文章编号: 1008-1542(2011)02-0138-05

汇编语言程序相似性检测混合算法

石陆魁, 张 军, 陈 飞, 李金钊

(河北工业大学计算机科学与软件学院, 天津 300401)

摘 要: 根据汇编语言自身的特点, 提出了结合属性计数和结构度量技术的相似性检测混合算法。在该方法中, 将程序段的数目、子程序定义和调用的次数、循环指令 loop 出现的次数、转移指令出现的次数作为结构信息, 73 个使用频率较高的关键字作为属性信息。在从汇编语言程序中提取这些信息后, 利用卡方检验来判断 2 个程序的相似性。实验结果表明, 从混合算法得到的结果与人工检测的结果相一致, 优于从属性计数和结构度量技术得到的结果。

关键词: 汇编语言; 相似性检测; 抄袭; 属性计数; 结构度量

中图分类号: TP313 文献标志码: A

Hybrid algorithm of similarity detection for assembly language programs

SHI Lu-kui, ZHANG Jun, CHEN Fei, LI Jin-zhao

(School of Computer Science and Engineering, Hebei University of Technology, Tianjin 300401, China)

Abstract: Plagiarism often occurs in programming assignments submitted by students. Similarity detection techniques can help teachers find the suspicious plagiarism. Most similarity detection techniques use identical algorithm for different programming languages, which leads to the redundant checking algorithms and degrades the checking accuracy. In this paper, a hybrid algorithm of similarity detection adapting to the characteristic of assembly language was presented, which combined attribute counting with structure metrics technique. In the algorithm, the number of paragraphs, the number of definition and calling of subroutines, the number of loop and branch occurrences in assembly programs were extracted as the structure information. And 73 high frequent keywords were taken as the attribute information. The similarity of two programs was judged with the chi-square test after getting the attribute and structure information. Experiments demonstrated that results from the proposed algorithm were consistent to those from the manual check. The hybrid algorithm was superior to the methods based on attribute counting and structure metrics.

Key words: assembly language; similarity detection; plagiarism; attribute counting; structure metrics

在程序设计课程中, 上机作业多以电子版的形式提交, 由于电子版易于复制和修改, 经常出现抄袭现象。这种抄袭行为不仅严重影响了教学效果, 而且也影响了成绩的公平性。人工评判方式主要依靠教师的短期记忆来判断程序代码是否存在抄袭问题, 并借助一些文本比较工具进行判断。这种传统方式不仅要花费教师大量的时间, 而且由于人的短期记忆性等问题出现误判、遗漏的可能性较大, 同时检测结果的好坏与检测者经验和代码规模也有很大的关系。而程序代码相似性检测技术正好可以帮助教师从大量的学生作业中找出存在抄袭嫌疑的作业对象, 从而减轻教师的工作负担并提高工作效率。

收稿日期: 2010-09-02; 修改日期: 2011-01-06; 责任编辑: 李 穆
基金项目: 天津市应用基础及前沿技术研究计划(10JCZDJC16000)
作者简介: 石陆魁(1974-), 男, 河北邯郸人, 副教授, 主要从事数据挖掘方面的研究。

1　程序相似性检测技术概述

程序相似性检测技术的研究最早可以追溯到 20 世纪 70 年代，OTTENSTEIN[1] 基于 HALSTEAD 软件度量方法[2] 提出了用属性度量的方式检测代码的相似性。程序代码相似性检测是指通过计算机计算 2 个程序文件之间的接近程度，根据计算结果在一定范围内给出 1 个度量值，并根据这个值进一步检测程序代码是否存在抄袭。现在，已有很多的软件工具来辅助教师检测代码的相似性[3-4]。在这些系统中，都使用一些相关的算法来检测程序的相似性。

2　程序相似性检测算法

目前，程序相似性检测算法可以分为 4 种：第 1 种是基于属性计数的检测算法，其基本思想是统计程序中一些特征值的数量，例如某种操作符、变量或循环体、判断体所出现的频率；第 2 种是基于结构度量的检测算法，主要判断程序是否具有相同的结构；第 3 种是属性计数和结构度量混合检测算法，它结合前 2 种判定方法，给出一个综合的判定；第 4 种是使用其他技术的一些算法，如使用静态语法树、神经网络、优化编译[5-6] 等方法来比较代码的相似性。

