代码相似性检测技术:研究综述

熊 浩[1,2] 晏海华[1] 郭 涛[2] 黄永刚[2] 郝永乐[2] 李舟军[1]
(北京航空航天大学计算机学院 北京 100191)[1] (中国信息安全评测中心 北京 100083)[2]

摘 要 程序代码的相似性检测是使用一定的检测手段度量程序代码间的相似程度,其对于提升高等教育中计算机课程教学效果和保护软件知识产权都有着重要的意义。介绍了代码相似性检测技术的研究意义和发展历程,阐述了本领域研究过程中的概念模型,深入分析了已有的几类代码相似性检测技术,总结了这几类技术各自的特点,同时探讨了一些相关研究,最后归纳了目前研究中的问题并展望了本领域研究的发展趋势。

关键词 代码相似,自动检测,概念模型
中图法分类号 TP311 **文献标识码** A

Code Similarity Detection: A Survey

XIONG Hao[1,2] YAN Hai-hua[1] GUO Tao[2] HUANG Yong-gang[2] HAO Yong-le[2] LI Zhou-jun[1]
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)[1]
(China Information Technology Security Evaluation Center, Beijing 100083, China)[2]

Abstract Identifying program code similarity is to measure similar degree between codes with a kind of detection method, which is very important in the fields of computer science education and intellectual property protection. We introduced the research purpose, the history and some conceptual models of code similarity detection. By analyzing several approaches of code similarity detection, the characteristic of each method was summarized. Then we discussed some code similarity based researches. Finally, the conclusion of the problems in current technologies was given before the discussion about some promising tendency of code similarity detection.

Keywords Code similarity, Automatic detection, Conceptual model

1 研究意义与发展历程

1.1 代码相似性检测的研究意义

相似性检测(Similarity detection)也称为抄袭检测(Plagiarism detection),或副本检测(Duplicate detection),可应用于知识产权的保护和信息检索等方面[9]。所谓相似性检测,就是判断一个文件的内容是否与其他某个或多个文件相似。相似不仅仅只是文件的复制和少量文件布局的修改,还包括同义词替换以及说法重述等方式。

相似性检测在文档方面又分为两类[9,34]:代码相似性检测和自然语言文本相似性检测。这两类检测技术有一些共性和类似之处,有些检测工具能够同时检测代码与文本的相似性。但二者仍存在明显的不同:自然语言格式较为随意,结构特征一般不明显,所表达的语义存在一定的模糊性;程序语言的语法非常规则,结构特征很明显,抄袭所付出的代价较低,通过编辑器改变外观并不影响程序的正常运行。

代码相似性检测是从重复代码检测和代码优化技术演变而来的[1-3]。重复代码通常是指软件系统中存在的相同或相似的代码片段,本质上是一种简单的代码复用机制。这种复用机制是非正式和不可控的,会带来一系列的危害[4,5]。软件系统中的重复代码大多是出于需要而必须频繁调用的,这不可避免地增加了程序代码的容量和程序运行的时间,如何有效地降低这些软件的开销正是代码优化的研究目标。现今,软件的发展达到了一个空前的规模。在复杂如大型软件系统,简单如程序设计课程的编程作业中,抄袭、雷同现象时有发生[6-8]。由此可见,研究代码相似性检测技术及相应的自动化工具具有重要的实际应用价值。

1.2 发展历程

代码相似性检测技术的研究工作已经有40多年的历史,其发展历程主要经过了两个阶段。早期的研究主要基于属性计数的方法,其基本原理是从源代码中抽取各种属性度量元作为相似性评判的依据:Ottenstein[11]使用Halstead软件科学参数[10]来检测Fortran程序的相似性。Berghel[12]在此基础上提出了更多的度量元来改善Fortran代码的检测效果。Grier[13]和Faidhi[14]分别使用了20和24种统计特征来检测代码的相似性。

到稿日期:2009-09-14 返修日期:2009-12-29 本文受国家863项目(2007AA010302,2008AA012114),国家自然科学基金(60703057,60573084)资助。

熊 浩(1985-),男,硕士生,主要研究方向为数据挖掘、软件工程等,E-mail:xionhao2005@cse.buaa.edu.cn;晏海华(1964-),副教授,主要研究方向为软件工程、软件测试等;郭 涛(1974-),博士,主要研究方向为信息安全、安全测试等;李舟军(1963-),博士,教授,博士生导师,CCF高级会员,主要研究方向为高可信软件、信息安全、数据挖掘与文本挖掘技术。

属性计数方法的优点是与所用的程序设计语言无关，无需分析代码的语法结构，实现较为简单。但是缺点也非常明显，属性计数是一种全局代码度量的模式，不能分析部分程序段的抄袭。而且 Verco[15] 认为仅仅使用属性度量的检测方法，增加向量维数并不能改善检测效果。此后，学者在研究过程中主要考虑的是代码的结构信息，我们称之为基于结构度量的相似性检测方法。

