社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2008-18 (2008-07)

アイテムセットマイニングを利用したコードクローン分析作業の効率向上

宮崎　宏海†　　肥後　芳樹†　　井上　克郎†

† 大阪大学 大学院情報科学研究科 〒 560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{h-miyazk,higo,inoue}@ist.osaka-u.ac.jp

あらまし　ソフトウェア保守を困難にしている要因の 1 つにコードクローンがある．コードクローンとはソースコード中に存在するコード片で，他のコード片と同一または類似しているものを指す．コードクローン分析支援の手法としてコードクローンの可視化がある．散布図は可視化手法の 1 つで，コードクローンの分布状況を瞬時に把握できるが，個々のコードクローンを分析する場合はどの部分から分析すべきか判断しづらい．これは，散布図上からはコードクローンの特徴を取得しづらいためである．本研究では，アイテムセットマイニング手法を用いてコードクローンの出現パターンを求め，その情報を散布図上に付加する手法を提案する．個々のコードクローンではなくその出現パターンに着目することで，より巨視的な視点で類似部分を知ることができ，効率的な分析が可能になると考えられる．そして，ツールを実装して適用実験を行い，手法の有用性を確認する．

キーワード　コードクローン，可視化，散布図，アイテムセットマイニング

Increasing Code Clone Analysis Efficiency Using Itemset Mining

Hiromi MIYAZAKI†, Yoshiki HIGO†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University, 1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan
E-mail: †{h-miyazk,higo,inoue}@ist.osaka-u.ac.jp

Abstract　One of the factors that makes software maintenance more difficult is code clone. A code clone is a code fragment that is similar or identical to other code fragments. Code clone visualization is one of the code clone analysis methods. Scatter plot is a widely used visualization, and we can figure out how code clones are distributed in the software system at once. However, it is hard to understand where to start analysis if we analyze individual code clone. This is because it is hard to get code clone features. Hence we propose a method using itemset mining to get code clone appearance patterns and adding the patterns into a scatter plot. We can identify larger or structurally similar sections of the software system, so that we can efficiently analyze code clones. We implement a tool and check usability of our method.

Key words　Code Clone, Visualization, Scatter Plot, Itemset Mining

1. はじめに

ソフトウェアの保守作業を困難にしている要因の 1 つにコードクローンがある．コードクローンとは，ソースコード中に存在するコード片で互いに同一もしくは類似したコード片を持つものを指し，主にコピーアンドペーストによって生成される．コードクローンであるコード片に修正が生じた場合，コードクローン全てに同一の修正を検討する必要があるが，大規模なソフトウェアに対してこのような修正を行うことは極めて困難である．

これまでに，コードクローンに関して様々な研究が行われてきた [2] [3] [4]．著者らの研究グループでもコードクローン検出ツール CCFinder [6] を開発している．CCFinder はコードクローンを高速で検出できるツールであるが，その出力結果であるコードクローン情報はテキスト形式であるため，そのままの形式でコードクローン分析作業を行うのは効率的ではない．そこで，分析作業を支援するために散布図 [10] を実装した．散布図を用いることで，コードクローンの分布状況を把握できる．

散布図はコードクローンの分布状況を瞬時に知り得る点で有用であるが，個々のコードクローンに対して分析を行う際にど

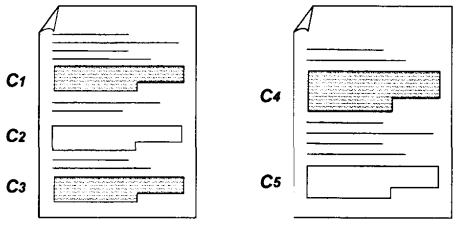


図 1　クローンペアとクローンセット

こから分析を行えばよいか判断しづらい．

本研究では，アイテムセットマイニングを用いて，コードクローンの出現パターンを取得し，その情報を散布図上に付加する手法を提案する．コード片はファイルが持つ機能の一部を表しているため，複数のコードクローンを共有するファイルは，そうでないファイルに比べて，より類似度が高いと考えられる．そこで，コードクローンのセットを散布図上に表示することで，より効率的なコードクローン分析を行うことができると考えられる．

