A System for Automatic Assessment and Plagiarism Detection of Student Programs

Changhai Zhao, Haihua Yan
School of Computer Science and Engineering
Beihang University
Beijing, China
zch@buaa.edu.cn, yhh@buaa.edu.cn

Cong Song
Graduate School
Beihang University
Beijing, China
songcong@buaa.edu.cn

***Abstract*—We describe a web-based system which supports automatic assessment and plagiarism detection of student programs. This system employs dynamic testing, efficiency testing, static code checking and program characteristic checking to analyze student program. It provides detailed and constructive feedback to students. An approach based on compiling optimization and disassembling is proposed to detect similarity in student 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 students are urged to do their homework independently. The system can lighten the teacher's burden and do the work which is impossible for the traditional teaching. We achieved great teaching effect by using the system in the last several years.**

Keywords-automatic grading; student program; plagiarism detection; automatic assessment

I. 引言

程序设计是计算机专业学生的必修课之一，也是进入IT行业必备的技能。程序设计是一个多层次的知识体系，完全通过课堂教学并不能使学生领会程序设计的内涵，必须通过大量的编程练习去感悟。练习过程中，教师应对学生程序给予及时的指导，逐步培养其良好的程序设计风格与解决实际问题的能力。但现实情况是，由于每届学生人数太多(多数高校超过 200 人)，评判学生练习往往要耗费教师大量的精力与时间，导致学生无法得到及时的指导，练习的次数也受到限制。

传统的程序设计通常采用纸质考试，学生只需要通过手写程序片段来答题，并不是在一个真实的开发环境编写程序，这就会出现有些学生虽然动手能力不强，但靠临考前的强化记忆，也可能会获得高分的问题；另外由于程序本身的特殊性，教师在批改的时候往往很难判断程序的正确性，导致得分的主观性比较强，特别是多人参与阅卷的时候，评估者间信度[1,2](inter-rater reliability)是一个相当严重的问题，同一份试卷不同的阅卷人得出的分数不一致，失去了考试的客观性。

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[3,4]。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。人工查找抄袭的工作量太大，必须借助于计算机。

CourseGrading 系统[5]是北京航空航天大学教改资助项目，它的主要特色是支持学生程序的自动评判与抄袭检测，2005 年开始使用至今，已有两千多人使用过该系统，此系统已应用于高级语言程序设计、算法与数据结构等课程，学生在该系统上进行作业练习与考试；研究生复试也使用该系统测试研究生的编程能力。学生通过系统提供的大量练习，提高了动手能力和编程水平，基本杜绝了互相抄袭的现象；教师也从繁重的批改程序的任务中解放出来，腾出更多的时间通过系统的答疑论坛与学生进行交流。

本文贡献在于：运用动态测试、运行效率测试、程序特征识别、代码静态检查等技术对学生程序进行分析，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。

本文首先介绍程序自动评判方法与抄袭检测方法，以及我们的应用与开发经验，结合实际数据分析该系统的应用效果，最后总结此类系统的发展方向。

II. 相关研究

从编程语言进入大学课堂，就有科研人员进行程序自动评判的研究与开发工作，美国北科罗拉多大学的 Isaacson 与 Scott[6]介绍了一个学生程序自动评判工具，学生需要将程序提交到服务器指定的目录内，服务器端的 Shell 脚本程序自动对该目录内的源文件进行编译，然后采用黑盒测试方法进行测试，将测试结果写入指定的文件。Dawson-Howe 在文献[7]中描述了另外一个基于黑盒测试的自动评判工具，学生通过 Email 提交源程序，评判结果也通过 Email 返回。WebToTeach[8]是基于 Web 的程序自动评判系统，采用黑盒测试方法，该系统支持在线考试，但不支持抄袭检测。浙江大学和北京大学开发的

978-0-7695-3997-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ICEE.2010.909