基于结构度量的方法主要是分析程序的结构信息以及执行流程。Donaldson[16] 最早在抄袭检测系统中考虑了语句的顺序，并且将源程序按语句出现顺序进行特征化，Plague 工具[17] 不仅比较了更加详细的代码结构信息，还首次将代码相似性检测方法分为了两个阶段：代码的格式转化和相似度比较算法。MOSS[18]，JPlag[19]，SIM[20] 和 YAP 系列[21,22] 都是当前比较著名的代码相似性检测工具，它们对代码进行度量时都考虑到了结构信息，在具体算法上各有特色。其中 MOSS 和 YAP3 利用程序相似性检测的方法，既可检测程序也可检测文本的相似性。YAP 系列主要包括 YAP1[21]，YAP2[21] 和 YAP3[22] 3 个版本。YAP3 的主要思路是对代码进行预处理后使用 RKR-GST 算法度量。JPlag 优化了 YAP3 的性能。SIM 工具的检测过程与 Plague 大同小异，只是在相似度比较算法上采用了 DNA 串匹配的 Local Alignment 算法。除了上述的两类方法外，还有学者提出了新的检测方法。比如，基于案例推理的 Cogger 工具[23]、基于 PDG 图的 GPLAG 工具[24]、基于神经网络的 Plague Doctor[25]、基于优化编译与反汇编的 BuaaSim[26] 和基于 XML 框架的 XP-Dec[27] 等等。

目前只有 JPlag 和 MOSS 提供了程序相似性检测的网上服务[28,29]。关于这些服务和功能的评测可参见文献[30]。

2 概念模型

2.1 代码相似性的定义

目前还缺乏一致公认的代码相似性的权威定义。早期的研究者认为："如果一份代码由另一份代码通过少量的常规变换手段得到，此两份代码就是相似的"[31]，这里所说的少量是指所付出的代价较少，而常规手段是指不需要理解源代码的文本置换。但随着计算机技术的发展，文本编辑器可以对代码进行大规模的变换，抄袭手段也不再局限于常规手段，逐渐出现了很多高级的代码抄袭方式。文献[19,24,32]认为："两份代码满足一些抄袭变换方式，就可以认为是相似的"。

2.2 抄袭手段研究

通过对代码相似性检测技术的研究，我们发现和总结出如下的抄袭手段：

1) Faidhi 和 Robinson 的代码变换光谱图[14]，将代码抄袭可能采取的手段分为：L0. 未更改、L1. 修改注释、L2. 修改标识符、L3. 调整变量位置、L4. 过程组合、L5. 变换程序语句、L6. 等价控制逻辑。L0－L6 抄袭所付出的代价和难度不断提升。

2) Joy 的两类抄袭方式归纳[33]：一为词法变换，如代码注释变换、格式变换、修改变量名和行数修改。二为结构变换，如 Loops 语句替换、等价语句和操作符变换以及不改变执行序列的语句调整。

3) Whale 的抄袭手段举例[34]：转换注释、格式、变量名、操作符的顺序、数据类型，等价语句置换，冗余代码添加，调整时序无关语句的顺序，循环结构、选择结构调整，取消函数调用，声明无关变量，重构代码片段。

4) Jones 总结了代码相似中常出现的 10 种抄袭手段[35]，根据所付出的努力从易到难为：完整拷贝、修改注释、重新排版、标识符重命名、更改代码块顺序、调整代码块里语句的顺序、更改表达式中操作符和操作数的顺序、更改数据类型、添加冗余代码、控制结构的等价变换。

此外，还有学者在研究过程中关注了跨语言抄袭[36]、生成新函数[37]、常量替换和表达式拆分[26] 等抄袭手段。

目前，还没有见到关于代码抄袭手段的精确分类。一般来说，可以从代码布局、语法的等价变换和语义的等价变换 3 个方面来对代码抄袭手段进行分类。上面列出的抄袭手段大多集中在代码布局和语法变换两个层次。语义层面的等价变换对代码外观的影响很大，而且由于程序状态的组合爆炸问题和程序的停机性不可判定[5]，没有任何方法能够完全检测出语义等价的相似代码。

2.3 代码相似性检测框架

Whale[17] 在研究过程中首次提出将检测过程分为两个阶段：代码格式转换和相似度确定。后来的很多检测方法都参考了这一框架，并将整个代码相似性检测的过程分为 4 个部分：预处理、中间代码转换、比较单元生成和匹配算法。预处理过程的目的是剔除一些与程序相似性无关的信息，尽可能地屏蔽一些简单的抄袭手段对代码外观的影响。其操作通常包括：统一代码的布局，消除代码中一些无关字符（如空行、注释行）等[30,32]。程序代码转换，是在不改变计算机程序语言的逻辑结构前提下，将程序代码转换成其他形式，以提供方便的、快速的程序比较。一般采用程序语言分析器[18,20,24,44,45]、编译器[19,21,22,26,36,38,39,43,50,53] 和自定义规则[27,42,51] 3 种方式来转换源代码文件。中间代码的形式有规格化文本（标识符或规格的符号表示）、树型结构（抽象语法树或 XML 文档）、图结构（程序结构与数据流分析）等。然后将中间代码形式切分成细粒度的单元与其他的单元进行相似性比较，这些切分后的单元一般称为比较单元。常见的比较单元有词（Token）[36,50,52]、字符串（String）[20,46,47,48]、特征向量（Vector）[25,41,53]、树结点（Node）[39,41]、子图结构（Graph）[24,44,45] 等。依据不同形式的比较单元，采用不同的相似性匹配算法来分析比较单元中的相似现象，如：串比较、神经网络、图匹配和子树匹配方法等等。

