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]

JPlag [Prechelt and Malphol, Journal of Universal Comp. Sci. ’02]

Cluster Level Detection

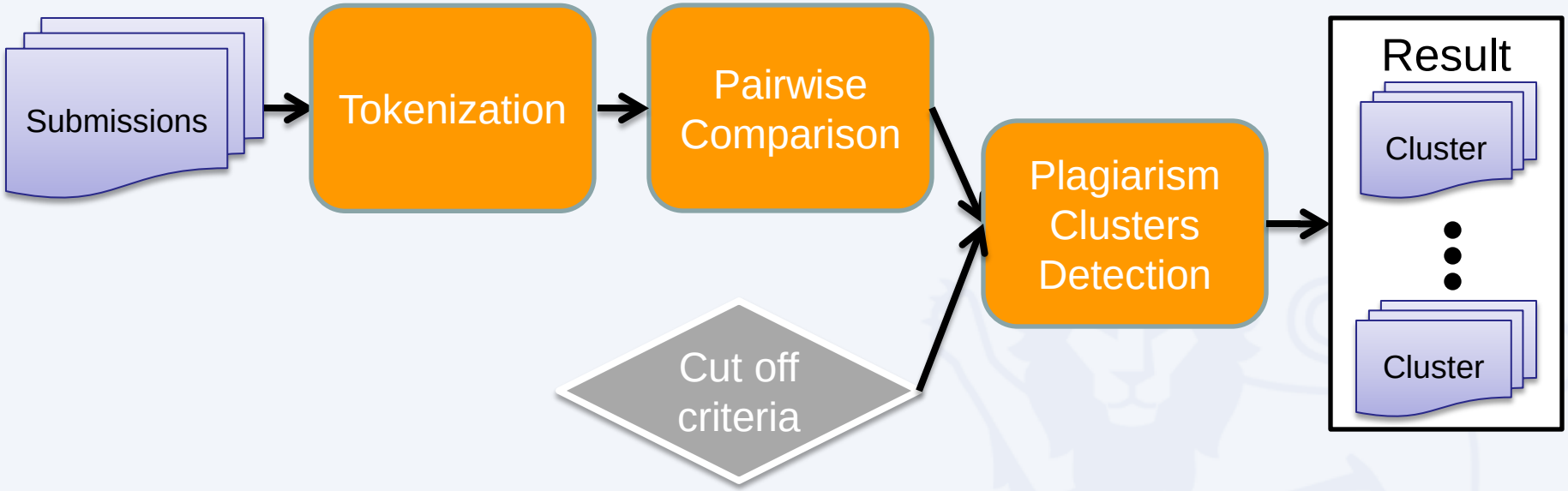
PDetect [Moussiades and Vakali, The Comp. Journal ’05]

PDE4Java [Jadalla and Elnagar, Journal of BI and DM ’08]

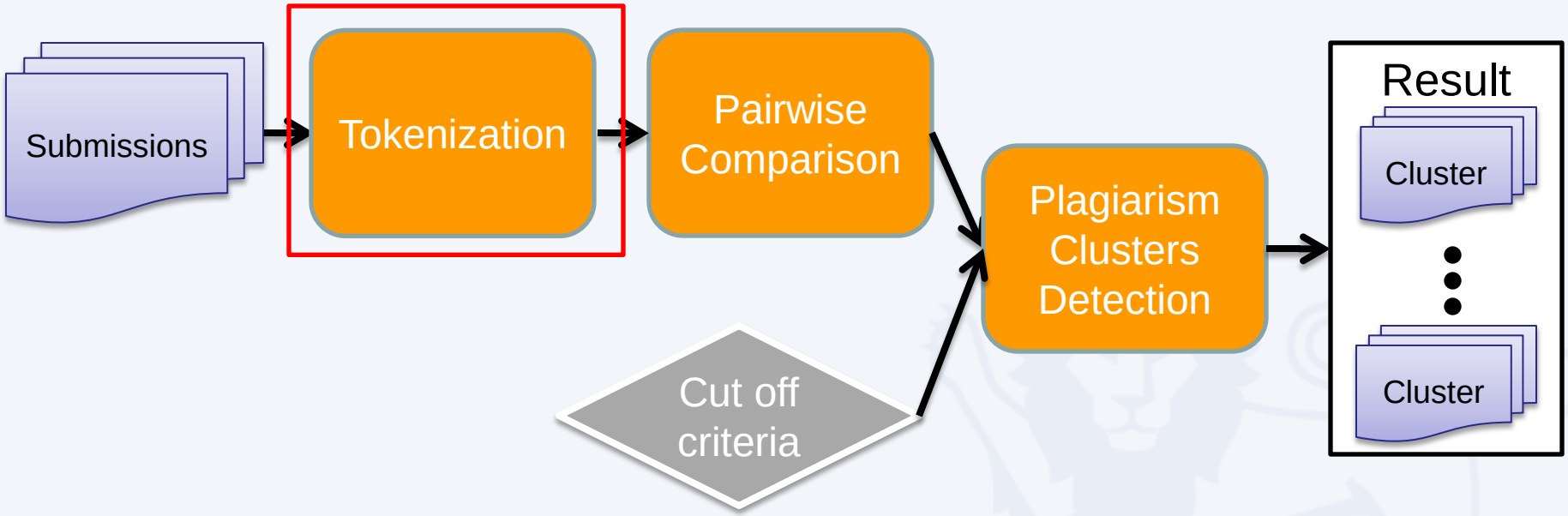
- **Plagiarists can easily confuse the system by inserting non-functional code that are larger than *MML*.**
- **Most of the systems employ pairwise level detection.**

