SourcererCC

Code Clone Detection

Hanhan, Michael, Robert, Xiaoyu
March 14, 2017

Agenda

- Introduction to the problem
- Algorithms
  - Core idea: bag-of-tokens
  - Optimizations (the main contributions of the paper)
- Evaluation
- Discussion

Code Clone

What is Code Clone ?

- Copy & Paste
- With/Without Minor Modifications

Why Code Clone ?

- Development Strategy
- Maintenance Benefits
- Overcome Underlying Limitations
- By Accident

Drawbacks

- Bug Propagation
- New Bugs
- Bad Design
- Challenging System Improvement
- Increased Maintenance Cost
- Increased Resources

Clone Detection is Necessary!

Clone Detection

Advantages

- Detect Library Candidates
- Find Usage Patterns
- Detect Malicious Program
- Detect Plagiarism
- Other

Terminology in Clone Detection

Clone Pair

- A pair of code fragments, (f1, f2)
- Clone type, φ

Clone Class

- A set of code fragments, (f1, f2,... fn)
- Clone type, φ

Clone Types

- Textual Similarity
    - Type-1
    - Type-2
    - Type 3
- Functional Similarity
    - Type-4

Code Clone Types Example - Type 1, 2

Type-1: Identical, except **Whitespace, Layout and Comments**

```
if (a >= b) {
    c = d + b; // Comment1
    d = d + 1;}
else
    c = d - a; //Comment2
```

```
if (a>=b) {
    // Comment1'
    c=d+b;
    d=d+1;}
else  // Comment2'
    c=d-a;
```

Type-2: Identical, except **Names of Identifiers, Types, Values**

```
if (a >= b) {
    c = d + b; // Comment1

   d = d + 1;}
else
   c = d - a; //Comment2
```

```
if (m >= n)
    { // Comment1'
    y = x + n;
    x = x + 5; //Comment3
    }
else
    y = x - m; //Comment2'
```

Code Clone Types Example - Type 3, 4

Type-3: Identical, except **Statements changed/added/deleted**

```
if (a >= b) {
     c = d + b; // Comment1
     d = d + 1;}
else
     c = d - a; //Comment2
```

```
if (a >= b) {
    c = d + b; // Comment1
    e = 1; // This statement is added
    d = d + 1; }
else
    c = d - a; //Comment2
```

Type-4: Functional Similar, may not be copied from each other

Iteration

```
int i, j=1;
  for (i=1; i<=VALUE; i++)
       j=j*i;
```

vs

Recursion

```
int factorial(int n) {
 if (n == 0) return 1 ;
 else        return n * factorial(n-1) ;
}
```

Highlights of SourcererCC

- Accurate Type-3 Detection
- Large Scale Clone Detection (250M LOC)
- Single Machine
- Small Index
- Language Agnostic

Bag-of-tokens

- Context: strategy used by SourcererCC to compare code blocks
    - Similar to bag-of-words-model in Information Retrieval

Some definitions

- A project P has a collection of code blocks B
    - $P : \{B_1, ..., B_n\}$
- A code block B has a collection of tokens (i.e. bag-of-tokens)
    - $B : \{T_1, ..., T_k\}$
- Tokens: programming language keywords, literals, and identifiers stored as a pair (token, frequency)

