基于多层次分析的软件重复度检测系统

学生：郝进　薛郁文　徐翔　　指导教师：张玉清

（信息工程学院）

【摘要】 软件抄袭在广大高校中一直饱为诟病，学生上机作业的抄袭非常普遍，这严重影响了学生能力的培养和计算机教学的效果。本文研究了目前六种进行软件重复度检测的算法模型，将 LCS、属性统计、Winnowing 指纹特征、语法分析树、最大权重流、深度学习的查重方法整合为多层次的抄袭检测软件。最终在三个应用场景下进行数据测试和效果对比，并对涉嫌抄袭学生进行了包括但不限于：取消比赛成绩，通知任课教师等处罚，维护了比赛及考试的公平性和严肃性。

【关键词】代码抄袭检测；语法分析树；token 特征；最大权重流；

【项目编号】201911415038

1．绪论

1.1　研究背景以及意义

如今，随着计算机领域的发展，软件开发人员和科技工作者开发出了越来越多的简单易用的框架和标准库，开发一款具有一定功能的软件变得愈发简单，通过黑盒调用即可完成绝大多数的开发操作，那么衡量软件开发的实际代码量来评估开发者的工作量变成了一个亟待解决的问题。与此相关的，软件抄袭是一种重用大量代码的行为，通常为达到特定目的而非法拷贝并使用他人代码。从学生上机作业的抄袭到软件产品的抄袭、移动端 App 的重打包，都属于软件抄袭的范畴。

软件抄袭在广大高校中一直饱为诟病，学生上机作业的抄袭非常普遍，这严重影响了学生能力的培养和计算机教学的效果。从国外最近的一项调研中，约有 33%的学生承认曾经抄袭[1],而且这一比例呈逐年增长趋势;另外在我们的调查中,根据 MOSS(一款国外软件抄袭检测服务)的统计结果显示,提交其检测的所有学生作业都会有 10%左右的抄袭比例,而且这还是在学生提前被告知其上机作业会被检测的情况下。近些年,软件抄袭也已成为正规的软件公司和开源软件组织所迫切关心的问题,其对软件知识产权的保护构成了严重威胁。现在有非常多的开源软件项目和社区,比如 GitHub。开源项目的蓬勃发展一方面促进了软件行业的繁荣,同时也给不法的抄袭者提供了更多以及更便捷的抄袭他人代码的机会,因为源码相比可执行的二进制代码而言更容易理解和修改,抄袭者还可能会利用自动化的混淆工具进行简单包装来盗用全部代码。通常这些开源项目允许用户在遵守特定的 License(如众所周知的 GPL、BSD 等)的前提下使用、修改以及对其进行重新发布,然而受经济利益的驱使,有些软件公司或个人会在其产品中集成第三方代码而不遵守相应的 License。所以如何快速有效且准确的识别出高重复代码，减少误判率，并提供一个简单易用的服务也成为了一个亟待解决的问题。

1.2　发展和研究现状

我们通过分析得到的两个问题，对市面上已有的项目进行了详细调研，首先对于实际代码量评估而言，互联网上暂未有相关的项目和研究；对于软件抄袭而言，国内暂无相关成熟项目，国外虽然有 MOSS 等，但是获取途径需要申请，而且操作复杂，对于代码的比较是基于 token 字串，简单的控制流改变就可以轻松

绕过追踪；而相关更复杂的技术虽然有研究员在研究，但是仍旧没有相关项目整合放出。

1.3 主要研究内容

我们决定复现整合市面上已有的软件抄袭检测的研究成果，并在此基础上做出改进，包括但不限于：根据任务情况划分使用多层次手段，开发整合美化软件界面等，并在最后开发出一套高效易用的软件重复度检测系统。

我们需要解决的问题如下：

1、如何实验检测过程的自动化或更少的人工参与。
2、如何在源码缺失情况下开展检测。
3、如何用多层次多重技术应对各种各样的代码混淆技术。
4、如何解决部分抄袭检测的问题。
5、如何有效评估代码工程的实际工作量。
6、如何提供更高效准确的检测服务。
7、如何开发一套简单易用的操作界面。

