GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis

Chao Liu, Chen Chen, Jiawei Han, Philip S. Yu
Presented by Jungyoon Lee
Software Plagiarism

- In Class?
- The Blossom of Open Source Projects
  - I/O
  - Graphical User Interface
  - etc
- “Professional” Plagiarist
  - Core-part plagiarism
  - With serious disguises
Serious Disguises

- Format Alteration
  - Insert and remove blanks and comments
- Identifier Renaming
- Statement Reordering
- Control Replacement
  - for → while
- Code Insertion
How to Detect?

- String-based
  - fragile to identifier renaming
- AST-based
  - fragile to statement reordering, control replacement
- Token-based (MOSS, JPLAG)
  - fragile to statement reordering, code insertion
Graph Modeling
Graph Isomorphism
Prune Search Space
Knowledge Discovery

- Interesting Patterns
  - Intrinsic representation of programs

- Anomaly detection
  - A small portion of the entire program
Program Dependency Graph (PDG)

- Graph representation of the source code of a procedure

- Basic statements $\rightarrow$ vertices
  - e.g., variable declarations, assignments, and procedure calls
Control Dependency Edge
ex) if \( v1 \) is true, then \( v2 \)

Data Dependency Edge
- \( \text{var} \leftarrow v1 \)
- \( \text{var} \leftarrow v2 \)
- An execution path from \( v1 \) to \( v2 \) along which there is no assignment to \( \text{var} \)
Definition 3 (Program Dependence Graph). The program dependence graph $G$ for a procedure $P$ is a 4-tuple element $G = (V, E, \mu, \delta)$, where

- $V$ is the set of program vertices in $P$
- $E \subseteq V \times V$ is the set of dependency edges, and $|G| = |V|$  
- $\mu : V \rightarrow S$ is a function assigning types to program vertices,
- $\delta : E \rightarrow T$ is a function assigning dependency types, either data or control, to edges.
Program Dependency Graph (PDG)

(a) Program Dependence Graph of the Procedure \texttt{sum}

(b) Summation over an Array

\begin{verbatim}
int sum(int array[], int count)
{
    int i, sum;
    sum = 0;
    for(i = 0; i < count; i++)
        sum = add(sum, array[i]);
    return sum;
}

int add(int a, int b)
{
    return a + b;
}
\end{verbatim}

Figure 1: An Illustrative Example for Program Dependence Graphs
Program Dependency Graph (PDG)

- Robust to disguises
  - “Creative” plagiarist may be able to overhaul the PDGs, but the cost is higher than rewriting

- Graph Similarity?
  - CloseGraph fails to terminate in two days with 30 vertices and 43 edges
Detection of Software Plagiarism Using PDG

Original program $\mathcal{P} (n \text{ procedures})$
Plagiarism suspect $\mathcal{P'} (m \text{ procedures})$
Two PDG sets $G$ and $G'$, respectively

- Given $g$ in $G$ and $g'$ in $G'$, how can we decide whether $g'$ is a plagiarized PDG of $g$?
Claim 1. Restricted to the five kinds of disguises, if $g$ ($g \in \mathcal{G}$) is subgraph isomorphic to $g'$ ($g' \in \mathcal{G}'$), the corresponding procedure of $g'$ is regarded plagiarized from that of $g$.

Claim 2. If $g$ ($g \in \mathcal{G}$) is $\gamma$-isomorphic ($0 < \gamma \leq 1$) to $g'$ ($g' \in \mathcal{G}'$), the corresponding procedure of $g'$ is regarded plagiarized from that of $g'$, where $\gamma$ is the mature rate for plagiarism detection.
Pruning Search Space

- How can we efficiently locate real plagiarized PDG pairs, while in principle $n \times m$ pairs are to be checked?
  - Lossless Filter
  - Lossy Filter
Pruning Search Space

Lossless Filter

- Exclude PDGs smaller than an interesting size $K$

- Based on the defn. of $\gamma$-isomorphism, a PDG pair $(g, g')$ can be excluded if $|g| < \gamma |g'|$
Exclude dissimilar pairs

- summarized representation of the PDG
  \[ h(g) = (n_1, n_2, \ldots, n_k) \]
  where \( n_i \) is the freq. of the \( i \)th kind of vertices

\[ h(g') = (m_1, m_2, \ldots, m_k) \]