Problem Formulation

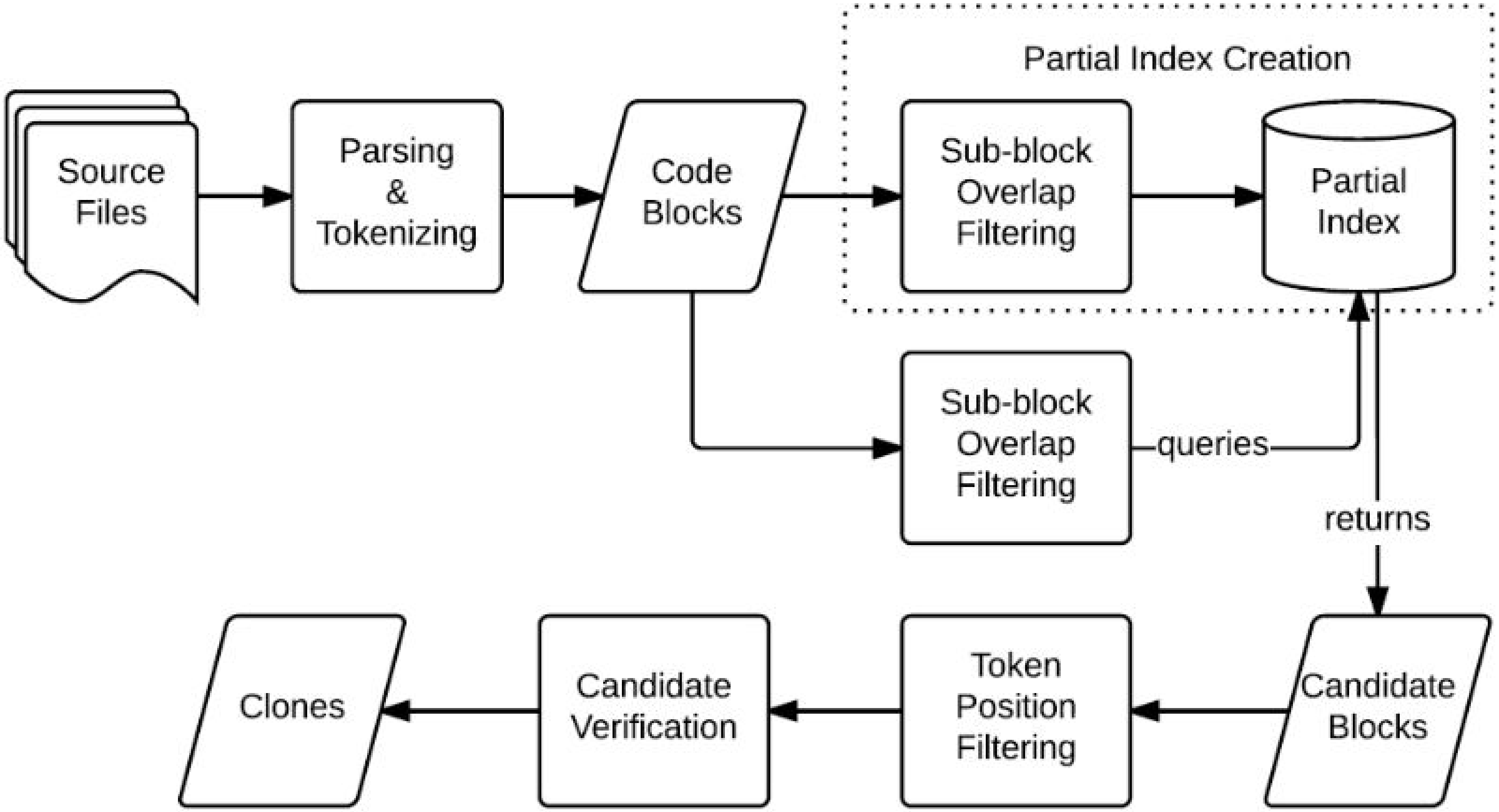
- Use a function to measure the degree of similarity between code blocks, and those with a value higher than some threshold are identified as clones.
- Given:
  - Two projects P_x and P_y
  - a similarity function f
  - a threshold θ
- Aim:
  - Find all block pairs/groups $P_x.B$ and $P_y.B$

    s.t. $f(P_x.B, P_y.B) \geq \text{ceiling}(\theta \times \max(|P_x.B|, |P_y.B|))$

Problem Formulation

- Overlap is used for similarity function *f* here.
  - The overlap similarity $O(B_x, B_y)$ is computed as the number of source tokens shared by B_x and B_y.
  - $O(B_x, B_y) = |B_x \cap B_y|$
  - i.e. if $\theta = 0.8$, $\max(|B_x|, |B_y|) = t$, then B_x and B_y are clones if they share at least ceiling($\theta|t|$) tokens
- Complexity: $O(n^2)$ on number of code blocks vs. number of comparisons (with method granularity)
  - We can improve this using filtering heuristics to reduce # of comparisons.

SourcererCC's Workflow



Baseline & Inverted Index

- Baseline
  - 2-level for loop to enumerate all pairs
- Inverted index
  - A map of (token -> code blocks)
  - Many methods to exploit this data structure
  - How to make the most use of it?