2．算法设计

2.1 基本概念与定义

软件抄袭在各个领域有不同的定义综合其他文献的描述,广义而言我们将其描述为一种非法使用他人软件的各种基本要素的行为,其中非法是指未得到软件拥有者的允许或未能遵守既定的条款规约， 软件基本要素包括软件代码、设计思想、用户界面、软件文档、输入数据等,这种行为可以是有意的也可以的无意的。这样的描述相当宽泛,简单的软件拷贝再发布属于抄袭,花费大量时间和精力逆向分析出他人代码的核心思想然后用其他语言实现也属于抄袭,而且很多时候抄袭与否难以界定。

鉴于此， 我们对要解决的抄袭问题进行限定,将其定义为攻击者(抄袭者)在对软件没有任何或很少了解的情况下花费很少的精力对软件代码实施的抄袭;也就说抄袭者不会花时间破解或理解代码,甚至完全没有任何代码知识,而是通过专业工具或者很少的人工修改实施抄袭,这也是在学生之间常用的代码抄袭手段。为了准确描述本文讨论的软件抄袭,下面给出其定义：

定义 1. 软件抄袭。对于两个程序 P 和 Q,只要满足以下任一条件， 就说 Q 抄袭了 P:

1) Q 集成了程序 P 的大部分代码;

2) Q 是抄袭者通过对 P 不变换语义修改得到。

代码抄袭手段是通过的特定代码变换手段将程序 P 转换成另一个程序 Q 的一种技术,并且 P 与 Q 至少保持语义等价,其目的是有意地隐藏程序逻辑,使抄袭后的程序 Q 更加难以理解和分析。代码抄袭手段被广泛用于隐藏抄袭的意图， 加大分析难度以躲避检测。

常见的代码抄袭手段有：标识符重命名、注释删除、格式及布局调整等,降低代码的可读性; 此类抄袭技术实现容易,同时不会给程序带来额外开销， 但混淆程度较弱,仅可以对抗部分基于文本和词法分析的检测技术。而通过改变程序的控制流结构进行的代码变换， 如代码内联和外联、循环条件扩充、基于不透明谓词的死码及垃圾代码植入、控制流扁平化等则会对程序的控制流结构造成较大改变， 因而基于控制流分析的某些检测技术可能会失效。除此之外对程序的数据结构进行改变的代码变换包括变量分割、参数重排、标量合并、数组重构则会混淆会影响数据的存储、编码等， 因而会干扰单纯基于数据流分析的抄检测技术。

2.2 传统抄袭检测方法

源码抄袭检测技术成功应用的前提是相应的源码可获取,而很多情况下如商业软件的源码是不公开的,所以源码抄袭检测研究的一个典型应用场景就是高校学生上机作业的抄袭。由于学生群体的特殊性,这类抄袭通常是以人工修改方式来实现,而且通常不会进行大量复杂的修改。一般来讲,抄袭手段限定在词法、语法、语义三方面的变换,而且大部分学生作业的抄袭集中在前两种。源码抄袭检测技术由于语义分析的复杂性,同时考虑到效率问题(通常要分析上千份的编程作业), 所以在传统方法中使用的是基于 LCS 的抄袭检测技术。

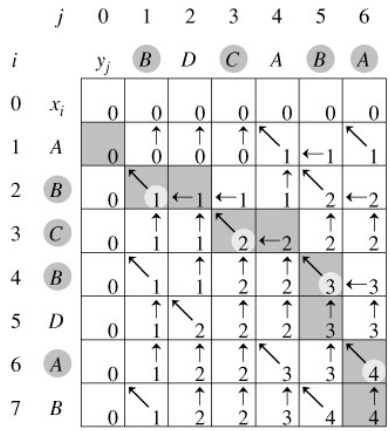
LCS (Longest Common Sub-sequence) 最长相同子序列问题，是一个典型的计算机算法问题，问题描述为给定两个序列，求这两个序列的最长相同子序列（LCS）的长度。举例说明为：序列 1：CDEFGBAH；序列 2：HEABCCA 则序列 1 和序列 2 的 LCS 是 EBA，长度为 3。则源码抄袭问题，可以转化为相应的 LCS 比例问题，当我们人为设置阈值，在 LCS 长度所占比例超过该阈值时即可判定为抄袭。