预处理的目的在于剔除一些与程序代码无关的信息。中间代码的转化需要尽可能地消除抄袭手段所带来的噪音，以提高检测效果。切分比较单元需要选用合适的粒度（固定或自由粒度）。固定粒度的值太大，会导致无法检测小于这个粒度的相似现象；粒度太小，检测结果会失去意义。而自由粒度最主要的问题是可能不合理地跨越程序的语法边界。

3 代码相似性检测技术的分析和比较

3.1 基于树型结构的检测

文献[39-41,43]等都使用工具获得源代码的抽象语法树结构。文献[39]直接在语法树上寻找相似子树，使用公式 $\text{Similarity}=2*S/(2*S+L+R)$ 计算语法树的相似度。S 为共有的结点，L，R 分别两树中的不同结点。文献[43]使用切

片技术分割语法树，并且在文献[39]的研究基础上提出了改进的相似度计算公式。文献[40]检测代码相似性的方法是匹配马尔科夫模型。而文献[41]的基本方法是定义了9维特征：[标识符、常量、赋值、自增/减、数组、条件、表达式、声明、循环]来表示语法树中每一个结点，父结点的值是其所有子结点的值和本身特征向量的矢量和，最后使用LSH算法度量源代码的相似性。此外，XPDec[27]及其改进版EXPDec[42]都是将源代码转变成XML文档，从中提取特征矩阵用于相似性的计算。

上述的检测方法有原树[27,39-42]和树结构串型化[43]两种操作模式。而相似匹配的基本单元有树[27,40,42,43]和单个结点[39,41]两种结构。遗憾的是，以树为匹配单位会丢失一些信息，同时产生一些无用的信息，单个结点匹配会忽略树的结构特征，从而降低抄袭检测的效果。而且，对于有 N 个结点的语法树，逐一地对子树进行比较需要 $O(N^3)$ 的计算时间。一般来说，一行程序语句平均会生成10个左右的抽象语法树结点，那么子树比较的时间复杂度为 $O(L^4)$（L 为代码的语句数）。显然，这样的方法无法处理大型的程序。

3.2 基于图的检测

基于图检测方法的基本思路与基于树型结构的方法类似。通过分析源代码的语法结构以及函数调用关系、控制依赖关系、数据流等，构建程序的依赖关系图（PDG），匹配图中的结点，由这些结点组成的连通图被称为相似子图，由此判断代码的相似性。文献[44]使用传统的程序依赖关系图，文献[45]充分考虑了抽象语法树和程序依赖关系图的关系，将传统的PDG划分成更小的比较单元，提高了检测的效果。GPLAG[24]根据PDG图建立并优化了抄袭检测空间，然后使用形式化的子图相似方法计算源代码的相似性。

在该种检测思路中，图的结点和连接关系体现了程序的语法，而数据流可视为程序语义的抽象表达，因而具有较好的抄袭检测能力，且有可能检测到文本结构完全不同的抄袭代码。但是，通过静态分析工具建立代码的PDG图所耗费的代价很高。图比树结构更为复杂，在其上寻找相似的子图的时空复杂度很高，故检测效率低。

3.3 基于串的检测

Dup工具[46]以源代码行作为比较单元，并构建比较单元的后缀树模式来计算相似度。其最大的问题在于忽略了代码空间布局的调整，检测能力不足。DupLoc[47]解决了这一问题，首先使用代码布局器对源代码进行处理，再使用一种基于字符串的动态模式匹配算法来分析源代码行的相似性。

Sim[20]通过标准的词汇分析器将源程序转化为解析树模式，而后切分成一个个模块作为相似性比较的基本单元。比较算法方面使用DNA串比较方法获得抄袭评判的量化值。而Bandit工具[48]是将程序代码分解成词汇标识符串、独立的程序结构、注释和变量名几部分，通过Dotplot方法查找相似的标识符号序列来获取相似度值。结果反馈是源程序中匹配序列的长度和位置，而不是上述工具的相似值结果。

基于串的检测是从源代码的文本结构及词法的角度去度量程序的相似度，不需要使用重量级的编译工具。方法支持多种编程语言甚至是纯文本文件的相似性检测。基于串的方法较树或图的检测思路具有更佳的时空效率，适用于检测大型软件系统的抄袭现象。

3.4 基于标识符的检测

文献[17-19,21-23,36,38,49-52]都是典型的基于标识符检测代码相似性的方法。而在标识符的生成、操作和匹配等方面各有特色。CCFinder[49]首先通过一系列的预处理（词法分析、针对特定编程语言的冗余信息消除等）将源代码转化成统一的特殊符号，而后把所有的符号构造成后缀树，使用时间复杂度仅有 $O(n)$（n 为树中符号的个数）的高效后缀树算法寻找程序中相似的片段，检测结果具有较高的召回率。文献[38]采用的方法与CCFinder类似，能够更加精确地定位相似代码，而且具有发现完整语法结构相似代码的功能。Plague工具[17]创建目标代码的标识符序列后用变体的最长公共子序列算法对其进行比较。YAP系列[21,22]是在Plague的基础上开发的，目的是解决Plague工具执行效率不高的问题。实验结果表明，YAP在不牺牲检测精确度的前提下，大大地提高了效率，而且具有较好的移植性。Jplag[19]在互联网上提供在线的程序代码抄袭检测服务，是目前较为流行的检测工具。使用RKR-GST算法来度量标记字符的相似程度。实验结果表明，Jplag较其他检测工具更为出众。文献[18,36,50,51]在对程序的转化字符操作中都考虑了切片技术（N-Grams）。一方面便于快速索引相似的字符片段，另一方面提高了检测的效率（任何对局部程序的改变只会影响很少的切片序列）。此外，基于案例推理的Cogger[23]和基于聚类的PDetect[52]都把程序设计语言中的关键字作为核心的程序属性（Metric）来判定程序之间的抄袭。

