情報処理学会論文誌 Vol.62 No.6 1350–1357 (June 2021)

ギャップを含むクローンセットの検出と評価

土居 真之[1,a)] 肥後 芳樹[1,b)] 楠本 真二[1,c)]

受付日 2020年3月13日, 採録日 2021年3月2日

概要：これまでに多数のクローン検出手法が提案されている．検出手法には互いに類似するコード片のペアであるクローンペア形式で出力する手法と互いに類似するコード片の集合であるクローンセット形式で出力する手法がある．どちらの形式で出力すべきかはクローン情報の用途によって異なるため，クローンの検出は両方の形式で出力できることが好ましい．しかし多くの既存検出手法では文の挿入や削除といったギャップを含むクローンはクローンペアの形式でしか出力できない．さらにクローンセットを検出できる手法は検出の精度やスケーラビリティの面で課題が残る．本研究では極大クリーク列挙アルゴリズムを利用してクローンペア形式で表されたクローンの情報からクローンセットを検出する手法を提案する．提案手法ではコード片を頂点，2つのコード片がクローンペアであることを辺と見なしたグラフを作成したときに極大クリークを1つのクローンセットと見なす．実験では提案手法を実装したツールを5つのオープンソースソフトウェアに対して適用した．提案手法と検出の精度に課題がある既存手法に対して同じクローンペア情報を与えたところ，すべての対象ソフトウェアに対して提案手法の方が多くのクローンセットを検出しており，既存手法で検出されるクローンセットが提案手法では細かく分割されていることが確認できた．また，提案手法はクローンペア情報から数秒でクローンセットを高速に検出できた．

キーワード：ギャップを含むクローン，クローンセット，クリーク

Detection and Evaluation of Gapped Clone Sets

Masayuki Doi[1,a)] Yoshiki Higo[1,b)] Shinji Kusumoto[1,c)]

Received: March 13, 2020, Accepted: March 2, 2021

Abstract: Many clone detection techniques have been proposed before now. There are two output formats of clone detection results. One is the clone pair format, which is a pair of code fragments that are similar to each other. The other is the clone set format, which is a set of code fragments that are similar to one another. The clone pair format is better in some situations while the clone set format is better in other situations. Hence, clone detection techniques should be able to output in both formats. However, most of the existing clone detection techniques that are capable of detecting gapped clones, which contains some gaps such as insertions and/or deletions of statements, can detect only clone pairs. In addition, in the case of the techniques that can detect clone sets, there is another problem that their detection accuracy is low and their scalability is poor. In this paper, we propose a new approach to obtain clone sets from detected clone pairs by using the maximum clique enumeration algorithm. Our proposed approach constructs a graph whose vertices are code fragments of the clone pairs and whose edges are clone pair relationships. Thereafter, maximal cliques in the graph are detected as clone sets. We applied our proposed approach to five open source software. We gave the same clone pairs to the proposed approach and an existing one, which has an issue in detection accuracy. The proposed approach detected more clone sets for all the target software, which means that the clone sets detected by the existing approach were finely devided by the proposed approach. In addition, our proposed approach could detect clone sets from clone pairs for each project in several seconds.

Keywords: gapped clone, clone set, clique

1. はじめに

コードクローン（以下，クローン）とは，ソースコード中に存在する互いに同一，あるいは類似したコード片である．クローンの主な発生要因はコピーアンドペーストで

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0971, Japan
[a)] m-doi@ist.osaka-u.ac.jp
[b)] higo@ist.osaka-u.ac.jp
[c)] kusumoto@ist.osaka-u.ac.jp

ある [17], [19]．クローンは保守性の低下 [13], [15], [23] やバグの伝播 [10], [16], [28] といったソフトウェアに悪影響を与えるとされている．したがって，クローンの検出はリファクタリング [14], [25] やデバッグ [7], [8] といった様々な場面に利用されている．

クローンを表す用語としてクローンペアとクローンセットがある [9] [*1]．**クローンペア**は互いに類似したコード片の組（ペア）を表す．**クローンセット**は互いに類似したコード片の集合（セット）を表す．クローンセット内に存在するコード片の任意のペアはクローンペアである．これまでに多数のクローン検出器が開発されている [17]．検出器は対象のソースコードからクローンペア形式もしくはクローンセット形式で検出したクローンを出力する．どちらの形式で出力すべきかはクローン情報の用途によって異なる．たとえばソースコードの重複度の算出や，ある箇所からバグが見つかりその類似箇所の確認 [10] を行う場合はクローンペア形式の出力で十分であるが，類似するコード片の集約 [25], [27] や，類似するコード片のライブラリへの抽出 [9] やコードの一貫性チェック [11] を行う場合にはクローンセット形式で出力する必要である．