对于 LCS 问题，我们通常可以通过动态规划进行求解，在本软件设计中，使用的便是动态规划方法，其相应的状态转移方程如下：

$$C[i,j]=\begin{cases}0 & \text{若 } i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0, x_i\neq y_j\end{cases}$$

其中 C[i,j]表示序列 1 的前 i 个数和序列 2 的前 j 个数的最长 LCS 长度。

算法执行过程如下：



（图 1）LCS 算法执行过程

2.3 基于属性统计的抄袭检测

基于属性统计的抄袭检测[2]是最早一批出现的检测方法,其基本思想是通过对软件代码的各项度量指标进行统计, 将软件转换成一个 n 维向量, 每个维度指定了软件的一项度量指标, 从而将其映射成 n 维笛卡尔空间中的一个点, 最后基于该空间中不同点间的距离实现抄袭的判定。

在我们的设计中，我们统计了变量总数、输入语句、条件语句、循环语句、赋值语句及输出语句等度量指标，将其统计数值组成一个向量作为该程序的属性统计向量。在比较不同程序时，我们将两程序的属性统计向量的夹角余弦作为其相似度的度量，夹角余弦值越大则相似度越小。

首先我们将所有的代码格式化后，再进行属性统计。在属性统计方法中，对于变量总数我们按照变量类型关键字进行统计，分别统计了 int、long、long long、float、double、char、bool、auto、string

以及各类型的指针，为了方便起见将不同的类型的变量统一加和成为一个数值代表其变量总数属性；对于输入语句，我们按照关键字进行统计，分别统计了 cin、scanf、getline、gets 的数量，并为了方便起见将不同的类型的输入统一加和成为一个数值代表其输入语句总数属性；对于条件语句，我们按照 if 关键字进行统计，以一个数值代表其条件语句总数属性；对于循环语句，我们按照关键字进行统计，分别统计了 while、for 两种循环的数量，将不同的类型的循环数量统一加和成为一个数值代表其循环语句总数属性；对于赋值语句，统计了=（包含左右两边的空格）的数量，以一个数值代表其循环语句总数属性；对于输出语句，分别统计了 cout、cerr、printf、puts 等输出语句数量，将不同的类型的输出语句数量统一加和成为一个数值代表其输出语句总数属性。最后使用两向量的点乘除以两向量的模的积，得到两程序的属性统计向量的夹角余弦，比较其相似度。

2.4 基于 token 的抄袭检测

这类方法是将软件代码转换成一系列 token， 然后通过 token 比较衡量代码相似性。token 可以是编程语言无关的字符串，也可以是编程语言中的基本元素。

前者是将程序代码当做文本进行分析，典型代表便是在国外提供的 MOSS 抄袭检测服务。它利用字符串匹配算法将程序代码划分为一系列 k-gram， 每个 k-gram 是一个长度为 k 的子串(也就是 token)； 然后对每个 k-gram 进行 Hash 运算,通过 Winnowing 算法[3]筛选出部分 Hash 值作为程序指纹， 两个程序共享的指纹越多就越有可能存在抄袭。后者则在对代码 token 化时考虑了编程语言的基本特性， 其基本思路是对代码进行词法扫描构建 token 字串， 然后通过 token 字串的比较决定程序对的相似性。这类方法的关键在于 token 的选择和比较算法的设计,筛选的 token 应当尽可能不太容易被简单的代码混淆破坏掉。

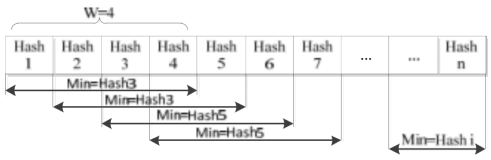
在我们的算法设计中，我们设计结合了两种特性，采用了同 MOSS 一样的 Winnowing 算法，并选取反汇编后的指令符序列作为 token 字串。这种方法结合了两者的优点，效率同前者，在准确度上可以与后者媲美:

