

NIL: Large-Scale Detection of Large-Variance Clones

Tasuku Nakagawa, Yoshiki Higo, Shinji Kusumoto

29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). 830–841.

背景

- 近年、多数行変更が加えられたクローンを検出する手法が構築されている
- 多数行変更が加えられたクローン検出は計算量を要するため計算効率が悪くなる

多数行変更クローン

• Large Gap

- ひとまとまりの複数の文が追加または削除されたクローン
- 検出器：CCAligner [37]

```
1 protected int run(Commandline cmd) {
2   try {
3     Execute exe = new Execute(new LogStreamHandler(this,
4       Project.MSG_INFO, Project.MSG_WARN);
5     exe.setAntRun(getProject());
6     exe.setWorkingDirectory(getProject().getBaseDir());
7     exe.setCommandline(cmd.getCommandline());
8     exe.setVMLauncher(false);
9     return exe.execute();
10  } catch (java.io.IOException e) {
11    throw new BuildException(e, getLocation());
12  }
```

(a) Clone A

```
1 protected int run(Commandline cmd) {
2   try {
3     Execute exe = new Execute(new LogStreamHandler(this,
4       Project.MSG_INFO, Project.MSG_WARN);
5     if (serverPath != null) {
6       String[] env = exe.getEnvironment();
7       if (env == null) {
8         env = new String[0];
9       }
10      String[] newEnv = new String[env.length + 1];
11      System.arraycopy(env, 0, newEnv, 0, env.length);
12      newEnv[newEnv.length - 1] = "SSDIR=" + serverPath;
13      exe.setEnvironment(newEnv);
14    }
15    exe.setAntRun(getProject());
16    exe.setWorkingDirectory(getProject().getBaseDir());
17    exe.setCommandline(cmd.getCommandline());
18    exe.setVMLauncher(false);
19    return exe.execute();
20  } catch (java.io.IOException e) {
21    throw new BuildException(e, getLocation());
22  }
```

(b) Clone B

Figure 1: Example of large-gap clones

• Large Variance

- 複数の文が散らばって追加または削除されたクローン
- 検出器：LVMapper [40]

```
1 protected String getPrompt(InputRequest request) {
2   String prompt = request.getPrompt();
3   if (request instanceof MultipleChoiceInputRequest) {
4     StringBuffer sb = new StringBuffer(prompt);
5     Enumeration e = ((MultipleChoiceInputRequest) request)
6       .getChoices().elements();
7     boolean first = true;
8     while (e.hasMoreElements()) {
9       if (!first) {
10        sb.append(",");
11      }
12      sb.append(e.nextElement());
13      first = false;
14    }
15    sb.append(" ");
16    prompt = sb.toString();
17  }
18  return prompt;
19 }
```

(a) Clone A

```
1 protected String getPrompt(InputRequest request) {
2   String prompt = request.getPrompt();
3   String def = request.getDefaultValue();
4   if (request instanceof MultipleInputChoiceRequest) {
5     StringBuilder sb = new StringBuilder(prompt).append(" ");
6     boolean first = true;
7     for (String next : ((MultipleInputChoiceRequest) request)
8       .getChoices()) {
9       if (first) {
10        sb.append(",");
11      }
12      if (next.equals(def)) {
13        sb.append(' ');
14      }
15      sb.append(next);
16      if (next.equals(def)) {
17        sb.append(' ');
18      }
19      first = false;
20    }
21    sb.append(" ");
22    return sb.toString();
23  }
24  else if (def != null) {
25    return prompt + "[" + def + "]";
26  }
27  else {
28    return prompt;
29  }
```

(b) Clone B

Figure 2: Example of large-variance clones

[37] Pengcheng Wang, Jeffrey Svajlenko, Yanzhao Wu, Yun Xu, and Chanchal K Roy. 2018. CCAligner: a token based large-gap clone detector. In Proceedings of the 40th International Conference on Software Engineering. 1066–1077.

[40] Ming Wu, Pengcheng Wang, Kangqi Yin, Haoyu Cheng, Yun Xu, and Chanchal K Roy. 2020. LVMapper: A Large-Variance Clone Detector Using Sequencing Alignment Approach. IEEE Access 8 (2020), 27986–27997.

