基于编译优化和反汇编的程序相似性检测方法

赵长海 晏海华 金茂忠

(北京航空航天大学 计算机学院, 北京 100191)

摘 要: 提出了基于编译优化和反汇编的程序相似性检测方法, 能够检测出标识符重命名、增加冗余语句、等价的控制结构替换等12种学生常用的抄袭手段. 基于该方法, 设计和实现了一个程序相似性检测系统BuaaSim, 采用编译优化和反汇编技术将源程序转化为汇编指令集合, 删除和替换汇编指令中对程序本质特征影响不大的易变元素, 使用一个与指令顺序无关的决策函数计算程序相似度; 还给出一个简单有效的聚类算法, 从程序集合中聚类出相似的程序子集. 通过与著名的JPlag系统针对两份典型的抄袭样本集进行评测对比, 表明本文方法的检测效果具有明显的优势.

关 键 词: 抄袭; 程序相似性; 相似性检测; 编译优化

中图分类号: TP 311.56

文献标识码: A 文 章 编 号: 1001-5965(2008)06-0711-05

Approach based on compiling optimization and disassembling to detect program similarity

Zhao Changhai Yan Haihua Jin Maozhong

(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: An approach based on compiling optimization and disassembling was proposed to detect similarity in computer programs. It can detect 12 modification strategies that are often used by students, such as renaming identifiers, adding redundant statements and replacing control structures with equivalent structures. The implemented software, called BuaaSim, translates source code into assembly instructions with the help of compiler and disassembler, removes and replaces those easily changed elements in the assembly instructions, and applies a decision function to calculate the similarity, which doesn't depend on the order of assembly instructions. A simple clustering algorithm was also introduced to find all groups of similar programs. By using two sets of plagiarized transcripts as testing programs, the comparative evaluation shows that BuaaSim has more advantages than JPlag, a famous similarity detection system.

Key words: plagiarism; program similarity; similarity detection; compiling optimization

当前国内外的大学教育中学生作业抄袭现象比较普遍, 尤其是计算机专业的程序设计课程, 抄袭现象更加严重. 国外很多教育机构针对程序设计课程的源代码抄袭现象进行的调查显示, 高达85.4%的学生承认抄袭过别人的编程作业[1-2]. 相对于自然语言, 程序语言语法非常规则, 抄袭起来简单的多, 完全不用理解程序, 通过文本编辑器进行简单的变量替换、添加冗余代码、变换代码顺序等手段就可以改变代码的外观, 且不影响程序的正常运行.

如今已经有很多软件工具可以协助教师检测代码相似性, 如斯坦福大学的Moss系统[3]、德国卡尔斯鲁厄大学的JPlag[4-5]、威奇塔州立大学的Sim[6]、悉尼大学的YAP3[7]等. 这些工具普遍采用属性计数和结构分析相结合的方法计算程序的相似度, 只有相似度大于某个阈值时, 才能认定学

收稿日期: 2007-05-23

基金项目: 国家自然科学基金资助项目(60703057)

作者简介: 赵长海(1979-), 男, 河南驻马店人, 博士生, zch@sei.buaa.edu.cn.

生抄袭,这个阈值很难确定,阈值过大可能会错过很多相似的程序,阈值过小可能会把很多本来独立编写的程序认定为抄袭,实际操作过程中,教师需要人工对比查看很多学生程序,才能选择一个合适的阈值. 本文作者在使用 JPlag和 Moss过程中还发现,如果在学生程序中加入一些冗余的语句、声明冗余的变量或者拆分语句,相似度会明显下降.

相比上述几个著名的程序相似性检测系统,本文提出的相似性检测方法的独到之处在于:引入编译优化技术和反汇编技术,从程序语义层面消除类似代码冗余、语句拆分、控制结构等价替换等高级抄袭手段带来的干扰,并给出一个相似度经验阈值和一个基于该经验阈值的简单有效的聚类算法. 基于该相似性检测方法的一个实现——BuaaSim系统已应用于北京航空航天大学高级语言程序设计教学辅助平台,该教学平台可以自动评判学生提交的源程序,然后使用 BuaaSim对学生提交的程序进行相似性比较,经过 3届学生的使用,取得了非常好的效果.

