SourcererCC
-- Scaling Code Clone Detection to Big-Code
What did this paper do?

SourcererCC

a *token-based clone detector*, that can *detect* both exact and *near-miss clones* from *large* inter project repositories using a standard workstation.
Code Clones -- separate fragments of code that are very similar.

Why are there so many duplicated code?
- Reusing code fragments by copying and pasting

Why it need to be detected?
- harmful in software maintenance and evolution
int[] s;
int i;
s = new int[5];

for (i = 0; i < 5; i++) {
    s[i] = i;
}

int[] s;
int i;
s = new int[5];
//initiation
for (i = 0; i < 5; i++) {
    s[i] = i;
}
int[] s;
int i;
s = new int[5];

for(i = 0 ; i < 5 ; i++){
s[i] = i;
}

int[] array;
int i;
array = new int[6];

for(i = 0 ; i < 6 ; i++){
    array[i] = i;
}
Type 3 (near-miss clones)  
-- statements, added, modified

```java
int[] s ;
int i ;
s = new int[5] ;

for(i = 0 ; i < 5 ; i++){
    s[i] = i ;
}
```

```java
int[] s = new int[5] ;

for(int i = 0 ; i < 5 ; i++){
    s[i] = i ;
}
```
Type 4
-- same functionality, different Syntax

```java
int[] s ;
int i ;
s = new int[5] ;

for(i = 0 ; i < 5 ; i++){
    s[i] = i ;
}
```

```java
int[] array = {0, 1, 2, 3, 4};
```
DEFINITIONS

**Code Fragment:** A continuous segment of source code

**Clone Pair:** A pair of code fragments that are similar

**Clone Class:** A set of code fragments that are similar

**Code Block:** 1) A sequence of code statements within braces, 2) is represented as a bag-of-tokens

**A token:** programming language keywords, literals, and identifiers.
DEFINITIONS

**Degree of similarity** -- The higher the value, the greater the similarity between the code blocks.

**Threshold** -- if the **Degree of similarity** is greater than **threshold**, the code blocks are considered as code clones.
```java
int i;
int[] s = new int[5];
for (i = 0; i < 5; i++) {
    s[i] = i;
}
```

To find: \(|B_1 \cap B_2| \geq \theta \cdot \max(|B_1|, |B_2|)\)

\(B_1 = \{a, b, c, d, e\} \) (5 tokens)

\(B_2 = \{b, c, d, e, f\} \) (5 tokens)

\(|B_1 \cap B_2| = 4\)

If threshold \(\theta = 0.8\)

\(\theta \cdot \max(|B_1|, |B_2|) = 0.8 \cdot \max(5, 5) = 4\)
Intuitive way

\[ B_1 = \{a, \ b, \ c, \ d, \ e\} \]

\[ B_2 = \{b, \ c, \ d, \ e, \ f\} \]

\[ O(n^2) \]
Heuristics Filter 1

B1 = \{a, b, c, d, e\}

B2 = \{b, c, d, e, f\}

with 5 tokens (t = 5)

threshold θ = 0.8

if clones : 0.8 \times 5 = 4 \text{ tokens (i=4)}.

How many different tokens

\[
t - i + 1 = 2\ 	ext{tokens match at least 1 token.}
\]

Which ORDER is best?

Similar to natural languages - the popularity of tokens

Change the order means change the source code meaning.
Ineffective case

\[ B_1 = \{a, b, c, d\} \text{ (4 tokens)} \]
\[ B_2 = \{b, c, d, e, f\} \text{ (5 tokens)} \]

Let threshold \( \theta = 0.8 \)

to be considered clones: \( 0.8 \times 5 = 4 \text{ tokens (i=4)} \)

2 tokens match at least 1 token
Heuristics Filter 2

\[ B_1 = \{ a, b, c, d \} \text{ (4 tokens)} \]

If clones --- 4 same tokens

Filter 1: 1 same token

Upper bound: \( 1 + \min(2, 4) = 3 \)

\[ B_2 = \{ b, c, d, e, f \} \text{ (5 tokens)} \]
Result
Clone Detection Algorithm

Two stages:
(1) **Partial Index Creation** - creates indexes for tokens only in sub-blocks.
(2) **Clone Detection** - a) establishes a map of candidates; b) Candidate Verification (check candidates that were not rejected in step a)

Filter 1 is involved in (1)
Filter 2 is involved in (2)
Detection of Near-miss (Type-3) clones

Type-3: The fragments have statements **added**, **modified** and/or **removed** with respect to each other

Semantic difference, but syntactic similarity at a token level to be detected as similar.
### Evaluation -- Execution time

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

Also can be executed for the IJaDataset (250MLOC)

Can out of memory issue indicate the scalability?
Evaluation -- Recall
(Percent of genuine clones that are reported)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Java</th>
<th></th>
<th></th>
<th>Java</th>
<th></th>
<th></th>
<th>Java</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td>SourcererCC</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>CCFinderX</td>
<td>99</td>
<td>70</td>
<td>0</td>
<td>100</td>
<td>77</td>
<td>0</td>
<td>100</td>
<td>78</td>
<td>0</td>
</tr>
<tr>
<td>Deckard</td>
<td>39</td>
<td>39</td>
<td>37</td>
<td>73</td>
<td>72</td>
<td>69</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>iClones</td>
<td>100</td>
<td>92</td>
<td>96</td>
<td>99</td>
<td>96</td>
<td>99</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NiCad</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>98</td>
<td>98</td>
<td>98</td>
</tr>
</tbody>
</table>

