プログラム依存グラフを用いたTemplate Method パターン適用によるコードクローン集約支援

堀田圭佑[†1]　肥後芳樹[†1]　楠本真二[†1]

近年，ソフトウェアの保守性を低下させる要因の 1 つとしてコードクローンが注目されており，これまでにその集約を支援する手法が多数提案されている．中でも Template Method パターン適用による集約支援手法は，差分を含むコードクローンを集約可能なため高い効果が期待できるが，既存手法にはいくつかの課題点が存在する．本稿ではプログラム依存グラフを用いることで既存研究の課題点を改善した，Template Method パターン適用によるコードクローン集約支援手法を提案する．

A Technique to Support Removing Code Clone by Applying Template Method Pattern with Program Dependence Graph

KEISUKE HOTTA,[†1] YOSHIKI HIGO[†1]
and SHINJI KUSUMOTO[†1]

Recently, code clone has received much attention. And many research efforts have performed on removing code clones. Especially, it is highly expected that the removing technique with Template Method pattern has high applicability because it can remove code clones which has some gaps. However previous research efforts remain some issues. In this paper, we propose a new technique of removing code clones by applying Template Method pattern with program dependence graph to resolve these issues.

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

1. まえがき

近年，ソフトウェア工学における研究対象として，コードクローンが注目されている．コードクローンとは，ソースコード中の同一，あるいは類似するコード片のことをいい，主にコピーアンドペースト等によって発生するといわれている．一般的に，コードクローンの存在はソフトウェアの保守を困難にすると考えられている．これは，あるコード片に修正を加えた際，そのコード片とコードクローン関係にある他のコード片についても同様の修正を検討しなければならない可能性が高いためである．このため，コードクローンを自動的に検出する，あるいはその集約を支援する手法が多数提案されている[1)]．

コードクローンの集約を支援する手法の 1 つとして，Template Method パターンと呼ばれるデザインパターンを適用する手法が提案されている[2),3)]．Template Method パターンとは，共通の親クラスを持つ類似メソッドを対象とし，メソッド間で共通の処理を親クラスに記述し，メソッドごとに異なる処理を子クラスに記述する，というパターンである．このパターンを用いたコードクローン集約手法では，メソッド間でコードクローンとなっている箇所を共通の処理として親クラスに引き上げ，コードクローンとなっていない箇所を子クラスに記述することで，コードクローンの集約を実現している．この手法の最大の特長として，対象となるメソッドがコードクローンとなっていない箇所を含んでいても適用が可能である，という点が挙げられる．しかし，既存手法には以下に挙げる課題点が存在する．

- 変数名などのユーザ定義名のみが異なるコードクローンを集約できない．
- 意味的に同じ処理を行っているコードでも，その表現方法が異なる場合は集約できない．

そこで本稿ではこれらの課題点を改善するため，プログラム依存グラフを用いた Template Method パターンの適用支援手法を提案する．

2. 準　　備

2.1 Template Method パターン

Template Method パターンとは，Gamma らによって提案されているデザインパターンの 1 つである[4)]．このパターンは共通の親クラスを持つメソッドペアに対し，メソッド間で共通の処理を親クラスに，異なる処理をそれぞれの子クラスに記述するというものである．

Template Method パターンの適用例を**図 1** に示す．この例では，*Site* を共通の親クラスとする 2 つのクラスの間に類似メソッド getBillableAmount() が存在している．このメソッドの共通部分を親クラスに引き上げ，異なる部分をそれぞれ新たなメソッドとして抽出している．

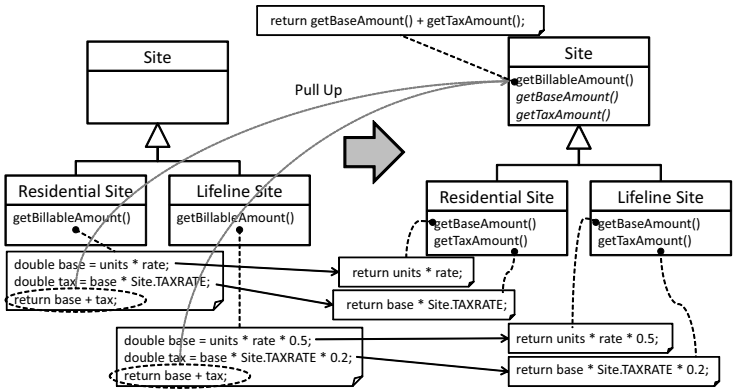

図 1 Template Method パターンの適用例
Fig. 1 An Example of the Application of Template Method Pattern