基于串的检测和基于标识符的检测在本质上并没有太多的区别，均可归结为基于文本的检测方法，可在自然语言文档复制检测的研究中找到雏形[9]。一些方法[17-19,21-23,51]在代码转化过程中同时涉及到了标识符和串结构，仅仅是研究者对于单个标识符在相似性判定的重要程度方面有所区别而已。基于标识符的检测无须依赖功能强大的编译器，检测算法时间复杂度低，如文献[38]使用普通的PC机比较177081行源代码的Antl.6.2工具，仅需要94385ms的运行时间。

3.5 基于度量值的检测

基于度量值的代码相似性检测思路，是用一系列的度量值代替源代码进行抄袭判定。该类检测方法由于没有对源代码进行转换，因此可以方便地定位源代码中的相似片段。早期检测抄袭的属性计数方法就是利用程序的某些度量值来分析程序中的相似片段[10-14]。但是由于检测效率不高，基于度量值的检测思路逐渐向两个方向发展。一是沿用属性计数的基本方法，充分考虑程序的结构信息。该种检测思路最早运用到文献[16]的检测系统中。后续工作加入了函数调用关系、扇入扇出系数、McCabe圈复杂度等与结构相关的信息；二是将交叉学科诸如数据挖掘、信息论的有关研究成果应用于基于度量值的代码相似性检测中[25,53-55,36,56-58]。比如Plague Doctor[25]试图将比较代码集分成相似类和非相似类，提取比较程序对的12个度量值并组成特征向量，输入至BP神经网络中进行训练。通过反复调节学习参数降低训练误差，从而使分类器具有相似检测能力。但该类方法的限定条件较多，检测效果的好坏无法判定，其具体实施还有待进一步测试。

对于上述各类代码相似性检测方法，其技术特点可用表

1概括。

表1 代码相似性检测技术分类

分类	源代码的中间表示	匹配算法	优点	缺点
基于树型结构(Tree-structure)	抽象语法树(AST)[39-41,43] XML树型文档[27,42]	计算公式[27,39,42,43] 信息距离[41,42] 模式匹配[40]	考虑了程序的语法结构信息，对于抄袭手段有良好的检测能力，检测结果较准确。	建立树结构的代价高，寻找树相似的匹配算法复杂度高，时空效率低，较难运用于大型软件的抄袭检测。
基于图结构(Graph-structure)	程序依赖关系图(PDG)[24,44,45]	图匹配[24,44] 后向切片[45]	综合考虑程序的语法和语义特征，对于抄袭手段有最好的检测能力，检测结果较准确。	建立程序的依赖关系图代价非常高，对图结构的操作时空间效率低，较难运用于大规模的软件系统中。
基于串结构(String-structure)	字符串(String)[20,46-48]	字符串匹配[20,46-48]	轻量级词法分析器或自定义规则，方法可适用于多种编程语言中，可应用于大规模的软件中。时空效率佳。	较少地反映程序的语法和语义信息，抄袭手段检测效果低，报告结果需要更多的人工分析。
基于标识符结构(Token-structure)	字符、标识符(Token)[17-19,21-23,36,38,49-52]	公共子串[17,21,23] 信息距离[18,36,50,52] 字符串匹配[19,22,51] 后缀树[38,49]	无需语法分析工具，容易扩展到多类编程语言中，时空效率佳，可应用于大规模的软件中。	忽略了程序的语法和语义信息，只对词法级别的抄袭有良好的检测的效果。相似值的参考价值低。
基于度量值(Metric-based)	程序属性度量值[10-14,25,36,53-58]	计算公式[10-14] 神经网络[25,53,54] 决策树[55] 信息距离[36,56-58]	考虑了程序的语法特征，容易扩展到多类编程语言中，可方便定位源代码中的抄袭片段。	度量值不能代表比较单元的全部特征，检测效果的好坏需要进一步验证。

4 代码相似性检测技术的相关研究

4.1 代码克隆检测技术

克隆代码自动检测技术的目的是发现那些在软件系统中相同或相似的代码片段[2,5]，主要应用于软件重构与再工程[68]，以提高软件复用性[69]。克隆的代码也可从词法、语法和语义3个层面来考虑。其检测技术的核心是代码相似性判定，即寻找软件系统中超过某一阈值的代码片段集。代码克隆检测和代码相似性检测在检测思路和原理上都有很大的一致性，两种方法具有一定的通用性[1,4,5,41,46,47,49,63]。