Plagiarism Detection Method

Our approach focuses on how plagiarism is carried out.



Plagiarism Detection Method



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

```
[1] public class MyClass {
[2]     public static void main(String[] args) {
[3]         int value = 1;
[4]         for (;value<10;value++) System.out.println(value + “”);
[5]     }
[6] }
```

Example of Parsing Code

```
[1] public class MyClass {
[2]     public static void main(String[] args) {
[3]         int value = 1;
[4]         for (;value<10;value++) System.out.println(value + “”);
[5]     }
[6] }
```

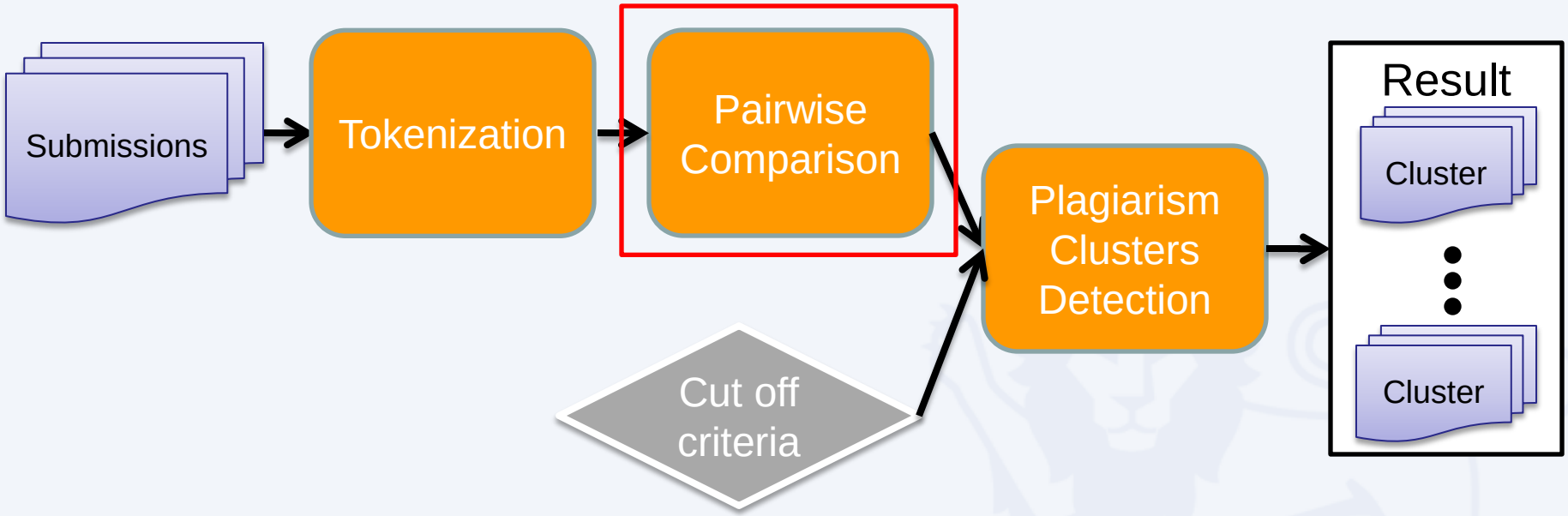
Line ID	Keyword Tokens
[1]	class
[2]	void
[3]	int
⋮	⋮