1 相关研究

文献[8]总结了学生常用的 10种抄袭手段,在此基础上,本文又增加了常量替换和表达式拆分 2种手段,根据抄袭所付出的努力,从易到难依次为:①完整拷贝;②修改注释;③重新排版;④标识符重命名;⑤代码块重排序;⑥代码块内语句重排序;⑦常量替换;⑧改变表达式中的操作符或者操作数顺序;⑨改变数据类型;⑩增加冗余的语句或者变量;⑪表达式拆分;⑫替换控制结构为等价的控制结构.

除了上述 12种,也存在其它抄袭手段,但需要对程序语言有较深入的了解,不是抄袭检测系统关注的重点. 一个好的检测系统应该对上述 12种抄袭手段有比较强的检测能力,否则学生就会利用系统无法检测的抄袭手段逃避检测,但目前还没有一个系统能消除上述所有抄袭手段带来的干扰.

程序相似性检测技术已经有 30多年的历史,早期的程序相似性检测主要基于属性计数方法,该方法从源代码中抽取各种度量元,例如关键字数、操作符数、循环数等,由于属性计数法没有考虑程序的结构,如果修改了源程序的结构,这种方法就会失效[9];后来出现了基于结构度量的相似性检测方法,MOSS, JPlag, SIM 和 YAP3是当前比较知名的基于结构度量的程序相似性检测系统,它们对程序进行度量的时候都考虑到了程序的结构,例如函数调用关系、扇入扇出系数、McCabe圈复杂度等与结构相关的信息.

上述系统所采用的方法在检测前 4种抄袭手段方面效果都很好,但由于它们都是在源代码层次基于大量的度量信息来判别相似性,而且没有对程序进行数据流和控制流分析,如果程序规模比较小,在代码中增加冗余变量或语句、拆分语句、增加冗余的头文件等,导致计算出的相似度陡降,使得原本相似的程序无法检测出来. 本文提出的相似性检测方法引入编译优化技术,能有效消除上述所有抄袭手段带来的干扰.

2 基于编译优化和反汇编检测相似性

抄袭程序为了保证正确性,其运行逻辑和执行结果必须与原始程序一致. 对于那些不理解原始程序的逻辑,但又担心把程序改错的学生,低级的等价变换(如更改变量名、修改注释、重新排版等)是首要的选择;一些对编程语言掌握稍微好点的学生,可能会采用比较高级的等价变换(如增加冗余变量、拆分语句、掉换语句顺序等). 对于低级的等价变换,不会影响编译生成的目标代码;高级的等价变换采用的手段,一般在编译器优化阶段即可消除. 下面给出一个基于编译优化和反汇编的程序相似性检测方法,其核心思想是利用编译器和反汇编工具对程序进行规一化(normalize)生成特征文本,最后对比特征文本确定程序两两之间的相似度,根据该相似度进行聚类,分析出相似的程序子集. 该方法的流程如图 1所示.

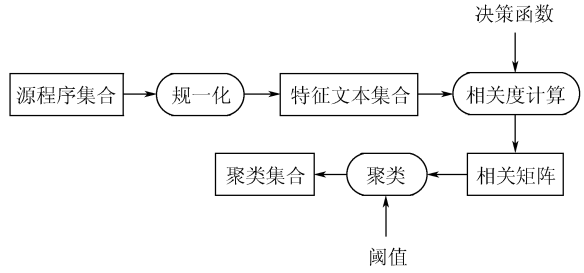

图 1 程序相似性检测流程

2.1 概念定义

规一化:为了进行相似性检测而对程序所执行的修剪和变换称为规一化. 规一化有 2个主要目的:①将代码中或者编译生成的二进制文件中的某些对程序的骨架影响较小的符号替换为通用的符号,例如将常量等统一替换为某一特殊符号、对表达式的简化和统一(例如 $a+=b$统一替换为 $a=a+b$)等等;②去除代码中的噪声,例如删

除代码中的注释行、对代码重新排版、编译优化消除冗余语句等.

2.2　规一化过程

规一化过程应用于所有的程序代码, 消除抄袭手段带来的噪声, 并生成代表该程序本质特征的文本. 规一化过程分为 3步, 如图 2所示. 首先优化编译生成与平台相关的二进制文件, 然后反汇编该二进制文件生成汇编代码, 最后对该汇编代码进行过滤生成特征文本并放入特征文本集合, 等待后续阶段应用决策函数计算相关度.

