验结果显示 CCLearner 比 SourcererCC 和 NiCad 检测到更多的真实克隆,并且比 Deckard 能够更有效地发现克隆[31]。

Yuki 等[29]实现了针对 JAVA 语言的在文件、方法、代码段 3 个粒度从粗到细逐层过滤检测克隆的工具 Decrescendo,利用 MD5 哈希判断克隆,在检测速度和召回率上达到很好的平衡,并生成容易分析的检测结果,但是该工具没有和其他的检测工具做对比实验,也没有评估其召回率、精确率等性能。

SAGA[31]是为了解决在段粒度(函数内的代码段)上高效检测大规模代码库中近似克隆难题而实现的克隆工具,它采用后缀数组数据结构,利用 GPU 加速算法,合并相邻的 1 型和 2 型克隆作为 3 型克隆,能在函数和段粒度上高效检测 1 型—3 型克隆。该技术有 3 个优点:1)可扩展性强,可以在 11min 内检测出 1 亿行代码中的代码克隆(召回率和精度与最先进的方法相当),比 CloneWorks 快 10 倍以上,比 SourcererCC 快 70 倍以上,是在大型代码库中以段粒度有效检测 3 型近似克隆的唯一工具(11h 检测 10 亿行代码);2)精确率高,在函数粒度上的精确度超过 99%,在片段粒度上超过 70%;3)使用不依赖第三方的词法分析器,可扩展到其他语言。文献[31]值得借鉴的思想如下:3 型克隆是 1 型和 2 型的组合,并把 3 型克隆按照通常克隆检测中的最小克隆长度分成两组并分别检测,一组是通常克隆检测中不小于最小克隆长度的 1 型和 2 型组合而成的 3 型克隆,另一组是小于最小克隆长度 1 型和 2 型构成的 3 型克隆。该文献得出对克隆检测相关领域有统计学意义的结论,即普遍存在的克隆是 API 使用模式、通用算法和类似的 switch-case 结构,并进一步分析了函数粒度的 3 个误报原因:1) 非常短的连续赋值语句;2)switch-case 结构的 case 子句中重复不同的 API 调用;3)非常长的单个语句执行大量字符串连接。

VCIPR[30]是一个可扩展的、与语言无关的未修补函数级漏洞检测技术。该技术使用 MD5 哈希值的精确匹配实现了低错误检测率和高速检测,处理近 10 亿行源代码仅花费了 12h35min,而 ReDeBug 需要 1d3h,SourcererCC 需要 25d3h,VUDDY 需要 14h17min。

基于 token-by-token 的克隆检测技术的特点如下:

1)精度高:精确地比较两个代码片段之间的相似度,因此可以检测出较为相似的代码片段。

2)易于实现:实现比较简单,很多开源的克隆检测工具都采用了这种方法。

3)效率低:需要对代码进行分词和构建 token 序列,因此计算复杂度较高,处理大规模的代码库时需要消耗大量的时间和计算资源,平均时间复杂度是 $O(n^2)$。

4)受代码格式影响:代码格式敏感,因此代码格式不同可能会影响检测结果。

4.2.3 基于词袋基本比较单元的克隆检测技术

本文把仅考虑共享词汇、不考虑词汇顺序、通过计算原始词汇的覆盖率来测量相似性的克隆检测算法归类成词袋法,SourcererCC[14]是这类算法的典型代表。

SourcererCC 是首个能在单机上检测 250MLOC 和 3 型克隆的通用克隆检测技术。它采用和语言无关的扫描器解析源代码中的代码块并进行符号化,然后利用低频词汇建立倒序索引,最后采用重叠函数(overlap)计算共享词汇。在检测阶段,SourcererCC 采用子块重叠和位置过滤技术减少假阳性,降低时间复杂度。由于词袋法忽略了代码行的顺序和语法结构,因此能检测 3 型克隆,即使行的插入、删除、更改较大时也能保持较高的查全率和查准率。另外,由于 SourcererCC 只考虑共享词汇,不考虑词汇次序,并且为了精度和索引效率