在这些算法和系统中，针对不同的编程语言使用相同的检测算法和技术[7-8]。然而由于不同的编程语言有不同的特点，造成程序设计语言编程模式和语法的多种多样，导致检测算法具有很大的冗余性，检测结果也常常不尽如人意。笔者根据汇编语言自身的特点，结合属性计数和结构度量方法的优点，提出了一种适合汇编语言程序相似性检测的混合算法，效果较好。

2.1　程序抄袭的定义及常用手段

代码抄袭除了对别人的代码直接进行拷贝外，还包括对他人代码无关紧要的地方稍作修改。PARKER 等定义代码抄袭为一个程序通过对另一个程序的少量简单修改而得到[9]。比较常见的修改主要包括：1）添加、删除或修改注释和空白；2）改变代码的行数和格式；3）重新命名标识符或改变数据类型；4）改变代码块的顺序及代码内部语句的顺序；5）改变表达式中操作数或操作符的顺序；6）添加冗余语句或变量；7）将某一控制结构用等价的控制结构替代[10]。

FAIDHI 等将代码变换的难度分为 6 个等级[11]：修改注释和空白，修改标识符，修改变量的位置，合并过程，修改程序语句和修改控制逻辑。这 6 个等级由易到难，对前 4 种难度等级，程序的修改不需要理解代码，这种程度的抄袭无法达到教学效果，应该禁止；而对后 2 种难度等级，需要对课程有一定程度的理解，并在对代码结构、内容等进行了深入理解的基础上才能做出相应的修改，这样的修改具有较大的难度，单单对于教学而非科研创作来说已经是可以接受了。因此，对于学生作业来说做到判断是否为前 4 种的代码变换即可。

2.2　基于属性计数的相似性检测算法

基于属性计数的检测方法从源代码中抽取各种度量元素，如关键字数、操作符数、循环数等，不考虑程序的内部结构。HALSTEAD 提出的软件科学度量方法是最早和最典型的属性计数法[2]，许多相似性检测算法都是基于这一软件科学理论，它们从程序中提取出数个软件度量特征，并使用这些特征来比较程序，其提出的属性计数法中统计如下 4 个属性：

N_1＝所有操作符的总数；N_2＝所有操作数的总数；n_1＝操作符的种类数；n_2＝操作数的种类数。

由此定义程序的长度 $N=N_1+N_2$ 和词汇量 $n=n_1+n_2$，并计算程序的“容量 V”如下：

$$V=N\mathrm{lb}n。\tag{1}$$

最后将以上信息表示成 HALSTEAD 特征向量，即 $\boldsymbol{H}(n, N, V)$。通过计算 2 个特征向量之间的欧几里德距离即可得到最终的相似度。计算公式见式(2)：

$$D=\mathrm{sqrt}((n_1-n_2)^2+(N_1-N_2)^2+(V_1-V_2)^2)。\tag{2}$$

欧几里德距离越大，说明两者差异越大；反之，说明两者越相似，抄袭嫌疑就越大，2 个相同程序的相似度为 0。

2.3　基于结构度量的相似性检测算法

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。而基于结构度量的相似性检测方法则要对程序的内部结构进行分析比较，如分析控制结构、计算代码嵌套深度、分析数据依

赖关系等。MCCABE提出的圈复杂度方法[12]是典型的结构度量技术。

圈复杂度方法通过计算执行路径的数量来衡量1个程序中的控制流。这种方法要求预先对程序代码进行分析,给出程序的控制流图。控制流图中的每条控制流线或弧表示1个可能的执行路径,每个节点表示1个处理语句或1个判断的入口。定义圈复杂度为[12]

$$V(G)=e+n+2p。\tag{3}$$

式中:e表示控制流图中的边数;n表示控制流图中节点数;p表示控制流图中的模块数。圈复杂度给出了程序的一个结构特征,它往往需要跟其他特征结合来表示一个程序。

3 适合汇编语言程序相似性检测的混合算法

目前,多数程序相似性检测算法都没有考虑编程语言自身的特点,然而不同的编程语言使用相同的检测算法和技术,会导致检测算法具有很大的冗余性。汇编语言都有其自身的特点,在程序结构、数据存储、结构化等方面都与高级语言存在较大的差别,设计适合汇编语言程序自身特点的相似性检测算法将极大地减少算法的冗余程度,并提高检测算法的精度。