このように修正することで，多態性によってそれぞれのクラスに応じた getBaseAmount() 及び getTaxAmount() が呼び出され，適用前後で振る舞いが保たれる[5]．

このように，Template Method パターンの適用によってメソッド間に共通していた処理が親クラスにまとめて記述されるため，適用前にコードクローンとなっていたコードが集約され，ソフトウェアの保守性向上が期待できる．また，Template Method パターンはメソッド間に異なる処理が含まれていても適用可能であり，コピーアンドペースト等でコードを生成した後に修正が加えられた場合でも適用が可能であるという特長がある．

2.2 プログラム依存グラフ

プログラム依存グラフ (Program Dependence Graph[6], 以降 PDG) とは，プログラム中の要素 (文) 間の依存関係を表した有向グラフである．依存関係には次の 2 種類が存在する．

データ依存 文 s で変数 v を定義し，変数 v が再定義されることなく文 t において参照される場合，文 s から文 t にデータ依存があるという．

制御依存 文 s が条件文または繰り返し文の条件式であり，その条件判定の結果によって文 t が実行されるか否かが直接決定される場合，文 s から文 t へ制御依存があるという．

図 2 の例では，4 行目で変数 y，z の値が参照されているため，2，3，5 行目から 4 行目へのデータ依存関係がある．また，5 行目は 4 行目によって実行の有無が定まるため，4 行目から 5 行目への制御依存関係がある．また，一般的に PDG はメソッドの入り口に相当する頂点を持ち，その頂点からメソッド直下のすべての文へ制御依存関係が存在する．

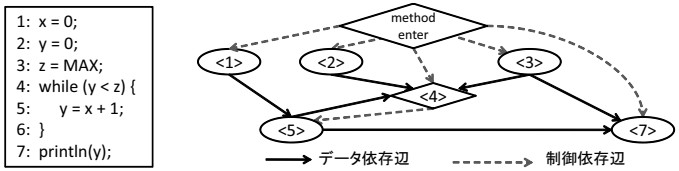

図 2 PDG の例
Fig. 2 An Example of PDG

2.3 PDG を用いたコードクローン検出

これまでにコードクローンの自動検出手法は多数提案されており，それらは検出時に用いるデータ構造によって分類できる．その 1 つである PDG を用いた検出は，PDG 上で同形部分グラフとなっている箇所をコードクローンとして検出する手法である．この手法の最大の利点として，コード順序が異なる場合や，for 文と while 文等の表現の異なる場合といった，他の検出手法では検出不可能なコードクローンを検出できるという点が挙げられる．

本稿で用いる検出ツール Scorpio[7] はこの PDG を用いた検出に分類される検出ツールである．Scorpio は PDG の比較時に，各頂点が表現するコード片からハッシュ値を生成し，そのハッシュ値を用いて頂点同士の一致性を判定している．このハッシュ値の生成時に特殊な処理を行っており，ユーザ定義名のみが異なるコードクローンを検出可能である．

3. 提 案 手 法

3.1 概　　要

提案手法は，対象とするメソッドの対，各メソッドの PDG，及びそのメソッド間のコードクローン情報を入力とし，各メソッドについて親クラスに引き上げるべき頂点集合，及び子クラスに残すべき頂点集合を出力する．子クラスに残すべき頂点集合については，1 つのメソッドとして抽出すべき頂点集合も併せて出力する．

3.2 用語の定義

定義 3.1 (対象メソッドペア) $ownerclass(m)$ をメソッド m が実装されているクラス，$ParentClasses(c)$ をクラス c の先祖クラスの集合であると仮定する．このとき，式 (1) を満たす 2 つのメソッド (m_0, m_1) の対を対象メソッドペアと呼ぶ．

$$(ownerclass(m_0) \neq ownerclass(m_1)) \wedge \exists c(c \in ParentClasses(ownerclass(m_0)) \wedge c \in ParentClasses(ownerclass(m_1))) \quad (1)$$