クローンペアは，その 2 つのコード片の間の類似度に応じて，以下の 3 種類に分類できる [3]．

Type-1 空白やタブの有無，括弧の位置などのコーディングスタイルに依存する箇所を除いて，完全に一致するコード片の組．

Type-2 変数名や関数名などのユーザ定義名，また変数の型など一部の予約語のみが異なるコード片の組．

Type-3 文の挿入や削除，変更が行われたコード片の組．

また，上記の分類を利用して，クローンセットも以下の 3 種類に分類できる．

Type-1 含有するコード片のすべての組が Type-1 のクローンペアであるコード片の集合．

Type-2 含有するコード片の組のうち少なくとも 1 つが Type-2 のクローンペアであり，残りのすべての組が Type-1 のクローンペアであるコード片の集合．

Type-3 含有するコード片の組のうち少なくとも 1 つが Type-3 のクローンペアであり，残りのすべての組が Type-1 もしくは Type-2 のクローンペアであるコード片の集合．

また Type-3 のクローンペアやクローンセットは**ギャップを含む**クローンペアやクローンセットとも表現される [26]．以降，本論文ではギャップを含むクローンペアおよびギャップを含むクローンセットを総称してギャップを含むクローンと呼ぶ．既存研究ではギャップを含むクローンがそうでないクローンよりも多く存在すると報告されている [18], [21]．したがってギャップを含むクローンは最も検出の必要があるクローンである．

ギャップを含むクローンを検出可能な検出器も多数開発されている．しかしこれらの検出器の多くはクローンペアの検出のみ可能であり，クローンセットの検出が可能な検出器は限られている．たとえば，SourcererCC [20] は大規模なソフトウェアに対してギャップを含むクローンのペアを検出できるが，クローンセットの検出はできない．一方 CCFinder [9] は suffix tree アルゴリズム [6] により高速にクローンセットを検出するが，ギャップを含むクローンは検出できない．iClones [5] は suffix tree アルゴリズムを拡張してギャップを含むクローンセットを検出するが，検出には膨大なメモリを必要とし，ギャップが大きいクローンは検出できない．NiCAD [4] はクローンセット形式でクローンを検出できるが，NiCAD が出力するクローンセットにはそのなかに存在するコード片の組がクローンペアとなっていない場合がある．たとえば，あるクローンセットにコード片 ABC が含まれており，コード片 A とコード片 B がギャップを含むクローンペアおよびコード片 B とコード片 C がギャップを含むクローンペアであっても，コード片 A とコード片 C がクローンペアとはなっていない場合がある．このため，NiCAD が検出するクローンセットは本節で述べたクローンセットの定義には厳密には従っていない．以上のことから，著者らは高速に定義どおりにギャップを含むクローンセットを検出する手法が必要であると考えた．現在，ギャップを含むクローンを検出可能/不可能の違いにかかわらず多くの検出器がクローンペア形式のみでの出力を行うという現状から，本研究ではクローンペアからクローンセットを生成する手法を提案する．提案手法は特定の検出ツールに依存しているわけではない．提案手法はクローンペアを入力とすることで，任意の既存検出器のポストプロセスとしてクローンセットを得る．ギャップを含むクローンペアを出力する検出器の検出結果を入力することで，本研究の目的であるギャップを含むクローンセットの抽出を可能にしている．

本研究では極大クリーク列挙アルゴリズムを利用して，クローンペアからクローンセットを検出する手法を提案する．提案手法は，検出されたクローンペアのコード片を無向グラフの頂点，クローンペアの関係をそれらの頂点間の辺として表現し，その無向グラフ内に存在している極大クリークを列挙する．列挙された各極大クリークがクローンセットである．極大クリーク列挙アルゴリズムを利用することにより，含有するすべてのコード片がクローンペアとなっているクローンセットを検出できる．

5 つのオープンソースソフトウェアに提案手法を適用した結果，ギャップを含まないクローンセットと比較して平均 3.0 倍多くのギャップを含むクローンセットが検出でき

[*1] 文献 [9] ではクローンセットではなくクローンクラスという名称が用いられているが，近年はクローンクラスという名称よりもクローンセットという名称が多く用いられるようになっているため，本論文でもクローンセットという名称を用いる．

た．さらに提案手法は各プロジェクトのクローンペアからクローンセットを 130 ミリ秒以下で検出できた．

2. 極大クリーク列挙

無向グラフ G の頂点集合 $V(G)$ の部分集合 $C \subseteq V(G)$ のうち，C に含まれる任意の頂点をつなぐ辺が存在する場合に，C を**クリーク**と呼ぶ．すなわち，クリーク C は完全グラフとなる G の誘導部分グラフである．クリークは密な構造を表す最も基礎的な構造である．G の任意のクリークにおいて，自身を除く任意のクリークの部分集合ではないようなものを，極大クリークと呼ぶ．G に存在するすべての極大クリークを検出することを**極大クリーク列挙**と呼ぶ．極大クリーク列挙はデータマイニング [1], [2] や Web マイニング [12] といった様々な研究に利用されている．

極大クリーク列挙の例を**図 1** に示す．図 1 (a) は NiCAD が 1 つのクローンセットとして検出した 6 つのコード片であり，極大クリーク列挙アルゴリズムを利用することで，その中から 2 つの極大クリーク $\{A, B, C, D\}$ と $\{C, D, E, F\}$ が列挙されたことを示している．

