一种面向功能类似程序的高效克隆检测技术

董加星 许 畅
(南京大学计算机软件新技术国家重点实验室 南京 210023)
(南京大学计算机科学与技术系 南京 210023)

摘 要 程序克隆检测被普遍应用于检测软件市场中是否有被恶意修改后重新发布的软件,或是应用于识别与重构克隆代码。但是其应用领域不仅限于此,面向功能类似程序的克隆检测有着独有的特点,同时也有着很大的应用前景。功能类似程序之间的克隆检测比一般的克隆检测稍复杂,它是在一些具有相似框架代码、实现功能类似的程序中寻找克隆程序对。现有的克隆检测技术在检测功能类似的程序时很难控制误报率。文中提出了一种改进的克隆检测方法,即通过分析功能类似程序克隆检测的特点,从中获取有用的信息,最终改进克隆检测技术并将其应用于实践。实验结果表明,该克隆检测技术可以有效地进行克隆检测,结合功能类似程序克隆检测的特点可以很好地控制误报率。

关键词 程序分析,克隆检测,技术移植
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.003

Efficient Clone Detection Technique for Functionally Similar Programs

DONG Jia-xing XU Chang
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)
(Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract Clone detection techniques are widely used for but not limited to maliciously modified software detection, code recognition and reconstruction, etc. However, it is difficult to precisely detect code clones in functionally similar programs since they have their own unique features. Existing clone detection techniques have high false positive rates when applied to functionally similar programs. In this paper, we proposed a novel clone detection approach. After analyzing the features of functionally similar programs, we improved the clone detection technique's performance. The experimental results show that our technique can effectively control the false positive rate when conducting the clone detection.

Keywords Program analysis, Clone detection, Technique transplantation

1 引言

克隆检测有着很多不同种类的检测技术和应用场景[1-2]。现在主流的应用场景是检测软件市场中的恶意软件和识别与重构克隆代码。这两个应用场景的主要特征是,从众多功能不同的程序中找出有克隆特征的程序对或代码块。面向功能类似程序的克隆检测是要在功能类似的程序中找到克隆程序对。现有的程序克隆检测技术虽然可以对功能类似的程序进行检测,但是检测结果的误报率往往很高,这是由功能类似程序独有的特点决定的。

本文则利用这些特点来改进克隆检测技术,并将其用于检测功能类似的程序。主要贡献如下:

1)提出了一种克隆检测方法,用于处理面向功能类似程序的克隆检测;

2)针对功能类似程序的特点,在特征提取和阈值设定方面对一种基于度量值的克隆检测技术进行了改进;

3)原技术是用于检测 Android 应用的,现用另一种完全不同的实现方式将其移植应用于检测 C 语言程序。

本文第 2 节介绍功能类似程序克隆检测的特点;第 3 节针对功能类似程序克隆检测的特点,介绍如何改进克隆检测技术;第 4 节是对克隆检测工具实现的说明以及对实验的设置和结果的分析;第 5 节介绍相关工作;最后对整个工作进行相应的总结。

2 功能类似程序克隆检测的特点

功能类似程序的克隆检测有以下几个特点:

到稿日期:2015-11-30 返修日期:2016-02-27 本文受国家"九七三"重点基础研究计划(2015CB352202),国家自然科学基金(61472174,61690204)资助。

董加星(1993—),男,硕士生,主要研究领域为软件测试与分析,E-mail:sydongjx@163.com;许 畅(1977—),男,博士,教授,CCF 高级会员,主要研究领域为大数据软件工程、软件测试与分析、自适应与嵌入式系统,E-mail:changxu@nju.edu.cn。

1)功能类似的程序往往会有相似的代码框架，在实现功能的过程中功能性函数的名字会相似；

2)功能类似的程序往往不可能只有一两个版本，一般之前就会有很多已经实现的程序版本；

3)面向功能类似程序的克隆检测往往需要检测技术才能够及时地得到检测结果并反馈相应信息。

在处理好这些特征并合理利用后，克隆检测可以达到更快速、准确的效果。

3　克隆检测技术及其改进

在分析了功能类似程序克隆检测特点的基础上得到了一些改进克隆检测技术的启发。以 Centroid[3] 作为检测工具的雏形，对其进行改进，使其能够很好地处理功能类似的程序的克隆检测。