4.2 代码的天然相似

代码之间存在的相似现象有一些客观原因[33,65,67]，可以从代码本身和编写人员的角度来分析。程序代码的语法比较规则，一些简单的操作所使用的程序语句基本一致，程序代码大部分是由这些简单的语句所构成，那么不可避免地在代码中存在着一些相似的语句。特别地，如果程序代码所解决的问题难度较低，流程就比较简单。这样的程序也会有很大的相似性。文献[26,53]在收集实验的样本时都提到了这一问题，而文献[66]提出了一种自适应的算法来降低这一问题所带来的影响。对于不同的编程人员，受技能培训和思维定式的影响，解决问题的思路也可能存在一致性，使用类似的逻辑开发的代码也会很相似。

4.3 全局代码的相似

文献[55,59,60]在研究中关注了全局的代码相似现象(所谓全局，即是代码源并非本地提交的，而是存在于网络、书籍和音像制品中的参考代码)。文献[59]认为与局部的代码相似应用领域相比，全局的抄袭现象缺乏更有效的检测手段，导致其问题日益严重。文献[55]主要分析了Web上的代码抄袭现象，而在文献[60]中探讨了应对网络抄袭的一些有效的策略。

4.4 代码抄袭的预防和管理

基于代码抄袭者需要参考一份或多份代码的事实，文献[62,63]认为在代码编写的初期施加一些策略就可捕获抄袭。文献[61]提出的代码水印技术，能够有效地记录、预防和管理抄袭现象。

结束语 目前，代码相似性检测仍然存在很多困难和问题。总结已有的研究成果并结合我们的研究实践，我们认为目前代码相似性自动检测技术中还存在以下的不足：

1) 在检测的时空效率和检测结果的准确度上难以取得较好的折衷。不同应用背景和代码规模对于时空效率的需求各不相同。5类检测思路各有优缺点，但由于它们所采用的代码抽象表示各异，因此很难将这几类技术的优势结合在一起，以形成一个更好的相似性检测方法。

2) 由于没有代码相似性的权威定义，不同学者对于相似代码的理解有差异。“两份代码满足一些抄袭变换方式，就可以认为是相似的”这一说法缺乏量化值，以至于在研究过程中无法确定代码相似性的客观值，代码相似性的检测效果存在着一定的失真性。

3) 文献[24]中总结了当前流行的检测工具能够检测出的抄袭手段，结果表明大部分的检测工具和方法只能检测出一些使用很少量的抄袭手段的相似代码，抄袭手段主要集中在词法和语法变换两方面。如果抄袭者获知相似性检测工具的技术原理，则完全可以使用一些更具有针对性的、更高级的基于语法和语义等价的代码变换方法来逃避抄袭检测。此外，一些检测方法只是针对某一种特定的抄袭手段，缺乏实用性[36,37]。

4)由于目前没有一个权威的代码相似性检测的测试集，因此代码相似性检测的效果难以准确评估。虽然，目前大多使用Jplag，MOSS等系统作为检测效果的对比工具[26,36,43,37,50,53,57,58]，但这些工具本身的检测效果并不理想[25,26,43,50,53]。

展望本领域的发展趋势，我们认为未来的代码相似性检测技术的研究将主要集中于：

1) 多类技术的综合。两份代码之间是否存在相似的部分，很大程度上可通过其文本的外观特征表现出来，基于串和标识符的检测的显著特点是代价低，通用性强，因此可以将基于文本特性的检测方法作为一个基础。在此基础上，结合具体的任务目标对结果进行后续处理，通过使用诸如树和图结

构的方法检测出相似代码片段的语法结构、语义信息等，使得检测结果更有意义。概括而言，就是在总体上使用一些文本检测的方法快速确定相似的代码片段，然后对这些片段使用语法、语义、代码特征度量值等分析手段得到更加准确的结果。文献[38,41,43,45,64]研究的方法都体现了这一基本思想。在基于树和图等相似性检测效果较好的方法中引入一些文本或度量值的方法，能够有效地降低时空复杂度，提升检测方法的适用性。此外，分类器思想的引入能够成功地整合多类检测方法[25,53]，而且在时空效率和检测效果上还有很大的改进空间，也不失为非常有价值的研究方向。

2）聚类方法。在一份程序集中寻找相似的代码，除了采用代码相似性检测技术外，还需要适当的聚类方法。文献[52]将研究的重点放在如何提出一种有效的聚类方法上。随着代码集规模的扩大，有必要寻找更有效的聚类方法。

3）检测对象的扩展。前面介绍的检测方法（见第3节）都是基于源代码的、文件级别的。而模块级别和源代码不可见的代码相似性检测方法是下一步关注的重点，比如检测软件系统中功能模块的相似性、Java软件的字节码相似性检测等。此外，由于代码自动化生成技术的不断发展，软件中不可避免地存在很多自动产生的代码。这些代码会降低相似性检测的效率，如何有效地剔除这些冗余信息将成为代码相似性检测的一个重要课题。

4）新的研究领域。如何将程序分析与理解的技术与代码相似性检测相结合，是一个值得研究的课题。此外，如何将代码演化理论和代码差异分析方法应用于代码相似性检测，也是值得关注的研究方向。

参 考 文 献

[1] Bilenko M, Mooney R J. Adaptive duplicate detection using learnable string similarity measure[C]// Proceeding of ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 2003:39-48,

[2] 曹羽中，金茂忠，刘超. 克隆代码检测技术综述[J]. 计算机工程与科学，2006，28(A2)：9-13

[3] Baker B S. On finding duplication and near duplication in large software systems[C]// Proceedings of 2nd Working Conference on Reverse Engineering. 1995:86-95