そして，手法を実装して，あるソフトウェアに対して適用し，取得した出現パターンの内容について調査を行った．

2. コードクローンの可視化

本節では，コードクローンとその可視化について述べる．

2.1 コードクローン

コードクローンとは，ソースコード中に存在するコード片で，他のコード片と同一もしくは類似しているものを指す．ここで，コードクローンをより良く表現するために，クローンペアとクローンセットの 2 つの言葉を説明する [6]．**クローンペア**とは，互いに同一もしくは類似しているコード片の対を指す．**クローンセット**とは互いに類似したコード片の集合を表しており，同一のクローンセットに属する任意の 2 つのコード片はクローンペアとなる．

図 1 はクローンペアとクローンセットの例である．この図には 5 つのコードクローンがあり，C_1,C_3,C_4 は互いに類似しており，C_2 と C_5 も互いに類似している．従って，図 1 には（C_1,C_3）（C_1,C_4）（C_3,C_4）（C_2,C_5）という 4 つのクローンペアと，{C_1,C_3,C_4} {C_2,C_5} という 2 つのクローンセットが存在することになる．

2.2 散布図

CCFinder ではコードクローンの情報をテキスト形式で扱っているため，そのままの形式で人間が分析を行うのは困難である．そこで，コードクローンを可視化する手法として散布図 [10] がある．

図 2 は散布図をモデル化したものである．散布図では左上を起点として，右方向および下方向にファイルの絶対パス順にソースファイルが並べられている．この時各ソースファイルはトークン単位に分割され，縦方向の要素と横方向の要素が一定数以上連続して一致している場合，そのセルに点をプロットする．つまり，散布図上では，コードクローンはある一定以上の長さを持った線分として出現する．例えば，図 2 で F3 の 1 番目〜3 番目のトークンと F4 の 3 番目〜5 番目のトークンが「d,e,f」であるため，この部分をトークン数が 3 のコードクローンとして抽出できる．このように，散布図を見ることでコードクローンの分布状況を瞬時に把握することが可能である．

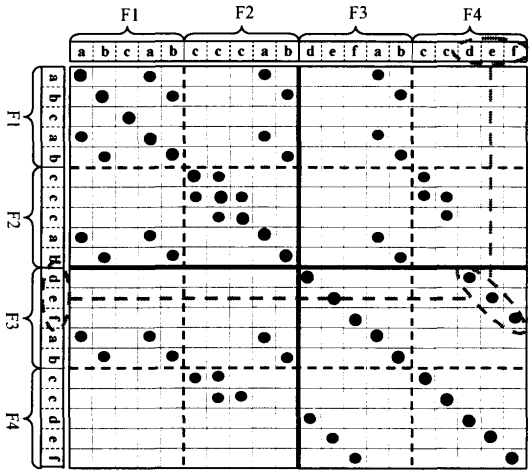


図 2　コードクローンの散布図のモデル

しかし，散布図にはコードクローンの分布状況を瞬時に把握できるという利点があるが，個々のコードクローンを分析したい場合にどこから調査を行うべきか判断が難しいという問題点がある．図 2 の散布図を見た場合に，コードクローンが多い箇所を特定できても，そこに分析を行うべきコードクローンがあるとは限らない．これは，散布図上からはコードクローンの特徴やコードクローン間の関係，コードクローンを含むファイルの特徴など，分布状況以外の情報を取得出来ないからである．そのため，コードクローン分析の効率化を図るには散布図上で特徴的なコードクローンを表示する必要がある．

2.3 関連研究

コードクローンの可視化を行った研究としては，Kapser らが理解支援を目的としたツールとして CLICS を開発している [7]．CLICS はファイル間の依存関係を表すグラフにコードクローン情報を付加して表示している．また，クエリによるコードクローンの検索機能も実装しており，ユーザーはコードクローンのサイズなどをクエリとして与えることで，その条件を満たすコードクローンの情報を取得することができる．

Adar らは，複数バージョン間のクラスやメソッド間の依存関係とコードクローン関係を表示するツール SoftGUESS を開発しており [1]，バージョン毎のコードクローンの比較を行っている．このツールを用いることで，バージョンを通して出現し続けているコードクローンや，バージョン間で修正が加わったことで削除されたコードクローン，新たに生成されたコードクローンなど，コードクローンの遷移を把握することができる．

Rieger らは Polymetric View [8] を用いてコードクローンの可視化を行っている [9]．Polymetric View では要素の位置，高さ，幅，色，要素間の辺などがそれぞれコードクローンの特徴を表しているため，特徴的な箇所を瞬時に判断できる．Rieger