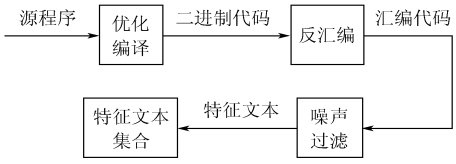

图 2　规一化过程

2.2.1　优化编译

学生常用的抄袭方法中, 一般是在代码层面上添加一些混淆信息, 改变代码的外观, 增加教师通过读代码辨别抄袭的难度, 编译生成的二进制代码通常不会发生变化.

代码排版格式只会影响代码的可读性, 编译器会忽略它们; 注释在编译的预处理阶段就会被删除, 所以修改注释和重新排版带来的噪声最容易也是最早被排除的; 代码中的标识符名并不会对程序的执行效果产生任何影响, 所以可忽略编译后二进制代码中的标识符, 则标识符重命名带来的噪声也被消除.

编译优化采用控制流分析、数据流分析和依赖分析等技术, 消除公共子表达式, 降低计算强度, 优化循环与跳转等. 优化编译可以将等价的程序逻辑表达方式转化为统一的形式, 因此采用代码冗余、表达式拆分或者等价控制结构替换手段修改过的代码, 经过编译优化编译后, 抄袭代码与原始代码生成的目标代码是一样的.

在优化编译阶段, 更改注释、重新排版、标识符重命名、增加冗余的语句或者变量、表达式的简单拆分和替换控制结构为等价的控制结构, 共 6种抄袭手段带来的噪声, 在源代码被转化为二进制目标代码后被消除, 但是改变代码块或者语句顺序带来的噪声还未消除, 改变语句顺序不仅仅是导致指令顺序发生改变, 还可能由于数组或者结构体的声明顺序变动, 导致偏移地址发生改变. 下一阶段通过反汇编技术, 过滤更多的噪声.

2.2.2　反汇编

第一步通过优化编译产生平台相关的二进制代码, 可以借助反汇编工具, 将程序中的代码段转化为汇编代码, 删除与程序特征无关的信息. 相对于二进制指令, 汇编代码比较容易理解, 而且每一条汇编指令都代表一定的语义, 是天然的程序"指纹", 一个执行逻辑不同的程序, 其对应的反汇编代码必然不同.

2.2.3　噪声过滤

经过上一步的处理, 程序被转换为汇编代码, 汇编代码中的偏移地址、函数地址、立即数和部分跳转指令需要进行特殊处理.

偏移地址 (例如: jmp* 0x804981c)非常容易变化, 语句或者变量声明顺序改变就可能导致偏移地址的变化, 为了消除这个差异, 统一使用 OFFSET代替.

函数地址 (例如: call 0x8048324)跟偏移地址类似也是易变的, 如果改变函数体的顺序就会导致函数地址改变, 统一使用 FUNCTION 替代函数地址, 屏蔽函数地址的差异.

立即数 (例如: push $ 0x8)代表了程序代码中的常量, 为了避免常量替换带来的噪声, 统一使用 CONSTANT替换汇编代码中的立即数.

还有一种情况需要注意, 如果关系表达式中操作数顺序, 例如将 $x < y$, $x <= y$, 改成 $y > x$, $y >= x$, 相应的汇编指令 jl就会改为 jg, jle变为 jge, 为了屏蔽这个抄袭手段的影响, 统一将 jl替换为 jg, jle替换为 jge.

2.3　决策函数

源代码经过规一化, 最后映射为汇编指令集合. 用 P_1和 P_2表示 2个待检测的程序, $F(P_1)$和 $F(P_2)$表示 P_1和 P_2规一化后的汇编指令集合, $Sim(P_1, P_2)$表示 2个程序的相似度, 本文选用的决策函数如下:

$$Sim(P_1, P_2) = \frac{F(P_1) \cap F(P_2)}{F(P_1) \cup F(P_2)} \qquad (1)$$

相似度 Sim满足 $Sim(P, P) = 1$, 且 $Sim(P_1, P_2) = Sim(P_2, P_1)$. 根据经验, 该公式相比其它常用的相关系数计算公式, 计算出的结果有更好的区分度. 选用公式 (1)另外一个优势是它的计算结果不受指令顺序的影响, 进而可以消除代码块或者语句顺序变动带来的噪声.

