極大頻出系列マイニングを用いたJavaコードクローンの検出

東京工芸大学工学部コンピュータ応用学科

宇田川 佳久

1

目　次

2

1. 研究の背景と概要

- コードクローン（類似したソースコード片）は、開発効率を高めるが、保守では有害とされている
- 例えば、コピー元のソースコードにバグが含まれていた場合は，ソースコードをコピーする度にバグが拡散する
- バグ修正のためには、類似ソースコードを検索する必要がある

3

コードクローンの分類

タイプ1：
- コメント，空白，タブの有無，括弧の位置などを除き，ソースコードとして完全に一致するソースコード

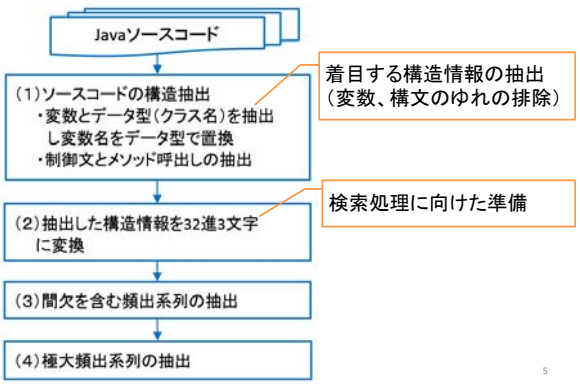
タイプ2：
- タイプ1のソースコードの内，変数名，リテラル，メソッド名などのユーザ定義名，および，変数の型などの予約語だけが異なるソースコード

タイプ3：
- タイプ2のソースコードの内，コピー＆ペースト後に文の変更，追加および削除が行われた結果によって生成されたソースコード

ソースコードの間欠（ギャップ）を考慮した検索が必要

4

コードクローンの検出処理の流れ



5

実験対象としたJavaソースコード

*Java SDK 1.7.0.45*の *java.lang* パッケージ
- ファイル数: 210個
- クラス数: 301個
- メソッド数: 2,527個
- ソース総行数: 67,677行

6

構造情報の抽出例

```
ThreadGroup::remove(ThreadGroup g)
{
    synchronized{
        if{
        return
        }
        for{
            if{
            System.arraycopy()
            break
            }
        }
        if{
        notifyAll()
        }
        if{
        destroy()
        }
    }
}
```

7

類似検索に向けた準備

- 類似度は、文字列の一致／不一致を検出する最長共通部分列（LCS）アルゴリズムを採用した
  - Needleman-Wunschのアルゴリズム ← LCSよりも短い共通部分列を抽出
  - Smith-Watermanのアルゴリズム ← 処理が複雑、パラメータに依存

◆LCSでは類似度が文字列の長さに依存する

➢ 1行の制御文は、文字列の長さに関わらず、同じ重さとして検索したい
  - *if*文 と *synchronized*文 を同等に扱いたい
- 抽出した構文要素を 3桁の32進数に変換する方法を採用した

8

001, {
002, super()
003, }
004, return
005, if{
...
00B, for{
...
00E, System.arraycopy()
...
	01U, break
	...
	058, synchronized{
	...
	0QH, SwitchPoint.mcs.setTarget()
	...
	0VL, Finalizer.runFinalizer()
	...
	16R, notifyAll()
	16S, destroy()

ThreadGroup::remove(ThreadGroup g)
→{→synchronized{→if{→return→}→for{→if{→
System.arraycopy()→break→}→}→if{→notifyAll()→}→
if{→destroy()→}→}→}

ThreadGroup::remove(ThreadGroup g)
→001→058→005→004→003→00B→005→
00E→01U→003→003→005→16R→003→
005→16S→003→003→003

9

頻出系列抽出アルゴリズムの概要

- Apriori アルゴリズム原理に基づくもの
  あるアイテムを含む集合(Y)の発生頻度は，その部分集合(X)の発生頻度より低い．（大きい集合ほど発生しにくい）
- この原理を系列（シーケンス）に適用した。
- 同類のアルゴリズムに CM-SPADE, PrefixSpan, CloSpan, ClaSP, MaxSP などがある
  - 本研究の方法： 直線上に並んだ文字の系列
  - PrefixSpanなど： 文字列の集合の系列（バスケット問題を想定）