Line ID	Variable Tokens
[1]	MyClass
[2]	main
[2]	String
⋮	⋮

Line ID	Symbol Tokens
[1]	{
[2]	(
[2]	[
⋮	⋮

Line ID	Constant Tokens
[3]	1
[4]	10
⋮	⋮

Plagiarism Detection Method



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

ABCDEFGH

EFGABCDH

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

ABCDEFGH

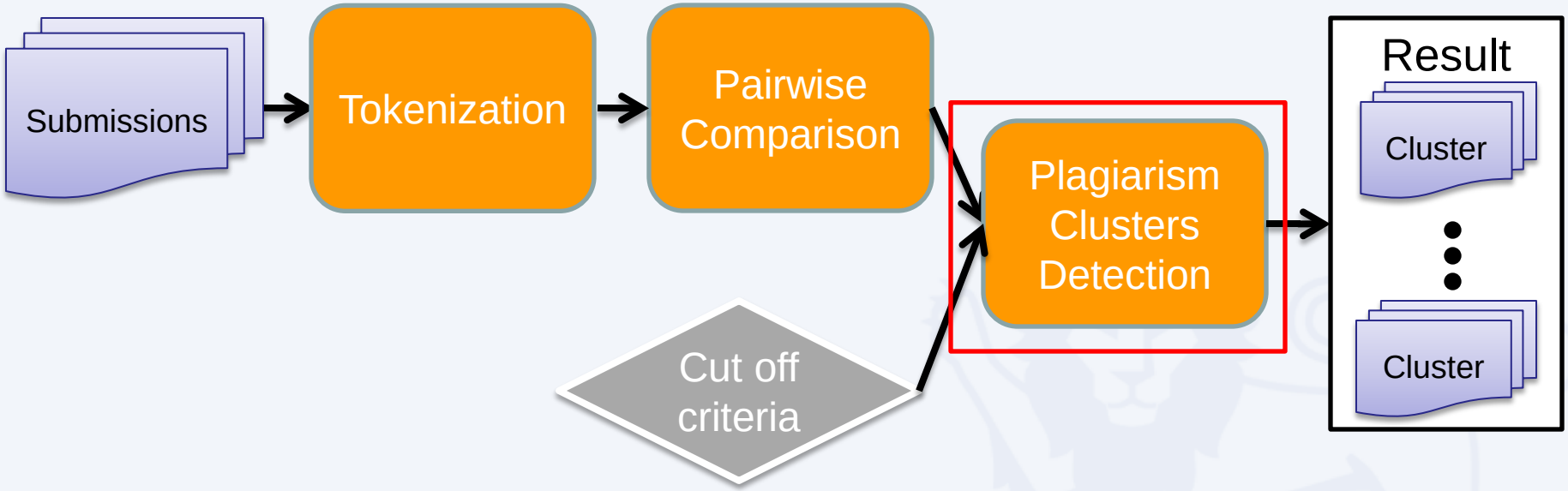
EFGABCDH

Example of Pairwise Comparison

```
        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);
}
```

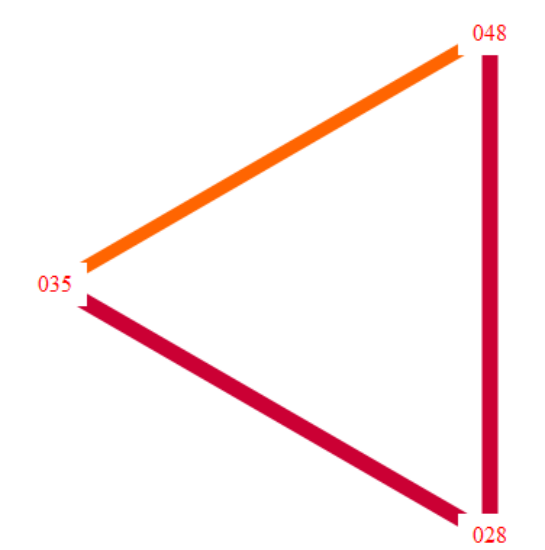
```
private void drawLine(Graphics g,
    int xOld, int yOld, int x, int y) {
        g.setColor(Color.white);
        g.drawLine(xOld + 25, yOld +
            25, x + 25, y + 25);
}

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) {
```