定義 3.2 (到達辺集合, 出発辺集合, 出発頂点, 到着頂点) ある頂点 n について，n を到

達頂点とする辺の集合を到達辺集合と定義し,$BackwardEdges(n)$ と表記する．また頂点 n を出発頂点とする辺の集合を出発辺集合と定義し,$ForwardEdges(n)$ と表記する．また，ある辺 e について，e の出発頂点を $fromnode(e)$，到達頂点を $tonode(e)$ とする．

定義 3.3 (PDG 頂点集合，PDG 辺集合) あるメソッド m について $Nodes(m)$ を m の PDG 頂点集合，及び頂点集合 N について $Edges(N)$ を N の PDG 辺集合と定義する．

定義 3.4 (クローンペア) 入力として与えられた対象メソッドペア (m_0, m_1) 間のすべてのクローンペアの集合を $ClonePairs(m_0, m_1)$ とする．このとき，式 (2) を満たす頂点集合 N_0，N_1 の対をクローンペアと呼び，$cp(N_0, N_1)$ と表記する．

$$(N_0, N_1) \in ClonePairs(m_0, m_1) \tag{2}$$

定義 3.5 (共通頂点集合，メソッド内共通頂点集合) 対象メソッドペアの共通頂点集合 $CommonNodePairs(m_0, m_1)$ を式 (3) で定義する．

$$CommonNodePairs(m_0, m_1) := \{(s,t) : \exists cp(N_0, N_1) \in ClonePairs(m_0, m_1) \\ ((s \in N_0) \wedge (t \in N_1))\} \tag{3}$$

また，対象メソッドペア (m_0, m_1) の各メソッドについて，そのメソッドのメソッド内共通頂点集合 $CommonNodes(m_i)\,(i = 0, 1)$ を式 (4) で定義する．

$$CommonNodes(m_i) := \{n \in Nodes(m_i) : (\exists (s,t) \in CommonNodePairs \\ (m_0, m_1)\,((s = n) \vee (t = n)))\} \tag{4}$$

定義 3.6 (差異頂点集合) 対象メソッドペア (m_0, m_1) の各メソッドについて，そのメソッドの差異頂点集合 $DiffNodes(m_i)\,(i = 0, 1)$ を式 (5) で定義する．

$$DiffNodes(m_i) := \{n \in Nodes(m_i) : n \notin CommonNodes(m_i)\} \tag{5}$$

3.3 処　　理

本節では提案手法の処理内容について述べる．なお，提案手法は下記の手順で実行される．

Step1 Template Method パターンの適用に必要な条件の解決.

Step2 1 つのメソッドとして抽出すべき頂点集合の特定.

Step3 Step2 で特定した頂点集合同士のメソッドペア間での対応付け.

以降の小節で各ステップについて詳細に述べる．

3.3.1 Step1:必要条件の解決

Template Method の形成にはいくつかの必要条件が存在するため，対象メソッドペアへの適用が可能か否かを判定する必要がある．適用ができない場合，共通頂点集合を必要条件を満たすように操作し，共通頂点集合の操作では必要条件の解決ができない場合は適用が不可能であると判断する．以下に満たさなければならない必要条件とその解決方法を述べる．

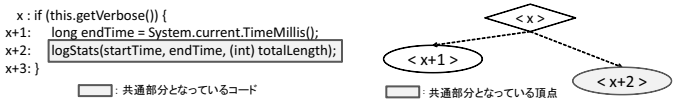

図 3 差異部分から共通部分への制御依存
Fig. 3 An Example of Control Dependence Edge from Different-Part to Common-Part

条件 1:差異頂点集合が return 文を含まない

差異頂点集合が表現する文に return 文が含まれる場合，差異頂点集合を新規メソッドとして抽出した際，作成したメソッドはそのメソッド自身とそのメソッドを呼び出す親クラスに引き上げたメソッドの 2 つのメソッドを同時に脱け出さなければならない．このような場合，提案手法は入力されたメソッドペアに対しての Template Method パターン適用は不可能と判定する．すなわち，$statement(n)$ を頂点 n が表現する文，$return(s)$ を文 s が return 文のときのみ真となる述語論理式であるとするとき，以下の論理式 $hasReturn(m_0, m_1)$ を充足する対象メソッドペア (m_0, m_1) を適用不可能であると判定する．

$$hasReturn(m_0, m_1) := \exists i \in \{0,1\}(\exists n \in DiffNodes(m_i)(return(statement(n)))) \tag{6}$$