Online Judge[9,10]也可以对程序进行自动评判，该系统也是基于黑盒测试方法对程序进行正确性测试，但因为该系统是 ACM 国际大学生编程竞赛训练平台，融入了很多竞赛的规则，并不适合程序设计课程使用。ASSYST[11]是利物浦大学计算机系的 David Jackson 和 Michelle Usher 开发的基于 C/S 结构的自动评分系统，它从正确性、程序运行效率、复杂度和程序风格方面对学生程序进行评判，相比以前只采用黑盒测试对学生程序评判的系统，有了很大的进步，但该系统不支持抄袭检测，也不支持在线考试。BOSS[12]在线提交系统由英国沃里克大学(University of Warwick)资助开发，支持的评判手段包括：正确性测试，程序风格度量，程序抄袭检查。该系统只支持 Java 语言的自动评判，最新的版本集成了 JUnit 测试框架，将测试驱动开发引入课堂教学，BOSS 系统的程序抄袭检查过程需要人为调整相似度，能够检查出的抄袭手段很有限。CourseMaster[13](现已改名为 CourseMarker)是一个 C/S 结构的学生程序自动评判系统，可以评判 C、C++和 Java，支持的评测手段非常丰富，包括：程序正确性，程序风格，复杂度，程序特征识别，代码静态检查和抄袭检测。但录入题目的时候，需要编写很多配置文件，易用性方面有待提高。

抄袭检测方面的研究也非常丰富，Jones[14]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两种手段，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[15]；后来出现了基于结构度量的相似性检测方法，MOSS[16]、JPlag[17]、SIM[18]和 YAP3[19]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。

III. 评判与抄袭检测流程

只有可编译执行的源程序才能被自动评判，评判与抄袭检测流程如图 1所示，学生将源程序提交给系统，系统首先编译源程序，如果编译通过，则进行动态测试，即运行程序，使用预先设定好的测试数据检验程序的正确性，根据学生程序通过测试数据的个数，系统自动打分；如果动态测试期间程序崩溃或者死循环，向学生反馈执行期间错误原因，否则，继续对程序进行运行效率测试、静态代码检查和程序特征识别，最后生成评判报告。

教师可以对学生提交的源程序进行抄袭检测，并将同一来源的代码聚类，抄袭检测结果只供教师参考，并不参与自动评分。

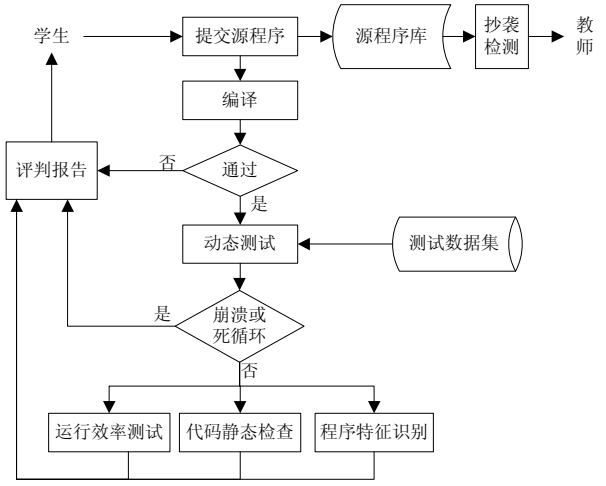

图 1 程序自动评判与抄袭检测流程

IV. 自动评判方法

A. *题目类型*

我们设计了三类编程题目，覆盖当前传统纸质考试中出现的题目类型，根据编程约束由弱至强分别是：题目描述编程、接口编程和程序片段编程，这三种题目类型增加了教师编写题目的灵活度。