3. 提案手法

本研究では極大クリーク列挙アルゴリズムを用いてクローンペアの情報からクローンセットを検出する手法を提案する．提案手法は，クローンペアを検出する任意の既存検出器のポストプロセスとして利用できる．提案手法はすべてのタイプ（Type-1，Type-2，および Type-3）のクローンセットを検出可能である．提案手法は，検出されたクローンペアのコード片を頂点，クローンペアの関係をそれらの頂点間の辺とする無向グラフを構築し，グラフに存在する極大クリークを検出するというアプローチを取る．たとえば，コード片 C_1 とコード片 C_2，コード片 C_2 とコード片 C_3，コード片 C_3 とコード片 C_1 という 3 つのクローンペアが検出されている状況では，各コード片を表す頂点はそれぞれ 2 つのクローンペアで共有されており，頂点 $C_1C_2C_3$ からなるグラフはクリークを成している．このアプローチを取ることにより，クローンペア情報からクローンセット情報を得るという問題を，無向グラフにおける極大クリーク列挙の問題として解くことができる．なお，すべてのクリークを列挙するのではなく，極大クリークのみを列挙する目的は，最大限大きなクローンセットを検出するためである．たとえば，4 つのコード片 $C_4C_5C_6C_7$ からなるクローンセットが検出される場合は，それらの部分集合であるクローンセット $C_4C_5C_6$ や $C_4C_6C_7$ は検出されない．

提案手法の概要を**図 2** に示す．提案手法は入力としてクローンペアを受け取り，クローンペアから形成されるクローンセットを生成する．提案手法はクローンペアを必要とするため，提案手法を利用するためには，図 2 にあるよ

tomcat/bdcp/bdcp2/datasources/PoolKeys.java

```
65 @Override
66 public int hashCode() {
67   final int prime = 31;
68   int result = 1;
69   result = prime * result + ((dataSourceName == null) ? 0 : dataSourceNam…
70   result = prime * result + ((userName == null) ? 0 : userName.hashCode());
71   return result;
72 }
```
A

tomcat/util/descriptor/web/TaglibDescriptorImpl.java

```
42 @Override
43 public int hashCode() {
44   final int prime = 31;
45   int result = 1;
46   result = prime * result
47           + ((location == null) ? 0 : location.hashCode());
48   result = prime * result + ((uri == null) ? 0 : uri.hashCode());
49   return result;
50 }
```
B

tomcat/util/descriptor/web/MessageDestinationRef.java

```
94 @Override
95 public int hashCode() {
96   final int prime = 31;
97   int result = super.hashCode();
98   result = prime * result + ((link == null) ? 0 : link.hashCode());
99   result = prime * result + ((usage == null) ? 0 : usage.hashCode());
100  return result;
101 }
```
C

tomcat/util/descriptor/web/ContextResourceLink.java

```
84 @Override
85 public int hashCode() {
86   final int prime = 31;
87   int result = super.hashCode();
88   result = prime * result + ((factory == null) ? 0 : factory.hashCode());
89   result = prime * result + ((global == null) ? 0 : global.hashCode());
90   return result;
91 }
```
D

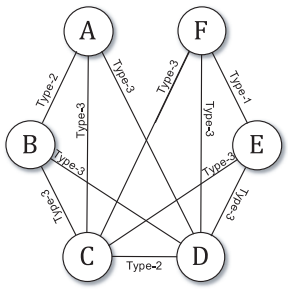
tomcat/util/descriptor/web/ContextEbj.java

```
116 @Override
117 public int hashCode() {
118   final int prime = 31;
119   int result = super.hashCode();
120   result = prime * result + ((home == null) ? 0 : home.hashCode());
121   result = prime * result + ((link == null) ? 0 : link.hashCode());
122   result = prime * result + ((remote == null) ? 0 : remote.hashCode());
123   return result;
124 }
```
E

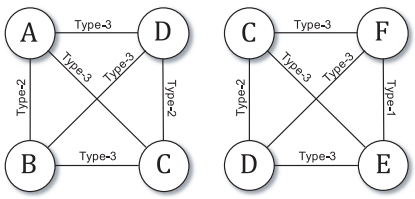
tomcat/util/descriptor/web/ContextLocalEbj.java

```
114 @Override
115 public int hashCode() {
116   final int prime = 31;
117   int result = super.hashCode();
118   result = prime * result + ((home == null) ? 0 : home.hashCode());
119   result = prime * result + ((link == null) ? 0 : link.hashCode());
120   result = prime * result + ((remote == null) ? 0 : remote.hashCode());
121   return result;
122 }
```
F

(a) hashCode メソッド



(b) 構築されたグラフ



(c) 検出された極大クリーク

図 1 極大クリーク列挙の例
Fig. 1 An example of maximal clique enumeration.