3.1　Centroid 方法总述

Centroid 是一种自底向上的基于度量值的克隆检测技术，它主要分 3 个阶段进行。首先，解析程序中所有的函数，提取函数控制流图中基本块的信息并转换成基本块坐标，根据坐标和控制流图中节点间边的关系，计算得到函数的质心向量。其次，根据得到的质心向量比较函数间的相似度。根据向量归一化距离的远近来判断两个函数的相似度。设定函数层面的阈值 δ 来判断两个函数间是否存在克隆关系。最后，根据得到的函数间的相似度判断程序间是否存在克隆关系。设定程序层面的阈值 Δ，当两个程序间的可疑函数占该程序所有函数的比例超过 Δ 时，就判定这两个程序之间有克隆关系。

3.2　Centroid 的不足之处

Centroid 可以满足克隆检测速度性能的需求，但是它仍存在局限性。一方面，它在处理功能类似程序时会有很多的误报；另一方面，它采用静态阈值的方法来设置阈值，这样的设置不利于功能类似程序的检测。

3.3　改进克隆检测技术

主要从特征提取和阈值设定两个方面来对克隆检测技术进行改进。

3.3.1　特征提取

在 Centroid 中，基本块坐标定义的本质是控制流图基本块中特征的提取。对于功能类似的程序的克隆检测，特征提取若过少则容易导致更多的误报。对此，提取了基本块中更多的有用信息。

定义 1(基本块坐标)　函数控制流图中每个基本块都有各自的坐标，该坐标用向量$\langle x,y,z,l,c,\omega\rangle$来表示。其中 x 是函数控制流图中节点的序号，y 是函数控制流图中节点的出边数目，z 是函数控制流图中节点所处的循环深度，l 是函数控制流图中节点所处循环的节点个数，c 是函数控制流图中节点所对应的基本块语句中 call 指令的个数，ω 是函数控制流图节点所对应的基本块中语句的数目。

记录 call 指令的个数间接地提取了函数间的调用关系。这样的特征提取，可以有效地降低误报率，而且漏报率不会有明显的提升。

3.3.2　阈值设定

阈值的设定直接决定着克隆检测技术的准确程度。函数层面阈值 δ 是闭区间[0,1]中的某个值，它表明了函数间的区分程度，若两个函数质心向量距离小于 δ，则这对函数对被认为是克隆函数对，否则就认为这对函数间无明显的克隆关系。δ 太大，则会造成很多误报；δ 太小，则会导致更多的漏报。程序层面阈值 Δ 也是闭区间[0,1]中的某个值，它用于表征程序间的相似程度，当两个程序的相似度超过 Δ 时，则认为这两个程序是克隆程序对，否则就认为这两个程序之间没有明显的克隆关系。程序层面阈值 Δ 的值直接影响技术检测克隆程序的能力。这使得我们需要选取合适的 δ 和 Δ，以使误报率和漏报率都在一个可控制的范围内。

函数层面的阈值 δ 是用来衡量两个函数是否有克隆嫌疑的。根据功能类似程序的特点 1)，可以将框架代码的函数排除在外考虑。对于功能性的函数，当函数名相似时，阈值 δ 根据函数名的相似程度做出动态的调整，从而使得检测更为准确。

算法 1　函数层面比较

输入：两个函数名 n_1 和 n_2 及其对应的质心向量$\vec{c_1}$和$\vec{c_2}$，框架代码函数名集合 frames，基础阈值 δ，系数 coef

输出：T/F

```
1. if n1∈frames ‖ n2∈frames then
2.     return F
3. ed=editDistance(n1,n2)
4. nd=normalizedDistance(c1⃗,c2⃗)
5. dt=(1−ed/(length(n1) + length(n2)))
6. if nd<δ+dt×coef then
7.     return T
8. return F
```