题目描述编程，是根据题目描述，编写一个完整的可编译执行的程序。题目描述形式如图 2 所示，学生要严格按照题目要求编写程序。

接口编程的灵感来自软件开发过程中的单元测试(unit testing)理论[20]，传统的结构化编程语言，比如C，学生需要实现一个函数；Java或者C++这样面向对象的语言，学生需要实现一个接口或者继承一个抽象类。教师编写的题目的时候，除了题目描述，还要提供头文件和main函数，这种类型的题目对学生编写程序提供了导向与约束的作用，一方面可以训练学生对函数或者类有更深入的了解，此外，数据结构课程可以利用这种约束作用，限定学生用某种类型的数据结构编写代码。

【问题描述】编写一个函数isprime(n).判断整数n是否为素数.编写程序使用此函数,当输入一个整数时,对它进行判断,当为素数时,输出1.否则,输出0.
【输入形式】控制台输入一个整数.
【输出形式】控制台输出判断结果0或者1.
【样例输入】45
【样例输出】0
【样例说明】45非素数,故输出为0
【评分标准】该题要求输出一个字符，答对得满分10分,否则得0分.上传C语言文件名为isprime.c

图 2 编程题目描述

程序片段编程，即补充源程序中缺失的代码段，代码段可以是一个表达式、语句、也可以是一个函数实现。系统根据代码段前后的调试打印语句或程序最终的输出结果评判程序的正确性。一般情况下，程序片段编程类的题目难度较小，在作业或考试中，可以与其它两种类型题目搭配。

B. *动态测试*

所谓动态测试，是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等，这种方法由三部分组成：构造测试实例、执行程序、分析程序的输出结果。动态测试是判断程序正确性的唯一途径，系统根据学生程序通过的测试数据个数进行打分。为了能够进行动态测试，需要预先设定一些约束，图 3展示了录入题目时需要填写的程序约束以及测试数据，接下来的4个小节分别介绍这些约束以及它们在自动评判过程中所起的作用。

图 3 程序题目输入

1) *输入输出形式*

对于没有图形界面且不存在交互的程序，共存在三种输入形式：标准输入、文件输入与命令行参数，输出存在两种形式：标准输出与文件输出，根据程序的输入输出不同可以将程序划分为6种类型，系统执行动态测试时会根据输入输出类型调用相应的例程(routine)，执行待测程序。由于待测程序脱离了控制台或者终端，具有标准输入的程序必须在待测程序执行前将标准输入重定向到测试数据所在的文件；文件输入相对简单，只需要将测试输入数据拷贝至待测程序的执行路径下；命令行参数输入则是在待测程序作为子进程启动时将测试数据作为命令行参数。待测程序的标准输出必须重定向到临时文件内，执行完毕后，临时文件内的输出内容与期望输出进行对比，判别程序的正确性。

2) *标准测试集录入*

测试数据的录入一直是一个难题，现有的自动评判系统都没有很好的解决这个问题，多数系统要求用户直接编辑测试数据，然后上传至服务器，这种方式需要用户了解评判的细节，才能编辑出正确格式的测试数据，可维护性和可移植性比较差；CourseMarker为用户提供了一个测试数据编辑模板，易用性方面有所改进。标准测试数据录入难点在于：一方面生成的测试数据文件必须符合特定的命名规则，才能被评判系统识别；此外，测试数据不仅仅包括前面介绍的6种形式的输入输出，可能还需要一些中间文件，例如，一个题目需要根据标准输入提供的密钥，对文件内的数据加密，然后输出加密后的内容。

本系统解决了标准测试数据录入难题，提出用XML来描述测试数据，在XML之上建可视化的编辑器，方便用户录入，编辑好的XML文件再转换为原始的测试数据，图 3展示的是一个使用XML表达的具有命令行输入文件输出，且有中间文件的测试数据，使用XML描述测试数据不仅方便移植，而且很容易编辑和修改。

3) *程序异常处理*

学生程序运行时，可能会发生运行时间过长甚至者崩溃的情况，评判系统需要对这两种异常情况进行处理。运行时间过长的主要原因是程序陷入了死循环，另外一个原因是程序性能非常差，不管是哪种原因必须及时终止程序运行，否则会增加服务器的负载，影响其它程序的评判。程序的最长运行时间由教师设定，如图 3所示，超过这个时间，进程被强行终止。