らの手法では，ディレクトリツリーと各ファイルに含まれるコードクローンの量や同型のコードクローンの量を同時に表示しており，ファイル内で閉じたコードクローンなのか，多くのファイルで共有されているコードクローンなのかを容易に把握できるようになっている．

散布図による可視化を行った研究もこれまでにいくつか行われている．Ducasse ら [4] は，バージョン毎の散布図から，コードの追加や削除が行われた箇所を把握出来るかを調査した．そして，縦軸を古いバージョン，横軸を新しいバージョンとする散布図を調べ，修正を加えた箇所はコードクローンとならないことから，バージョン間でコードの追加や削除が行われた箇所を把握できるという結果を得ている．

3. FP-Growth

本手法では，対象とするソースコード中に頻出するコードクローンの出現パターン (以後，頻出パターン) を効率的に見つけるために，アイテムセットマイニング手法である FP-growth [5] を用いている．FP-growth では FP-tree という木構造のグラフを利用しており，各ノードはアイテムとカウントを保持している．アイテムとは出現パターンを構成する要素であり，トランザクションには複数のアイテムが含まれ，データベースには複数のトランザクションが含まれている (表 1 参照)．また，カウントはそのアイテムを含むパターンが出現するトランザクションの数を表す．

FP-tree には以下のような性質がある．これらの性質を利用して頻出パターンを取得する．なお，頻出パターンかどうかは，データベース上でのそのパターンの出現回数が閾値以上かどうかで判断するが，この閾値をサポート値と呼ぶ．

性質 1: FP-tree 上に存在する全てのアイテムは，データベース中にサポート値以上の回数出現するアイテム (以後，頻出アイテム) である

性質 2: ノードが保持するカウントは，親ノードが保持するカウント以下かつ子ノードが保持するカウント以上である (ただし root は除く)

性質 3: 各ノードから root までの経路に存在するノードが保持するアイテムは全て同一トランザクションに含まれる

性質 4: 性質 3 より，ノード N から root までの経路をたどることで，N が保持するアイテムを含む出現パターンを検出できる

性質 5: 性質 4 より，検出された出現パターンを持つトランザクションは，ノード N のカウント数だけ存在する

表 1　データベースの例

トランザクション ID	アイテム ID	頻出アイテム ID
1	a,c,d,f,i,m,p	f,c,a,m,p
2	a,b,c,f,l,m,o	f,c,a,b,m
3	b,f,h,j,o	f,b
4	b,c,k,p,s	c,b,p
5	a,c,e,f,l,m,n,p	f,c,a,m,p

```
1. データベース中の総出現回数がサポート値以上である頻出
アイテムの集合CFを抽出
2. 各トランザクションでアイテムを総出現回数の降順でソート
3. FP-treeの根の部分となるrootを作成
4. for each t ∈ T(全てのトランザクション){
5.    t が含む頻出アイテムの集合Ctを抽出(Ct ⊆CF)
6.    insert_item (Ct, root){
7. }

8. insert_item (アイテムの集合C, ノードN){
9.    Cで先頭のアイテムxを取得
10.   if(Nの子ノードに x を持つノードNxが存在){
11.       Nxのカウントを1増やす
12.       N = Nx
13.   }
14.   else{
15.       アイテム x とカウント1を持つNewNodeを作成
16.       Nの子ノードにNewNodeを追加
17.       N = NewNode
18.   }
19.   Cからxを除去
20.   if(Cに属するアイテムがある){
21.       insert_item(C, N)
22.   }
23. }
```

図 3　FP-tree 作成のアルゴリズム

上記の性質を持つ FP-tree は，図 3 のアルゴリズムで構築される．なお，入力はデータベースとし，出力は root を根とする木である．

表 1 のデータベースに図 3 のアルゴリズムを適用した場合，図 4 のような FP-tree が構築される．図 4 では FP-tree の他に，各頻出アイテムを保持するノードへのリンクを保持したテーブルを作成している．破線の矢印をたどれば，その頻出アイテムを保持するノードに辿りつくようになっている．