若输出 T，则表示两个函数间有克隆关系；若输出 F，则表示没有。第 1 行就是处理框架代码的情况，将框架代码排除在外考虑可以极大地降低误报率。editDistance 函数是两个函数名的编辑距离，normalizedDistance 是向量每个维度归一化距离的最大值。第 5 行是动态阈值 dt 的设定，两个函数名相似时，它们实现的功能很有可能相似。为防止人为干扰的误导，将阈值调整得稍大。第 6 行根据基础阈值加上附有比例系数的动态阈值，最终决定两个函数是否有克隆关系。

程序层面的阈值 Δ 是用来衡量程序间是否有克隆关系的。根据功能类似程序的特点 2)，可以预先对已有版本程序进行检测，得到程序相似度的分布，再结合以往的一些 oracle，确定一个合适的阈值区间，最后根据阈值区间对需要进行克隆检测的程序进行检测，这样可以使检测的结果更加准确。

4　实现及实验分析

实验的样本取自《编译原理》实验课程。每次实验取出两组实验代码进行评估，一组作为训练组，另一组作为实验组。实验机器为 Dell Inspiron 14R Turbo，拥有 CORE i5 2.5GHz 的处理器和 6GB 内存。

4.1　工具实现

实现了一个可以应用于功能类似 C 语言程序克隆检测的原型工具。整个工具的系统框架如图 1 所示。

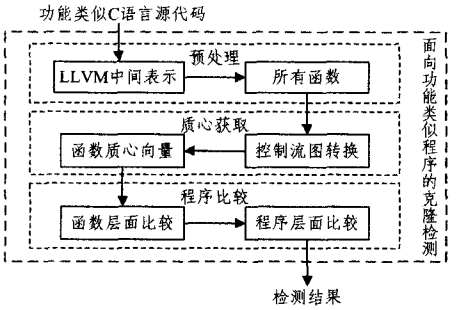

图 1 系统框架

系统的整个框架从功能类似 C 语言源代码出发，通过面向功能类似程序的克隆检测技术得到检测结果。整个检测过程分为预处理、质心获取和程序比较 3 个部分。这个原型工具是由 1000 多行 C++代码和 200 多行 python 代码实现的。

预处理阶段是将 C 语言源代码转换成一种中间表示。通过包含 LLVM 和 Clang 处理命令的 python 脚本来重新编译 C 语言代码，获得包含程序中所有函数以及程序指令的 LLVM 中间表示文件。使用 LLVM 和 Clang 对源文件进行重新编译而不是直接解析可执行文件，是为了防止编译器的过度优化对克隆检测造成影响。

质心获取阶段是根据 LLVM 中间表示文件中的信息得到所有函数的质心向量。解析中间表示文件，提取函数控制流图中所有边的信息，接着按照基本块坐标的定义方式给函数控制流图中的节点编码，最后根据函数控制流图中节点的坐标和边之间的关系获取函数的质心向量。将程序编号，并记录程序中所有函数名及其对应的质心向量。当有新的程序时，只需计算出程序中所有函数的质心向量并将其添加到记录中即可。

程序比较阶段是根据函数的质心向量将程序进行比较得到最终的检测结果。根据设定函数层面动态阈值和分析训练组得到的函数层面阈值和程序层面的阈值区间，将一个程序中的每个函数和其他程序中的所有函数进行比较，从而得到结果。

4.2 实验及分析

4.2.1 实验主体和步骤

实验主体是两组实验代码。在进行实验时，根据训练组代码和相应的代码克隆检测报告，得到一些初步的信息，根据这些信息排除框架代码并合理地设定函数层面阈值和程序层面阈值。

《编译原理》实验是在 flex 和 bison 两个工具的基础上完成的，这使得实验有了一个相似的框架基础。将 flex 和 bison 自动生成的函数排除在外考虑，函数名极其相似的函数间阈值的设定根据编辑距离做相应的调整。通过对训练组进行函数层面的检测分析，函数层面的基础阈值 δ 设置为 0.010，动态阈值系数 $coef$ 设置为 0.008。这样的设置可以有效地检测函数间的克隆关系。