Optimizations

- Heuristic filtering
- Purpose
    - Reduce the number of candidate clones for each code block during clone detection (and reduce memory usage)

Optimizations

- Heuristic filtering
- Purpose
  - Reduce the number of candidate clones for each code block during clone detection (and reduce memory usage)
- Two methods
  - Sub-block Overlap Filtering
  - Token Position Filtering

Sub-block Overlap Filtering

- General idea
  - By Pigeonhole Principle, when two ordered lists have a large intersection, their subsequences must have a certain number of overlaps.
  - Original problem is large - take its complement
- Property
  - Given blocks Bx and By consisting of t tokens each ***in some predefined order***, if $|Bx \cap By| \geq i$, then the subblocks SBx and SBy of Bx and By respectively, consisting of first $(t-i+1)$ tokens, must match at least one token.
- Usage
  - For given Bx and candidate By consisting of t tokens, if first $(t-i+1)$ have no matching tokens, then By cannot be a clone of Bx.

Sub-block Overlap Filtering - Example

- Given Bx = {a, b, c, d, e} By = {b, c, d, e, f}, $t = 5$, $\theta = 0.8$
  - $i = \text{ceiling}(0.8*5) = 4$
  - $t - i + 1 = 5 - 4 + 1 = 2$
  - Bx' = {a, **b**} and By' = {**b**, c} share one common token
  - By stays in candidate list

- If Bx = {a, b, c, d, e} By = {c, d, e, f, g}, $t = 5$, $\theta = 0.8$ is given
  - Bx' = {a, b} and By' = {c, d} share no common token
  - Reject By from candidate list
  - (In practice, we do it the other way around - only add By to the candidate list of Bx if there is a shared token, queried via Inverted Index.)

Token Position Filtering

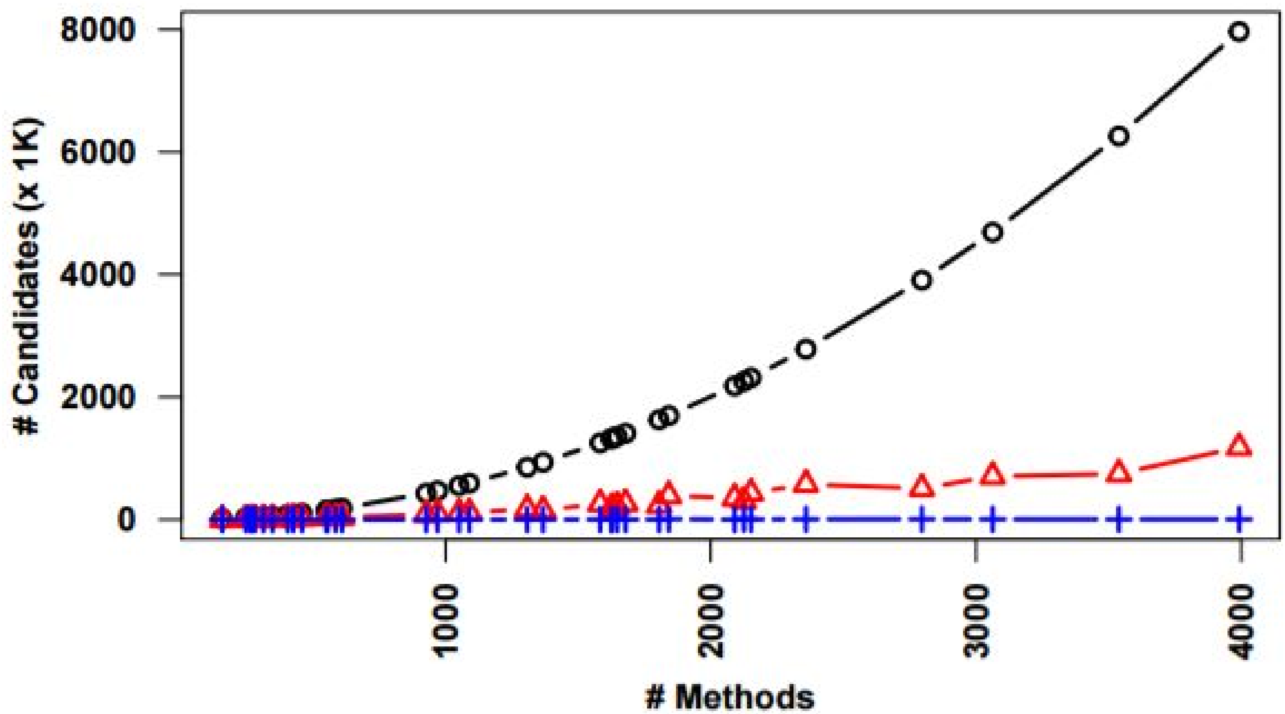
- Given Bx = {a, b, c, d} By = {b, c, d, e, f}, $t = 5$, $\theta = 0.8$
  - Still, $i = 4$ and $t - i + 1 = 2$
  - Bx' = {a, b} and By' = {b, c} share a common token
  - But since the two blocks have only 3 tokens in common, they **cannot** be identified as a clone pair.