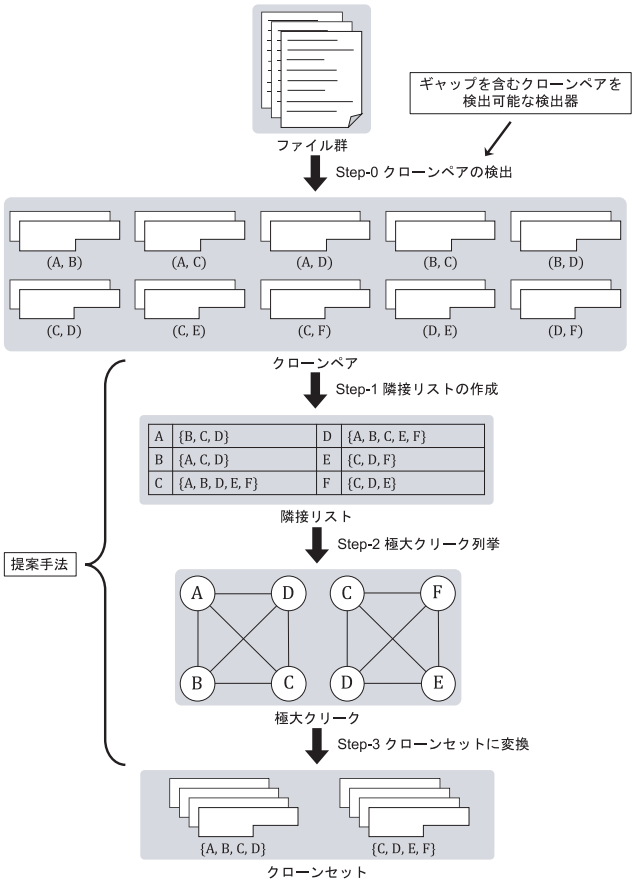


図 2 提案手法の概要
Fig. 2 An overview of the proposed technique.

うに，既存のギャップを含むクローンペアを検出可能な検出器を利用してクローンペアを検出しておく必要がある．

Step-1 隣接リストの生成
Step-2 極大クリーク列挙
Step-3 クローンセットに変換

以降それぞれのステップについて説明する．

Step-1：隣接リストの生成

入力として与えられたクローンペアを基に隣接リストを生成する．隣接リストとはグラフを表すデータ構造の 1 つで，各頂点について直接辺でつながっている頂点集合をリスト構造で表現している．図 2 には隣接リストの例を記載されている．図 2 の隣接リストは図 1 (b) のグラフを表している．図 2 では，Step-0 の結果として，コード片 A を含むクローンペアとして AB，AC，AD が存在するため，頂点 A の隣接リストには頂点 B，C，D が含まれる．

Step-2：極大クリーク列挙

このステップでは Step-1 で生成された隣接リストを基に極大クリーク列挙を行う．2 章で述べたように，クリーク内の任意の頂点間に辺が存在する．提案手法におけるグラフの辺はクローンペアであることを示すため，クリーク内に存在するコード片は同じクリーク内に存在する他のすべてのコード片とクローンペアとなっている．したがって極大クリーク列挙により得られるクリークは，互いに類似したコード片の集合であるクローンセットとなる．

Step-3：クローンセットに変換

極大クリーク列挙により得られたクリークからクローンセットに変換する．極大クリークとクローンセットは一対一に対応しており，図 2 の場合，2 つのクローンセットが得られる．

3.1 提案手法によるクローンセット検出と NiCAD 方式によるクローンセット検出の違い

図 1 を用いて，提案手法によるクローンセット検出と NiCAD によるクローン検出の是非について述べる．図 1 では，6 つの Java メソッドが記載されており，提案手法を利用すると，$\{A, B, C, D\}$ と $\{C, D, E, F\}$ の 2 つのクローンセットが検出されるのに対して，NiCAD を利用すると $\{A, B, C, D, E, F\}$ の 1 つのクローンセットが検出される．

類似するコード片の集約 [25], [27] や類似するコード片のライブラリへの抽出 [9] を行う場合を考える．このような場合は，対象となる複数のコード片は，そのプロジェクト内もしくは外部のライブラリ内に単一のコード片としてまとめられる．複数のコード片を 1 つにまとめるためには，それらは互いに一定以上類似している必要がある．たとえば，$\{A, B, C, D\}$ と $\{C, D, E, F\}$ は，その内部のコード片のすべての組がクローンペアであることから，一定以上の類似度があることが保証されている．一方，$\{A, B, C, D, E, F\}$ では，コード片 A とコード片 F はクローンペアを成していないため，それらの間には十分な類似度が存在しておらず，それらを単一のコード片としてまとめるのは難しいことを示唆している．よって，$\{A, B, C, D\}$ や $\{C, D, E, F\}$ の方が，まとめる対象となるコード片の集合としては適切といえる．この場合，$\{A, B, C, D\}$ を単一のコード片としてまとめると，$\{C, D, E, F\}$ を単一のコード片としてまとめることはできなくなってしまう．このような場合，どちらのクローンセットを優先してまとめるのか，もしくはどちらもまとめないのかは開発者が判断する必要がある．