この FP-tree からアイテム p を含む頻出パターンを抽出する場合について考える．アイテム p とカウント 2([p:2] と表記) を持つノードから root までの経路には [f:4],[c:3],[a:3],[m:2],[p:2] を持つノードがある．また，[p:1] を持つノードから root までの経路には [c:1],[b:1],[p:1] を持つノードがある．前者からはアイテムの集合 {f,c,a,m,p} が 2 つのトランザクションに表れていることが，後者からは {c,b,p} が 1 つのトランザクションに表れていることがそれぞれ分かる．これらの情報から，p は 3 つのトランザクションで c と，2 つのトランザクションで a,f,m と，1 つのトランザクションで b と同時に出現していることが求められる．そのため，3 つ以上のトランザクションに含まれる出現パターンを頻出パターンとする場合，このデータベース中でアイテム p を含む頻出パターンは (c,p) と (p) の 2 つであることが分かる．

4. 提案手法

本節では，コードクローンの頻出パターンを取得する手法について説明する．

4.1 概　　要

本手法は以下の手順で，コードクローンの頻出パターンを構築する．

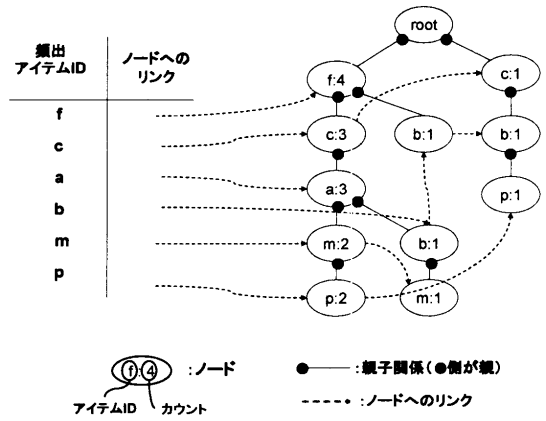
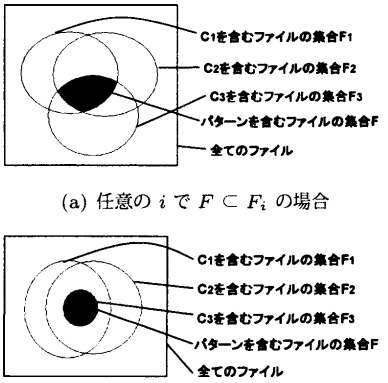


図 4　表 1 から作成した FP-tree(サポート値は 3)



(a) 任意の i で $F \subset F_i$ の場合

(b) ある i で $F = F_i$ の場合

図 5　パターンを含むファイルとパターンを構成するクローンセット ID を含むファイルの包含関係

手順 1：　コードクローンを検出する
手順 2：　コードクローン情報から，クローンセット ID をアイテム，ファイルをトランザクションとする FP-tree を構築する
手順 3：　構築した FP-tree からクローンセット ID の頻出パターンを取得する

以下，各手順について説明していく．

4.2　手順 1:コードクローンの検出

既存のコードクローン検出ツールを用いて，対象ソースファイルからコードクローンを検出する．また，手順 2 でクローンセットの情報を用いるためクローンセットの情報を構築する．

4.3　手順 2：FP-tree の作成

本手法では，クローンセット ID をアイテムとした．

FP-tree の作成の手順としては，まず頻出パターンを構成するコードクローン数の閾値と，含むファイル数の閾値を設定する．次に，各ファイルにどのコードクローンが含まれているか調べ，トランザクションを構築する．この時，閾値以上のファイルに含まれるクローンセットの ID を頻出クローンセット ID とする．なお，1 つのクローンセットに属するコードクローンが，2 つ以上同じファイルに含まれている場合は，それらは異なる ID を持つものとする．そして，頻出クローンセット ID をアイテムとする FP-tree を作成する．FP-tree の各ノードはクローンセット ID とカウントを保持している．

4.4　手順 3：頻出パターンの取得

手順 2 で作成した FP-tree から，頻出パターンを求める．この時，頻出パターンを構成するコードクローンの数とそれを含むファイルの個数は共に閾値以上である．

なお，同じファイルに含まれており，構成するクローンセット ID が包含関係になっている (例：表 2)2 つの頻出パターンがある場合，包含されているものを削除する．これは，包含されている頻出パターンは，包含しているものに比べて新たに得られる情報がないと判断したためである．

