Project Update #2

Dataset creation & GPLAG summary

Nick Montanaro Bernie Cecchini

CSCI 729, Spring 2017

Outline

1. Initial dataset creation
   - Thoughts
   - Code

2. GPLAG
   - Program Dependence Graph approach
   - GPLAG algorithm

Outline

Initial dataset thoughts

- Need control group to compare effectiveness of several tools
- Non-GPLAG clone detection uses tokenization, string comparison
  - GPLAG uses PDG to detect plagiarism, even when just semantic
  - Dataset should be simple so we know our implementation works

Outline

Common problem templates

```
public class FizzBuzz{
  public static void main(String[] args){
    for(int X = 1; X <= 100; X++){
      if(X % 15 == 0){
        System.out.println("FizzBuzz");
      }else if(X % 3 == 0){
        System.out.println("Fizz");
      }else if(X % 5 == 0){
        System.out.println("Buzz");
      }else{
        System.out.println(X);
      }
    }
  }
}
```

- Solutions to common introductory code problems from Rosetta Code

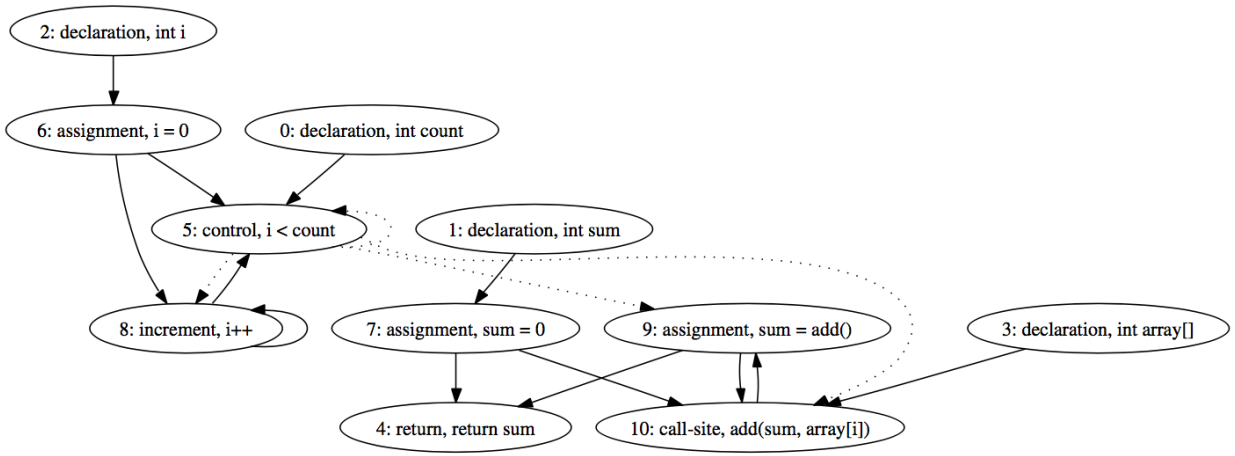
Simple generator code

```
while ((line = br.readLine()) != null) {
  bw.write(line.replaceAll("X", Character.toString(c))
    .replaceAll("Y", Character.toString(c2))
    [further replacements]
  bw.newLine();
}
```

- Iterating through all lines of template files, replacing and randomizing certain variables
- Output as many variations of templates as needed
- Simple changes ensure all tools should yield similar results

Outline

PDG approach vs. others



(a) Program Dependence Graph of the Procedure `sum`

```
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;
}
```

(b) Summation over an Array

- PDGs catch semantic similarities while other methods do not
  - JPLAG and Moss most common plagiarism detection alternatives, both use tokenization

Outline

GPLAG algorithm overview

Algorithm 1 GPLAG($\mathcal{P}$, $\mathcal{P}'$, K, γ, α)

Input: $\mathcal{P}$: The original program
$\mathcal{P}'$: A plagiarism suspect
K: Minimum size of nontrivial PDGs, default 10
γ: Mature rate in isomorphism testing, default 0.9
α: 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: **for each** $g \in \mathcal{G}_K$
6: **let** $\mathcal{G}'_{K,g} = \{g' | g' \in \mathcal{G}'_K, |g'| \geq \gamma|g|, (g, g') \text{ passes filter}\}$
7: **for each** $g' \in \mathcal{G}'_{K,g}$
8: **if** g is γ-isomorphic to g'
9: $\mathcal{F} = \mathcal{F} \cup (g, g')$
10: **return** $\mathcal{F}$;

Pre-generated PDGs
Paper used CodeSurfer

Discard PDGs below certain size
Easy search space reduction

Implemented in C++
Used VFLib

Summary

- Dataset creation is trivial, but need to test across all tools
- PDGs are a more effective way to compare source code files for plagiarism
- GPLAG uses PDGs and subgraph isomorphism with filters to efficiently and accurately detect similarities

- Next steps
  - Implement GPLAG if possible - need tool for PDG generation
  - Compare other tools, including GRAAL, to GPLAG using our dataset
  - Test on extended, real world dataset