没有对词汇进行规范化，因此减少了为检测 2 型克隆带来的高误报，其精确率达到 91%，高于文献中选取的当前先进工具。在 VUDDY 实验中，SourcererCC 不能识别 IF 修补和未修补代码而产生较高的假阴性[7]。

Nishi[33]的目标是可扩展的检测 3 型克隆，该技术利用自适应扩展了 SourcererCC 的前缀过滤启发式算法，在文件粒度为每个 2-prefix 策略建立增量反向索引数据结构，该技术可以无修改地应用于代码搜索。当相似度阈值为 40%～80%时，SourcererCC 的可扩展性在最佳情况下，执行时间减少了 11%，过滤候选数增加了 63%。

Golubev 等[34]提出了基于词汇的词袋法的改进技术，该文献的内容中有两点值得思考：1）与常用的根据功能研究最佳参数配置不同，提出针对代码块的大小研究最佳参数对配置思想，即代码块中最小词汇数和相似性阈值；2）针对这两个参数会在不同的参数对中产生大量的重复克隆问题，进一步提出了减少时间开销的优化算法，对于每个具体的相似性阈值，通过实验找到最小词汇数和最大词汇数。该技术对增加克隆检测数量的效果明显，在 SourcererCC 和 CloneWorks 中对 1 型—3 型克隆的召回率均有提升，约为 40.5%～56.6%。特别是在 ST3 中，提升效果最为显著，约为 20%。

CloneWorks[32]是可定制、快速、可扩展的大规模克隆检测工具，可以自定义源代码转换、规范化、检测类型等，支持 Java，C 和 C# 这 3 种语言在代码块、函数和文件 3 种粒度上进行克隆检测。该工具使用修改的 Jaccard 度量相似性，使用部分克隆索引的子块过滤启发式算法过滤克隆。CloneWorks 在和 iClones，NiCad，SourcererCC 的对比实验中，在召回率、精确率和可扩展性上有明显优势[38]。

MSCCD[35]（多语言语法代码克隆检测器）是第一个可以与语法定义文件一起使用的 3 型克隆检测工具，它允许用户输入 ANTLR 语法定义文件。首先借助 PT（解析树）的语义信息提高 3 型克隆的检测性能并实现语言无关性，通过简化 PT 和关键词过滤技术降低检测时间复杂度，可以在 6h 内完成 100MLOC 检测任务。MSCCD 使用和 SourcererCC 基本相同的相似度计算方法和克隆检测算法，但 ST3 和 MT3 上的召回率高于 SourcererCC，文献[35]认为其原因是 MSCCD 创建了多粒度的令牌包，在进行完整检测时，对同一代码段进行多粒度检测提高了检测能力。

CCStokener[36]旨在提高大规模代码库中对 3 型克隆的检测能力，利用来自 AST 路径的结构信息和来自令牌关系的语义信息来生成语义令牌。我们采用并改进了定位-过滤-验证流程，以实现快速克隆检测的目标。

基于词袋比较基本单元的克隆检测技术不考虑词汇顺序，因此能检测语法相似的 3 型克隆，但当重命名的标识符太多时，相似性会急剧下降，检测性能降低。

基于词袋的克隆检测技术的特点如下。

1）可扩展性：可以处理大规模的代码库，因为只需要对每个代码片段构建一个词袋，而不需要对整个代码库进行分析。

2）鲁棒性：对代码的格式、注释等因素并不敏感，并且不考虑词汇顺序，能够适应代码变形的情况。

3）无法处理复杂的克隆类型：该技术主要适用于基于复制-粘贴的克隆类型，对于其他类型的克隆（如函数调用、继承等），其效果可能不佳。

4.2.4 基于行比较单元的克隆检测技术

本文逐行（line-by-line）对源代码进行归一化，将一行或多行作为克隆检测的基本单元，并将通过其代码行的公共子序列的覆盖率来计算代码对的相似性的技术归类到这类。