3.1 获取结构信息

不同于高级语言,汇编语言不是一种完全结构化的程序设计语言,没有明显的分支结构和循环结构。多数情况下循环不是通过循环语句实现的,而是通过转移语句实现的,这些都使得对结构度量的分析难度加大。但是,汇编语言程序中程序段的分配、子程序的定义和调用非常明显,转移指令使用频率较高,数据存取主要针对寄存器操作。因此,为了从程序中提取结构信息,必须简化现有的基于结构度量的相似性检测算法,使其适合汇编语言的特点。

由于汇编语言自身的特点,从汇编语言程序中自动地提取所有的结构信息较为困难,因此,只选择一些关键点进行比较。这些关键点包括程序段的分配、子程序的判别、转移指令“jxx”出现的次数、循环指令“loop”出现的次数等。

1)判断程序段的数目和类型是否一致。例如,如果一个程序中包括4个程序段,而另一个程序只包含3个程序段,那么可以判断2个程序抄袭概率较小。又如,如果一个程序使用了堆栈段,而另一个没有使用,那么也可以判断2个程序抄袭概率较小。

2)判断子程序定义和调用是否一致。统计子程序定义的个数和子程序调用的次数,如果统计结果相差较大,那么可以判断2个程序抄袭概率较小。

3)判断所有转移指令“jxx”出现的次数是否一样。如果2个程序中所有转移指令出现的次数相差较大,那么可以判断2个程序抄袭概率较小。

4)判断循环指令“loop”出现的次数是否一样。如果相差较大,那么可以判定2个程序的设计方法不一样,抄袭概率较小。

通过分析汇编语言程序的关键点,就可以统计出相关的结构信息。如果这些结构信息都非常相近,那么说明2个程序可能较为相似。

3.2 获取属性信息

对汇编语言进行属性度量并进行相似性检测,关键在于属性的确定。汇编语言中有较多的指令,也就是关键字较多,这些关键字都可以作为属性进行统计,而属性个数的增加无疑会提高程序检测的准确性。在本文中,采用汇编语言中使用频率较高的73个关键字,如“mov”,“pop”,“push”,“jmp”等作为属性进行统计。这些关键字不包括与结构信息有关的“segment”,“call”,“loop”,“proc”等。

3.3 相似性检测混合算法

将前面提取的结构信息和属性信息结合起来,得到混合算法。为了将二者结合在一起,需要将结构信息数值化,也就是统计程序中的程序段的数目、子程序的个数、子程序调用的次数即“call”指令的个数、所有转移指令“jxx”的个数、循环指令“loop”的个数等,将这些信息和属性信息一起作为程序的度量特征来检测相似性。得到这些信息之后,如何检测2个程序的相似性呢?

在传统的属性计数方法中,通过计算2个程序的属性之间的距离,如欧式距离或夹角余弦,来判断2个程序的相似性。在得到结构和属性信息后,笔者采用统计学中的卡方检验来判断2个程序的相似性[13]。设

共有 n 个结构和属性项，每个程序提取结构和属性信息后可表示为 1 个 n 元特征向量 $\boldsymbol{P}_i=(p_{i1}, p_{i2}, \cdots, p_{in})$，2 个程序分别用 P_i 和 P_j 表示，则 2 个样本的卡方可表示为

$$\chi^2=\sum\frac{(P_i-P_j)^2}{P_j}=\sum_{k=1}^{n}\frac{(p_{ik}-p_{jk})^2}{p_{jk}}。\tag{4}$$

对 2 个程序 P_i 和 P_j，设相同统计项的值不完全为 0 的项数为 m，则自由度 k 可设为 $m-1$。确定自由度 k 后，在 0.05 的显著性水平下，通过查表作为参考，可得出 2 个程序是否相似。

由于汇编语言程序中可能会存在一些注释和空白，而这些又不影响程序的功能，但会影响相似性检测，因此，需要对这些信息进行处理，即对源程序文件进行格式化处理，去掉与程序功能无关的内容。具体检测算法如下。

