Detecting Code Plagiarism on Student Submissions

Yanyan Jiang and Chang Xu

SPAR Group, Institute of Computer Software
Department of Computer Science and Technology, Nanjing University

ACM TURC'18 SIGCSE China Symposium

May 19, 2018

Needle[1]: Detecting Code Plagiarism on Student Submissions

1. Background and Motivation
   - Code plagiarisms among students
2. Measurement of Program Similarities
   - Approximating the editing distance $d(P_1, P_2)$
3. Empirical Results
   - Applying Needle to student submissions
4. Lessons and Challenges
   - Lessons and challenges

[1]Needle is a utility project in Project-N, the computer systems lab series at Nanjing University: processors (NPC, NOOP), emulator (NEMU), operating system (Nanos), and compiler (NCC).

Outline

1. Background and Motivation
   - Code plagiarisms among students
2. Measurement of Program Similarities
   - Approximating the editing distance $d(P_1, P_2)$
3. Empirical Results
   - Applying Needle to student submissions
4. Lessons and Challenges
   - Lessons and challenges

Students Copy Code[2]

T1	change comments, names, or cases	`cur->lineno` $\rightarrow$ `head->line`
T2	reformat or reorder code fragments	indention/code style change, etc.
T3	add or delete redundant elements	add unused variables or redundant computations
T4	refactor program constructs	`for` $\rightarrow$ `while`, function split/merge, function rewrites, ...

[2] Neal R. Wagner. Plagiarism by student programmers. UTSA Tech Rep, 2000.

Students Copied Lots of Code

Without plagiarism control

A conservative post-mortem analysis (a single anonymous lab) showed that

- ~82% students plagiarized
- ~42% of the plagiarized copies are of similarity $>$ 99%.

With *Needle* $\leftarrow$ this talk's topic, *penalties*, and *honor grades*

Reduced plagiarism rate

- (average) ~5% project groups plagiarized in a single lab
- ~20% project groups plagiarized in a semester

Outline

Problem Definition

Calculate the (minimum) editing distance between two *optimized* program binaries[3] P_1 and P_2:

1. Reordering of two functions with zero cost
2. Deletion of an instruction costing c_d
3. Modification of an instruction (either instruction type or operands) costing c_m
4. Insertion of an arbitrary instruction costing c_i[4]

Programs with small $d(P_1, P_2)$ are likely to be plagiaristic copies

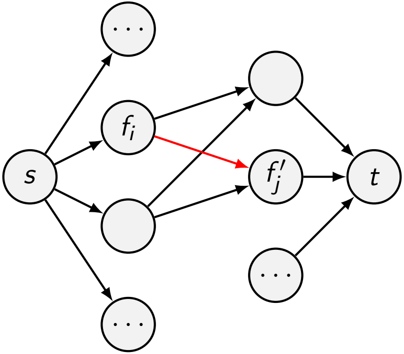
- Unfortunately, computing $d(P_1, P_2)$ is NP-Complete

[3] Compiler optimizations are good at "normalizing" local semantics changes.

[4] $c_d = c_m = c_i = 1$ suffices for code plagiarism detection in practice.

The Network Flow Solution

"A lower-bound estimation of the editing distance $d(P_1, P_2)$"



Let each unit of flow (l_i, r_j) to denote "embedding an instruction in function f_i to function f'_j". The similarity between P_1 and P_2 is denoted by the *maximum weighted matching*.

The Network Flow Solution (cont'd)

Similarity between functions (windowed longest-common-subsequence):

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

A weighted bipartite $G(L, R, c, w)$

- Capacity $c(\ell_i, r_j) = \sigma(f_i, f_j')$
  - cannot embed too many instructions from $f_i \to f_j'$
- Weight $w(\ell_i, r_j) = \left(1 + e^{-\alpha \cdot \frac{\max\{\sigma(f_i, f_j'), \sigma(f_j', f_i)\}}{\min\{|f_i|, |f_j'|\}} + \beta}\right)^{-1}$
  - embedding $f_i \to f_j'$ is more profitable with a larger $\sigma(f_i, f_j')$

$$\text{Program Similarity } \sigma(P_1, P_2) = \frac{\text{MaximumWeightFlow}(G, c, w)}{\sum_{i \in [n]} |f_i|} \in [0, 1]$$

Why It Works?