ReDeBug[15]是第一个在 21 亿行代码查找未修补的代码克隆工具，具有高速、可扩展、低错误检测率和语言无的特点。其中高速度是利用基于语法的精确匹配算法通过复杂的数据结构实现的；语言无关性是通过执行简单的规范化、删除空白，并将所有字符转换为其小写等效字符实现的；低错误检测率通过精确匹配用数量换取质量实现。

ReDeBug 能够检测到部分 3 型克隆，但无法检测变量重命名或数据类型更改的 2 型克隆，并且在测试 Android 智能手机固件（15 MLoC）时，有 17.6%的假阳性[8]。在查准率上，ReDeBug 能实现 0 漏报，但是由于行粒度缺少足够的上下文信息，可能出现误报（当底层编译器未将易受攻击的代码检测为死代码，并且以前识别的易受攻击代码段以不可利用的方式使用时，ReDeBug 会出现误报），但通过调整参数可以降低误报率。

ReDeBug 在 Debian，Ubuntu，Linux，SourceForge 等 OS 上检测了未修补的代码克隆性能，默认设置每窗口 4 行（其中 3 行上下文），对每个窗口（连续 4～7 个 token）应用 djb2，sdbm，和 jenkins 这 3 个不同的哈希函数，然后利用 bloom 过滤器检测文件级的代码克隆。实验结果表明，Deckard 漏报的代码克隆比 ReDeBug 多 6 倍。

CCAligner[11]是大差异克隆的检测工具，采用滑动窗口中的 6 行作为基本单元，允许一个窗口的误配，利用非对称相似性技术检测克隆，在对 100 万行代码检测中，1 型－3 型克隆都有良好精确率和召回率，并且由于采用分布式计算赋予其更多改进的地方。

CCAligner 分为两个阶段。1）利用 TXL 工具提取源代码中的代码块，对代码块的每一行格式化，该阶段与编程语言相关，然后利用基于 Flex 生成的扫描仪词法分析代码块，通过规范化和标识化可以检测 1 型－2 型克隆；2）在设定的滑动窗口和编辑距离下，计算窗口的匹配率，找到满足相似性阈值的 3 型克隆。例如，滑动窗口为 6 行、编辑距离阈值为 1 行时，两个窗口的任何一个 5 行的顺序子序列的 MurmurHash 哈希值相等，则认为这两个窗口匹配，然后根据非对称相似性阈值 0.7 判定克隆，即两个代码块的匹配窗口数/较小代码块中的窗口数商大于等于 0.7，则判定两个代码块相似。在大差异克隆检测中，CCAligner 平均精确率、召回率和 F1 值分别为 86%，83%，84%，相比 SourcererCC 的 88%，19%，31%有明显优势。在通用的 1 型—3 型克隆检测中，CCAligner 与 SourcererCC，iClones，Deckard 等工具相比，也具有出色的执行时间、良好的召回率和精确度。

LVMapper[12]是 Wu 等提出的大方差克隆检测技术。该算法受第三代序列比对算法启发，收集 CCAligner 每三行连续的代码作为种子，然后利用 MurmurHash 计算每个种子的哈希值，并建立〈哈希值、块 ID〉索引，通过计算两个代码块的

共享种子数尽可能找到更多的候选克隆，即两个代码块的共享种子数除以较大块的种子总数大于等于 0.1，根据代码的保序性原理，依据较小代码块的长度设置动态分段相似性阈值验证克隆。在与 CCAligner 相同的数据集上验证了 LVMapper 大方差克隆检测的性能，LVMapper 检测到的大差异克隆比 SourcererCC，CCAligner 和 Oreo 明显增多，其中能检测到比 Oreo 多一倍的大方差克隆，在一般 1 型－3 型克隆检测中，与最先进的克隆检测工具相当。

