A Method to Evaluate CFG Comparison Algorithms

Patrick P.F. Chan
Christian Collberg
Research problem

- Which CFG similarity algorithm is better?
- I come up with a new algorithm, how does it compare to the existing ones?
- Is there a systematic way to compare CFG similarity algorithms?
Research outcomes

- A methodology to evaluate and compare CFG similarity algorithms
- Comparison results of four CFG similarity algorithms
- A survey of existing CFG similarity algorithms
- A publicly available evaluation framework
What is CFG?

- CFG stands for control-flow graph
- A CFG represents all possible execution paths of a function
- And thus, it encodes its behavior
```python
a = input()
if a % 2 == 0:
    print("even")
else:
    print("odd")
```
Why do we compare CFGs?
Why do we compare CFGs?

- Malware detection / classification
Why do we compare CFGs?

• Software theft detection
Why do we compare CFGs?

- Programming assignments grading
Why do we compare CFGs?

- Code clones detection

```plaintext
function sort(s) {
    for (int i = 0; i < s.length; ++i)
        for (int j = i + 1; j < s.length; ++j)
            if (s[i] > s[j]) swap(s[i], s[j]);
}
... ...
function sort(s) {
    for (int i = 0; i < s.length; ++i) {
        swapped = false
        for (int j = i + 1; j < s.length; ++j) {
            if (s[i] > s[j]) {
                swap(s[i], s[j]);
                swapped = true;
            }
        }
    } if (!swapped) break;
}
... ...
```

How similar?
Why do we compare CFGs?

- Detection of changes between different versions of a program
Why do we compare CFGs?

• Detection of changes between different versions of a program

Match the nodes of the enhanced CFGs
This leads to many algorithms to compare CFGs...
Let’s use two existing algorithms to compare these two CFGs

CFG A

CFG B
Algorithm 1 from Kruegel et al.

- Extract subgraphs that have \( k \) nodes (\( k \)-subgraphs) from CFGs and match them
No match!
Algorithm 2 from Hu et al.

- Approximates the minimum number of edit operations needed to transform one graph into another graph.
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>5</th>
<th>3</th>
<th>∞</th>
<th>∞</th>
<th>∞</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>∞</td>
<td>5</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CFG A

CFG B

Total cost = 5
And there are many other algorithms...

- Algorithm from Vujosević-Jančić et al. iteratively builds a similarity matrix between the nodes of the two CFGs, based on the similarity of their neighbor.

- Algorithm from Sokolsky et al. models the control flow graphs using *Labeled Transition Systems* (LTS).
But which one is the best?
Evaluation of CFG similarity algorithms

• Start by generating CFGs $G_1, G_2, \ldots, G_i$ with increasing edit distances with respect to a seed CFG $G_0$

  • i.e. $\text{ED}(G_0, G_i) = i$

• Use the algorithm under evaluation to rank the CFGs such that the higher is the similarity score between $G_i$ and $G_0$ given by that algorithm, the higher $G_i$ is ranked

• Get a “goodness score” for the algorithm by comparing the ranking it produces to the ground truth $\langle G_1, G_2, G_3, \ldots \rangle$, using ranking correlation algorithms such as sortedness or Pearson correlation
Example

$G_0$
Example

\[ G_0 \]

- \[ G_1 \]
  - \[ \text{ED} = 1 \]
- \[ G_2 \]
  - \[ \text{ED} = 2 \]
- \[ G_3 \]
  - \[ \text{ED} = 3 \]
Example

Ranking: \langle G_1, G_2, G_3 \rangle
Example

G0
  └─ G1
      └─ G2
          └─ G3

G0
  └─ G1
      └─ SimA = 0.4
          └─ G3
              └─ SimA = 0.8
                  └─ G2
                      └─ SimA = 0.1

ED = 1
ED = 2
ED = 3

Ranking: \langle G_1, G_2, G_3 \rangle
Example

$G_0$  
\[ \text{ED} = 1 \]

$G_1$

\[ \text{ED} = 2 \]

$G_2$

$G_3$

Ranking: $\langle G_1, G_2, G_3 \rangle$

$G_0$

$G_1$

\[ \text{Sim}_A = 0.4 \]

$G_2$

$G_3$

\[ \text{Sim}_A = 0.8 \]

Ranking: $\langle G_3, G_1, G_2 \rangle$
Example

\[ \text{Ranking: } \langle G_1, G_2, G_3 \rangle \]

\[ \text{Ranking: } \langle G_3, G_1, G_2 \rangle \]

Pearson correlation = -0.5
Two questions remain...