T1	change comments, names, or cases	do not affect the compiled binary
T2	reformat or reorder code fragments	matching is order insensitive
T3	add or delete redundant elements	instructions can be aligned by LCS
T4	refactor program constructs	limited changes to the optimized binary; function split/merge can be detect by matching

Outline

An Anonymous Programming Assignment: Case Study

In all 79 students' submissions

- 65 of 79 (82%) submissions are plagiaristic (by a manual inspection)
- 42 of 79 (398 pairs) simiarlity > 99% ("nearly perfect embedding")

Interview Results

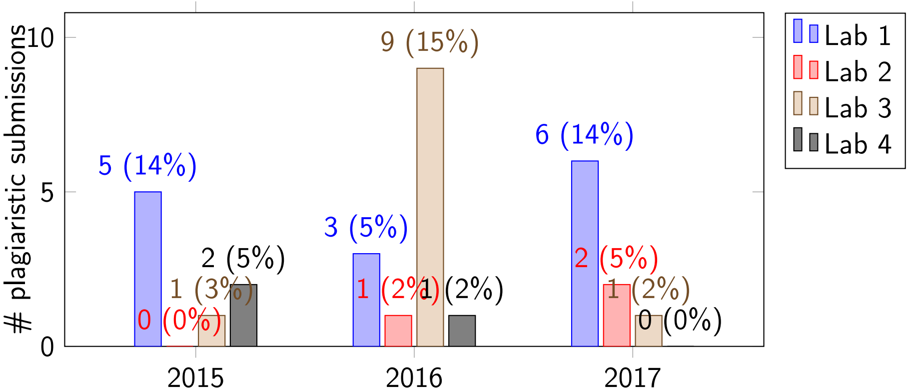
The teaching assistant: "...Looking at their lab reports, I knew that many plagiarized. However, it would be impossible to handle such many cases ..."

A student: "...There are some self-motivated students who worked on the assignment. However, it's too easy to get a working copy from the Internet. Some simply copied it, and the others submitted modified versions ..."

Deploying Needle in the Teaching Practice

- The "Principles and Techniques of Compilers" course
  - `C--` $\rightarrow$ IR $\rightarrow$ MIPS32 assembly
- Code plagiarism control
  - let the students know (important)
  - use both tools (Needle, and a faster tool for cross-semester)
  - manual inspection and interview

Overall Results: Plagiaristic Project Groups Confirmed



Cheater's Code Examples

```
cur->lineno = temp->lineno;
strcpy(cur->type, type);
cur->isLexical = 0;
cur->children = temp;
```

```
head->number_signal = 0;
head->line = temp->line;
strcpy(head->type,type);
head->child_left = temp;
```

```
$$->is_root=1;
$$->no_leaves=1;
$$->leaves[0]=(Node*)$1;
if(exit_error==0)
  {print_tree($$,0);}
```

```
$$->final=0;
$$->num_children=1;
$$->children=(Node**)malloc
   (sizeof(Node*)*$$->
       num_children);
$$->children[0]=(Node*)$1;
if(!wrong) printNode($$,0);
```

```
temp->line = a->line;
temp->lChild = a;
while(num > 1){
  a->rChilds = va_arg(list,
      node*);
  a = a->rChilds;
  num--;
}
```

```
p_node->left_child = temp;
p_node->line = temp->line;
for(int i=0;i < num-1;++i){
  temp->right_child =
    va_arg(valist, struct
        Node*);
  temp = temp->right_child;
}
```

Another Example (From Another Programming Assignment)

A piece of code to trick code plagiarism detectors

```
...
draw_string("Game␣Over");
...
```

```
...
draw_string("G");
draw_string("A");
draw_string("M");
draw_string("E");
draw_string("␣");
draw_string("O");
draw_string("V");
draw_string("E");
draw_string("R");
...
```

Outline

Lessons and Challenges

- Plagiarism is a seductive way for students to obtain unethical grades
  - ~20% of the project groups plagiarized in the study
  - students attacked the plagiarism detection tool
  - cross-semester plagiarism (~30%) is also common
- Using plagiarism control policies *alleviate* but not *eliminate* the issue
  - there is a heavy burden on the instructors to conduct code interviews

Thank You!