[4] Mayrand J, Leblanc C, Merlo E M. Automatic detection of function clones in a software system using metrics[C]// Proceeding of International Conference on Software Maintenance (ICSM). 1996

[5] Rieger M. Effective clone detection without language barriers [D]. Bern University, Switzerland, 2005

[6] Georgina C, Mike J. Source-code plagiarism: A UK academic perspective [R]. RR-422. Department of computer Science, University of Warwick, 2006

[7] Sheard J, Dick M, Markham S, et al. Cheating and plagiarism: perceptions and practices of first year it students[C]// Proceedings of the 7th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. 2002:183-187

[8] McCabe D. Levels of Cheating and Plagiarism Remain High [OL]. Center for Academic Integrity. Duke University, 2005. http://academicintegrity.org

[9] 鲍军鹏，沈钧毅，刘晓东，等. 自然语言文档复制检测研究综述[J]. 软件学报，2003，14(10)：1753-1760

[10] Halstead, Howard M. Elements of Software Science [Z]. Elsevier, 1977

[11] Ottenstein K J. An Algorithmic Approach to the Detection and Prevention of Plagiarism[J]. SIGCSE Bulletin, 1977, 8(4):30-41

[12] Berghel H L, Sallach D L. Measurements of program similarity in identical task environments[J]. Sigplan Notices, 1984, 19:65-76

[13] Grier S A. A tool that detects plagiarism in PASCAL programs [J]. SIGCSE Bulletin, 1981, 13:15-20

[14] Faidhi J A W, Robinson S K. An empirical approach for detecting program similarity and plagiarism within a university programming environment[J]. Computer Education, 1987, 11(1): 11-19

[15] Verco K L, Wise M J. Software for detecting suspected plagiarism: comparing structure and attribute-counting systems[C]// Proceeding of the 1st Australian Conference on Computer Science Education. 1996:3-5

[16] Donaldson J L, Lancaster A-M, Sposato P H. A plagiarism detection system[J]. SIGCSE Bulletin, 1981, 13:21-25

[17] Whale G. Plague: Plagiarism Detection Using Program Structure [R]. Dept. of Computer Science Technical Report 8805. University of NSW, Kensington, Australasian, 1988

[18] Schleimer S, Wilkerson D, Aiken A. Winnowing: Local Algorithms for Document Fingerprinting[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. 2003:76-85

[19] Prechelt L, Malpohl G, Philippsen M. Finding plagiarisms among a set of programs with JPlag[J]. Journal of Universal Computer Science, 2002, 8(11):1016-1038

[20] Gitchell D, Tran N. Sim: A Utility for Detecting Similarity in Computer Programs[C]// Proceedings of 30th SIGCSE Technical Symposium. New Orleans, LA, USA, 1999

[21] Wise M J. Detection of similarities in student programs: YAP'ing may be preferable to Plague' ing[C]// Proceedings of 23th SIGCSE Technical Symposium. Kansas City, USA, 1992: 268-271

[22] Wise M J. YAP3: Improved Detection of Similarities in Computer Program and Other Texts [C]// ACM SIGCSE. 1996:130-134

[23] Cunningham P, Mikoyan A. Using CBR techniques to detect plagiarism in computing assignments [R]. Department of Computer Science, Trinity College, Dublin, 1993

[24] Liu Chao, Chen Chen, Han Jia-wei, et al. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis [C]// Proceedings of ACM SIGKDD 2006. 2006:872-881

[25] Engels S, Lakshmanan V, Craig M. Plagiarism Detection Using Feature-Based Neural Networks [C] // Proceedings of ACM SIGCSE 2007. 2007

[26] 赵长海，晏海华，金茂忠. 一个基于编译优化和反汇编的程序相似性检测方法[J]. 北京航空航天大学学报，2008，34(6)：711-715

[27] Noh Seo-Young, Gadia S K. An XML plagiarism detection model for procedural programming languages[C]// Proceedings of the 2nd International Conference on Computer Science and its Applications. 2004:320-326

[28] JPlag services 2002[OL]. http://www.ipd.uni-karlsrube.de/

jplag/

[29] Measure of software similarity[OL]. 2003. http://www.cs.berkeley.edu/~moss/general/moss.html

[30] Bull J, Collins C, Coughlin E, et al. Technical review of plagiarism detection software report [R]. http://www.jisc.ac.uk/. 2003

[31] Parker A, Hamblen J O. Computer Algorithms for Plagiarism Detection[J]. Proceedings of IEEE Transaction on education, 1989, 32(2)

[32] Clough P. Plagiarism in natural and programming languages: An overview of current tools and technologies[M]. Research Memoranda: CS-00-05. Department of Computer Science, University of Sheffield, 2000

[33] Joy M, Luck M. Plagiarism in Programming Assignments[J]. Proceedings of IEEE Transaction on education, 1999, 42(2)

[34] Whale G. Detection of plagiarism in student programs [C]// Proceedings of the 9th Australian Computer Science Conference. 1986: 231-241

[35] Jones E L. Metrics based plagiarism monitoring[J]. Proceedings of the 6th Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges, 2001, 16(4): 253-261

[36] Arwin C, Tahaghoghi S M M. Plagiarism Detection across Programming Languages[C]// Proceedings of the 29th Australasian Computer Science Conference (ACSC2006). 2006