选取 Karp-Rabin 算法作为预处理 Hash 算法：

$$Hash(s[0,m-1]) = (2^{m-1}s[0] + 2^{m-2}s[1] + ... + 2^0 s[m-1]) \bmod p$$

$$Hash(s[1,m]) = (2Hash(s[0,m-1] - 2^{m-1}s[0] + s[m])) \bmod p$$

其中，s 表示对应转化后的指令符序列，m 则为整个序列的长度，p 为很大的质数，避免出现循环节导致 Hash 冲突。



（图 2）winnowing 算法过程

接着我们会应用 Winnowing 算法提取相应的指纹特征，算法过程为使用一个大小为 W 的 filter，在整个 Hash 序列上滑动，再经过一个一维池化，得到最终目标特征。通过与 2.3 相同的方式，得到程序的相似度。

下面举例说明其基本工作流程：

（1） 输入程序：

```
#include<iostream>
```

……

（2） 转化为指令符序列：

MOV ADD SUB MOV MOV ADD SUB

（3） 进行 Hash 值计算：

30 28 51 30 30 28 51

（4） 设置滑动窗口为 3 进行 filter 及池化处理：

(【30 28 51】 30 30 28 51)

(30 【28 51 30】 30 28 51)

(30 28 【51 30 30】 28 51)

(30 28 51 【30 30 28】 51)

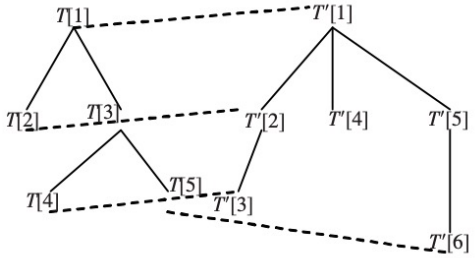
(30 28 51 30 【30 28 51】)

（5） 记录去后的指纹特征：

28 30 28

2.5 基于树的检测

这类技术是将软件代码转换成树型结构[4]进行分析。通过构建程序的抽象语法树(Abstract Syntax Tree, AST) 实现自底向上的累积运算和在 AST 遍历为每个节点计算一个 Hash 值, 并通过节点的 Hash 值匹配和匹配的节点比例衡量相似性。



（图 3）树编辑距离计算方法

在我们的实现中，我们通过 GCC 功能，增加编译参数-fdump-translation-unit 生成.tu 的文件，得到对应文本的语法分析树，将中间的文本抽象语法树通过树的数据结构建立起来，建立树之后再标准化，这样能够提高分析的准确性，最后通过树编辑距离的方式对树进行分析，从而最终得出其相似度。

1.3 基于图的检测

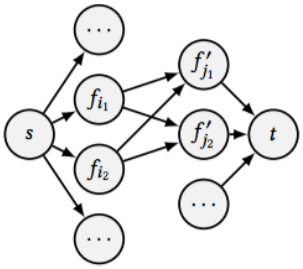
基于图的抄袭检测方法相对较少，一般是通过从软件代码中构造程序依赖图, 通过子图同构匹配实现相似性计算。

我们采用的是南京大学提出的网络流最大匹配流方法[5],我们定义两个函数之间的相似性：

$$\sigma(f_i,f_j') = \max_{k\in\{1,2,\ldots,|f_j'|\}} LCS(f_i,f_j'[k:k+\omega])$$

其中 LCS 表示两个指令序列的最长公共子序列，$f_j'[k{:}\,k+\omega]$表示其索引所在的f_j'的子序列。直观来看，$\sigma(f_i,f_j')$是在固定窗口大小 ω 中受约束的最长公共子序列。 较大的 σ 表示 fi 中的更多指令在f_j'中的长度为 ω 的连续指令序列中具有对应的等效指令，因此表示较小的距离 d。