10

開発したアルゴリズムの概要

MTHD1→00E→00C
MTHD2→00E→00F→00C→00E→00G
MTHD3→00E→00A→00C→00C
MTHD4→00E→00C→00H→00F→00E→00C

00F→ N=2 (2|4)
00E→ N=6 (1|2+2|3|4+4)
00C→ N=6 (1|2|3+3|4+4)
00E→00C→ N=3 (1|4+4)

図6 間欠を含まない場合の抽出例

minSup 50%（系列２個以上なら抽出する）

11

開発したアルゴリズムの概要

MTHD1→00E→00C
MTHD2→00E→00F→00C→00E→00G
MTHD3→00E→00A→00C→00C
MTHD4→00E→00C→00H→00F→00E→00C

00F→ N=2 (=2|4)
00E→ N=6 (1|2+2|3|4+4)
00C→ N=6 (1|2|3+3|4+4)
00E→00C→ N=5 (1| 2 | 3 |4+4)
00F→00C→ N=2 (2|4)
00F→00E→ N=2 (2|4)

図 7 間欠が1 (Gap=1)の場合の抽出例

00F→ N=2 (2|4)
00E→ N=6 (1|2+2|3|4+4)
00C→ N=6 (1|2|3+3|4+4)
00E→00C→ N=3 (1|4+4)

図6 間欠を含まない場合の抽出例

minSup 50%（系列２個以上なら抽出する）

12

開発したアルゴリズムのイメージ

```
GProve(String[] args){
  k= 1;
  LinkedList<String> Sk に探索の初期値(制御文)をセットする。
  do {
    Retrieve_Cand();  // 今回の検索キーよりも1個長い系列検索する
    k= k+1;
    Sk.clear();  //  ふるいに掛けた後の系列を記憶する
    while( Ck のすべての要素  e に対し){
      if ( Ck[e] の発生個数 >= minSup ){
          探索Key系列と発生個数＋発生箇所のリストを結果Fileに出力する
          while(Gapの範囲で探索Key系列に一致する系列をDBから検索する){
              Sk.add(Gap同義語);
          }
      }
    }
  } while (Sk.size() > 0);
}
```

```
Retrieve_Cand(){
  Ck.clear();   // 探索キー系列の発生個数を記憶する領域
  CkMD.clear();   // 探索キー系列の発生位置を記憶する領域
  CkSyn.clear();   // 探索キー系列にGap数の範囲で一致する系列を記憶する領域
  for (すべての Sk の要素 s に対し) {
      for (データベース内のすべてのデータ t に対し) {
          for ( t で  s の先頭の要素と一致する箇所 p に対し){
              LongestCommonSubsequence(s, t);
              if (一致した要素数 >= k && Gap数  <= maxGap ){
                  Ck.put(探索キー系列, 発生個数);
                  CkMD.put(探索キー系列, 発生した行の番号);
                  t からサブシーケンスを切り出す(= 検索キー同義語系列 )
                  CkSyn.put(検索キー同義語系列, 探索キー系列);
              }
          }
      }
  }
}
```

極大頻出系列の抽出

- Apriori準拠アルゴリズムでは、大量の頻出系列が抽出される(課題の一つ)

➢極大頻出系列(Maximal Frequent Sequence)

定義:

- 極大頻出系列とは,“頻出系列であり,かつ,一つ長い系列が頻出系列でない系列”
- 極大頻出系列とは、系列の長さを一つずつ成長させてゆき,抽出したすべての頻出系列の中で,最も長い頻出系列

14

極大頻出系列の事例

頻出系列

00F→　　N=2 (=2|4)
00E→　　N=6 (1|2+2|3|4+4)
00C→　　N=6 (1|2|3+3|4+4)
00E→00C→　N=5 (1| 2 | 3 |4+4)
00F→00C→　N=2 (2|4)
00F→00E→　N=2 (2|4)

極大頻出系列

00E→00C→　N=5 (1| 2 | 3 |4+4)
00F→00C→　N=2 (2|4)
00F→00E→　N=2 (2|4)

