Instructor-Centric Source Code Plagiarism Detection and Plagiarism Corpus

Jonathan Y. H. Poon, Kazunari Sugiyama, Yee Fan Tan, Min-Yen Kan

National University of Singapore
Introduction

Plagiarism in undergraduate courses

• 181 / 319 students admitted to committing source code plagiarism in School of Computing, the National University of Singapore
  [Ooi and Tan, CDTLink’05]

• 40% of 50,000 students at more than 60 universities admitted in plagiarism
  [Jocoy and DiBiase, Review of Research in Open and Distance Learning’06]
Related Work

Attribute-counting Metric Systems

Similarity between codes is computed based on counts of particular entities.

[Ottenstein, SIGCSE Bulletin ’76] Unique operators and operands

Improved approaches of [Ottenstein, SIGCSE Bulletin ‘02]
[Donaldson et al., SIGCSE ’81] Loops
[Grier, SIGCSE ‘81] Control statements
[Berghel and Sallach, SIGPLAN Notices ’84] Keywords
[Faidhi and Robinson, Comp. and Edu. ’87] Average length of procedure or function

All previous work uses pairwise level detection.
Related Work

Structure Metric Systems

Similarity between codes is computed based on code structure. The Minimum Match Length (MML) parameter is important.

- MOSS (Measure Of Software Similarity) [Aiken ’94]
- YAP (Yet Another Plague) family [Wise, SIGCSE ’92, ’96]
- sim [Gitchell and Tran, SIGCSE ’99]

Cluster Level Detection

- PDetect [Moussiades and Vakali, The Comp. Journal ’05]
- PDE4Java [Jadalla and Elnagar, Journal of BI and DM ’08]

- Plagiarists can easily confuse the system by inserting non-functional code that are larger than MML.
- Most of the systems employ pairwise level detection.
Plagiarism Detection Method

Our approach focuses on how plagiarism is carried out.

- Submissions
- Tokenization
- Pairwise Comparison
- Plagiarism Clusters Detection

Cut off criteria

Result
- Cluster
- Cluster
Plagiarism Detection Method

1. Submission
2. Tokenization
3. Pairwise Comparison
4. Plagiarism Clusters Detection
5. Result

Cut off criteria
Tokenization

• Parse code into four types of token $N$-grams
  • Keyword (“class,” “void,” “int,” etc.)
  • Variable (“MyClass,” “main,” “String,” etc.)
  • Symbol (“{,” “(,” “[,” etc.)
  • Constant (“1,” “10,” etc.)

• Language specific (currently, support Java)
  • Easily adapt to other program languages if a tokenizer for the target language is introduced.
Example of Parsing Code

```java
public class MyClass {
    public static void main(String[] args) {
        int value = 1;
        for (;value<10;value++) System.out.println(value + "");
    }
}
```
Example of Parsing Code

```java
public class MyClass {
    public static void main(String[] args) {
        int value = 1;
        for (; value < 10; value++) System.out.println(value + "");
    }
}
```
Plagiarism Detection Method

1. Submissions
2. Tokenization
3. Pairwise Comparison
4. Plagiarism Clusters Detection
5. Cut off criteria
6. Result

Clusters
Pairwise Comparison

- Greedy-String-Tiling algorithm [Wise,’93]

- Some improvements
  - Exclusion of skeleton code
  - Minimum Match Length (MML)
    - Statement-based matching
    - Two statements are identical if and only if their contiguous tokens are identical
  - Hash
    - Complexity of submission comparison
      \[ O(c^3) \rightarrow O(c^2) \]
      
      \( c \): Number of tokens in a submission
Greedy-String-Tiling Algorithm

Find the longest substrings more than Minimum Match Length (\(MML\))

[Example]

\[MML = 3\]

ABCDEF GH
EFG ABCD H
Greedy-String-Tiling Algorithm

Find the longest substrings more than Minimum Match Length ($MML$)

[Example]

\[ MML = 3 \]

ABCDEFGH

EFGABCDH
Greedy-String-Tiling Algorithm

Find the longest substrings more than Minimum Match Length \( (MML) \)

[Example]

\( MML = 3 \)