Linux下，进程崩溃时都会释放信号，如SIGFPE、SIGSEGV和SIGBUS等，评判系统内安装了信号处理函数，可以在运行时捕捉学生程序发出的信号，并对这些信号进行解释，方便学生查找崩溃原因，C语言中，引发学生程序崩溃最常见的原因是数组越界，其次是对指针的不当操作。

4) *输出格式问题*

题目描述中都会规定程序的输出格式，如果将学生程序的输出结果严格地与期望结果进行匹配，将会出现误判，即打印出来的结果与期望结果看不出差别，然而评判系统却判定程序输出错误。我们曾经发现学生在输出结果的最后打印了一个回退符“\b”，评判系统认为与期望结果不一致。

本系统支持两种输出结果匹配标准：完全匹配与内容匹配，完全匹配是将输出结果的行首与行尾的空格字符和

控制字符等不可见字符删除之后与预期结果匹配；内容匹配则将输出结果内所有的不可见字符删除，然后与预期结果的内容进行比较。教师可根据需要设定两种匹配标准的分值。

C. 运行效率测试

程序的运行效率包括时间效率和空间效率，它们可以通过让程序执行若干次，根据占用内存和运行时间的均值来度量，这两个数据在程序进行动态测试的过程中得到。程序的运行效率不仅仅与程序的算法复杂度、程序结构、服务器的体系结构，还与当时服务器的状态有关，即使是同样的测试数据，运行若干次，每次得到的运行时间也不会相同，所以运行效率并不适宜参与自动评分，但是可以给教师提供参考，由教师根据实际情况对最后得分进行调整。一种更加科学的运行效率度量策略参看Randal E.Bryant 和 David R. O'Hallaron 提出的 K 次最优测量方法[21]。

现在多数程序员重视程序的功能实现，但是忽略程序的性能[22]，为了训练学生性能调优的技能，本系统在执行完动态测试后，调用性能优化工具gprof[23]和OProfile[24]分析程序的cache命中率、分支预测、函数调用频率和函数执行时间等信息，帮助学生定位性能问题，培养他们优化程序性能的意识，掌握性能优化方法。

D. 程序特征识别

针对某一道编程题目，教师经常要求学生编写的程序具备某些特征，例如要求学生在代码中使用for循环，而不能用while循环；使用数组，不能使用指针；或者采用多进程技术，而不能用多线程。本系统采用正则表达式构建“包括/排除(include/exclude)”规则，对于不符合要求的程序，系统会自动扣分。

E. 代码静态检查

静态代码检查是指不运行被测试程序而寻找程序代码中可能存在的错误或评估程序代码的过程。静态测试的特点是不需要运行代码，也不需要对代码编译、连接和生成可执行文件[25]。代码中有些错误通过动态测试是无法暴露出来的，例如下面的代码：

```
int main(int argc, char* argv[]) {
   int   b = 0x12345678;
   char a[8];
   a[sizeof(a)] = 0xff;
   printf("b=0x%x\n", b);
   return 0;
}
```

在给变量a赋值时，发生了数组越界存取，程序没有崩溃，打印出来的b的值变成了0x123456ff，而不是原来的0x12345678。

代码静态分析可以对代码进行全局分析，能识别没有被适当检验的数组下标、报告未被初始化的变量、警告使用空指针、冗余的代码，等等。系统中使用Splint[26]等静态代码分析工具对学生代码进行检查，作为动态测试的补充，教师可以设定检查出问题数量的上限，如果超过则扣除一定的分数。

代码静态检查可以督促学生写规范的、安全的代码。

V. 抄袭检测方法

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：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：加冗余变量、拆分语句、调换句顺序等)。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。本系统提出了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生成特征文本，如图 5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 4所示，详细论述请参本文作者的论文[27]。