这类技术对代码行上的修改很敏感，当对代码行进行复杂修改时可能产生漏报。

基于行的克隆检测技术的特点如下。

1）高效性：该方法只需要对代码片段按行比较，因此速度较快，适合处理大规模的代码库。

2）对代码行修改敏感：当对代码行进行复杂修改时可能产生漏报。

4.2.5 基于词汇的克隆检测性能总结

在基于词汇的克隆检测技术中选择行作为比较单元的时间复杂度最低，约为 $O(n)$，并且行能检测到相似度较低的 WT 克隆；n-gram 和行能检测 3 型克隆中普遍存在的大方差和大差异克隆的精确度在 85% 以上。图 4 给出了 4 类技术的精确率和召回率。图 4 显示，在一般的 1 型—3 型克隆检测中，选择行作为基本比较单元的技术精确率偏低，为 84.2%，其他粒度技术在 90% 以上；在召回率上，词袋法在各个克隆类型的召回率上最低；各类技术在 1 型、2 型、VST3 上的召回率都有完美的表现，几乎达到 100%，但在 ST3，MT3 特别是 WT 的检测中，召回率有待进一步提高，有待展开深入研究。

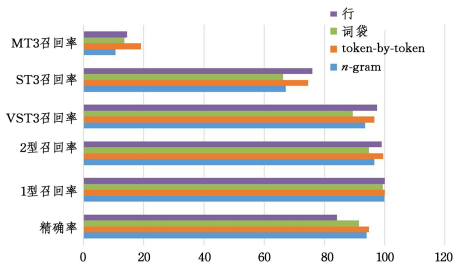


图 4 检测技术的精确率和召回率

Fig. 4 Accuracy and recall of studied works

5 基于词汇的克隆检测技术面临的挑战和未来可能的研究方向

基于词汇的克隆检测技术虽然已经被广泛应用于大规模克隆检测任务，但在多粒度、多编程语言、跨语言和大规模的实际需求下，该技术仍面临一些挑战，其中降低算法时空复杂度、提高检测 ST3，MT3 和 WT 的精确率和召回率以及代码结构语义信息表征等方面是需要解决的问题。

1）提升检测性能的可研究方向

基于词汇的克隆检测技术利用的符号序列或频率比较相似性，一般采用 LCS（最长公共子序列）、后缀树、后缀数组、哈希等匹配算法检测克隆，利用倒序索引、部分索引、过滤（长度、位置）等优化技术提高检测效率。

后缀树算法计算符号序列中的所有相同子序列，计算复杂度为 $O(n^2)$；DPM（动态模式匹配）计算两个序列之间最长的公共子序列，计算复杂度为 $O(mn)$。尽管可以使用优化技术降低时间复杂性，但仍然面临大规模系统克隆检测的挑战。目前只有 SourcererCC 和 CloneWorks 能通过单个工作站扩展到 250 MLOC 规模，因此在可扩展性方面，采用分块技术并利用 GPU 计算加速是可能的改进方法。

哈希值比较的匹配算法可以计算基本计算单元（词汇、窗口、语句）的哈希值，并利用哈希值排序，时间复杂性为 $O(n)$。通常采用长度过滤等优化技术降低时间复杂度，哈希算法中基本计算单元的选择会对技术的适用性、有效性和精确度产生影响，并且哈希和排序算法很大程度上依赖于输入语言。因此，采用层次过滤的混合方法值得进一步探索，即首先使用局部敏感哈希（LSH）找到近似克隆，然后使用特征散列过滤克隆。

在源代码不能编译或解析致使检测技术受限的情况，选择简单的测量方法（如 Kondrak 距离或 Jaccard 索引）可能是衡量代码克隆的更好选择。

2）赋予词汇丰富的语义信息提高 3 型和 4 型的克隆检测能力

传统的基于词汇的克隆检测技术不能充分利用代码的语法语义信息，虽然对 1 型和 2 型代码克隆检测的召回率几乎达到 100%，但对 3 型、4 型代码克隆检测的性能欠佳。赋予词汇丰富的结构信息和语义信息，如结合抽象语法树提取代码的结构信息和语法规则、融合 CPG（程序依赖图）抽象程序的语义信息，或融合多种代码表征方式提取代码间的复杂联系，提高对 3 型和 4 型代码克隆的检测性能是值得研究的方向。