- How to exploit this constraint?
  - Count the current matched tokens and the minimum number of unseen tokens in Bx and By!

Token Position Filtering - Example

- Given Bx = {a, b, c, d} By = {b, c, d, e, f}, $t = 5$, $\theta = 0.8$
    - Still, $i = 4$ and $t - i + 1 = 2$
    - Bx’ = {a, b} and By’ = {b, c} share a common token
    - Share token number = 1
    - Minimum number of unseen tokens are 2 for Bx and 4 for By
    - $1 + \min(2, 4) = 3 < 4 = i$
    - Safely reject By from the candidate list

How filtering reduce the comparisons



Red: Applied Sub-block Overlap Filtering

Blue: Applied Token Position Filtering

Evaluation - Methodology

- Four state-of-the-art competitors
  - Configurations “based on our extensive previous experiences with the tools, as well as previous discussions with their developers”
- Evaluate scalability using inter-project IJaDataset
- Evaluate recall & precision using BigCloneBench & Mutation Injection

Evaluation - Scalability

- Main variable: size of input (LoC)
- How to build inputs of different sizes?
  - Randomly sample from IJaDataset

Evaluation - Scalability

- Main variable: size of input (LoC)
- How to build inputs of different sizes?
    - Randomly sample from IJaDataset
    - Smaller inputs are subset of larger inputs.

Evaluation - Scalability

Table 2: Execution Time (or Failure Condition) for Varying Input Size

LOC	SourcererCC	CCFinderX	Deckard	iClones	NiCad
1K	3s	3s	2s	1s	1s
10K	6s	4s	9s	1s	4s
100K	15s	21s	1m 34s	2s	21s
1M	1m 30s	2m 18s	1hr 12m 3s	MEMORY	4m 1s
10M	32m 11s	28m 51s	MEMORY	—	11hr 42m 47s
100M	1d 12h 54m s5s	3d 5hr 49m 11s	—	—	INTERNAL LIMIT

Findings:

- Keep in mind: CCFinderX only finds Type-1 and 2 clones.
- Small inputs can be ignored (constant overhead dominates).
- Quadratic increase of time can be seen in 100K and above.
- SourcererCC really performs better.

Evaluation - Scalability

Table 2: Execution Time (or Failure Condition) for Varying Input Size

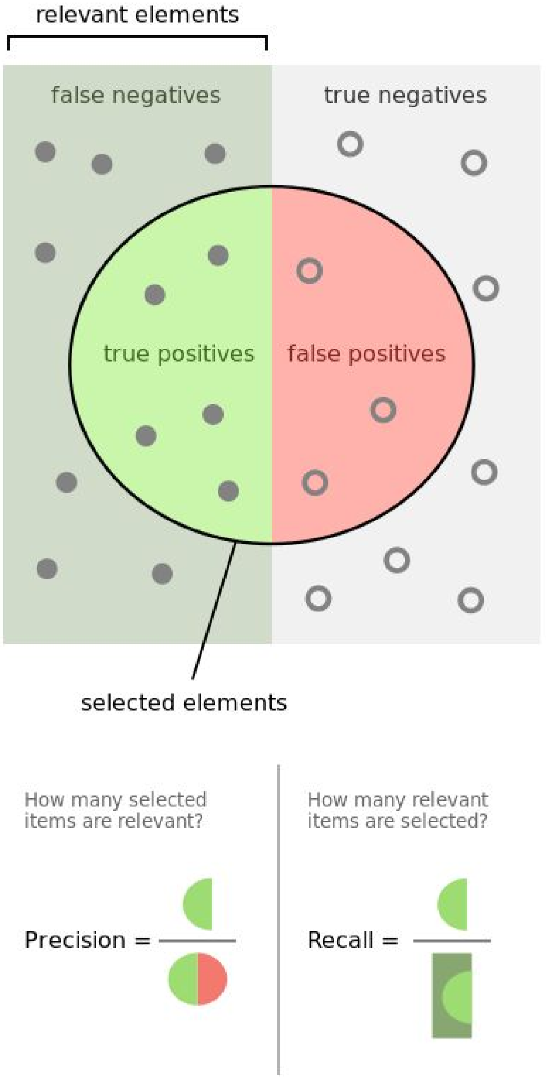
LOC	SourcererCC	CCFinderX	Deckard	iClones	NiCad
1K	3s	3s	2s	1s	1s
10K	6s	4s	9s	1s	4s
100K	15s	21s	1m 34s	2s	21s
1M	1m 30s	2m 18s	1hr 12m 3s	MEMORY	4m 1s
10M	32m 11s	28m 51s	MEMORY	—	11hr 42m 47s
100M	1d 12h 54m s5s	3d 5hr 49m 11s	—	—	INTERNAL LIMIT