条件 2:差異頂点集合からメソッド内共通頂点集合への制御依存が存在しない

$control(e)$ を辺 e が制御依存辺のときのみ真となる述語論理式であるとする．このとき，対象メソッドペア (m_0, m_1) について，論理式 $invalidControl(m_0, m_1)$ を以下に定義する．

$$invalidControl(m_0, m_1) := \exists i \in \{0,1\}(\exists e \in Edges(Nodes(m_i))((control(e)) \wedge \\ (fromnode(e) \in DiffNodes(m_i)) \wedge (tonode(e) \in CommonNodes(m_i)))) \tag{7}$$

$invalidControl(m_0, m_1)$ を充足する対象メソッドペア (m_0, m_1) の差異頂点集合を新規メソッドとして抽出することは不可能である．これは**図 3** のように，差異頂点集合が条件式とブロック文の一部のみを含んでいる場合に生じる問題である．この条件は共通頂点集合に含む頂点ペア情報を操作することで解決が可能である．以下にその解決方法 $resolveIC(CommonNodePairs(m_0, m_1))$ を示す.

Input: $CommonNodePairs(m_0, m_1)$
Output: $CommonNodePairs(m_0, m_1)(\neg invalidControl(m_0, m_1))$
for all (n_0, n_1) such that $(n_0, n_1) \in CommonNodePairs(m_0, m_1)$ **do**
  for all be such that $((be \in BackwardEdges(n_0)) \wedge control(be) \wedge fromnode(be) \in DiffNodes(m_0))$ **do**
    $CommonNodePairs(m_0, m_1) \leftarrow CommonNodePairs(m_0, m_1) - (n_0, n_1)$
  end for
  for all be such that $((be \in BackwardEdges(n_1)) \wedge control(be) \wedge fromnode(be) \in DiffNodes(m_1))$

```
    do
      CommonNodePairs(m_0, m_1) ← CommonNodePairs(m_0, m_1) − (n_0, n_1)
    end for
  end for
  return CommonNodePairs(m_0, m_1)
```

3.3.2 Step2:抽出頂点集合，抽出頂点集合群の特定

対象メソッドペア $(m_0,\ m_1)$ の各メソッドについて，その差異頂点集合から 1 つの新規メソッドとして抽出すべき頂点の集合である抽出頂点集合を特定し，それらの集合である抽出頂点集合群 $DiffGroups(m_i)\,(i=0,1)$ を特定する．

まず，頂点集合 N に含まれる頂点のうち，頂点 n から辺を辿って到達可能な頂点集合を特定する処理 $search(n,N)$ を以下に定義する．

```
Input: n:基点，N:探索可能頂点集合
Output: n から辺を辿って到達可能かつ N に含まれる頂点集合
T ← n
for all e such that e ∈ ForwardEdges(n) do
  if tonode(e) ∈ N then
    T ← T + search(tonode(e), N)
  end if
end for
return T
```

次に，2 つの頂点集合 S,T に対して $merge(S,T)$ を以下に定義する．

$$merge(S,T) := S \cup T \tag{8}$$

このとき，$DiffGroups(m_i)$ を特定する処理 $detectDiffGroups(m_i)$ を以下に示す．

```
Input: DiffNodes(m_i)
Output: DiffGroups(m_i)
DiffGroups(m_i) ← ∅
for all n such that n ∈ DiffNodes(m_i) do
  dg ← search(n, DiffNodes(m_i))
  DiffGroups(m_i) ← dg
  for all adg such that (adg ∈ DiffGroups(m_i) ∧ ∃n ∈ dg(n ∈ adg)) do
    DiffGroups(m_i) ← DiffGroups(m_i) − adg + merge(adg, dg)
  end for
end for
```

3.3.3 Step3:抽出頂点集合同士の対応付け

対象メソッドペア $(m_0,\ m_1)$ の各メソッドにおける抽出頂点集合群 $DiffGroups(m_0)$，$DiffGroups(m_1)$ の各要素同士で対応を取り，抽出頂点集合ペア $DiffGroupPairs(m_0,m_1)$ を特定する．まずこの手順に必要となる返り値集合の特定手法を述べた後，この手順のアルゴリズムを説明する．