5.　実　　験

本節では，提案手法を実装したツールを用いてコードクローンの頻出パターンを取得し，それらがコードクローン分析に有用なものであるかの調査を行う．

5.1　実験内容

本実験では，取得した頻出パターンがコードクローン分析に有用であるかどうかを，パターンの内容とそれを含むファイルの内容から判断する．

まずは，頻出パターンを構成する各コードクローンが，パターンを含むファイルが持つ機能を表すものとして適切であるかどうかを調べる．適切であれば，同じパターンを含む 2 つのファイルは，そうでない 2 つのファイルより類似しているといえる．

次に，パターンを含むファイルと，構成するクローンセット ID を含むファイルの包含関係を調べる．パターンを含んでいるファイルの集合を F，パターンを構成するコードクローンを C_1,C_2,..,C_n，C_i と同型のコードクローンを含むファイルの集合を F_i と表す．図 5 は，F と F_i の包含関係を表しており，F が各 F_i の積集合となっていることが分かる．もし，任意の i で $F \subset F_i$ ならば (図 5(a))，単一のコードクローンからは得られない，特定の機能を持つファイルの集合を得る (以後，ファイルを絞り込む) ことができる．しかし，ある i で $F = F_i$ ならば (図 5(b))，単一のコードクローンを含むファイルの集合と違いがなく，パターンを用いる意義が薄いといえる．

これら 2 つの項目を調べることで類似した機能を持つファイルの集合を取得でき，効率的な分析が可能になると考えている．

この実験では，コードクローン検出ツールは CCFinder を用いた．CCFinder では，コードクローンを構成するトークン数の最小値を定める必要があるが，この実験では 30 とした．実験の対象とするプロジェクトは JHotDraw(バージョン 5.4) とした．JHotDraw には 484 個の Java ファイル (総行数：71736 行) が含まれており，コードクローンは 1604 個，クローンセットは 323 個取得できた．また，頻出パターンを含むファイルの

表 2　クローンセット ID が包含関係になる例

頻出パターン ID	クローンセット ID	ファイル ID
1	a,b,c,d	f1,f2,f3,f4
2	a,b,c	f1,f2,f3,f4

数は最少で 2 個，構成するコードクローンの数は最少で 2 個とした．

5.2 実験結果

JHotDraw からは頻出パターンを 40 個取得することができた．取得したパターンの一部と，パターンを含んでいるファイルについて以下で紹介する．

5.2.1 取得した頻出パターン 1

1 つ目の頻出パターンとして，「PolygonScaleHandle.java」「TriangleRotationHandle.java」「RadiusHandle.java」の 3 つのファイルが含むものがある (図 6)．このパターンは 2 個のコードクローンで構成されている．図 6(a) のコード片は，3 つのファイルでメソッド名が等しい，図形を描画する draw メソッドの一部である．図 6(b) 図 6(c) 図 6(d) は類似したコード片で，メソッド名は異なるが，それぞれ図形を 1 つ前の状態に戻すメソッドの一部である．これらのことから，この頻出パターンには「図形を描画する」「図形を 1 つ前の状態に戻す」という 2 つの情報が含まれており，3 つのファイルが図形を処理するファイルであることが分かる．

5.2.2 取得した頻出パターン 2

その他の頻出パターンとして「ChangeAttributeCommand.java」と「SendToBackCommand.java」の 2 つのファイルが含むものがあった．図 7 は，この 2 つのファイルでパターンを構成するコード片である．メソッド execute() の一部であるコード片では複数の図形を対象としたコマンドの実行を記述しており，メソッド undo() の一部であるコード片ではそのファイルが記述するコマンドの取り消し処理を記述している．これらのことから，この頻出パターンでは「複数の図形を対象にしたコマンドを実行する」「そのコマンドを取り消す」という 2 つの情報が含まれており，2 つのファイルがコマンド処理を行うファイルであることが分かる．

5.2.3 パターンを含むファイル

ファイルの絞り込みが出来ているパターンは，JHotDraw から取得したパターン 40 個のうち 6 個であった．頻出パターン 2 もその 1 つである (表 3)．メソッド execute() の一部であるコードクローンを含むファイルは他に「BringToFrontCommand.java」がある．また，メソッド undo() の一部であるコードクローンを含むファイルは他に「ConnectedTextTool.java」「AlignCommand.java」がある．これらのことから，「ChangeAttributeCommand.java」と「SendToBackCommand.java」は「BringToFrontCommand.java」とは異なり「そのコマンドを取り消す」という機能を持ち，「ConnectedTextTool.java」「AlignCommand.java」とは異なり「複数の図形を対象にしたコマンドを実行する」という機能を持っていることが分かる．これにより，パターンを用いることで特定の機能を持つファイルの集合を求めることができたといえる．