1)程序内容格式化。由于汇编语言程序符号的特定化，直接阅读源程序比较困难，故而多数汇编语言程序的注释较多。同时一些学生通过修改注释、空白和大小写等来隐蔽抄袭行为。因此，需要先对程序进行格式化，去除与程序功能无关的内容，格式化内容包括去除注释和空白，统一大小写等。

2)去除完全相同的程序。在程序格式化之后，找出完全相同的程序，在每个完全相同的程序组中选出 1 个程序和剩下的程序作为新的样本集进行相似性检测。

3)对上一步骤中得到的样本集中的每个程序提取结构和属性信息，其中属性度量包括 73 个基本属性值的计数统计，结构度量包括程序段的个数、子程序的个数、子程序调用的次数、所有转移指令“jxx”出现的次数、循环指令“loop”出现的次数，并将每个程序表示成 1 个特征向量。

4)根据上一步骤中得到的统计信息，利用式(4)计算每对程序的卡方值，并确定相应的自由度。

5)产生结果。在 0.05 为显著性水平下，通过查表确定在与 2 个程序相同的自由度 k 下的卡方值，并将此值作为界限。如果 2 个程序之间的卡方值大于该界限，则 2 个程序存在抄袭的可能性较小；反之，则 2 个程序存在抄袭的可能性较大。抄袭具有传递性，如果程序 P_i 和 P_j 相似，P_j 和 P_k 相似，则 P_i，P_j 和 P_k 相似。

4　实　验

为了测试所提出的混合算法的性能，利用学生所提交的汇编语言实验程序进行测试。在实验中，将所提出的算法与基于属性计数的方法和基于结构度量的方法进行了比较。

实验分为 2 组，每组都针对一个特定的题目。一组实验以一个班所提交的同一题目的汇编语言程序为样本，共包括 25 个长度不同的程序，将程序统一按照“Pi”的方式命名，实验结果如表 1 所示。另一组实验样本也包括 25 个长度不同的程序，这 25 个程序不是针对一个班级，而是从一个年级所提交的有关同一题目的程序中任意选取。将程序统一按照“Si”的方式命名，实验结果如表 2 所示。

表 1　第 1 组程序的测试结果

Tab.1　Test results of the 1st group of programmes

算法	检测结果
基于结构度量的算法	{P12}，{P23}，{P1, P2, P4, P5, P7, P8, P9, P10, P14, P15, P16, P18, P19, P20, P21, P22, P24, P25}，{P3, P6}，{P11, P13, P17}
基于属性度量的算法	{P2}，{P7}，{P1, P3, P4, P5, P6, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P21, P22, P23, P25}，{P19, P20, P24}
相似性检测混合算法	{P7}，{P18}，{P1, P2, P3, P4, P5, P6, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P21, P22, P23, P25}，{P19, P20, P24}

在第 1 组实验中，结构度量算法将 25 个程序分为 5 组，属性度量算法和混合算法分为 4 组。在结构度量方法的结果中，程序 P12 和 P23 不相似且与其他程序相似性较小；同样程序 P3 和 P6 相似而与其他程序不相似。实际上通过人工检测发现这 4 个程序和 P1，P2 等其他 16 个程序非常相似。程序 P11，P13 和 P17 的情况类似。在属性度量算法的结果中，程序 P2 与其他程序相似性较小，实际情况是 P2 与 P1，P3 等 19 个程序非常相似；而程序 P18 实际上与其他程序相似性较小，在该算法的结果中却与 P1，P3 等相似。混合算法得到的结果与人工检测结果一致。该组程序由于是一个班级中的同学所提交，相互之间抄袭的可能性更大，实验结果也说明了这一点。

表 2 第 2 组程序的测试结果

Tab. 2 Test results of the 2nd group of programmes

算法	检测结果
基于结构度量的算法	{S2}, {S8}, {S9}, {S11}, {S16}, {S20}, {S21}, {S01, S14, S15}, {S3, S6, S7, S13, S23}, {S4, S17, S18, S19, S24, S25}, {S5, S10, S12, S22}
基于属性度量的算法	{S2}, {S3}, {S8}, {S9}, {S11}, {S14}, {S16}, {S17}, {S20}, {S01, S15}, {S4, S18, S19, S24, S25}, {S5, S10, S12, S22}, {S6, S7, S13}, {S21, S23}
相似性检测混合算法	{S2}, {S8}, {S9}, {S11}, {S16}, {S20}, {S1, S14, S15}, {S3, S6, S7, S13}, {S4, S17, S18, S19, S24, S25}, {S5, S10, S12, S22}, {S21, S23}