コードの一貫性チェック [11] を行う場合にはどちらのクローンセットも有益であると著者らは考える．たとえば，$\{A, B, C, D\}$ に含まれる各コード片は，ハッシュ値を得るための変数（`result`）の初期値が 0 であり，`result` へと足し込まれる回数が 2 回であるという特徴を持つ．これらの共通の処理において，用いられている変数名の整合性やフォーマットやコードコメントの統一性チェックを行う場合には提案手法を利用する方が望ましい．NiCAD 方式では必ずしもコード片間に十分な類似度がないために，一貫性をチェックする必要がないコード片が含まれてしまう可能性が，提案手法を利用した場合に比べて高くなってしまう．一方，NiCAD 方式が役に立つ場合もある．たとえば，$\{A, B, C, D, E, F\}$ のクローンセットを開発者が見た

際に，コード片 A とコード片 B の `int result = 1;` は `int result = super.hashCode();` に変更した方がよい，と思うかもしれない．多くのコード片を提示し，そのなかの多数派と少数派の記述方法が明らかになることで，潜在的な不具合を発見するきっかけとなる場合がある．

ソースコードの重複度の算出や，ある箇所からバグが見つかりその類似箇所の確認 [10] を行う場合はクローンペアがあれば十分なため，提案手法や NiCAD によるクローンセットの生成は必要ない．

3.2 実装

提案手法ではクローンペアの情報を入力として受け取るために，クローンの検出器が出力したクローンペアの検出結果ファイルを読み込む必要がある．本研究では NiCAD [4] のクローンペアの出力結果を読み込めるように実装した．NiCAD は検出したクローンペアとクローンセットをそれぞれ別のファイルに出力する．本研究ではこのうち，クローンペアのファイルを利用した．NiCAD 以外の検出器に対しても検出結果を読み込む処理を実装することで提案手法を適用できる．

極大クリーク列挙を行うためにはコード片と頂点番号との紐づけが必要である．そのため検出結果に含まれるすべてのコード片に ID をつける．NiCAD の検出結果に含まれるすべてのコード片に pcid と呼ばれる ID がすでに割り振られているため，本研究ではこの pcid を頂点番号とした．

また，本研究では Makino らが提案した極大クリーク列挙のアルゴリズム [29] を実装しているツールの MACE [22] を用いた．MACE はグラフの隣接リストを入力に受け取り，そのグラフに存在する極大クリークを出力する．

4. 実験

本章では，提案手法を評価するために行った実験について述べる．

4.1 調査項目

この実験では下記の項目で提案手法を評価した．

クローンセット数 提案手法を利用することにより，既存のクローン検出器で検出されたクローンペアから，ギャップを含むクローンセットがどの程度生成されるのかを調査した．

実行時間とメモリ使用量 提案手法を利用してクローンペアからクローンセットを得るのに要した時間およびメモリ使用量を調査した．

4.2 対象プロジェクト

実験には 5 つのプロジェクトを用いた．**表 1** に対象のプロジェクトを示す．なお，クローン検出を行う前にテストコードは削除した．

表 1 対象のプロジェクト
Table 1 Target projects.

プロジェクト	コミット ID	ファイル数	行数
Eclipse.jdt.core	6958b52...	3,232	734,119
Guava	c15cd80...	1,818	407,870
JFreeChart	d03e68a...	637	219,187
RxJava	166c529...	850	174,085
Tomcat	cc48d0d...	1,855	465,227

4.3 検出器

提案手法の入力であるクローンペアの検出には NiCAD [4] を用いた．検出単位はメソッド単位とし，類似度の閾値はデフォルトの 0.30 とした．検出するクローンの最小の大きさはデフォルトの 10 行を用いた．また，この実験では比較のために iClones も用いた．iClones は検出するクローンの最小の大きさを字句数で指定するため，NiCAD と完全に同じ値を指定することはできないが，今回は Java のプロジェクトが対象ということもあり，1 行あたり平均で 5 つの字句が含まれると想定し，50 字句を検出する最小のクローンの大きさとした．NiCAD も iClones もその他の設定はすべてデフォルトの値を用いた．

4.4 クローンセット数の調査

提案手法が検出したクローンセット数を NiCAD が検出したクローンセット数と比較した．その結果を**表 2** に示す．表 2 (a) より，どのプロジェクトにおいても，NiCAD の検出したクローンセットが分解されて，極大クリークからなるクローンセットが生成されていることが分かる．しかし，それらの数の比はプロジェクトによって大きく異なる．Guava では，NiCAD は 1,206 のクローンセットを検出したのに対して，提案手法では 1,306 のクローンセットを検出した．つまり，平均では NiCAD が検出した単一のクローンセットから 1.08 の極大クリークをなすクローンセットが生成されたことが分かる．一方，JFreeChart では，平均で NiCAD が検出した単一のクローンセットから 11.08 の極大クリークをなすクローンセットが生成された．

提案手法を利用した場合に，JFreeChart から最も多くの Type-3 クローンセットが検出された要因は JFreeChart から検出されたクローンペアに `equals()` メソッドが多く含まれていたためである．`equals()` メソッドは引数のオブジェクトが自身のクラスと等しいかの比較を行うメソッドである．そのためクラスによって処理が異なり，`equals()` メソッドのクローンペアはギャップが含まれている場合が多い．一方で Guava から最も多くの Type-1 および Type-2 クローンセットが検出された要因は，共通処理がパッケージ間に多く存在するためである．Guava は Java のプログラミングで共通して使われるユーティリティライブラリである．android パッケージと android パッケージ以外の

表 2 検出されたクローンセット数
Table 2 The number of detected clone sets.