抄袭检测一般是在作业或者考试结束后执行，检测出来的结果不参与自动评分，由教师核对后酌情处理。

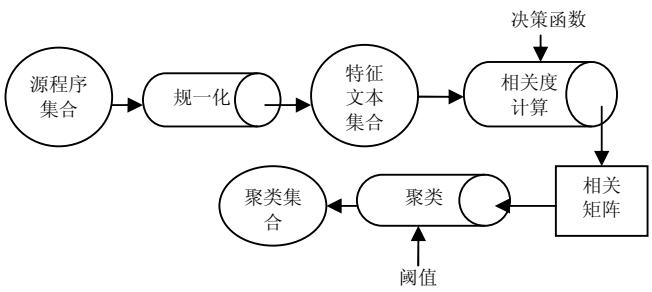

图 4 基于编译优化的程序相似性检测方法

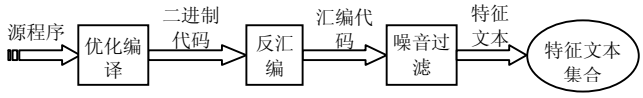

图 5 规一化过程

VI. 经验教训

开发一套能够在线考试的自动评判和抄袭检测系统挑战性非常大，根据几年的开发、维护和应用经历，我们总结了三点经验教训：

第一，性能非常关键。系统最初应用时，北航一届计算机学院的学生加上辅修计算机的学生将近350人，由于项目初始资金有限，只能采用普通的PC机做服务器，所有人同时在线考试的时候，要保证每个人不会感到系统的延迟，系统一旦有延迟就可能会影响学生的答题情绪，还要考虑延长考试时间；更不能因为集中的并发访问发生系统崩溃，一旦系统崩溃，将会导致教学事故。系统在设计与开发过程中必须采取相应的性能优化策略。

第二，系统安全性。学生提交的程序在服务器上编译执行，要防止服务器被植入木马或者系统中保存的数据被

破坏。学生成绩的安全性也非常重要，我们的系统设立四种角色：系统管理员、教师、助教和学生，必须采用一些必要的技术手段限制助教和学生的权限，保护密码的安全。

第三，源文件的组织与管理。每一届学生都要向服务器提交大量的源文件，例如北航每届学生要提交将近20000份源文件，这些源文件必须长期保存，所以要合理组织和管理这些源文件，一方面避免组织不当影响系统性能，另外要保证存储的可靠性，建议对过期的源文件进行打包压缩，减少磁盘空间占用。

VII. 应用效果

本文以高级语言程序设计课程为例介绍系统的应用情况，每届学生需要完成8次作业，每次作业有12道选择题和3道编程题；期中考试要求学生两个小时内完成10道简答题和2道编程题；期末考试要求学生在三个小时内完成20道简答题，3道编程题。