然后，我们通过计算所有函数$f_i \in P_1$可以嵌入到P_2中的函数的程度来计算P_1和P_2之间的相似性。 为了将P_1中的多个函数嵌入到单个函数$f_j' \in P_2$中，我们构造了一个加权流网络图 G =(V,E)，其容量函数为$c{:}\,E \mapsto R$和权重函数$w{:}\,E \mapsto R$. 设 $V = \{s,t,l_1,\ldots,l_n,\ r_1,\ldots,r_m\}$ 其中 s 是源点，t 是汇点，如图所示：



（图 4）建图方式

对于每个$i \in [n]$，构造一个边(s,l_i)，其中$c(s,l_i) = |f_i|$和$w(s,l_i) = 0$；对于每个$j \in [m]$，构造一个边(r_j,t)，其中$c(r_j,t) = |f_j'|$和$w(r_j,t) = 0$；对于每个$(i,j) \in [n] \times [m]$，构造一个边(l_i,r_j)，其中

$$c(l_i,r_i) = \sigma(f_i,f_j')$$

$$w(l_i,r_j) = \frac{1}{1+e^{-\alpha\cdot\frac{max\{\sigma(f_i,f_j'),\sigma(f_j',f_i)\}}{min\{|f_i|,|f_j'|\}}+\beta}}$$

具有常数α和β的逻辑函数用于归一化权重，使得一对函数之间的“成功”嵌入将具有接近 1 的权重，而失败的嵌入将具有接近 0 的权重。因此，两代码之间的相似性定义为:

$$\sigma(P_1,P_2) = \frac{MaximunWeightFlow(G,c,w)}{\sum_{i\in[n]}|f_i|}$$

1.3 基于深度学习方法的检测

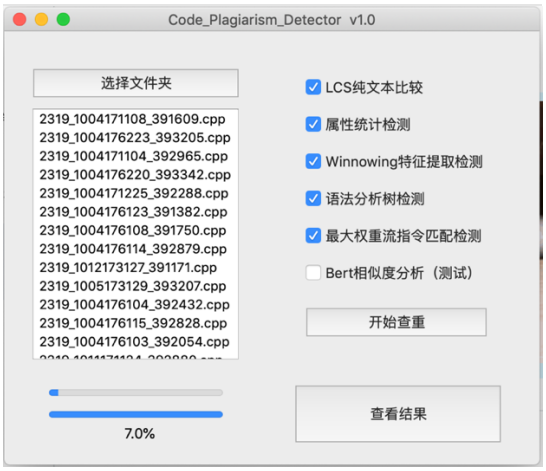
基于深度学习方法的检测，一般是采用群体学习的思想,在软件度量指标基础上将其他检测技术(如将要提到的基于 token 的检测方法)的分析结果考虑进来， 将它们作为 BP 神经网络的输入训练分类器达到检测的目的。但该方法限定条件较多， 如需要专业人员事先标定大量的训练数据,且存在数据稀疏性问题，是一个比较难以解决的问题。

近日，微软、哈工大在 arxiv 上联合发表了一篇论文，其提出了 CodeBert 预训练模型[6]，解决了编程语言方面一般 NLP 模型的预训练数据不足，收敛性差的问题。鉴于 CodeBert 尚未开源，我们仅在 Bert 预训练模型下，通过在 Bert 后接入一个 SoftMax 来实现二分类任务完成代码相似度检测，