(a) 全クローンセット

プロジェクト名	NiCAD	提案手法 (%)		
		合計	Type-3	Type-1 および Type-2
Eclipse.jdt.core	1,036	1,510	1,283 (84.97%)	227 (15.03%)
Guava	1,206	1,306	402 (30.78%)	904 (69.22%)
JFreeChart	244	2,685	2,625 (97.77%)	60 (2.23%)
RxJava	314	478	382 (79.92%)	96 (20.08%)
Tomcat	399	477	396 (83.02%)	81 (16.98%)

(b) 要素数が 3 以上のクローンセット

プロジェクト名	NiCAD	検出したクローンセット数 (%)		
		合計	Type-3	Type-1 および Type-2
Eclipse.jdt.core	300	529	497 (93.95%)	32 (7.05%)
Guava	207	328	289 (88.11%)	39 (11.89%)
JFreeChart	90	2,502	2,487 (99.40%)	15 (0.60%)
RxJava	118	234	223 (95.30%)	11 (4.70%)
Tomcat	99	131	121 (92.24%)	10 (7.76%)

パッケージとで共通する処理が多いため，ギャップを含まないクローンペアが多数検出された．

また，提案手法については，検出されたクローンセットが Type-3 とそうでないもので分類をした．表 2 (a) より，全クローンセットを対象とした場合，Guava を除く 4 つのプロジェクトにおいて Type-3 クローンの数が過半数であることが分かる．また，表 2 (b) より，要素数が 3 以上のクローンセットの場合だと，すべてのプロジェクトにおいて Type-3 クローンセットの割合が非常に高くなっており，特に JFreeChart ではその割合が 99.40%にもなっていることが分かる．この結果より，Type-1 および Type-2 のクローンセットに比べて Type-3 のクローンセットが多く存在していることが分かる．

4.5 実行時間とメモリ使用量の調査

NiCAD と提案手法を用いたクローンセット検出および iClones を用いたクローンセット検出の実行時間とメモリ使用量を調査した．調査結果を**表 3** に示す．表 3 (a) より，NiCAD によるクローンの検出は iClones のクローンの検出に比べて長い時間を必要としていることが分かる．これは，NiCAD が少ないメモリ使用量でクローン検出を行う実装の工夫をしているためである．提案手法の実行時間は対象が大きい場合であっても数秒である．提案手法の実行時間のうち，大半の時間はテキストファイルからクローンペア情報を読み込みメモリ内にオブジェクトとして保存する処理であり，極大クリーク列挙に必要であった時間はたかだか数百ミリ秒であった．

表 3 (b) から，NiCAD のメモリ使用量は最も大きい対象の場合（All projects）であっても約 76 MB であるのに対して，iClones ではその 100 倍以上の約 9.7 GB のメモリを使用している．iClones ほどではないが提案手法もメモリ使用量が大きく，より大きなプロジェクトからクローンセットを検出する場合は大容量のメモリが必要である．このことから，ギャップを含むクローンセットの検出にはいまだメモリ使用量についての課題があるといえる．

対象プロジェクトから構築されたグラフの規模を**表 4** に示す．この表より，JFreeChart のグラフの平均次数が 25 を超えており，他のすべてのプロジェクトの平均次数が 4 以下であることを考慮すると，JFreeChart のグラフ

表 3 提案手法と iClone の実行時間とメモリ使用量
Table 3 Execution time and memory usage of the proposed technqiue and iClones.

(a) 実行時間（提案手法の括弧内の数値は極大クリーク検出時間）

プロジェクト	NiCAD + 提案手法		iClones
	NiCAD	提案手法	
Eclipse.jdt.core	105.6 s	4.9 s (98 ms)	17.2 s
Guava	48.9 s	3.0 s (61 ms)	7.1 s
JFreeChart	16.8 s	3.3 s (130 ms)	2.2 s
RxJava	22.4 s	1.8 s (48 ms)	2.7 s
Tomcat	48.1 s	1.8 s (44 ms)	4.0 s
All projects	256.8 s	7.7 s (215 ms)	34.1 s

(b) メモリ使用量

プロジェクト	NiCAD + 提案手法		iClones
	NiCAD	提案手法	
Eclipse.jdt.core	74.1 MB	3,801.0 MB	4,155.3 MB
Guava	73.8 MB	938.7 MB	2,079.1 MB
JFreeChart	72.8 MB	1,292.6 MB	1,020.1 MB
RxJava	72.5 MB	649.1 MB	1,252.1 MB
Tomcat	72.3 MB	564.1 MB	1,891.9 MB
All projects	76.2 MB	6,918.1 MB	9,719.7 MB

表 4 対象プロジェクトのグラフ規模
Table 4 Graph size of the target projects.