3）利用深度学习建模提取代码结构信息

随着深度学习模型在提取语法语义结构信息中的成功应用及其可解释性研究的不断深入[42]，深度学习在代码克隆检测、漏洞检测[43]等领域的应用中也展现出了卓越的检测性能，如 CCLearner 利用它学习词汇的代码信息用于克隆检测、Russell[44]结合卷积神经网络（Convolutional Neural Networks，）和循环神经网（Recurrent Neural Networks，RNNs）提取代码特征进行漏洞检测。利用神经网络模型根据具体任务自动提取代码中所包含的高维特征，降低传统依靠人工制定特征的依赖和主观偏差，提高代码结构相似克隆检测性能，值得进一步研究。

4）进一步探索代码克隆技术在漏洞检测中的实际应用

Islam 等[45]对 34 个软件系统中 870 万行源代码进行定量分析发现，克隆中的安全漏洞比非克隆中的安全漏洞具有更高的安全风险，也有学者对缺陷检测做了相关研究：如 Yue 等[46]、Mondal 等[47]对缺陷检测的克隆进行研究；Zhu 等[48]针对提高脆弱性源代码相似度分析的准确性，提出了一种 Simhash 算法与 MD5 匹配算法相结合的解决方案，研究了传统哈希算法无法通过局部敏感哈希生成相同指纹来记录相似文件之间差异的问题，使用 Simhash 算法来补充基于 MD5 匹配的文件级同源性分析算法。虽然已有学者研究漏洞检测工具，如 VCIPR 只能检测 1 型漏洞，CP-Miner，VUDDY 能检测

1 型和 2 型漏洞，Redebug 虽然能检测到 3 型漏洞，但是错过了对 2 型漏洞的检测，提高了代码克隆漏洞检测和漏洞定位的性能，有紧迫的实际需求。

结束语 本文针对基于词法的克隆检测技术的研究现状进行了搜索，筛选出了近 5 年的相关研究文献，并对其进行了梳理；提出按照相似性算法的基本粒度将其分为 4 类，从 10 个技术特征对这 4 类研究技术进行了系统的分析，指出了每类技术的优势和不足。本文为对基于词汇的克隆检测技术在克隆检测中的效果和未来可能的研究方向感兴趣的研究者，提供了一定的参考价值。

参 考 文 献

[1] JUERGENS E, DEISSENBOECK F, HUMMEL B, et al. Do code clones matter? [C]// 2009 IEEE 31st International Conference on Software Engineering. IEEE, 2009: 485-495.

[2] SHENEAMER A, KALITA J. A survey of software clone detection techniques[J]. International Journal of Computer Applications, 2016, 137(10): 1-21.

[3] ISLAM J F, MONDAL M, ROY C K. Bug replication in code clones: An empirical study[C]// 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2016, 1: 68-78.

[4] MONDAL M, ROY B, ROY C K, et al. An empirical study on bug propagation through code cloning[J]. Journal of Systems and Software, 2019, 158: 110407

[5] MONDAL M, ROY B, ROY C K, et al. Investigating context adaptation bugs in code clones[C]// 2019 IEEE International Conference on Software Maintenance and Evolution(ICSME). IEEE, 2019: 157-168.

[6] MONDAL M, ROY C K, SCHNEIDER K A. A Summary on the Stability of Code Clones and Current Research Trends[M]// Code Clone Analysis: Research, Tools, and Practices. 2021: 169-180.

[7] MONDAL M, ROY C K, SCHNEIDER K A. A fifine-grained analysis on the inconsistent changes in code clones[C]// 2020 IEEE International Conference on Software Maintenance and Evolution(ICSME). IEEE, 2020: 220-231.