目的・アプローチ

- 目的

- 計算効率の良いLarge Varianceクローン検出手法を構築する

- アプローチ

- 前処理にN-gramsを用いてコードを加工し，字句単位で類似度を計算する

手法概要

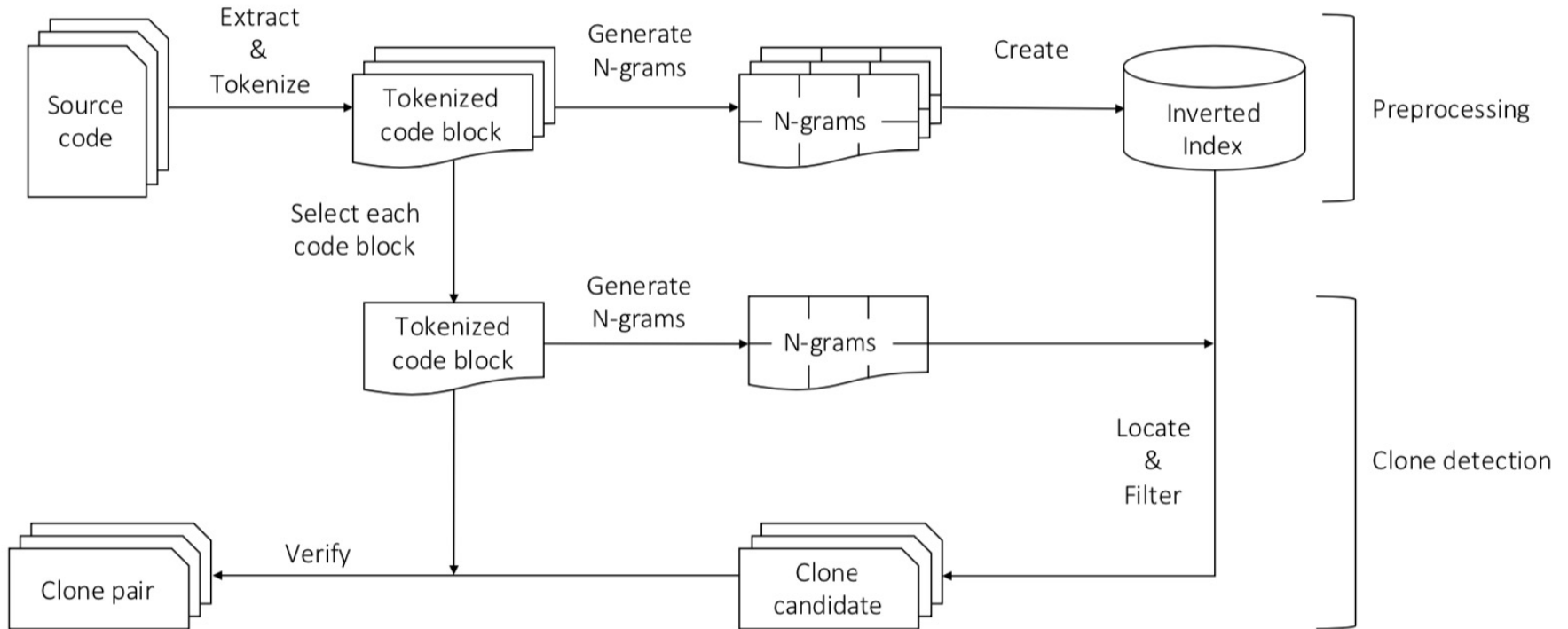


Figure 4: Overview of NIL

前処理

- 字句分割
 - ソースコードを空白や演算子，カッコ等で区切る
 - 字句解析を用いないことで言語依存がなくなる，高速になるという利点がある
- N-grams, 転置indexデータベースの作成
 - 字句からN-gramsを作成
 - 転置indexを割り当て，N-gramsと字句の対応付けと転置indexをクローン検出に利用出来るようにする

本処理

- Location Phase

- コードブロックからN-gramsを作成
- N-gramsをハッシュ値化
- 転置indexからハッシュ値を含むコードブロックをクローン候補に含める

Algorithm 1: Clone Detection

Input: C is a list of tokenized code blocks $\{c_1, c_2, \dots, c_n\}$,
Inverted Index I of C , N for size of N-gram, θ for filtering
threshold, δ for verifying threshold