プロジェクト	頂点数	辺数	最大次数	平均次数
Eclipse.jdt.core	3,738	19,060	90	3.71
Guava	3,265	5,991	26	2.90
JFreeChart	1,004	9,949	59	25.24
RxJava	1,159	3,004	24	3.70
Tomcat	1,169	1,810	15	2.77

がきわめて密であることが分かる．極大クリーク列挙には Makino-Uno アルゴリズム [29] と Tomita アルゴリズム [24] の 2 つが存在する．検出対象が疎グラフである場合は前者の方が高速であり，検出対象が密グラフである場合は後者の方が高速であるという特徴を持つ [22]．本研究では Makino-Uno アルゴリズムを実装した MACE [22] というツールを利用したため，JFreeChart からの極大クリーク検出時間が他のどのプロジェクトよりも長くなったと考えられる．また，Makino-Uno アルゴリズムは密なグラフからの極大クリーク列挙が得意でないことから，JFreeChart に対する提案手法のメモリ使用量が iClones よりも大きくなったと考えられる．

上述のとおり，提案手法は対象プロジェクトの規模ではなく，対象プロジェクトから検出されたクローンペアの数や，クローンペアから作成されたグラフの密度に依存する．そのため，対象プロジェクトが非常に大きい場合であっても検出されたクローンペアが少ない場合には提案手法は問題なく動作するが，対象プロジェクトが小さい場合でも非常に多くのクローンペアが検出された場合には，提案手法は実行に長い時間を必要とする，およびメモリが大量に必要である，という課題が存在する．クローンペアから構築したグラフが密である場合には，本研究で利用した Makino-Uno アルゴリズムではなく，密グラフからの極大クローン列挙が得意な Tomita アルゴリズムを利用することで実行時間の短縮やメモリ使用量の改善ができる可能性がある．

5. 妥当性への脅威

実験対象：外的妥当性

実験の対象には 5 つのプロジェクトに対して実験を行った．この 5 つのプロジェクトは検出されるクローンの評価によく用いられるプロジェクトであるが，他のオープンソースソフトウェアや企業のプロジェクトに対して同様の結果になるとは限らない．

検出器：内的妥当性

実験では NiCAD が検出したクローンペアに対して提案手法を適用した．しかし NiCAD 以外の検出器で得られたクローンペアに対しても同様のクローンセットが得られるとは限らない．

6. あとがき

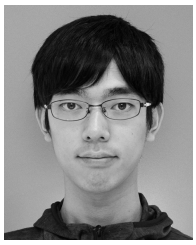
本研究では，極大クリーク列挙アルゴリズムを用いることでクローンペアからクローンセットを検出する手法を提案した．提案手法は，クローンペアを検出する任意の既存検出器のポストプロセスとして利用できる．実験では複数のオープンソースソフトウェアに適用し，すべての対象ソフトウェアから多数のギャップを含むクローンセットが検出された．また，提案手法によるクローンペアからクローンセットの検出時間は最も大きな対象ソフトウェアに対しても数秒で完了することも確認できた．今後の課題としては，提案手法が検出したクローンセットと NiCAD が検出するクローンセットを有用性の点から比較することがあげられる．たとえば，著者らが提案手法や NiCAD を用いてオープンソースソフトウェアからクローンセットを検出し，そのクローンセットを目視確認することにより，修正が必要と思われるコード片を見つけ，実際に修正を行いそれをプルリクエストにより開発者にフィードバックすることによる評価を検討している．

謝辞 本研究は，科学研究費補助金基盤研究（B）（課題番号：20H04166）の助成を得て行われた．

参考文献

[1] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I., et al.: Fast discovery of association rules, *Advances in Knowledge Discovery and Data Mining*, Vol.12, No.1, pp.307–328 (1996).
[2] Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules, *Proc. 20th Int. Conf. Very Large Data Bases VLDB*, Vol.1215, pp.487–499 (1994).
[3] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Trans. Software Engineering*, Vol.33, No.9, pp.577–591 (2007).
[4] Cordy, J.R. and Roy, C.K.: The NiCad clone detector, *2011 IEEE 19th International Conference on Program Comprehension*, pp.219–220, IEEE (2011).
[5] Göde, N. and Koschke, R.: Incremental clone detection, *2009 13th European Conference on Software Maintenance and Reengineering*, pp.219–228, IEEE (2009).
[6] Gusfield, D.: Algorithms on stings, trees, and sequences: Computer science and computational biology, *Acm Sigact News*, Vol.28, No.4, pp.41–60 (1997).
[7] Jiang, L., Su, Z. and Chiu, E.: Context-based detection of clone-related bugs, *Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp.55–64, ACM (2007).
[8] Juergens, E., Deissenboeck, F., Hummel, B. and Wagner, S.: Do code clones matter?, *2009 IEEE 31st International Conference on Software Engineering*, pp.485–495, IEEE (2009).
[9] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Software Engineering*, Vol.28, No.7, pp.654–670 (2002).