程序层面阈值 Δ 的设定需要参考训练组程序层面检测结果的分布情况。对训练组进行分析，根据相似度分布规律和克隆检测报告中的 oracle，程序层面阈值的取值区间为 [0.850,0.900]。在课程实验的检测中，还会人工核对检测到的程序，结果漏报的后果比误报要严重。所以，取最低的阈值 0.850 作为程序层面的阈值。检测结果阈值超过 0.850 的有 20 对程序对，其中有 6 对程序对是确实有克隆关系的。对相似程度在 0.850 以下的几组也进行人工比较和鉴定，没有发现明显的克隆程序对。以上结果直接验证了阈值设定的合理性和方法的有效性。

4.2.2 性能分析

衡量一个克隆检测技术是否有效的一个重要标准是它能否准确地找出程序克隆对，尽量减少漏报和误报。要了解程序克隆检测漏报的情况，则需要知道我们克隆检测技术无法找到的克隆程序对的数量。我们无法预先知道实验组中代码的克隆关系，所以漏报率很难衡量。我们的工具检测出了包含部分已知克隆关系的训练组克隆检测报告中报告有克隆关系的程序版本以及报告中漏报的克隆案例。

表 1 列出了我们的工具在控制误报率方面的性能。程序对的数目是由实验组程序数量决定的，每个版本的程序和其他程序一一成对。误报率＝(疑似的克隆对－确认的克隆对)/程序对总数。其中疑似的克隆对是指程序间相似度超过阈值 Δ 的程序对，确认的克隆对是经人工比较后，核实有克隆关系的程序对。根据表中数据可知该工具的误报率都在 0.015 以下，表明该工具能够有效地对功能类似的程序进行克隆检测。

表 1 克隆检测工具误报率分析

实验编号	程序对总数	疑似的克隆对	确认的克隆对	误报率
1	1806	26	4	0.012
2	2352	20	6	0.006
3	1806	32	6	0.014
4	1482	16	8	0.005
5	2484	14	0	0.006
6	2508	10	2	0.003

根据功能类似程序的特点 3)，克隆检测技术在速度性能方面是有一定要求的。每份实验组大概有 30～50 份实验程序，每份程序平均由 8000 多条指令构成，平均包含 60 多个功能性函数。通过人工的方法从这些程序中找出克隆程序对显然是不合时宜的，我们用面向功能类似程序的克隆检测技术来完成这个任务。我们的工具预处理某次实验的 39 份代码的耗时为 20s，相互比较耗时为 2min，这样的速度完全能达到克隆检测快速高效进行的要求。

4.2.3 对比实验

对原技术进行改进使其更好地适用于检测功能类似程序，目的是在保证克隆检测高效进行的前提下降低检测功能类似程序时的误报率。从误报率和时间开销两个方面将改进后的方法和原技术作对比，结果如表 2 所列。

表 2 Centroid 原技术和改进后技术的比较

实验编号	原技术		改进后	
	误报率	时间开销/s	误报率	时间开销/s
1	0.021	151	0.012	154
2	0.018	203	0.006	211
3	0.018	155	0.014	162
4	0.012	111	0.005	120
5	0.013	206	0.006	219
6	0.017	219	0.003	232

从表 2 中的数据可知，对原技术的改进是成功的。改进方法略微牺牲了时间开销，在保证克隆检测快速高效进行的前提下，误报率得到了显著的下降。需要说明的是，原技术找到的确认的克隆程序对，改进后的方法也都找到了，不存在原技术找到而改进后的方法漏报的情况。

5　相关工作

基于文本的检测[4-6]是从源代码的文本结构及词法的角度进行克隆检测。这种检测不需要使用重量级的语法分析器，具有良好的时空效率，但是其准确率不是很理想。

基于语法树的检测[7-8]是用静态分析工具把源代码构造成抽象语法树，然后在语法树上寻找相似子树进行克隆检测。基于语法树的检测有很大的时空开销。

基于图的检测[9-10]是分析语法结构、数据流以及调用关系等，构造出整个程序依赖关系图，然后寻找相似的子图。这个过程时空开销很高且效率不高。

基于度量值的检测[3,11]需先设定源代码的比较单元（类、函数等），然后提取比较单元中的一些特征并计算一系列的度量值，最后通过比较度量值进行克隆检测。该类检测方法普遍具有良好的时空效率。

结束语　本文提出了一种面向功能类似程序的高效克隆检测技术。我们根据功能类似克隆检测的特点，对克隆检测技术进行了改进，从而使得克隆检测更加快速，检测结果更加准确。本文实验样本取自真实的课程编程实践，实验结果表明，改进后的克隆检测技术可以快速、高效地检测功能类似程序，结合功能类似程序克隆检测的特点可以有效地控制误报率。

参 考 文 献

[1] ROY C K, CORDY J R, KOSCHKE R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach[J]. Science of Computer Programming, 2009, 74(7): 470-495.