ABCDEF GH
EFGABCDH
Example of Pairwise Comparison

```java
currentBox = ((int) (random.nextFloat() * 4));
}

private void drawLine(Graphics g, int xOld, int yOld, int x, int y) {
    g.setColor(Color.white);
    g.drawLine(xOld + 25, yOld + 25, x + 25, y + 25);
}

private void deleteLine(Graphics g, int xOld, int yOld, int x, int y) {
    g.setColor(Color.gray);
    g.drawLine(xOld + 25, yOld + 25, x + 25, y + 25);
}

private void drawSmile(Graphics g, int xOld, int yOld) {
    currentBox = ((int) (random.nextFloat() * 4));
}
```

```java
private void drawLine(Graphics g, int xOld, int yOld, int x, int y) {
    g.setColor(Color.white);
    g.drawLine(xOld + 25, yOld + 25, x + 25, y + 25);
}

private void deleteLine(Graphics g, int xOld, int yOld, int x, int y) {
    g.setColor(Color.gray);
    g.drawLine(xOld + 25, yOld + 25, x + 25, y + 25);
}

private void drawSmile(Graphics g, int xOld, int yOld) {
    currentBox = ((int) (random.nextFloat() * 4));
}
```
Plagiarism Detection Method

1. Tokenization
2. Pairwise Comparison
3. Plagiarism Clusters Detection

Cut off criteria

Result
- Cluster
- Cluster
Plagiarism Clusters Detection

• DBScan [Ester at el., KDD’96]
  • Groups submissions that are highly similar to each other.

• Performance
  • More than 80 introductory programming assignments (over 3,600 submission pairs)
  • Less than 4 seconds on average (on 2.8GHz Linux laptop)
Plagiarism Corpus

- 28 student volunteers plagiarize submissions
- 2 assignments
- 4 samples per assignment to generate plagiarized version of source code
  - 56 positive examples (plagiarized submissions)
  - 180 negative examples (original submissions)
Similarity Distribution for Various Sized $N$-gram ($MML=2$)

Our system successfully differentiates between $\text{ORG}$ and $\text{PLAG}$.
Attacks Performed by Student Volunteers

“Attacks”: plagiarism attempts

• Immutable attacks
• Size dependent attacks
• Successful attacks
# Immutable Attacks

Attacks that our system can protect

<table>
<thead>
<tr>
<th>Type of attacks</th>
<th>The number of confused attacks</th>
<th>The number of observed attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion, modification or deletion of comments</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Indention, spacing or line breaks modifications</td>
<td>0</td>
<td>38</td>
</tr>
<tr>
<td>Identifier renaming</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>Constant modification</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Insertion, modification, or deletion of modifiers</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>No change</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(122 attacks in total)
Identifier Renaming

(a) Original submission

int value = 1;

(b) Plagiarized copy

int v = 1;

Our system detect this type of plagiarism.
Size Dependent Attacks

Attacks that needs large modification

<table>
<thead>
<tr>
<th>Type of attacks</th>
<th>The number of confused attacks</th>
<th>The number of observed attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reordering of independent statements</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Reordering of methods</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>Insertion or removal of parentheses</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Inlining or refactoring of code</td>
<td>13</td>
<td>18</td>
</tr>
</tbody>
</table>

(64 attacks in total)
Reordering of Independent Statements

left = tree.getLeft();
right = tree.getRight();

(a) Original submission

right = tree.getRight();
left = tree.getLeft();

(b) Plagiarized copy

Our system detect this type of plagiarism.
## Successful Attacks

<table>
<thead>
<tr>
<th>Type of attacks</th>
<th>The number of confused attacks</th>
<th>The number of observed attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td><strong>Scope modification</strong></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Modification of control structures</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Declaration of variables</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Modification of method parameters</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Modification of import statements</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Introduction of bug</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Modification of temporary variables in expressions</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Modification of mathematical operations and formulae</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Structural redesign of code</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

(60 attacks in total)
Scope Modification

for(int i = 0; i < 10; i++){
    int k;
    ...
}

(a) Original submission

int k;
for(int i = 0; i < 10; i++){
    ...
}

(b) Plagiarized copy

Our system cannot detect this type of plagiarism.
User Interface Work Flow

Pairwise Comparison Interface

Instructors overview the code segments with several colors.
Instructors learn
- suspicious pairs of students,
- plagiarism cases.