返り値集合の特定

ある抽出頂点集合 dg をメソッドとして抽出した際，返すべき値の集合である返り値集合 $ReturnValues(dg)$ を特定する．はじめに，$data(e)$ を辺 e がデータ依存辺のときのみ真となる論理式とするとき，頂点集合 N からのデータ依存辺の集合 $output(N)$ を式 (9) で定義する．

$$output(N) := \{e \in Edges(N) : ((fromnode(e) \in N) \wedge (tonode(e) \notin N) \wedge data(e))\} \tag{9}$$

また，頂点集合 N を保持するメソッドを $ownermethod(N)$，$Variables(m)$ をメソッド m で使用される変数の集合とし，$var(de)$ をデータ依存辺 de が表す変数とする．このとき，返り値集合 $ReturnValues(dg)$ を式 (10) で定義する．

$$ReturnValues(dg) := \{v \in Variables(ownermethod(dg)) : \exists e \in output(dg)(v = var(e))\} \tag{10}$$

対応付けアルゴリズム

以下の説明では，対象メソッドペアを $(m_0,\ m_1)$ とし，各メソッドを $m_i\,(i=0,1)$ とする．

はじめに，ある抽出頂点集合 N に対し，その抽出頂点集合へのデータ依存辺を持ち，かつそのデータ依存辺の出発頂点がメソッド内共通頂点集合に含まれるような辺集合 $input(N,m_i)$ を式 (11) で定義する．

$$input(N,m_i) := \{e \in Edges(Nodes(m_i)) : ((fromnode(e) \in CommonNodes(m_i)) \wedge (tonode(e) \in N))\} \tag{11}$$

また，2 つの頂点 n_0,n_1 の対が共通頂点集合に含まれるか否かを判定する式 $clone(n_0,n_1)$ を式 (12) で定義する．

$$clone(n_0,n_1) := \exists (a,b) \in CommonNodePairs\ (m_0,m_1)\,((n_0 = a \wedge n_1 = b) \vee (n_0 = b \wedge n_1 = a)) \tag{12}$$

次に，$typeE(e)$ を辺 e の種類 (データ依存辺か制御依存辺か)，$typeV(v)$ を変数 v の型を表すものとし，$|S|$ を集合 S の要素数とする．このとき，2 つの抽出頂点集合 DG_0,DG_1 への依存辺の集合 B_0，B_1 に対し，それらの依存辺の出発頂点が共通頂点集合内で対応しているか否かを判定する論理式 $sameInput(B_0,B_1)$ を式 (13) で定義する．また,DG_0,DG_1 からのデータ依存辺が表す変数の集合 V_0，V_1 に対し，それらの変数が表す型及び変数の数が一致するか否かを判定する式 $sameOutput(V_0,V_1)$ を式 (14) で定義する．

$$\begin{aligned} sameInput(B_0,B_1) := (\forall b_0 \in B_0, \exists b_1 \in B_1((typeE(b_0) = typeE(b_1)) \wedge \\ clone(fromnode(b_0), fromnode(b_1))) \\ \wedge (\forall b_1 \in B_1, \exists b_0 \in B_0((typeE(b_0) = typeE(b_1)) \wedge \\ clone(fromnode(b_0), fromnode(b_1))) \end{aligned} \tag{13}$$

$$sameOutput(V_0, V_1) := (|V_0| = |V_1|) \wedge (\forall v_0 \in V_0, \exists v_1 \in V_1(typeV(v_0) = typeV(v_1))) \wedge (\forall v_1 \in V_1, \exists v_0 \in V_0(typeV(v_0) = typeV(v_1))) \quad (14)$$

このとき，抽出頂点集合間の対応の有無を判定する式 $same(S, T)$ を式 (15) に定義する．

$$same(S, T) := sameInput(input(S, ownermethod(S)), input(T, ownermethod(T))) \wedge sameOutput(ReturnValues(S), ReturnValues(T)) \quad (15)$$

以上の定義のもとで，抽出頂点集合の対応を取る処理 $detectDGPairs(DiffGroups(m_0), DiffGroups(m_1))$ を以下に示す。