由于规一化过程中已经基本消除了抄袭方法带来的噪声, 相似程序的相似度通常会达到一个比较高的值, 相似与不相似程序之间的相似度会有一个比较大的落差. 本文作者对 3届学生提交的将近 6 000份作业程序进行了统计分析, 发现抄袭程序与原始程序的相似度会达到 93%以上,

大部分相似度达到 100%; 而 2个独立编写的程序之间的相似度通常在 80%以下, 本方法选用 93%作为阈值, 高于 93%可认为相似.

2.4 聚类算法

程序两两之间的相似度计算出来之后, 还需要一个聚类的过程, 从提交的源程序集合中分析出相似的程序集合. 聚类过程中最重要的是要分析出原始程序, 因为通常原始程序与抄袭程序最类似. 假设 A 是原始程序, B 与 C 都是抄袭 A 得到的, 那么就存在 $Sim(A, B) > 93\%$, $Sim(A, C) > 93\%$, 而 $Sim(B, C) < 93\%$的情况. 如果用 B 作为聚类的种子, 聚类中就会丢失 C; 一个聚类也可能存在多个原始程序的情况, 例如 B 抄袭 A, C抄袭 B, 有可能 $Sim(A, C) < 93\%$, 这时就需要使用多个聚类种子.

本方法选定的阈值为 93%, 抄袭程序与原始程序间的相似度基本在 93%以上. 有了这个特点, 聚类算法就可简化, 若用 $C(P_1, P_2, \cdots)$表示一个正在构造中的聚类集合, 如果程序 P_k与该集合中的任何一个程序的相似度大于 93%, 即可将 P_k归入该聚类集合. 计算机编程实现的时候可以先任选一个程序作为聚类种子, 然后不断地迭代, 即可构造出所有的聚类集合. 该聚类算法使用计算机实现相当简单, 时间复杂度也比较低.

3 结果评测及相关工作比较

目前还没有一个权威的抄袭样本可以用来评测相似性检测系统的效果, 本文作者在 15位研究生的帮助下, 构造了 2组典型的评测程序集, 可以在 http://www.sei.buaa.edu.cn/buaasim 下载. 本方法使用该评测程序集与 JPlag系统的评测结果进行比较, 发现本系统的长处与不足. 文献[4]认为 JPlag相比 Moss系统检测效果更好, 性能方面优于 YAP3, 所以与 JPlag对比具有代表意义.

该评测程序集内包含了 2组 C语言程序, 分别对应 2个问题的实现: 统计出现次数最多的整数(简称题目 1)和交叉引用生成器(简称题目 2). 题目 1的程序平均源代码长度 50行左右, 题目逻辑比较简单; 题目 2的程序平均源代码长度 150行左右, 题目难度较大, 涉及多个函数调用. 每一组内有 3个程序, 由 3个学生独立完成, 组内的其它程序都是基于这 3个程序进行抄袭. 为了保证样本的公平性, 没有告知样本提供者本系统和 JPlag使用的相似性检测算法; 同时, 为了保证样本的典型性, 要求样本提供者在不改变程序最终输出结果的情况下, 尽量采用高级的手段抄袭程序, 而不仅仅是简单的修改注释、更改变量名和重新排版.

使用 BuaaSim和 JPlag分别检测上述评测集, 聚类结果如表 1所示. 编号为 0, 10和 20是 3个不同的原始程序, 其它编号的程序都是抄袭这 3个程序; JPlag系统中计算两程序间的相似度不具备可交换性, 即 $Sim(P_1, P_2) \neq Sim(P_2, P_1)$, 评判结果分别根据平均相似度和最大相似度进行聚类, 根据 JPlag结果说明, 平均相似度最能代表程序的相似程度, 适用于大多数情况, 最大相似度适用于 2个程序代码行相差较大的情形, 本测试样本都是基于同一个题目的实现, 代码行相差非常小, 所以本实验取根据平均相似度进行聚类的结果.