[8] KIM S, WOO S, LEE H, et al. Vuddy: A scalable approach for vulnerable code clone discovery[C]// 2017 IEEE Symposium on Security and Privacy(SP). IEEE, 2017: 595-614.

[9] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on Software Engineering, 2007, 33(9): 577-591.

[10] SVAJLENKO J, ROY C K. Bigcloneeval: A clone detection tool evaluation framework with bigclonebench[C]// 2016 IEEE International Conference on Software Maintenance End evolution (ICSME). IEEE, 2016: 596-600.

[11] WANG P, SVAJLENKO J, WU Y, et al. CCAligner: a token based large-gap clone detector[C]// Proceedings of the 40th International Conference on Software Engineering. 2018: 1066-1077.

[12] WU M, WANG P, YIN K, et al. Lvmapper: A large-variance clone detector using sequencing alignment approach[J]. IEEE Access, 2020, 8: 27986-27997.

[13] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.

[14] SAJNANI H, SAINI V, SVAJLENKO J, et al. Sourcerercc: Scaling code clone detection to big-code[C]// Proceedings of the 38th International Conference on Software Engineering. 2016: 1157-1168.

[15] JANG J, AGRAWAL A, BRUMLEY D. ReDeBug: finding unpatched code clones in entire os distributions[C]// 2012 IEEE Symposium on Security and Privacy. IEEE, 2012: 48-62.

[16] RATTAN D, BHATIA R, SINGH M. Software clone detection: A systematic review[J]. Information and Software Technology, 2013, 55(7): 1165-1199.

[17] ZHANG H, SAKURAI K. A survey of software clone detection from security perspective[J]. IEEE Access, 2021, 9: 48157-48173.

[18] CHEN Q Y, LI S P, YAN M, et al. Code Clone Detection : A Literature Review[J]. Journal of Software, 2019, 30(4): 962-980.

[19] ROY C K, CORDY J R. A survey on software clone detection research[J]. Queen's School of Computing TR, 2007, 541(115): 64-68.

[20] AIN Q U, BUTT W H, ANWAR M W, et al. A systematic review on code clone detection[J]. IEEE Access, 2019, 7: 86121-86144.

[21] MIN H, LI PING Z. Survey on software clone detection research [C]// Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences. 2019: 9-16.

[22] WALKER A, CERNY T, SONG E. Open-source tools and benchmarks for code-clone detection: past, present, and future trends[J]. ACM SIGAPP Applied Computing Review, 2020, 19(4): 28-39.

[23] KAUR A, SHARMA S, SAINI M. Code clone detection using machine learning techniques: A systematic literature review[J]. International Journal of Open Source Software and Processes (IJOSSP), 2020, 11(2): 49-75.

[24] LEI M, LI H, LI J, et al. Deep learning application on code clone detection: A review of current knowledge[J]. Journal of Systems and Software, 2022, 184: 111141.

[25] SEMURA Y, YOSHIDA N, CHOI E, et al. CCFinderSW: Clone detection tool with flexible multilingual tokenization[C]// 24th Asia-Pacific Software Engineering Conference(APSEC 2017). IEEE, 2017: 654-659.

[26] NAKAGAWA T, HIGO Y, KUSUMOTO S. NIl: large-scale detection of large-variance clones[C]// Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021: 830-841.

[27] LI Z, LU S, MYAGMAR S, et al. CP-Miner: Finding copy-paste and related bugs in large-scale software code[J]. IEEE Transactions on software Engineering, 2006, 32(3): 176-192.

[28] LI L, FENG H, ZHUANG W, et al. CClearner: A deep learning-

based clone detection approach[C]// IEEE International Conference on Software Maintenance and Evolution(ICSME 2017). IEEE, 2017: 249-260.

[29] YUKI Y, HIGO Y, KUSUMOTO S. A technique to detect multi-grained code clones[C]// 2017 IEEE 11th International Workshop on Software Clones(IWSC). IEEE, 2017: 1-7.