Output: All clone pairs CP

```
1:  $CP \leftarrow \phi$ ;  
2: for all each  $c_i$  in  $C$  do  
3:   // Location phase  
4:   //  $CC$  represents clone candidates  
5:    $CC \leftarrow \phi$   
6:   for  $j = 1, 2, \dots, (c_i.len - N + 1)$  do  
7:     //  $c_i[j]$  is  $j$ -th token in  $c_i$ 's token sequence  
8:      $n\_gram = \text{concat}(c_i[j], c_i[j + 1], \dots, c_i[j + N - 1])$ ;  
9:      $key = \text{hash}(n\_gram)$ ;  
10:    /*  $get$  is a function that returns values to which a given  
    key is mapped in a given dictionary */  
11:     $CC = CC \cup get(I, key)$ ;  
12:  end for
```

本処理

- Filtration phase

- 次のフェーズのLCS*の計算コストを削減するために、クローン候補からN-gramsで類似度が低いものを除去

- $$\text{filtration_sim}(c_1, c_2) = \frac{\text{common_ngrams}(c_1, c_2)}{\min(\text{ngrams}(c_1), \text{ngrams}(c_2))}$$
- $$\text{common_ngrams}(c_1, c_2) = |\text{ngrams}(c_1) \cap \text{ngrams}(c_2)|$$

```
14: // Filtration phase
15: for all each  $cc_j$  in  $CC$  do
16:   /* common_ngrams is a function
      that computes the number of common N-grams between
      two given code blocks */
17:    $cn = \text{common\_ngrams}(c_i, cc_j)$ ;
18:    $m = \min(c_i.\text{len}, cc_j.\text{len})$ ;
19:    $\text{filtration\_sim} = cn / (m - N + 1)$ ;
20:   if  $\text{filtration\_sim} < \theta$  then
21:      $CC = CC \setminus \{cc_j\}$ ;
22:   end if
23: end for
```

本処理

- Verification phase

- 残ったクローン候補とLCSを計算し、類似度が閾値であればLarge-Varianceクローン

- $$verification_sim(c_1, c_2) = \frac{lcs(c_1, c_2)}{\min(\min|c_1|, \min|c_2|)}$$

```
25: // Verification phase
26: for all each  $cc_j$  in  $CC$  do
27:   /*  $lcs$  is a function that computes the length of the LCS
   between token sequences of two given code blocks */
28:    $lcs\_len = lcs(c_i, cc_j)$ ;
29:    $verification\_sim = lcs\_len / \min(c_i.len, cc_j.len)$ ;
30:   if  $verification\_sim \geq \delta$  then
31:      $CP = CP \cup (c_i, cc_j)$ ;
32:   end if
33: end for
```

評価実験

- Large Variance クローン検出精度
 - precision：先行研究で用いたプロジェクトに提案手法を実行し手動で正誤判定
 - recall：ミュートーション[31]を用いて擬似的なLarge Variance クローンの正解集合を作成
- 通常クローン検出精度
 - ミュートーション と BigCloneEval の2つのベンチマークを用いて precision, recall を算出
- 計算効率
 - 1, 10, 100, 250MLOC のそれぞれの規模のデータセットで実行時間を計測

データセット, パラメータ

- データセット
 - BigCloneEval[33]
 - ミューテーション
 - 既存手法で用いたプロジェクト

System	# Files	LOC
Ant 1.10.1	895	109,073
Maven 3.5.0	698	60,471

- パラメータ
 - filtration threshold θ : 0.7
 - verification threshold δ : 0.1
 - N-grams : 5

Table 2: Recall and execution time results for each N value

N	1	2	3	4	5	6	7	8	9
Type-2	97.5	97.5	97.5	97.5	96.6	96.3	96.1	95.8	93.3
Very Strongly Type-3	93.5	93.5	93.5	93.5	93.5	93.1	92.9	92.7	92.5
Strongly Type-3	68.4	68.4	68.4	68.3	67.1	65.3	64.6	63.7	62.0
Moderately Type-3	11.2	11.2	11.2	11.1	10.6	9.9	9.3	8.7	7.8
Execution time	> 24h	2h 13m 52s	21m 42s	5m 56s	2m 58s	1m 13s	1m 4s	1m 2s	57s

Large Variance クローン検出精度

- 既存手法よりも大きい差で改善した
- 追加行が増えてもrecallが高いまま

Table 4: Large-variance clone detection results

Tool	System	# Large-variance clones	Precision (%)
NIL	Ant	354	86.0
	Maven	398	88.0
LVMapper	Ant	355	64.0
	Maven	389	60.0
CCAligner	Ant	184	43.0
	Maven	284	40.0