表 1 BuaaSim和 JPlag的检测结果

	题目 1	题目 2
人工聚类	{0, 1, 2, 3, 4, 5} {10, 11, 12, 13, 14, 15, 16} {20, 21, 22, 23, 24}	{0, 1, 2, 3, 4, 5} {10, 11, 12, 13, 14, 15, 16} {20, 21, 22, 23, 24}
BuaaSim	{0, 1, 2, 3, 4, 5} {10, 11, 13, 14, 15, 16} {20, 21, 22, 23, 24}	{0, 1, 2, 3, 4, 5} {10, 11, 13, 14, 15, 16} {20, 21, 22, 23, 24}
JPlag	{0, 2, 3, 4, 5} {10, 11, 13, 16} {20, 22, 23, 24}	{0, 1, 2, 3, 4, 5} {10, 11, 13, 16} {20, 22, 23}

从表 1的结果可以看出, 2个工具都没有出现误判的情况, 程序代码毕竟反映了人的思维, 2个独立编写的程序一般情况下差异会比较大, 除非程序要解决的问题非常简单.

检测效果 BuaaSim明显好于 JPlag, BuaaSim没有检测到题目 1和题目 2对应的编号为 12的抄袭程序, 这 2个程序都是由同一个样本提供者改写编号为 10的原始程序得到的, 与编号 10的程序的相似度分别为 84%和 86.9%, 经过仔细分析, 发现该样本提供者对原始程序做了如下改动:

1) 将 1个函数分解为 2个函数;
2) 在结构体内增加冗余的结构成员;
3) 修改函数参数的顺序;
4) 将 for循环改为 while循环;
5) 替换变量名;
6) 改变函数内变量声明循序;
7) 改变函数定义顺序.

主要是前 2项改动导致 BuaaSim无法检测出抄袭程序, 包括修改函数参数顺序在内的其它几项改动在规一化过程中可以被消除. 采用上述的

抄袭手段需要付出比较大的努力,而且需要具备相当的编程能力,虽然调低相似度阈值可以检测出这种类型的抄袭程序,但是对于教学来说没有太大必要.

JPlag遗漏的抄袭程序较多,掉换函数定义顺序、加入冗余的变量声明、拆分语句都会对程序间的相似度产生较大影响,但是修改注释、改变变量名、重新排版不会影响 JPlag计算相似度.

BuaaSim的规一化过程中要将源程序编译生成可执行码,这就要求源代码必须是可编译的,而 JPlag没有这个限制,这是 BuaaSim不及 JPlag之处.

4 结束语

通过分析与实验可以看出,本相似性检测方法具有较强的抗干扰能力,可以有效消除学生常用的抄袭方法带来的噪声.本方法没有采用传统的基于属性或者结构度量的方法检测程序间的相似性,而是深入到程序的语义层面,引入编译优化技术和反汇编工具对源程序进行规一化,消除增加代码冗余、代码块重排序、控制结构等价替换等常规方法无法检测到的干扰;本文还给出了一个相似度经验阈值,根据该阈值给出了一个简单的聚类算法,可以有效地提取相似的程序子集.

相比其它系统,本系统的限制在于要求源代码必须是可编译的,而且编译优化后丢失了代码段的定位信息,只能进行整体的相似性比较,无法检测 2个程序内部代码段的相似性,不适合检测大规模的源程序的相似性.这也是本系统今后努力的方向,考虑使用插装技术,解决源代码与目标代码的定位问题.

参考文献 (References)

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

[2] 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. New York: Association for Computing Machinery, 2002: 183-187

[3] Aiken A. Moss: a system for detecting software plagiarism[EB/OL]. 2006-09[2008-04-05]. http://theory.stanford.edu/~aiken/moss/

[4] 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

[5] Emeric K, Moritz K. JPlag: a system that finds similarities among multiple sets of source code files[EB/OL]. 2005[2008-04-05]. http://www.ipd.uni-karlsruhe.de/jplag/

[6] Gitchell D, Tran N. Sim: A utility for detecting similarity in computer programs[C]//The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education. New York: Association for Computing Machinery, 1999: 266-270

[7] Wise M J. YAP3: improved detection of similarities in computer program and other texts[C]//Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education. New York: Association for Computing Machinery, 1996, 28(1): 130-134

[8] Jones E L. Metrics based plagiarism monitoring[C]//Proceedings of the Sixth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges. USA: Consortium for Computing Sciences in Colleges, 2001, 16(4): 253-261

[9] Verco K L, Wise M J. Plagiarism à la mode: A comparison of automated systems for detecting suspected plagiarism[J]. The Computer Journal, 1996, 39(9): 741-750

(责任编辑:文丽芳)