Plagiarism Detection Method



Plagiarism Clusters Detection

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



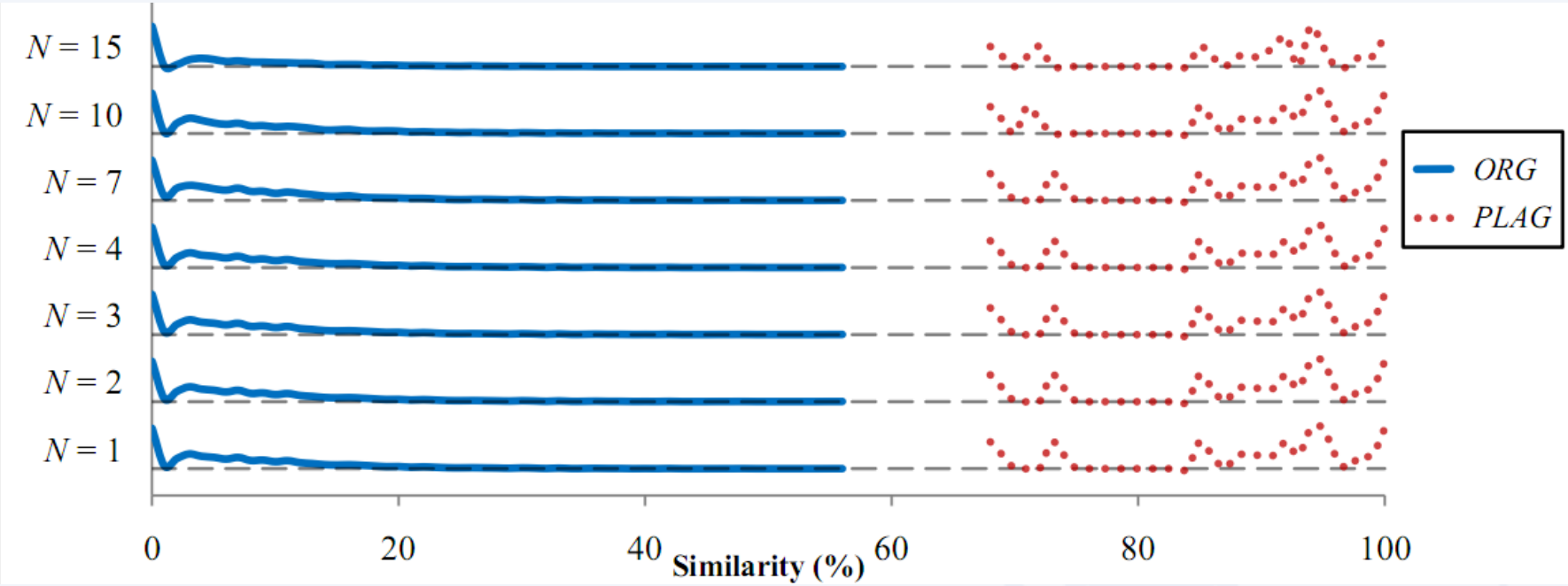
- Performance
  - More than 80 introductory programming assignments (over 3,600 submission pairs)

    ⇨ 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)



ORG: Original non-plagiarized submissions
PLAG: Plagiarized submissions

Our system successfully differentiates between *ORG* and *PLAG*.

Attacks Performed by Student Volunteers

"Attacks": plagiarism attempts

- Immutable attacks
- Size dependent attacks
- Successful attacks

Immutable Attacks

Attacks that our system can protect

Type of attacks	The number of confused attacks	The number of observed attacks
Insertion, modification or deletion of comments	0	35
Indention, spacing or line breaks modifications	0	38
Identifier renaming	0	41
Constant modification	0	2
Insertion, modification, or deletion of modifiers	0	6
No change	0	0

(122 attacks in total)

Identifier Renaming

```
int value = 1;
```

(a) Original submission

```
int v = 1;
```

(b) Plagiarized copy

Our system detect this type of plagiarism.