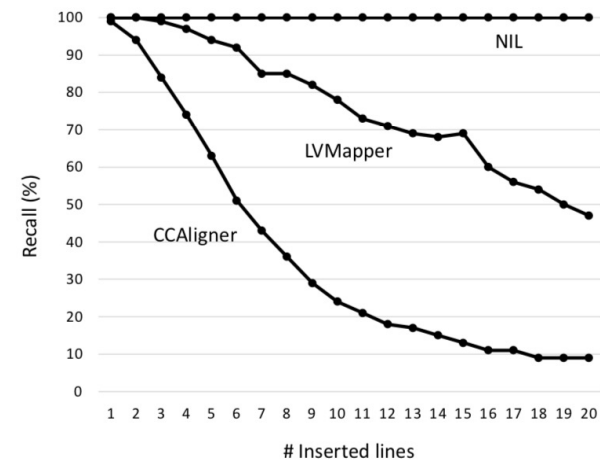


Figure 7: Recall results for various numbers of inserted lines

通常クローン検出精度

- 通常クローンでも精度が高い
- Type3ではLVMapperでしか取得できないクローンがあることがわかった

Table 5: Recall results for Mutation Framework

Tool	NIL	LVMapper	CCAligner	SourcererCC	NiCad
Type-1	100.0	100.0	100.0	100.0	100.0
Type-2	100.0	100.0	100.0	100.0	100.0
Type-3	100.0	99.9	99.9	100.0	100.0

Table 6: Recall and precision results for BigCloneBench

Tool	NIL	LVMapper	CCAligner	SourcererCC	NiCad
Type-1 Recall	99.9	99.9	99.8	93.8	99.9
Type-2 Recall	96.6	99.2	98.9	96.6	99.2
Very Strongly Type-3 Recall	93.5	98.1	97.4	68.2	98.4
Strongly Type-3 Recall	67.1	81.8	69.0	59.0	69.7
Moderately Type-3 Recall	10.6	19.1	10.0	4.8	0.5
Precision	94.0	58.5	33.7	99.2	80.2

計算効率

- Filtration phaseと Location phaseがあることでクローン候補を絞り込める
- 他の検出器に比べて大きく計算効率が高い

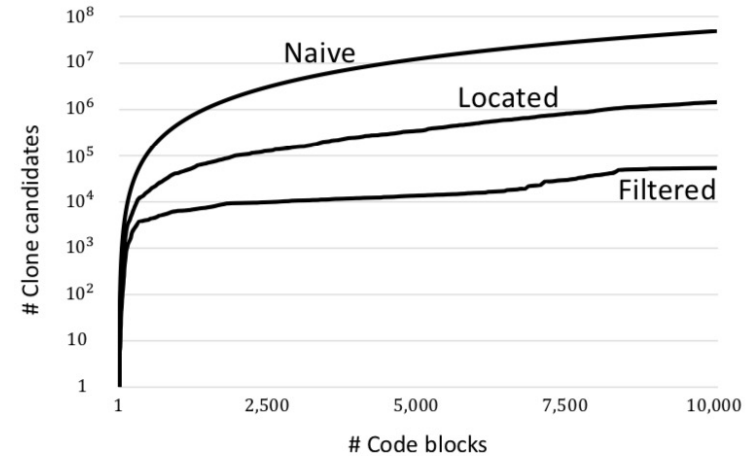


Figure 8: Growth in the number of clone candidates with the increased number of code blocks

Table 7: Scalability results

Tool	NIL	LVMapper	CCAligner	SourcererCC	NiCad
1 M	10s	29s	52s	3m 1s	1m 48s
10 M	1m 38s	13m 38s	26m 3s	27m 37s	—
100 M	1h 38m 29s	17h 23m 39s	—	19h 38m 5s	—
250 M	7h 40m 7s	3d 13h 47m 39s	—	5d 6h 55m 1s	—

まとめ

- N-gramsを用いて計算効率の向上を目的とした Large Variance クローンの検出手法を構築
- Large Variance, 通常どちらのクローンでも精度を大きく向上させた
- 計算効率も大きく向上させた

所管

- 全体的に大きく改善したのが印象的
- **Large Variance** クローンの欠陥混入のしやすさが気になった
- 変更行が大きすぎるともはやクローンではないのではないかと思った