3．软件设计

3.1 界面设计

界面设计以轻量化、界面简洁为原则。



（图 5）软件界面

左侧为文件目录下所有待查重文件名，右侧为进行查重所采用方法，采用方法越多，查重耗时越长。点击查看结果则可以打开输出报告所在目录。

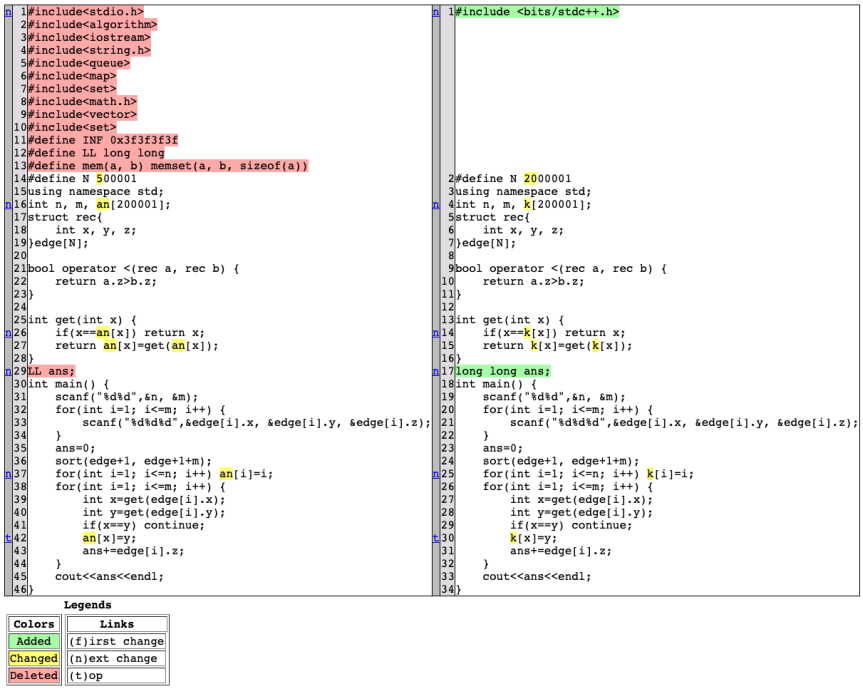
程序框架采用 PythonQT,并采用多线程写法，避免了在执行查重过程中的假性卡死。并通过进度条的形式实时查看查重进度。

3.2 输出报告设计

输出报告采用 HTML5 加 Echarts 形式

每对相似度达到阈值的查重文件生成一份报告。

报告共分两部分，第一部分为仿照 GitHub 设计的增删改对比模块，通过不同色块标记显示提交代码之间的修改情况，并且代码部分不进行格式化保留原格式，更进一步可以展示出抄袭者是如何对另一份代码进行的抄袭。



```
#include<stdio.h>
#include<algorithm>
#include<iostream>
#include<string.h>
#include<queue>
#include<map>
#include<set>
#include<math.h>
#include<vector>
#include<set>
#define INF 0x3f3f3f3f
#define LL long long
#define mem(a, b) memset(a, b, sizeof(a))
#define N 500001
using namespace std;
int n, m, an[200001];
struct rec{
    int x, y, z;
}edge[N];

bool operator <(rec a, rec b) {
    return a.z>b.z;
}

int get(int x) {
    if(x==an[x]) return x;
    return an[x]=get(an[x]);
}
LL ans;
int main() {
    scanf("%d%d",&n, &m);
    for(int i=1; i<=m; i++) {
        scanf("%d%d%d",&edge[i].x, &edge[i].y, &edge[i].z);
    }
    ans=0;
    sort(edge+1, edge+1+m);
    for(int i=1; i<=n; i++) an[i]=i;
    for(int i=1; i<=m; i++) {
        int x=get(edge[i].x);
        int y=get(edge[i].y);
        if(x==y) continue;
        an[x]=y;
        ans+=edge[i].z;
    }
    cout<<ans<<endl;
}
```

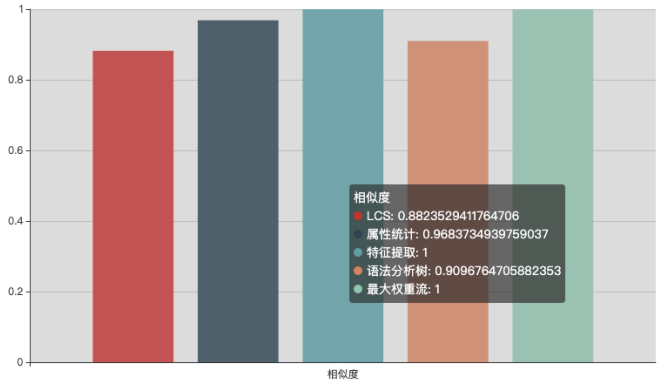
```
#include <bits/stdc++.h>
#define N 2000001
using namespace std;
int n, m, k[200001];
struct rec{
    int x, y, z;
}edge[N];

bool operator <(rec a, rec b) {
    return a.z>b.z;
}

int get(int x) {
    if(x==k[x]) return x;
    return k[x]=get(k[x]);
}
long long ans;
int main() {
    scanf("%d%d",&n, &m);
    for(int i=1; i<=m; i++) {
        scanf("%d%d%d",&edge[i].x, &edge[i].y, &edge[i].z);
    }
    ans=0;
    sort(edge+1, edge+1+m);
    for(int i=1; i<=n; i++) k[i]=i;
    for(int i=1; i<=m; i++) {
        int x=get(edge[i].x);
        int y=get(edge[i].y);
        if(x==y) continue;
        k[x]=y;
        ans+=edge[i].z;
    }
    cout<<ans<<endl;
}
```