Size Dependent Attacks

Attacks that needs large modification

Type of attacks	The number of confused attacks	The number of observed attacks
Reordering of independent statements	6	10
Reordering of methods	6	16
Insertion or removal of parentheses	0	20
Inlining or refactoring of code	13	18

(64 attacks in total)

Reordering of Independent Statements

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

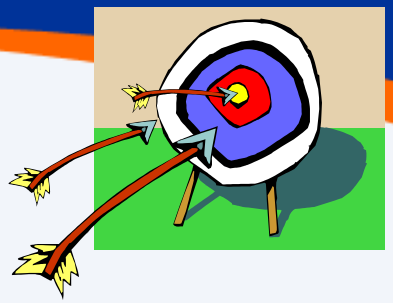
(a) Original submission

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

(b) Plagiarized copy

Our system detect this type of plagiarism.

Succesful Attacks



Type of attacks	The number of confused attacks	The number of observed attacks
Redundancy	8	8
Scope modification	7	7
Modification of control structures	14	14
Declaration of variables	10	10
Modification of method parameters	1	1
Modification of import statements	2	2
Introduction of bug	1	1
Modification of temporary variables in expressions	10	10
Modification of mathematical operations and formulae	2	2
Structural redesign of code	5	5

(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.

Submitted By: 036

```
37    int totalQuery = sc.nextInt();
38    for (int i = 0; i < totalQuery; i++){
39     int start = sc.nextInt();
40     int end = sc.nextInt();
41     if (SolveQuery(start, end))
42      System.out.println("YES");
43     else
44      System.out.println("NO");
45    }
46   }
47
48   // Solve a Query, return if and only if ther
49   public static boolean SolveQuery(int start,
50    if (start == -1 || end == -1) return false;
51    if (!safeCity[start] || !safeCity[end]) ret
52    if (start == end) return true;
53
54    return SolveQuery(parentCity[start], parent
55   }
56
57  }
58
```

Submitted By: 039

```
19    int totalQuery = sc.nextInt();
20    for (int i = 0; i < totalQuery; i++){
21     int start = sc.nextInt(); int end = sc.nex
22     if (SolveQuery(start, end)) System.out.pri
23    }
24   }
25
26   public static void traceTree() {
27    pC = new int[aCs];
28    for (int i = 0; i < aCs; i++) pC[i] = -1;
29    for (int m = 0; m < aCs; m++){
30     int lt = sc.nextInt(); int rt = sc.nextInt
31     if (lt != -1) pC[lt] = m;
32     if (rt != -1) pC[rt] = m;
33    }
34   }
35
36   public static boolean SolveQuery(int start,
37    if (start == -1 || end == -1) return false;
38    if (!sfC[start] || !sfC[end]) return false;
39    if (start == end) return true;
40    if (SolveQuery(pC[start], pC[end]) || Solve
```

Selected matched segment | Matched segments | Skeleton code matched segment | Non-matched segment