表 3 頻出パターン 2 を構成するコードクローンを含むファイル

execute() の一部である コードクローンを含むファイル	undo() の一部である コードクローンを含むファイル
BringToFrontCommand.java	AlignCommand.java
ChangeAttributeCommand.java	ChangeAttributeCommand.java
SendToBackCommand.java	ConnectedTextTool.java
	SendToBackCommand.java

```
public void draw(Graphics g) {
    Rectangle r = displayBox();

    g.setColor(Color.yellow);
    g.fillOval(r.x, r.y, r.width, r.height);

    g.setColor(Color.black);
    g.drawOval(r.x, r.y, r.width, r.height);
```

(a) 3 つのファイルで定義されている draw メソッド

```
protected boolean resetPolygon() {

    FigureEnumeration fe = getAffectedFigures();

    if (!fe.hasNextFigure()) {
        return false;
    }

    PolygonFigure figure = (PolygonFigure)fe.nextFigure();
    Polygon backupPolygon = figure.getPolygon();
    figure.willChange(
```

(b) PolygonScaleHandle.java が含む類似コード片

```
protected boolean resetRelationAngle() {

    FigureEnumeration fe = getAffectedFigures();

    if (!fe.hasNextFigure()) {
        return false;
    }

    TriangleFigure figure = (TriangleFigure)fe.nextFigure();
    double backupAngle = figure.getRotationAngle();
    figure.willChange(
```

(c) TriangleRotationHandle.java が含む類似コード片

```
protected boolean resetRadius() {

    FigureEnumeration fe = getAffectedFigures();

    if (!fe.hasNextFigure()) {
        return false;
    }

    RoundRectangleFigure currentFigure =
        (RoundRectangleFigure)fe.nextFigure();
    Point FigureRadius = currentFigure.getArc();
    currentFigure.setArc(
```

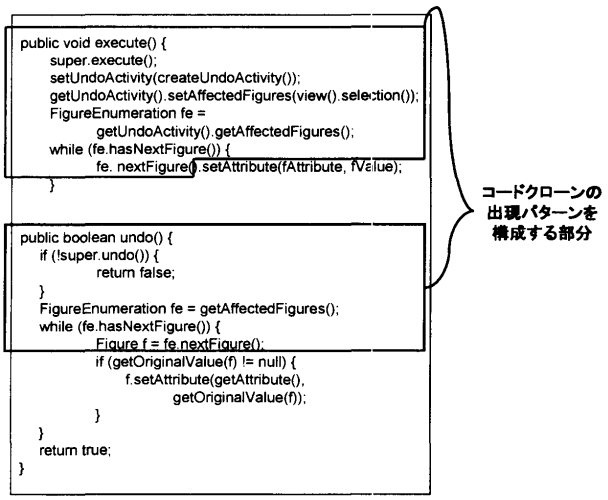
(d) RadiusHandle.java が含む類似コード片

図 6 3 つのファイルから取得した頻出パターンの例

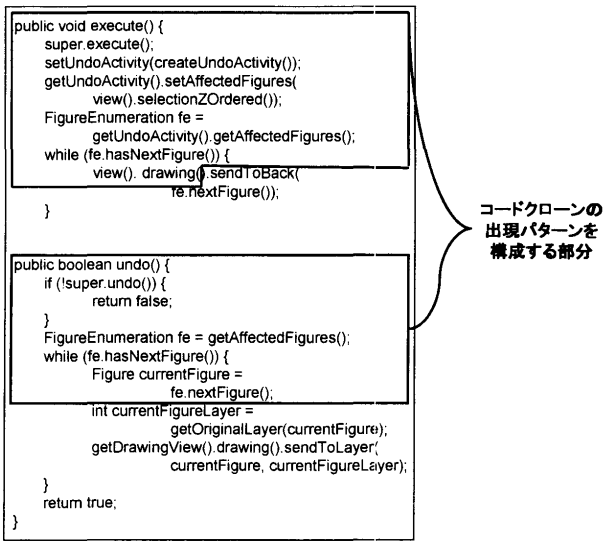
6. 考　　察

JHotDraw に対して実験を行い，取得した頻出パターンを調べたところ，ファイルが持つ機能を表すものとして適切なものが見つかった．また，頻出パターンを含むファイルを調べたところ，パターンを構成するクローンセット ID を含むファイルとの間に包含関係が成り立つものがあり，特定の機能を持つファイルに分けられていた．これらのことから，頻出パターンを用いることで，単一のコードクローンからは得られない情報を得られることが分かった．