[2] GU T X, CAO C, XU C, et al. Low-disruptive dynamic updating of Java applications[J]. Information and Software Technology, 2014, 56(9): 1086-1098.

[3] CHEN K, LIU P, ZHANG Y J. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets[C]// Proceedings of the 36th International Conference on Software Engineering. 2014: 175-186.

[4] 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.

[5] LI Z M, LU S, MYAGMAR S, ZHOU Y Y. CP-Miner: finding copy-paste and related bugs in large-scale software code[J]. IEEE Transactions on Software Engineering, 2006, 32(3): 176-192.

[6] HITESH S, VAIBHAV S, CRISTINA L. A parallel and efficient approach to large scale clone detection[J]. Journal of Software: Evolution and Process, 2015, 27(6): 402-429.

[7] JIANG L X, MISHERGHI G, SU Z D, et al. DECKARD: Scalable and accurate tree-based detection of code clones[C]// Proceedings of the 29th International Conference on Software Engineering. 2007: 96-105.

[8] RAINER K. Large-scale inter-system clone detection using suffix trees and hashing[J]. Journal of Software: Evolution and Process, 2014, 26(8): 747-769.

[9] LIU C, CHEN C, HAN J W. GPLAG: detection of software plagiarism by program dependence graph analysis[C]// Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2006: 872-881.

[10] QIU J, SU X H, MA P J. Library functions identification in binary code by using graph isomorphism testings[C]// IEEE 22nd International Conference onSoftware Analysis, Evolution and Reengineering. 2015: 261-270.

[11] KAUR M, LAL M. Code clone detection using function based similarities and metrics[J]. International Journal of Emerging Research in Management and Technology, 2015, 4(7): 156-159.

（上接第 7 页）

[24] HABER S, STORNETTA W S. How to time-stamp a digital document[J]. Journal of Cryptology, 1991, 3(2): 99-111.

[25] BAYER D, HABER D, STORNETTA W S. Improving the efficiency and reliability of digital time-stamping[M]. New York: Springer New York. 1993: 329-334.

[26] Wikipedia. 区块链[EB/OL]. [2016-10-21]. https://zh.wikipedia.org/wiki/区块链.

[27] Wikipedia. Blockchain (database) [EB/OL]. [2016-9-15]. https://en.wikipedia.org/wiki/Blockchain_(database).

[28] 巴比特. 区块链是什么[EB/OL]. www.8btc.com/what-is-blockchain.

[29] SWANM. Blockchain: Blueprint for New Economy[M]. USA: O'Reilly Media Inc, 2015.

[30] ANTONOPOULOS A M. Mastering Bitcoin[M]. USA: O'Reilly Media, 2014.

[31] BUTERIN V. Ethereum: A next generation smart contractand decentralized application platform [EB/OL]. https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[32] HYPERLEDGER. About the hyperledger project[EB/OL]. https://www.hyperledger.org/about.

[33] HYPERLEDGER. Projects[EB/OL]. https://www.hyperledger.org/community/projects.

[34] WIKIPEDIA. Smart contract[EB/OL]. https://en.wikipedia.org/wiki/Smart_contract.

[35] CASSANO J. What are smart contracts?cryptocurrency's killer app[N/OL]. Fastcompany, 2014-09-17 [2016-10-23]. https://www.fastcompany.com/3035723/app-economy/smart-con-tracts-could-be-cryptocurrencys-killer-app.

[36] BROWN A R. A simple model for smart contracts[EB/OL]. https://gendal.me/2015/02/10/a-simple-model-for-smart-contracts.

[37] WILKINSON S, BOSHEVSKI T, BRANDOFF J, et al. Storj: A peer-to-peer cloud storage network(V0.2)[EB/OL]. [2016-11-15]. https://storj.io/storj.pdf.