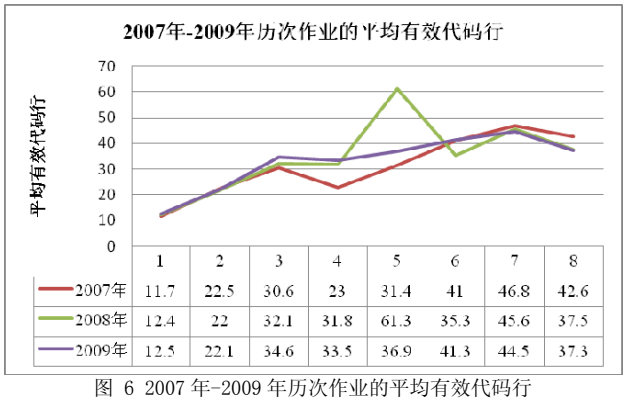

图 6 2007 年-2009 年历次作业的平均有效代码行

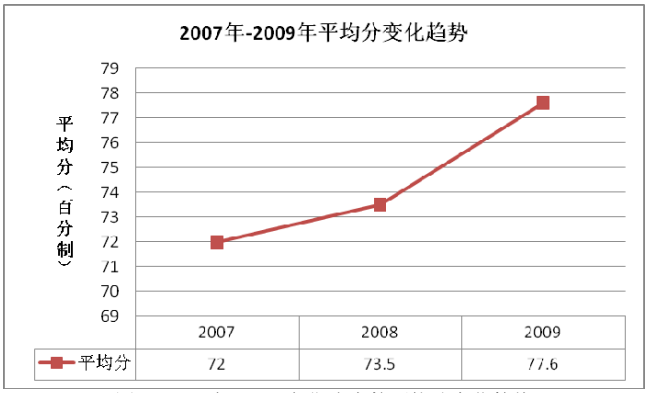

图 7 2007 年-2009 年期末考核平均分变化趋势

图 6展示了最近三年学生8次作业编写的有效代码行，平均代码行数维持在11~60行，随着课程的深入，学生编写的代码行数大体上也在增加。每一届的第一次作业，系统检查出来抄袭的程序相当多，课堂警告之后，接下来的几次作业基本可以杜绝抄袭现象，学生的学习态度明显好转，在线答疑积极踊跃。

近三年的期末考核平均分呈现显著上升的趋势(如图 7所示)，学生考试中编写的有效代码行数也呈上升趋势(如图 8所示)，有效代码行数基本上与编程题目难度成正比，这一现象说明，近几年题目越来越难，学生成绩反而越来越好。其他课程的教师也反映学生的编程水平在提高。经过我们的调查，主要的原因是系统应用以来，在学生当中逐渐形成一种舆论氛围，即高级语言程序设计这门课程考核严格，只能认真学习，无法蒙混过关。此外，学生能够即时得到自己编写的程序的反馈信息，产生了成就感，激发了学习兴趣。

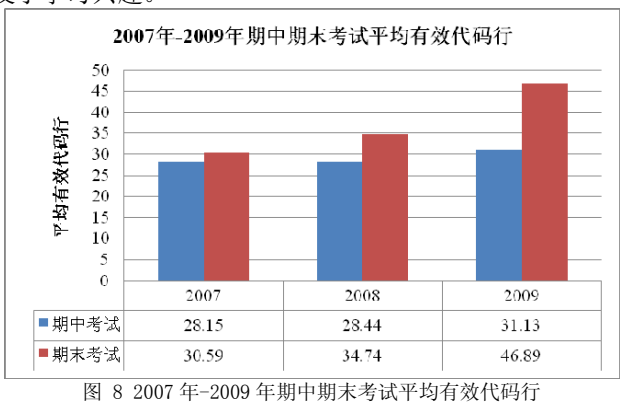

图 8 2007 年-2009 年期中期末考试平均有效代码行

VIII. 总结与展望

本文介绍的程序自动评判与抄袭检测系统利用动态测试、运行效率测试、程序特征识别、代码静态检查等技术对学生程序进行评测，向学生反馈丰富的且有建设性的信息；系统采用的一个基于编译优化和反汇编的程序相似性检测方法，可以准确的检测出抄袭代码，从根本上消除了学生的侥幸心理，督促他们踏实、独立地完成作业。本系统一方面减轻了教师的工作负担，完成传统教学中相当繁琐甚至根本无法进行的工作，另外一方面提高了教学质量。

本系统的发展方向是支持更多课程的自动评判和抄袭检测，目前系统支持中文文档的相似性检测与互联网相似文档搜索，能够对常用的算法进行自动评判，我们正在研究并行程序的自动评判方法。

REFERENCES

[1] Xiaomin Sun, Houcan Zhang, A Comparative Study on Methods Used in Estimating the Inter-rater Reliability of Performance Assessment--Methods Based on Correlation, Percentage of Inter-rater Reliability and the Generalizability Theory, Psychological Science, Vol 28, pp.646-649, 2005(In Chinese).