[30] AKRAM J, QI L, LUO P. VCIPR: vulnerable code is identifiable when a patch is released(hacker's perspective)[C]// 2th IEEE Conference on Software Testing, Validation and Verification (ICST 2019). IEEE, 2019: 402-413.

[31] LI G, WU Y, ROY C K, et al. SAGA: efficient and large-scale detection of near-miss clones with GPU acceleration[C]// 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering(SANER). IEEE, 2020: 272-283.

[32] SVAJLENKO J, ROY C K. Fast and flexible large-scale clone detection with CloneWorks[C]// ICSE(Companion Volume). 2017: 27-30.

[33] NISHI M A, DAMEVSKI K. Scalable code clone detection and search based on adaptive prefix filtering[J]. Journal of Systems and Software, 2018, 137: 130-142.

[34] GOLUBEV Y, POLETANSKY V, POVAROV N, et al. Multi-threshold token-based code clone detection[C]// 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering(SANER). IEEE, 2021: 496-500.

[35] ZHU W, YOSHIDA N, KAMIYA T, et al. MSCCD: grammar pluggable clone detection based on ANTLR parser generation [C]// Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. 2022: 460-470.

[36] WANG W, DENG Z, XUE Y, et al. Ccstokener: Fast yet accurate code clone detection with semantic token[J]. Journal of Systems and Software, 2023, 199: 111618.

[37] SVAJLENKO J, ISLAM J F, KEIVANLOO I, et al. Towards a big data curated benchmark of inter-project code clones[C]// 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, 2014: 476-480.

[38] ROY C K, CORDY J R. A mutation/injection-based automatic framework for evaluating code clone detection tools[C]// 2009 International Conference on Software Testing, Verification, and Validation Workshops. IEEE, 2009: 157-166.

[39] ISHIHARA T, HOTTA K, HIGO Y, et al. Inter-project functional clone detection toward building libraries — an empirical study on 13000 projects[C]// 2012 19th Working Conference on Reverse Engineering. IEEE, 2012: 387-391.

[40] ROY C K, CORDY J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]// 2008 16th IEEE International Conference on Program Comprehension. IEEE, 2008: 172-181.

[41] JIANG L, MISHERGHI G, SU Z, et al. Deckard: Scalable and accurate tree-based detection of code clones[C]// 29th International Conference on Software Engineering(ICSE'07). IEEE, 2007: 96-105.

[42] WAN Y, ZHAO W, ZHANG H, et al. What do they capture? a structural analysis of pre-trained language models for source code[C]// Proceedings of the 44th International Conference on Software Engineering. 2022: 2377-2388.

[43] LI Z, ZOU D, XU S, et al. SySeVR: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2244-2258.

[44] RUSSELL R, KIM L, HAMILTON L, et al. Automated vulnerability detection in source code using deep representation learning[C]// 2018 17th IEEE International Conference on Machine Learning and Applications(ICMLA). IEEE, 2018: 757-762.

[45] ISLAM M R, ZIBRAN M F, NAGPAL A. Security vulnerabilities in categories of clones and non-cloned code: An empirical study[C]// 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement(ESEM). IEEE, 2017: 20-29.

[46] YUE R, MENG N, WANG Q. A characterization study of repeated bug fixes[C]// 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017: 422-432.

[47] MONDAL M, ROY C K, SCHNEIDER K A. Bug-proneness and late propagation tendency of code clones: A comparative study on different clone types[J]. Journal of Systems and Software, 2018, 144: 41-59.

[48] ZHU C, TANG Y, WANG Q, et al. Enhancing code similarity analysis for effective vulnerability detection[C]// Proceedings of the 2nd International Conference on Computer Science and Software Engineering. 2019: 153-158.

LIU Chunling, born in 1981, master, lecturer. Her main research interests include code vulnerability detection and code similarity detection.

TANG Yonghe, born in 1983, Ph.D, lecturer. His main research interests include malware detection and classification, code similarity detection and computer security.

(责任编辑:喻藜)