[10] Kim, H., Jung, Y., Kim, S. and Yi, K.: MeCC: Memory comparison-based clone detector, *2011 33rd International Conference on Software Engineering*, pp.301–310, IEEE (2011).
[11] Krinke, J.: A study of consistent and inconsistent changes to code clones, *Proc. 14th Working Conference on Reverse Engineering*, pp.170–178, IEEE (2007).
[12] Kumar, R., Raghavan, P., Rajagopalan, S. and Tomkins, A.: Trawling the web for emerging cyber-communities, *Computer networks*, Vol.31, No.11-16, pp.1481–1493 (1999).
[13] Lozano, A. and Wermelinger, M.: Assessing the effect of clones on changeability, *2008 IEEE International Conference on Software Maintenance*, pp.227–236, IEEE (2008).
[14] McIntosh, S., Poehlmann, M., Juergens, E., Mockus, A., Adams, B., Hassan, A.E., Haupt, B. and Wagner, C.: Collecting and leveraging a benchmark of build system clones to aid in quality assessments, *Companion Proc. 36th International Conference on Software Engineering*, pp.145–154, ACM (2014).
[15] Mondal, M., Rahman, M.S., Saha, R.K., Roy, C.K., Krinke, J. and Schneider, K.A.: An empirical study of the impacts of clones in software maintenance, *2011 IEEE 19th International Conference on Program Comprehension*, pp.242–245, IEEE (2011).
[16] Rahman, F., Bird, C. and Devanbu, P.: Clones: What is that smell?, *Empirical Software Engineering*, Vol.17, No.4-5, pp.503–530 (2012).
[17] Rattan, D., Bhatia, R. and Singh, M.: Software clone detection: A systematic review, *Information and Software Technology*, Vol.55, No.7, pp.1165–1199 (2013).
[18] Roy, C.K. and Cordy, J.R.: Near-miss function clones in open source software: An empirical study, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol.22, No.3, pp.165–189 (2010).
[19] Roy, C.K., Cordy, J.R. and Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, Vol.74, No.7, pp.470–495 (2009).
[20] Sajnani, H., Saini, V., Svajlenko, J., Roy, C.K. and Lopes, C.V.: SourcererCC: Scaling code clone detection to big-code, *Proc. 38th International Conference on Software Engineering*, pp.1157–1168 (2016).
[21] Svajlenko, J., Islam, J.F., Keivanloo, I., Roy, C.K. and Mia, M.M.: Towards a big data curated benchmark of inter-project code clones, *2014 IEEE International Conference on Software Maintenance and Evolution*, pp.476–480, IEEE (2014).
[22] Takeaki, U.: Implementation issues of clique enumeration algorithm, *Special Issue: Theoretical Computer Science and Discrete Mathematics, Progress in Informatics*, Vol.9, pp.25–30 (2012).
[23] Thummalapenta, S., Cerulo, L., Aversano, L. and Di Penta, M.: An empirical study on the maintenance of source code clones, *Empirical Software Engineering*, Vol.15, No.1, pp.1–34 (2010).
[24] Tomita, E., Tanaka, A. and Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments, *Theoretical Computer Science*, Vol.363, No.1, pp.28–42 (2006).
[25] Tsantalis, N., Mazinanian, D. and Rostami, S.: Clone refactoring with lambda expressions, *Proc. 39th International Conference on Software Engineering*, pp.60–70, IEEE (2017).
[26] Ueda, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: On detection of gapped code clones using gap locations, *9th Asia-Pacific Software Engineering Conference*, pp.327–336, IEEE (2002).
[27] Yoshida, N., Ishizu, T., Edwards III, B. and Inoue, K.: How slim will my system be? estimating refactored code size by merging clones, *Proc. 26th Conference on Program Comprehension*, pp.352–360 (2018).
[28] Zhang, T. and Kim, M.: Automated transplantation and differential testing for clones, *Proc. 39th International Conference on Software Engineering*, pp.665–676, IEEE (2017).
[29] 宇野毅明：大規模グラフに対する高速クリーク列挙アルゴリズム，一般社団法人電子情報通信学会，Vol.103, No.31, pp.55–62 (2003).



土居 真之

平成 30 年大阪大学基礎工学部情報科学科卒業．令和 2 年同大学大学院情報科学研究科卒業．在学時，コードクローン検出の研究に従事．

肥後 芳樹 （正会員）

平成 14 年大阪大学基礎工学部情報科学科中退．平成 18 年同大学大学院博士後期課程修了．平成 19 年同大学院情報科学研究科コンピュータサイエンス専攻助教．平成 27 年同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析やリファクタリング支援に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．

楠本 真二 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部助手．平成 8 年同講師．平成 11 年同助教授．平成 14 年同大学大学院情報科学研究科助教授．平成 17 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．IPSJ，IEICE，JSSST，IEEE，JFPUG，PM 学会，SEA 各会員．