[2] Jon A. Preston and R.Shackelford, “Improving on-line assessment: an investigation of existing marking methodologies”, Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education. Cracow, Poland, 1999, pp.29-32.

[3] Cosma Georgina and Joy Mike. “Source-code plagiarism: A UK academic perspective”, Research Report No. 422, Department of Computer Science, University of Warwick, 2006.

[4] Judy Sheard, Martin Dick, Selby Markham, et al. “Cheating and plagiarism: perceptions and practices of first year it students”, ACM SIGCSE Bulletin, vol. 34, 2002, pp.183-187.

[5] http://course.buaa.edu.cn

[6] Peter C. Isaacson and Terry A. Scott, "Automating the execution of student programs", ACM SIGCSE Bulletin, vol.21, pp.15-22, June 1989

[7] K. M. Dawson-Howe, "Automatic submission and administration of programming assignments", ACM SIGCSE Bulletin, vol.27(4), 1995, pp.51-53.

[8] David Arnow, Oleg Barshay, "On-line programming examinations using WebToTeach", Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education. Cracow, Poland, pp.21-24, June 1999.

[9] http://acm.pku.edu.cn/JudgeOnline/

[10] http://acm.zju.edu.cn/

[11] David Jackson and Michelle Usher, "Grading student programs using ASSYST", Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education. San Jose, California, United States. pp.335-339, February 1997.

[12] MIKE JOY, NATHAN GRIFFITHS, and RUSSELL BOYATT, "The BOSS Online Submission and Assessment System", ACM Journal on Educational Resources in Computing, vol. 5, September 2005

[13] Foxley E., Higgins C., Hegazy T., Symeonidis P. and Tsintsifas A., "The CourseMaster CBA System: Improvements over Ceilidh", Fifth International Computer Assisted Assessment Conference, Laughborough University, UK, July 2001.

[14] Jones, E. L., "Metrics based plagiarism monitoring", Journal of Computing Sciences in Colleges, vol. 16, 2001, pp. 253-261.

[15] K. L. Verco and M. J. Wise, "Plagiarism à la Mode: A comparison of automated systems for detecting suspected plagiarism", The Computer Journal, 1996, vol. 39, pp. 741–750.

[16] MOSS: a system for detecting software plagiarism. http://theory.stanford.edu /~aiken/moss/

[17] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag", Journal of Universal Computer Science, vol. 8, 2002.

[18] David Gitchell, and Nicholas Tran. "Sim: A utility for detecting similarity in computer programs". The proceedings of the thirtieth SIGCSE technical symposium on Computer science education. New Orleans, Louisiana, United States, pp. 266-270, March 1999.

[19] Michael J. Wise, "YAP3: Improved detection of similarities in computer program and other texts". Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education. Philadelphia, Pennsylvania, United States, pp. 130-134, February 1996.

[20] Michael Olan, "Unit testing: test early, test often", Journal of Computing Sciences in Colleges, vol. 19, pp. 319-328, December 2003.

[21] Randal E. Bryant and David R. O'Hallaron. Computer Systems: A Programmer's Perspective. Prentice Hall, 2003.

[22] Sutter Herb, "The free lunch is over: A fundamental turn toward concurrency in software". Dr. Dobb's Journal, vol. 30, 2005.

[23] GNU gprof, http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html

[24] OProfile - A System Profiler for Linux, http://oprofile.sourceforge.net/news/

[25] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix and W. Pugh, "Using Static Analysis to Find Bugs," IEEE Software, vol. 25, pp. 22-29, Sep 2008.

[26] Splint: annotation-assisted lightweight static checking, http://www.splint.org/

[27] Changhai Zhao, Haihua Yan and Maozhong Jin, "Approach based on compiling optimization and disassembling to detect program similarity", Journal of Beijing University of Aeronautics and Astronautics, vol. 34, pp. 711-715, June 2008(In Chinese).