しかし，パターンを含むファイルと，パターンを構成するクローンセット ID を含むファイルとの間に包含関係があるパターンは 40 個中 6 個と少なかった．また，頻出パターン 2 で得ら



(a) ChangeAttributeCommand.java のコード片



(b) SendToBackCommand.java のコード片

図 7　2 つのファイルから取得した頻出パターンの例

れたファイルと類似した機能 (メソッド execute() と undo()) を持つファイルとして「CutCommand.java」が存在しており，検出漏れがあったことが分かった．

検出漏れは，より小さいサイズのコードクローンを検出することで改善できると考えている．そのために，コードクローンを構成するトークン数の最小値を変化させる方法がある．

7. まとめと今後の課題

本研究では，コードクローン分析を支援するために，コードクローンの出現パターンを取得する手法を提案した．提案手法では，アイテムセットマイニング手法の 1 つである FP-growth を用いてコードクローンの頻出パターンを取得した．そして，Java ソフトウェアである JHotDraw に対して適用したところ，頻出パターンはファイルの機能を表すものとして適当であることが分かった．次に，パターンを含むファイルと，パターンを構成するクローンセット ID を含むファイルとの間に包含関係が成り立つものがあることから，単一のコードクローンからは得られない，類似した機能を持つファイルの集合を得ることができ，出現パターンをもとにコードクローンを分析することが有用であることが分かった．しかし，包含関係が成り立っていないパターンも多く検出され，複数のコードクローンを用いる意義が薄いという結果も得られた．

今後の課題として，まず出現パターンの情報を散布図に付加することが挙げられる．次に，パターンを構成する要素の変更が挙げられる．具体的には，現在用いているコードクローンの代わりに 1 つの処理単位であるブロック文を使用する方法や，コードクローンを構成するトークン数を変更する方法などが挙げられる．その他に，頻出パターンだけではなく，コードクローンのメトリクスやソフトウェアメトリクスの情報を散布図上に付加することで分析支援を行う手法も考えている．

8. 謝　　辞

本研究は，IJARC(マイクロソフト産学連携機構) 第 3 回 CORE プロジェクト，文部科学省科学研究費補助金基盤研究 (A)(課題番号:17200001)，および若手研究 (スタートアップ)(課題番号:19800022) の助成を得た．

文　　献

[1] E. Adar and M. Kim. Visualization and Exploration of Code Clones in Context. In *29th International Conference on Software Engineering*, pp. 762–766, May 2007.

[2] B. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In *Proc. of the 2nd Working Conference on Reverse Engineering*, pp. 86–95, 1995.

[3] I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. Clone Detection Using Abstract Syntax Trees. In *Proc. International Conference on Software Maintenance 98*, pp. 368–377, 1998.

[4] S. Ducasse, M. Rieger, and S. Demeyer. A Language Independent Approach for Detecting Duplicated Code. In *Proc. International Conference on Software Maintenance 99*, pp. 109–118, 1999.

[5] H. JIAWEI, P. JIAN, Y. YIWEN, and R. M. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. In *Data Mining and Knowledge Discovery*, Vol. 29, pp. 1–12, 2000.

[6] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code. In *IEEE Transactions on Software Engineering*, Vol. 28, pp. 654–670, 2002.

[7] C. Kapser and M. W. Godfrey. Improved Tool Support for the Investication of Duplication in Software. In *21st International Conference on Software Maintenance*, pp. 305–314, 2005.

[8] M. Lanza and S. Ducasse. Polymetric Views: A Lightweight Visual Approach to Reverse Engineering. In *IEEE Transactions on Software Engineering*, Vol. 29, pp. 782–795, September 2003.

[9] M. Rieger, S. Ducasse, and M. Lanza. Insights into System-Wide Code Duplication. In *11th Working Conference on Reverse Engineering*, pp. 100–109, November 2004.

[10] 植田, 神谷, 楠本, 井上. 開発保守支援を目指したコードクローン分析環境. 電子情報通信学会論文誌, 第 86-D-I 巻, pp. 863–871, 2003.