Log System

Student: 038

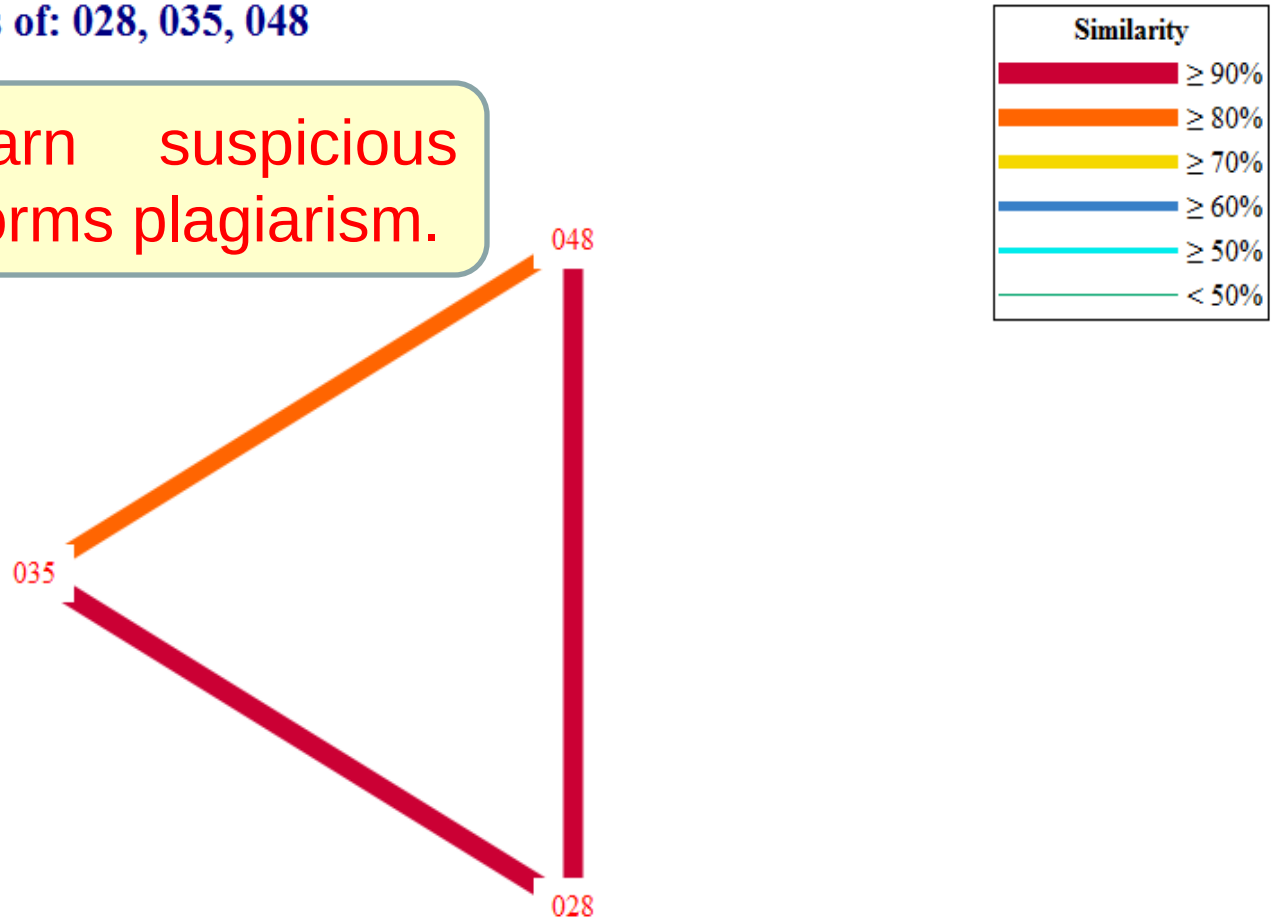
DATE/TIME	MODULE	ASSIGNMENT	GRADER	REMARKS
05/04/2010 19:53:18	CS3256	Individual Project	Yee Fan Tan	Reported submissions from students 038 and 053 as suspicious
01/04/2010 12:50:20	CS2143	1	Min-Yen Kan	The student is found guilty in plagiarism
01/04/2010 12:50:20	CS2143	1	Min-Yen Kan	Confirmed submissions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

Instructors learn
- suspicious pairs of students,
- plagiarism cases.

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 matric in red denotes the student is found guilty in plagiarism for the assignment
- To mark / unmark a student, click the student matric
- 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 matric

Assignment	1
Cut off criterion	83.825%
Plagiarism Clusters	038 [hide] 053 --- 028 [hide] 035 048

Assignment	2
Cut off criterion	88.324%
Plagiarism Clusters	053 [hide] 063 066 --- 043 [hide] 047 064

Ranking

Cut off criterion: ≥ 80%

Rank	Student	Found
1	053	2
2	028	1
3	063	1
4	064	1
5	043	1
6	066	1
7	035	1
8	047	1
9	048	1
10	038	1

Plagiarism Activities Monitoring

CS2105: Intro

- Student matric in red
- To mark / unmark a s
- To show / hide plagiarism cluster;
- To view student's summary, mov ursor to the student matric

Assignment	1
Cut off criterion	83.825%
Plagiarism Clusters	038 [hide] 053 028 [hide] 035 048

Assignment	2
Cut off criterion	88.324%
Plagiarism Clusters	053 [hide] 063 066 043 047 064

Instructors learn suspicious student pairs.