<table>
<thead>
<tr>
<th>DATE/TIME</th>
<th>MODULE</th>
<th>ASSIGNMENT</th>
<th>GRADER</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/04/2010 19:53:18</td>
<td>CS3256</td>
<td>Individual Project</td>
<td>Yee Fan Tan</td>
<td>Reported submissions from students 038 and 053 as suspicious</td>
</tr>
<tr>
<td>01/04/2010 12:50:20</td>
<td>CS2143</td>
<td>1</td>
<td>Min-Yen Kan</td>
<td>The student is found guilty in plagiarism</td>
</tr>
<tr>
<td>01/04/2010 12:50:20</td>
<td>CS2143</td>
<td>1</td>
<td>Min-Yen Kan</td>
<td>Confirmed submissions from students 038 and 053 as plagiarism</td>
</tr>
<tr>
<td>30/03/2010 18:21:43</td>
<td>CS2143</td>
<td>1</td>
<td>Jonathan Poon</td>
<td>Reported submissions from students 038 and 053 as suspicious</td>
</tr>
</tbody>
</table>
Plagiarism Clusters

Plagiarism cluster consists of: 028, 035, 048

Instructors learn suspicious group that performs plagiarism.
Plagiarism Activities Monitoring

CS2105: Introduction to Computer Networks

- Student metric in red denotes the student is found guilty in plagiarism for the assignment
- To mark/unmark a student, click the student metric
- To show/hide plagiarism cluster, click the show/hide link next to the plagiarism cluster
- To view student's summary, move mouse cursor to the student metric

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Cut off criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>83.825%</td>
</tr>
<tr>
<td>2</td>
<td>88.324%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plagiarism Clusters</th>
<th>[hide]</th>
</tr>
</thead>
<tbody>
<tr>
<td>038</td>
<td></td>
</tr>
<tr>
<td>053</td>
<td></td>
</tr>
<tr>
<td>028</td>
<td>[hide]</td>
</tr>
<tr>
<td>035</td>
<td></td>
</tr>
<tr>
<td>048</td>
<td>[hide]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plagiarism Clusters</th>
<th>[hide]</th>
</tr>
</thead>
<tbody>
<tr>
<td>053</td>
<td></td>
</tr>
<tr>
<td>063</td>
<td></td>
</tr>
<tr>
<td>064</td>
<td></td>
</tr>
</tbody>
</table>

Ranking

<table>
<thead>
<tr>
<th>Rank</th>
<th>Student</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>053</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>028</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>063</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>064</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>043</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>066</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>035</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>047</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>048</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>038</td>
<td>1</td>
</tr>
</tbody>
</table>

Cut off criterion: 80%
Plagiarism Activities Monitoring

Instructors learn suspicious student pairs.

A list of the top 10 students can help instructor in monitoring their plagiarism activities.
Similarity Between Students

“Assignment 1”:
Similarity(038, 053) ≥ 90%

“Assignment 2”:
Similarity(038, 053) < 50%
Similarity(053, 063) ≥ 90%
Similarity(053, 066) ≥ 90%

- 038 stopped plagiarizing 053’s assignments.
- 053 started plagiarizing 063’s and 066’s assignments.
Finding the Submissions Most Similar to the Target Student’s One

Instructors find the top $k$ students paired up with the target student “038.”

<table>
<thead>
<tr>
<th>Assignments</th>
<th>Similarity (in decreasing order)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>038 (99.609%)</td>
</tr>
<tr>
<td></td>
<td>007 (50.602%)</td>
</tr>
<tr>
<td></td>
<td>006 (26.846%)</td>
</tr>
<tr>
<td></td>
<td>031 (26.817%)</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>016 (33.245%)</td>
</tr>
<tr>
<td></td>
<td>002 (27.443%)</td>
</tr>
<tr>
<td></td>
<td>141 (25.572%)</td>
</tr>
<tr>
<td></td>
<td>133 (21.414%)</td>
</tr>
<tr>
<td></td>
<td>062 (20.168%)</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>141 (19.322%)</td>
</tr>
<tr>
<td></td>
<td>139 (18.919%)</td>
</tr>
<tr>
<td></td>
<td>133 (18.414%)</td>
</tr>
<tr>
<td></td>
<td>020 (17.672%)</td>
</tr>
<tr>
<td></td>
<td>007 (17.672%)</td>
</tr>
</tbody>
</table>

*Student matric in **red** denotes the student is found guilty in plagiarism for the assignment
*To view code comparison between a student and student 038, click the student matric
Conclusion

• Instructor-Centric Source Code Plagiarism Detection

• Improvements in “Pairwise Comparison”
  • Faster processing

• Construction of “Plagiarism Corpus”
  • Other researchers can enhance algorithm to detect plagiarism of source code.
  • Downloadable URL: http://wing.comp.nus.edu.sg/downloads/SSID/PlagiarismCorpus.html

• Improvements in “Interfaces”
  • Instructors can monitor students’ plagiarism activities.

Thank you very much!