1. What is the definition of the edit distance between two CFGs?

2. How to generate those CFGs such that they have increasing edit distances with the seed CFG $G_0$?
What is the definition of the edit distance between two CFGs?

- The Graph Edit Distance is a function $ED : (G_i, G_j) \rightarrow N$ that computes the smallest number of edit operations needed to transform $G_i$ into $G_j$.
- There are four possible edit operations.
What is the definition of the edit distance between two CFGs?

- Add a zero-degree node
What is the definition of the edit distance between two CFGs?

- Add an edge between two existing nodes
What is the definition of the edit distance between two CFGs?

- Delete an edge between two existing nodes
What is the definition of the edit distance between two CFGs?

- Delete a zero-degree node
How to generate those CFGs such that they have increasing edit distances with the seed CFG $G_0$?
How to generate those CFGs such that they have increasing edit distances with the seed CFG G0?

For every possible edit operation that can be applied to G0, apply that and generate a new graph.
How to generate those CFGs such that they have increasing edit distances with the seed CFG $G_0$?

Do the same for the newly generated graphs.

Obtain the Edit Distance Graph (EDG).
How to generate those CFGs such that they have increasing edit distances with the seed CFG $G_0$?

Randomly pick a CFG on each level and they become our $G_1, G_2, G_3, \ldots$
Implementation

• Re-coded four CFG similarity algorithms in Python
• Implemented the evaluation framework
• Generated an EDG with five levels
• Picked 100 test cases (each test case comprises five CFGs)
Evaluation results

(a) $A_{Hu}$
Evaluation results

(b) $A_{Kruegel}$
Evaluation results

(c) A Vujošević-Janičić
Evaluation results

(d) $A_{\text{Sokolsky}}$
Evaluation results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average</th>
<th>Max(Best)</th>
<th>Min(Worst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{Hu}$</td>
<td>0.885</td>
<td>1</td>
<td>-0.3</td>
</tr>
<tr>
<td>$A_{Kruegel}$</td>
<td>0.486</td>
<td>1</td>
<td>-0.9</td>
</tr>
<tr>
<td>$A_{Vujošević-Janičić}$</td>
<td>0.805</td>
<td>1</td>
<td>-0.4</td>
</tr>
<tr>
<td>$A_{Sokolsky}$</td>
<td>0.409</td>
<td>1</td>
<td>-0.8</td>
</tr>
</tbody>
</table>

“Goodness score” statistics of the four algorithms
## Evaluation results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total time used (sec)</th>
<th>Relative time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}_{\text{Hu}}$</td>
<td>1.996</td>
<td>1.1</td>
</tr>
<tr>
<td>$\mathcal{A}_{\text{Kruegel}}$</td>
<td>1.815</td>
<td>1.0</td>
</tr>
<tr>
<td>$\mathcal{A}_{\text{Vujošević-Janičić}}$</td>
<td>6.179</td>
<td>3.4</td>
</tr>
<tr>
<td>$\mathcal{A}_{\text{Sokolsky}}$</td>
<td>2.315</td>
<td>1.28</td>
</tr>
</tbody>
</table>

Time used by the four algorithms to finish 100 test cases
Related work

• An evaluation framework for text plagiarism detection

• Generate artificial plagiarism cases

• Shuffling, removing, inserting, or replacing words or short phrases at random
Related work

• An evaluation framework for code clone detection tools

  • Inject mutated code fragments into the code base
Future work

- Generate CFGs with instructions in the nodes

Edit instructions => huge EDG
Try our framework

- http://cfgsim.cs.arizona.edu/
- Evaluate existing algorithms
- Compare your own algorithm with the others
- Fine tune your algorithm
Summary

• A methodology to evaluate CFG similarity algorithms

• Publicly available evaluation framework

• Serves as a benchmark for CFG similarity algorithms users / researchers
Thank you!