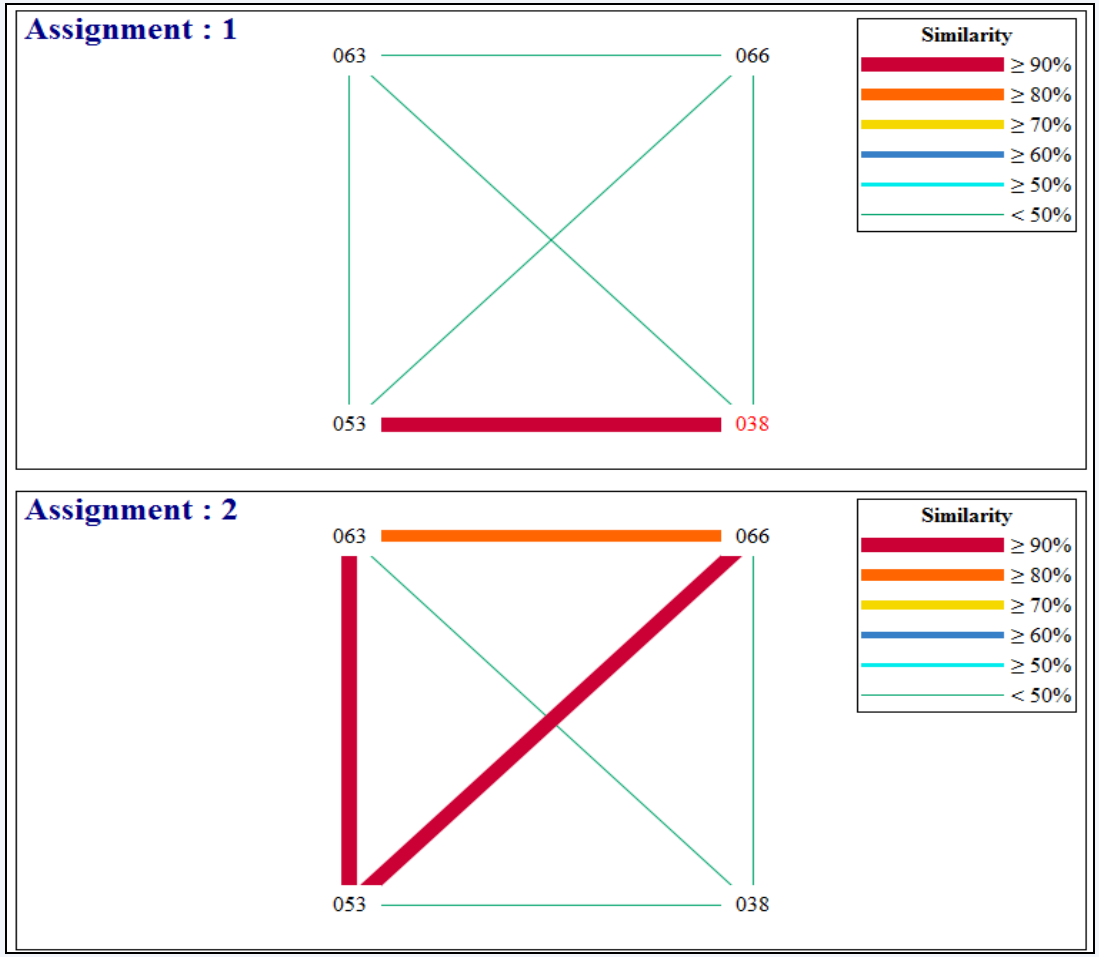
Ranking

ut off criterion: ≥ 80%

Rank	Student	Found
1	053	2
2	028	1
3	063	1
4	064	1
5	043	1
6	066	1
7	035	1
8	047	1
9	048	1
10	038	1

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

Top 5 similar submissions to student 038 target student

- 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

Assignments	Similarity (in decreasing order)
1	053 (99.609%) 024 (50.602%) 007 (32.041%) 006 (26.846%) 031 (26.817%)
2	016 (33.245%) 002 (27.443%) 141 (25.572%) 133 (21.414%) 062 (20.168%)
3	141 (19.322%) 139 (18.919%) 133 (18.414%) 020 (17.672%) 007 (17.672%)

Similarity
- ≥ 90%
- ≥ 80%
- ≥ 70%
- ≥ 60%
- ≥ 50%
- < 50%

Instructors find the top *k* students paired up with the target student "038."

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*!**