Legends

Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

（图 6）报告对比部分

报告另一部分设计为相似度查重结果，可以轻松地看到多层次不同手段进行查重所得到的结果，从而进一步确认抄袭者的抄袭行为。



（图 7）报告相似度结果部分

4. 实验结果及分析

4.1 数据测试

我们主要在三个场景下进行了大规模的算法效果及数据测试。

1　计算机-电子小学期《C 语言编程实践》

2　《算法设计与分析》期末考试

3　第十三届中国地质大学（北京）校程序设计竞赛

在以上的测试中，我们共分析了共计 5000 份以上的提交、查到进行代码抄袭的学生 100 余人次，对

其进行了包括但不限于： 取消比赛成绩，通知任课教师等处罚，维护了比赛及考试的公平性严肃性。

4.2 效果比对

算法	检测程度	运行速度	可检测类型
基于属性统计的检测算法	很差	极快,O(N)	完全重复文本
基于LCS的纯文本比较算法	差	快,O(N^2)	修改格式，小幅度修改的程序
基于语法分析树的检测算法	中	极慢,因为GCC生成的原语法分析树文件特大，需要剔除冗余信息	变量替换、修改格式
基于winnowing特征提取检测算法	良	快，O(KN)，只需	变量替换、格式修改、计算顺序交换、函数提取
基于最大权重流的指令匹配检测算法	优	慢，网络流需要时间过长	变量替换、格式修改、大规模无用信息塞入、函数提取、循环展开、计算顺序交换

（表 1）效果比对

在上面所述的场景中我们对所实现算法进行了数据测试，测试效果如上。

对于深度学习方法，由于目前模型效果不佳，不计入统计，等待 CodeBert 权重开源后再进行整合。

致谢

我们衷心感谢张玉清老师和季晓慧老师在大创期间给予我们的细致指导。没有他们的帮助我们不可能完成如今所有的代码实验以及报告。在课题研究过程中，柏宇学长、陈三星学长以及实验室的同学及学弟一直鼓励我们支持我们，在此向他们表示诚挚的谢意。

最后，我们马上毕业了，在此也十分感谢母校对我们的四年培育。

【参考文献】

[1] Chuda D, Navrat P, Kovacova B, et al. The issue of (software) plagiarism: A student view[J]. IEEE Transactions on Education, 2011, 55(1): 22-28.

[2] Osman A H, Salim N, Binwahlan M S, et al. An improved plagiarism detection scheme based on semantic role labeling[J]. Applied Soft Computing, 2012, 12(5): 1493-1502.

[3] Schleimer S, Wilkerson D S, Aiken A. Winnowing: local algorithms for document fingerprinting[C]//Proceedings of the 2003 ACM SIGMOD international conference on Management of data. 2003: 76-85.

[4] Zhang K, Shasha D. Simple fast algorithms for the editing distance between trees and related problems[J]. SIAM journal on computing, 1989, 18(6): 1245-1262.

[5] Jiang Y, Xu C. Needle: detecting code plagiarism on student submissions[C]//Proceedings of ACM Turing Celebration Conference-China. 2018: 27-32.

[6] Feng Z, Guo D, Tang D, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages[J]. arXiv preprint arXiv:2002.08155, 2020.