在第 2 组实验中, 属性度量方法将 25 个程序分为 14 组, 结构度量方法和混合方法分为 11 组。在该组实验中存在与第 1 组实验相同的情况。例如, 在结构度量方法的结果中, 程序 S21 与其他程序相似性较小, 通过人工检测发现它与程序 S23 非常相似。又如, 在属性计数方法的结果中, 程序 S3 与其他程序相似性较小, 实际情况是它与程序 S6, S7, S13 非常相似。混合算法得到的结果与人工检测结果相一致。由于该组程序是从一个年级中随机地选出, 抄袭情况没有第 1 组实验严重。

从测试结果可以看出, 在 2 组实验中用混合算法得到的结果与人工检测结果一致, 表明混合算法优于属性度量算法和结构度量算法。显然, 由于结合了汇编语言自身的特点, 检测算法的检测精度得到提高。

5 结 语

由于学生经常以电子版的形式提交程序作业, 出现抄袭现象时, 抄袭检测系统可以帮助教师发现可疑的抄袭对象。目前, 在多数抄袭检测系统中, 针对不同的编程语言都采用相同的相似性检测算法, 造成检测算法具有较大的冗余。针对汇编语言自身的特点, 提出了一种结合属性度量和结构度量技术的相似性检测混合算法。该方法从汇编语言程序中选择一些关键点作为结构信息, 如程序段的数目, 子程序定义和调用的次数、循环和分支出现的次数等, 并且将汇编语言中使用频率较高的 73 个关键字作为属性信息。每个程序用结构信息和属性信息表示成一个特征向量, 获取所有的特征向量后用卡方检验来判断 2 个程序的相似性。实验结果表明从混合算法得到的结果与人工检测的结果一致, 且算法的检测精度得到提高。

参考文献:

[1] OTTENSTEIN K J. An algorithmic approach to the detection and prevention of plagiarism[J]. Sigcse Bulletin, 1976(8): 30-41.

[2] HALSTEAD M H. Elements of Software Science[M]. Amsterdam: Elsevier Press, 1977.

[3] PRECHELT L, MALPOHL G, PHILIPPSEN M. Finding plagiarisms among a set of programs with JPlag[J]. Journal of Universal Computer Science, 2002, 8(11): 1 016-1 038.

[4] WISE M J. YAP3: Improved detection of similarities in computer program and other texts[A]. Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education[C]. New York: Association for Computing Machinery, 1996.

[5] 熊 浩, 晏海华, 赫建营, 等. 一种基于静态词法树的程序相似性检测方[J]. 计算机应用研究(Application Research of Computers), 2009, 26(4): 1 316-1 320.

[6] 赵长海, 晏海华, 金茂忠. 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(Journal of Beijing University of Aeronautics and Astronautics), 2008, 34(6): 711-715.

[7] CLOUGH P. Plagiarism in Natural and Programming Languages: An Overview of Current Tools and Technologies[R]. Sheffielel: The University of Sheffield, 2000.

[8] GOEL S, RAO D. Plagiarism and Its Detection in Programming Languages[R]. [S. l.]: JIITU, 2008.

[9] PARKER A, HAMBLEN J. Computer algorithms for plagiarism detection[J]. IEEE Transactions on Education, 1989, 32(2): 94-99.

[10] WHALE G. Identification of program similarity in large populations[J]. The Computer Journal, 1990, 33(2): 140-146.

[11] FAIDHI J A, ROBINSON S K. An empirical approach for detecting program similarity within a university programming environment[J]. Computers and Education, 1987, 11(1): 11-19.

[12] MCCABE T J. A complexity measure[J]. IEEE Transactions on Software Engineering, 1976, SE-2(4): 308-320.

[13] 张 莉, 周祖林. 代码相似性检测在程序设计教学中的应用[J]. 计算机教育(Computer Education), 2009(13): 116-118.