[37] Ji Jeong-Hoon, Woo Gyun, Cho Hwan-Guo. A Source Code Linearization Technique for Detecting Plagiarized Programs[C]// Proceedings of ITiCSE'07. United Kingdom, 2007

[38] 刘楠,刘超,李虎. 基于 Eclipse 平台的克隆代码检测插件的设计[J]. 计算机科学, 2008, 35(11. 专刊)

[39] Kim Young-Chul, Cho Yong-Yoon, Moon Jong-Bae. A plagiarism detection system using a syntax-tree[C]// Proceedings of International Conference on Computational Intelligence. 2004, 1: 23-26

[40] Kontogiannis K, Galler M, DeMori R. Detecting code similarity using patterns[C]// Proceedings of Third Workshop on AI and Software Engineering. Breaking the Toy Mold, 1995: 68-73

[41] Jiang Lingxiao, Misherghi G, Su Zhendong, et al. DECKARD: Scalable and Accurate Tree-based Detection of Code Clones[C]// Proceedings of the 29th International Conference on Software Engineering (ICSE'07). 2007: 96-105

[42] Noh Seo-Young, Kim Sangwoo, Jung Cheonyoung. A lightweight program similarity detection model using XML and levenshtein distance[C]// Proceedings of the 2006 World Congress in Computer Science, Computer Engineering and Applied Computing. 2006: 3-9

[43] Xiong Hao, Yan Haihua, Li Zhoujun, et al. A program plagiarism detection approach based on abstract syntax tree[C]// Proceedings of the 2009 International Conference on Artificial Intelligence and Education. 2009: 196-205

[44] Komondoor R, Horwitz S. Using slicing to identify duplication in source code[C]// Proceedings of the 8th International Static Analysis Symposium (SAS). 2001

[45] Krinke J. Identifying similar code with program dependence graphs[C]// Proceedings of 8th Working Conference on Reverse Engineering (WCRE'01). 2001: 301-309

[46] Baker B S. A Program for Identifying Duplicated Code[J]. Computing Science and Statistics, 1992, 24: 49-57

[47] Ducasse S, Rieger M, Demeyer S. A language independent approach for detecting duplicated code[C]// Proceedings of International Conference on Software Maintenance (ICSM). 1999: 109-118

[48] West A. Coping with plagiarism in computer science teaching laboratories [C]// Computers in Teaching Conference. 1995

[49] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multi-linguistic token based code clone detection system for large scale source code[J]. Proceedings of IEEE Transactions on Software Engineering, 2002, 28(6): 654-670

[50] Jadalla A, Elnagar A. PDE4Java: Plagiarism Detection Engine for Java source code: a clustering approach[J]. International Journal of Business Intelligence and Data Mining, 2008, 3(2): 121-135

[51] Burrows S, Tahaghoghi S M M, Zobel J. Efficient and effective plagiarism detection for large code repositories[C]// Proceedings of the Second Australian Undergraduate Students' Computing Conference (AUSCC04). 2004: 8-15

[52] Moussiades L, Vakali A. PDetect: A clustering approach for detecting plagiarism in source code datasets[J]. The Computer Journal, 2005, 48(6): 651-661

[53] 熊浩,晏海华,李舟军,等. 一种基于 BP 神经网络的代码相似性检测方法[J]. 计算机科学, 2010, 37(3): 159-164

[54] Singhe S, Tweedie F J. Neural networks and disputed authorship: New challenges[C]// Proceedings of the 4th International Conference on Artificial Neural Networks. 1995: 24-28

[55] Elenbogen B S, Seliya N. Detecting outsourced student programming assignments[J]. Journal of Computing Sciences in Colleges, 2007: 50-57

[56] Ciesielski V, Wu N, Tahaghoghi S. Evolving similarity functions for code plagiarism detection[C]// Proceedings of 10th Conference on Genetic and Evolutionary Computation (GECCO08). 2008: 12-16

[57] Chen X, Francia B, Li M, et al. Shared Information and Program Plagiarism Detection[J]. Proceedings of IEEE Transactions on Information Theory, 2004, 50(7): 1545-1551

[58] Zhang Liang, Zhuang Yue-ting, Yuan Zhen-ming. A Program Plagiarism Detection Model Based on Information Distance and Clustering[C]// Proceedings of International Conference on Intelligent Pervasive Computing. 2007: 431-436

[59] Butakov S, Scherbinin V. The toolbox for local and global plagiarism detection[J]. Computers and Education, 2009, 52: 781-788

[60] Austin M J, Brown L D. Internet plagiarism: Development strategies to curb student academic dishonesty[J]. The Internet and Higher Education, 1999, 2(1): 21-33

[61] Daly C, Horgan J. Patterns of Plagiarism[J]. ACM SIGCSE Bulletin, 2005, 31(1): 383-387

[62] Horwitz S. Identifying the semantic and textual differences between two versions of a program[C]// Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation. 1990, 25: 234-245

[63] Toomim, Begel M, Graham A, et al. Managing Duplicated Code with Linked Editing[C]// Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing. 2004: 173-180

(下转第 76 页)