- Measure the similarity between \( g \) and \( g' \) in terms of their vertex histograms \( h(g) \) and \( h(g') \)
First, estimate a k-dimensional multinomial distribution $P_g(\theta_1, \theta_2, \ldots, \theta_k)$ from $h(g)$

Consider whether $h(g')$ is likely to be an observation from $P_g$

If it is, $(g, g')$ should be checked; otherwise, prune it

For the details, please refer to the paper
Implementation of GPLAG

Algorithm 1 GPLAG(\(\mathcal{P}, \mathcal{P}', K, \gamma, \alpha\))

Input: \(\mathcal{P}\): The original program
- \(\mathcal{P}'\): A plagiarism suspect
- \(K\): Minimum size of nontrivial PDGs, default 10
- \(\gamma\): Mature rate in isomorphism testing, default 0.9
- \(\alpha\): Significance level in lossy filter, default 0.05

Output: \(\mathcal{F}\): PDG pairs regarded to involve plagiarism

1: \(\mathcal{G} = \) The set of PDGs from \(\mathcal{P}\)
2: \(\mathcal{G}' = \) The set of PDGs from \(\mathcal{P}'\)
3: \(\mathcal{G}_K = \{g|g \in \mathcal{G} \text{ and } |g| > K\}\)
4: \(\mathcal{G}'_K = \{g'|g' \in \mathcal{G}' \text{ and } |g'| > K\}\)
5: \text{for each } g \in \mathcal{G}_K
6: \quad \text{let } \mathcal{G}'_{K,g} = \{g'|g' \in \mathcal{G}'_K, |g'| \geq \gamma|g|, (g, g') \text{ passes filter}\}
7: \quad \text{for each } g' \in \mathcal{G}'_{K,g}
8: \quad \text{if } g \text{ is } \gamma\text{-isomorphic to } g'
9: \quad \quad \mathcal{F} = \mathcal{F} \cup (g, g')
10: \text{return } \mathcal{F};
Effectiveness of GPlAG

- Plagiarize a text join program with 667 LOC for two hours
- The plagiarism operations succeeded in confusing MOSS and JPlAG
- GPlAG detected the plagiarism in less than 0.1 sec

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Stmt Reorder</th>
<th>Ctrl Replace</th>
<th>Code Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfields</td>
<td>4</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>keycmp</td>
<td>11</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>prjoin</td>
<td>12</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>join</td>
<td>10</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>add_field_list</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>make_blank</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Detected Plagiarism Procedures
Efficiency of GPLAG

![Graph showing the efficiency of GPLAG across different significance levels.](image)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>No Filter</th>
<th>Lossless Only</th>
<th>Lossless &amp; Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tested Pairs</td>
<td>Time</td>
<td>Matches</td>
</tr>
<tr>
<td>bc</td>
<td>3,136</td>
<td>251.21</td>
<td>63</td>
</tr>
<tr>
<td>less</td>
<td>11,449</td>
<td>1,171.35</td>
<td>125</td>
</tr>
<tr>
<td>tar</td>
<td>6,889</td>
<td>853.1</td>
<td>110</td>
</tr>
</tbody>
</table>

Table 5: Efficiency of GPLAG
Core-Part Plagiarism

- GP\textsuperscript{LAG} is expected to find six plagiarized procedures.
- Not only can the lossy filter reduce the false positive rate, but it also makes time const acceptable.

<table>
<thead>
<tr>
<th>Carriers</th>
<th>Lossless Only</th>
<th>Lossless &amp; Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Matches</td>
</tr>
<tr>
<td>Orig.</td>
<td>Plag.</td>
<td></td>
</tr>
<tr>
<td>bc</td>
<td>less</td>
<td>245.457</td>
</tr>
<tr>
<td>bc</td>
<td>tar</td>
<td>620.394</td>
</tr>
<tr>
<td>less</td>
<td>bc</td>
<td>1069.72</td>
</tr>
<tr>
<td>less</td>
<td>tar</td>
<td>2655.13</td>
</tr>
<tr>
<td>tar</td>
<td>bc</td>
<td>1084.34</td>
</tr>
<tr>
<td>tar</td>
<td>less</td>
<td>1048.43</td>
</tr>
</tbody>
</table>

Table 6: Simulated Core-Plagiarism
Thank You