*Table 3: Mutation Framework Recall Results*
Evaluation -- Recall using BigCloneBench

Table 4: BigCloneBench Clone Summary

<table>
<thead>
<tr>
<th>Clone Type</th>
<th>T1</th>
<th>T2</th>
<th>VST3</th>
<th>ST3</th>
<th>MT3</th>
<th>WT3/T4</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Clone Pairs</td>
<td>35787</td>
<td>4573</td>
<td>4156</td>
<td>14997</td>
<td>79756</td>
<td>7729291</td>
</tr>
</tbody>
</table>

Table 5: BigCloneBench Recall Measurements

<table>
<thead>
<tr>
<th>Tool</th>
<th>All Clones</th>
<th></th>
<th>Intra-Project Clones</th>
<th></th>
<th>Inter-Project Clones</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T2</td>
<td>VST3</td>
<td>ST3</td>
<td>MT3</td>
<td>WT3/T4</td>
</tr>
<tr>
<td>SorcererCC</td>
<td>100</td>
<td>98</td>
<td>93</td>
<td>61</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>CCFinderX</td>
<td>100</td>
<td>93</td>
<td>62</td>
<td>15</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Deckard</td>
<td>60</td>
<td>58</td>
<td>62</td>
<td>31</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>iClones</td>
<td>100</td>
<td>82</td>
<td>82</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NiCad</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>95</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Evaluation -- Precision

<table>
<thead>
<tr>
<th></th>
<th>SourcererCC</th>
<th>CCFinderX</th>
<th>Deckard</th>
<th>iClones</th>
<th>NiCad</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>83</td>
<td>72</td>
<td>28</td>
<td>91</td>
<td>56</td>
</tr>
<tr>
<td><strong>Precision (10LOC)</strong></td>
<td>86</td>
<td>79</td>
<td>30</td>
<td>93</td>
<td>80</td>
</tr>
<tr>
<td><strong>Recall</strong>¹</td>
<td>90</td>
<td>75</td>
<td>53</td>
<td>78</td>
<td>99</td>
</tr>
<tr>
<td>**Recall(T3)**²</td>
<td>68</td>
<td>26</td>
<td>38</td>
<td>38</td>
<td>96</td>
</tr>
</tbody>
</table>

¹ Including T1, T2, VST3, ST3.  
² Including VST3, ST3.
Summary of its evaluation methodology

- Evaluated the **scalability, execution time, recall** and **precision**
- Compared 4 publicly available and state-of-the-art **tools**.
- Used two recent **benchmarks**: (1) a big benchmark of real clones, BigCloneBench, and (2) a Mutation/Injection-based framework of thousands of fine-grained artificial clones.
Related work: How is the research different from prior work?

Rattan et al. --- found very few tools target scalability to very large repositories.

Liveri et al. --- partition the input for the large repositories, distribution of the executions over a large number of machines.

Svajlenko et al. --- non-deterministic shuffling heuristic to reduce the number of tool execution, reduction in recall.

**SourcererCC** --- scale to **large** repositories on a **single** machine.
Related work: How is the research different from prior work?

Ishihara et al. --- MD5 hashing to scale method-clone detection, fast execution time, cannot detect Type-3 clones

Hummel et al. --- first to use an index-based approach, detect only Type-1 and Type-2 clones

Others have scaled clone detection in domain-specific ways

**SourcererCC** --- small index, detects **Type-3 clones**, **generic** method
What is learnt from the paper?

Idea is simple, implementation may be simple too

Why can it be published on ICSE?
Formally, given two projects $P_x$ and $P_y$, a similarity function $f$, and a threshold $\theta$, the aim is to find all the code block pairs (or groups) $P_x.B$ and $P_y.B$ s.t $f(P_x.B, P_y.B) \geq [\theta \cdot \max(|P_x.B|, |P_y.B|) ]$. Note that for intra-project similarity, $P_x$ and $P_y$ are the same. Similarly, all the clones in a project repository can be revealed by doing a self-join on the entire repository itself.

While there are many choices of similarity function, we use Overlap\textsuperscript{3} because it intuitively captures the notion of overlap among code blocks. For example, given two code blocks $B_x$ and $B_y$, the overlap similarity $OS(B_x, B_y)$ is computed as the number of tokens shared by $B_x$ and $B_y$.

$$OS(B_x, B_y) = |B_x \cap B_y|$$ (1)
Algorithm 1 SourcererCC’s Algorithm - Partial Index Creation

INPUT: $B$ is a list of code blocks $\{b_1, b_2, ..., b_n\}$ in a project/repository, $GTP$ is the global token position map, and $\theta$ is the similarity threshold specified by the user.

OUTPUT: Partial Index($I$) of $B$

1: function CREATEPARTIALINDEX($B, \theta$)
2:     $I = \phi$
3:     for each code block $b$ in $B$ do
4:         $b = \text{SORT}(b, GTP)$
5:         tokensToBeIndexed = $|b| - [\theta \cdot |b|] + 1$
6:         for $i = 1 : \text{tokensToBeIndexed}$ do
7:             $t = b[i]$
8:             $I_t = I_t \cup (t, i)$
9:         end for
10:     end for
11: return $I$
12: end function
Evaluation

1. Show impressive results
2. Cover the features we mentioned before
Discussion:
Should we change the order of tokens?

Change the order means change the meaning

Same material can generate different code
Discussion: Is code clone totally bad?

Learning is kind of code clone. We clone the knowledge from professors, from books ...

Understand it and build up our own knowledge system
How can we partition the program to different blocks?