15

実験結果

実験対象としたJavaソースコード

Java SDK 1.7.0.45の java.lang パッケージ

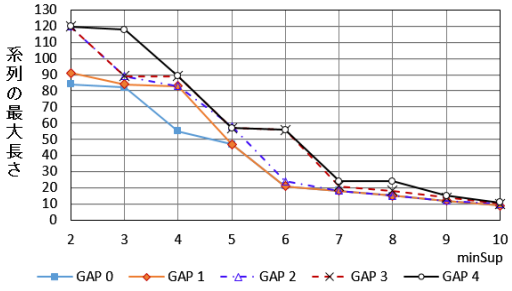
- ファイル数:　210個
- クラス数:　301個
- メソッド数:　2,527個
- ソース総行数:　67,677行

・有意なメソッドは 2,522 個である.
・識別子の種類は 1,286 個である.
・識別子の総数は 18,205 個である.

16

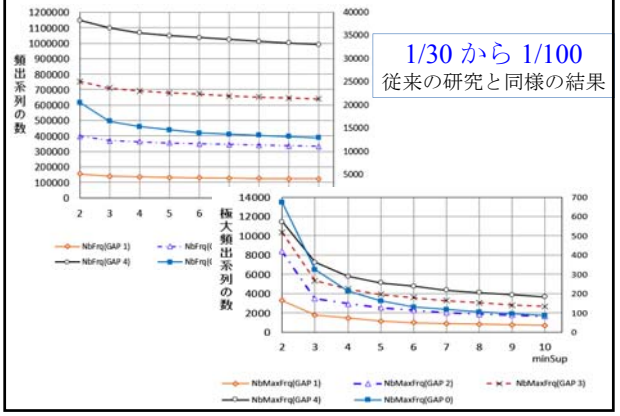
間欠と検出された系列の最大長さ

間欠が大きくなるほど、
検出される系列の最大長さは長くなる.



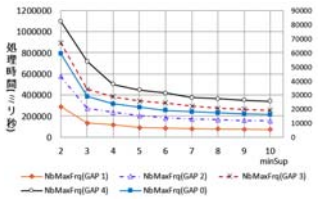
17

頻出系列の個数 V.S. 極大頻出系列の個数

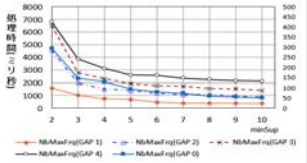


1/30 から 1/100
従来の研究と同様の結果

処理時間



頻出系列を抽出するのに要した時間



極大頻出系列を抽出するのに要した時間

検索されたソースコードの事例

No.	メソッドの構造
1	ThreadGroup::remove(ThreadGroup g)→001→058→005→004→003→00B→005→00E→01U→003→003→005→16R→003→005→16S→003→003→003
2	ThreadGroup::remove(Thread t)→001→058→005→004→003→00B→005→00E→01U→003→003→003→003
3	SwitchPoint::invalidateAll(SwitchPoint[] switchPoints)→001→005→004→003→00B→005→01U→003→0QH→003→0QI→003
4	LazyPattern::run()→001→005→004→003→00B→058→005→01U→003→003→0VL→003→003

図2に示したメソッド

コードクローン

20

構造情報の抽出例

```
ThreadGroup::remove(ThreadGroup g)
{
    synchronized{
        if{
        return
        }
        for{
            if{
            System.arraycopy()
            break
            }
        }
        if{
        notifyAll()
        }
        if{
        destroy()
        }
    }
}
```

```
SwitchPoint::invalidateAll(SwitchPoint[] switchPoints
{
    if{
    return
    }
    for{
        if{
        break
        }
    SwitchPoint.mcs.setTarget()
    }
MutableCallSite.syncAll()
}
```

21

おわりに

間欠を含む極大頻出系列を抽出するアルゴリズムを使ったコードクローン検索の試みについて述べた

今後の研究方針

1. 類似度だけでは、クローンの判断が難しい事例が散見した
2. 抽出された極大頻出系列は，数千件に及ぶことから、更なる絞込みの技法を開発する必要がある

22

ご清聴ありがとうございました

23

24