制后，信任评价的正确率也会受到一定影响。

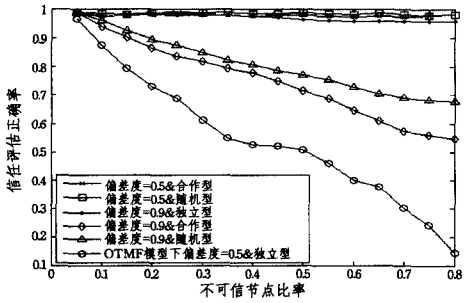

图 2　不可信节点比率 vs. 信任评估正确率

5.2　收敛性比较

由于 OTMF 模型采用了间接推荐的机制，因此信任度收敛速度比一般信任模型已经有了较大的提高，但 OTMF 模型的收敛速度是在没有考虑虚假推荐攻击下的。图 3 反映了在不可信节点占 10%，虚假推荐偏差度不超过 0.5，采用随机型的虚假攻击方式下，OTMF 模型和 RTM 之间在收敛性能上的差异。

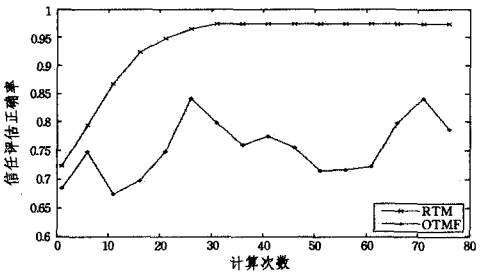

图 3　计算次数 vs. 信任评估正确率

由于 RTM 具备对不可信节点的识别能力，在有限次信息交互后，对不可信节点的间接推荐基本采取极低信任度，即不可信节点的间接推荐将对模型的信任值不会产生影响，因此在该模型下信任评估会逐步达到稳定的状态。而 OTMF 不具备对不可信节点的识别能力，所以系统的评估正确率不会随着间接推荐的增加而趋于稳定，而会随着不可信节点的随机攻击性而出现不稳定的状态，基本信任评估正确率在[0.65，0.85]区间徘徊。

结束语　针对 Ad hoc 网络中源路由的特性，在 OTMF 模型基础之上提出的鲁棒信任机制(RTM)提升了对虚假推荐的抵御能力并且加快了信任值的收敛速度。但通过性能分析表明，该系统在恶意节点和不可信节点协作攻击下，系统的评估正确率有待进一步提高，这些将是我们今后研究工作的重点。

参 考 文 献

[1]　孙玉星，黄松华，等. 自治网络中信任/信誉模型的安全现状的研究[J]. 计算机科学，2009，36(4)：5

[2]　Li Ruidong，Li Jie，Liu Peng，et al. An Objective Trust Management Framework for Mobile Ad hoc Networks[C]// Vehicular Technology Conference，VTC 007-Spring. April 2007：56-60

[3]　Theodorakpoulos G，Baras J S. On Trust models and trust evaluation metrics for ad-hoc networks [J]. IEEE Journal on Selected Areas in Communications，2006，24(2)：318-328

[4]　Theodorakpoulos G. Distributed trust evaluation in ad-hoc networks[D]. Maryland：University of Maryland，2004

[5]　Jiang Tao，Baras J S. Trust Evaluation in Anarchy：A Case Study on Autonomous Networks [C]// Proceedings of the 25th Conference on Computer Communications. Barcelona，Spain，April 2006

[6]　Buchegger S，Le Boudec J-Y. A Robust Reputation System for P2P and Mobile Ad-hoc Networks[C]// Proceedings of P2PEcon 2004. Harvard University，Cambridge MA，U. S. A. http：// www. sonja. ws/robust. pdf

[7]　Sun Y，Yu W，Han Z. Information theoretic framework of trust modeling and evaluation for ad hoc networks[J]. IEEE Journal on Selected Areas in Communications，2006，249(2)：674-679

[8]　Lamsal. Understanding trust and security[J/OL]. http：//www. cs. helsinki. FI/u/lamsal/papers/

[9]　Marti S，Giuli T J，Lai K. Mitigating routing misbehavior in mobile ad hoc networks[C]// Proceedings of the 6th annual international conference on Mobile computing and networking. Boston，Massachusetts，United States，2000：255-265

[10]　Sun Yu-xing，Huang Song-Hua，et al. Bayesian Decision-Making Based Recommendation Trust Revision Model in Ad hoc Networks [J]. 软件学报，2009，20(9)：2574

(上接第 14 页)

[64]　Ohno A，Murao H. Measuring Source Code Similarity Using Reference Vectors[J]. Proceedings of International Conference on Innovative Computing，Information and Control (ICICIC' 06)，2006，2(30-01)：92-95

[65]　Mann S，Frew Z. Similarity and originality in code：Plagiarism and normal variation in student assignment[C]// Proceedings of the 8th Australian conference on computing education. 2006，52：37-58

[66]　Ji J H，Park S-H，Woo G，et al. Source code similarity detection using adaptive local alignment of keywords[C]// Proceedings of 8th International Conference on Parallel and Distributed Computing，Applications and Technologies. 2007：179-180

[67]　Cosma G，Joy M. Towards a definition of source-code plagiarism [C]// Proceedings of IEEE Transactions on Education. 2008，51：195-200

[68]　Fowler M，Beck K，Brant J，et al. Refactoring：Improving the Design of Existing Code [M]：63

[69]　Gamma E，Helm R，Johnson R，et al. Design patterns：Elements of reusable object-oriented software[M]. Addison-wesley，1997