Potential problems:

- Do they really hit the “limit”?
- Deckard and NiCad may be too slow eventually, but iClones seems promising.

Evaluation - Recall & Precision

- Precision measured by humans via random sampling
  - More on this later
- Recall is hard to measure
  - Impractical to reliably sample because of low ratio of true positives
  - Lack of “oracles”
  - Two benchmarks previously created by the same authors
    - Synthetic mutation-and-injection
    - Real-world BigCloneBench



Evaluation - Recall

Table 3: Mutation Framework Recall Results

Tool	Java			C			C#		
	T1	T2	T3	T1	T2	T3	T1	T2	T3
SourcererCC	100	100	100	100	100	100	100	100	100
CCFinderX	99	70	0	100	77	0	100	78	0
Deckard	39	39	37	73	72	69	-	-	-
iClones	100	92	96	99	96	99	-	-	-
NiCad	100	100	100	99	99	99	98	98	98

Mutation-and-Injection:

- High recall rates across the board
- Might be biased?

Evaluation - Recall

BigCloneBench:

- Mined from IJaDataset and manually validated (?) based on *semantical similarity*
- Categorized into finer types based on *syntactical similarity*

Table 4: BigCloneBench Clone Summary

Clone Type	T1	T2	VST3	ST3	MT3	WT3/T4
# of Clone Pairs	35787	4573	4156	14997	79756	7729291

Evaluation - Recall

Table 5: BigCloneBench Recall Measurements

Tool	All Clones						Intra-Project Clones						Inter-Project Clones					
	T1	T2	VST3	ST3	MT3	WT3/T4	T1	T2	VST3	ST3	MT3	WT3/T4	T1	T2	VST3	ST3	MT3	WT3/T4
SorcererCC	100	98	93	61	5	0	100	99	99	86	14	0	100	97	86	48	5	0
CCFinderX	100	93	62	15	1	0	100	89	70	10	4	1	98	94	53	1	1	0
Deckard	60	58	62	31	12	1	59	60	76	31	12	1	64	58	46	30	12	1
iClones	100	82	82	24	0	0	100	57	84	33	2	0	100	86	78	20	0	0
NiCad	100	100	100	95	1	0	100	100	100	99	6	0	100	100	100	93	1	0

Findings:

- SourcererCC suitable for VST3+ (lack of identifier normalization)
- NiCad has better recall rate
- Clones generated by mutation-and-injection are mostly at least VST3

Evaluation - Precision

- Randomly selected 390 detected clones from BigCloneBench
  - 95% confidence level and ±5% confidence interval
- Split validation across 3 clone experts (?)
- SourcererCC 91%
  - Higher than or comparable to the claimed precision of competitors
  - Argue against the high precision of Deckard & NiCad

Discussion

Sub-block Overlap Filtering - If not preordered

- Given $S1 = \{1, 2, g, f, e, a, b, c, d, e, h\}$

  $S2 = \{3, 4, a, b, c, d, e, f, g, h\}$ $t = 5, \theta = 0.8$

  $i = 0.8*10 = 8$

  $t - i + 1 = 3$

  $S1' = \{1, 2, g\}$ $S2' = \{3, 4, a\}$