Input: $DiffGroups(m_0), DiffGroups(m_1)$
Output: $DiffGroupPairs(m_0, m_1)$
$DiffGroupPairs(m_0, m_1) \leftarrow \emptyset$
$RDG \leftarrow \emptyset$
for all dg_0 such that $dg_0 \in DiffGroups(m_0)$ **do**
  $pd \leftarrow false$
  for all $dg1$ such that $dg_1 \in DiffGroups(m_1)$ **do**
    if $same(dg_0, dg_1)$ **then**
      $DiffGroupPairs(m_0, m_1) \leftarrow DiffGroupPairs(m_0, m_1) + (dg_0, dg_1)$
      $RDG \leftarrow RDG + dg_1$
      $pd \leftarrow true$
      break
    end if
  end for
  if $pd = false$ **then**
    $DiffGroupPairs(m_0, m_1) \leftarrow DiffGroupPairs(m_0, m_1) + (dg_0, \text{NULL})$
  end if
end for
for all dg_1 such that $dg_1 \in DiffGroups(m_1) \wedge dg1 \notin RDG$ **do**
  $DiffGroupPairs(m_0, m_1) \leftarrow DiffGroupPairs(m_0, m_1) + (\text{NULL}, dg_1)$
end for
return $DiffGroupPairs(m_0, m_1)$

ここで対応関係の取られた抽出頂点集合 $dg_{pair}(dg_0, dg_1) \in DiffGroupPairs(m_0, m_1)$ は同じシグネチャを持つメソッドとして抽出すべき箇所となる．いずれか一方が NULL の場合，その処理は一方のメソッドにしか含まれない処理であり，その処理を含まないメソッドには同じシグネチャを持つ何もしないメソッドを作成する必要が生じる．

3.4 処理の流れ

提案手法の処理の流れを以下に示す．なお，入力情報から $CommonNodePairs(m_0, m_1)$ を特定する処理を $detectInitCN$ とし，$CommonNodePairs(m_0, m_1)$ から $DiffNodes(m_i)\,(i = 0, 1)$ を特定する処理を $detectDN$ と表記している．

Input: (m_0, m_1), $Nodes(m_0)$, $Nodes(m_1)$, $Edges(Nodes(m_0))$, $Edges(Nodes(m_1))$, $ClonePairs(m_0, m_1)$
Output: $(CommonNodePairs(m_0, m_1),\ DiffGroupPairs(m_0, m_1))$
$DiffGroupPairs(m_0, m_1) \leftarrow \emptyset$
$CommonNodePairs(m_0, m_1) \leftarrow detectInitCN$
$DiffNodes(m_0) \leftarrow detectDN(m_0)$
$DiffNodes(m_1) \leftarrow detectDN(m_1)$
if $hasReturn(m_0, m_1)$ **then**
  return (NULL, NULL) /* invalid method pair */
end if
if $invalidControl(m_0, m_1)$ **then**
  $CommonNodePairs(m_0, m_1) \leftarrow resolveIC(CommonNodePairs(m_0, m_1))$
  $DiffNodes(m_0) \leftarrow detectDN(m_0)$
  $DiffNodes(m_1) \leftarrow detectDN(m_1)$
end if
$DiffGroups_0 \leftarrow detectDiffGroups(m_0)$
$DiffGroups_1 \leftarrow detectDiffGroups(m_1)$
$DiffGroupPairs(m_0, m_1) \leftarrow detectDGPairs(DiffGroups_0, DiffGroups_1)$
return $CommonNodePairs(m_0, m_1), DiffGroupPairs(m_0, m_1)$

4. 実　　装

提案手法を Java を用いて実装した．入力となる PDG 情報及びコードクローン情報の特定には，ソースコード解析ツール MASU[8] とコードクローン検出ツール Scorpio を用いた．

4.1 入 力 支 援

提案手法では入力として対象となるメソッドペアをあらかじめ指定する必要がある．しかし，膨大なソースコードから対象メソッドペアを特定することは困難であると考えられる．

このため，本実装では Scorpio で検出されたコードクローン情報から適用可能な候補を自動的に特定し，その一覧を利用者に提示することで適用可能候補の特定の支援を行う．

4.2 言語依存支援

提案手法はプログラミング言語に依存せずに適用可能であるが，適用対象のプログラミング言語に応じた処理を追加することでより的確な支援が実現可能であると考えられる．MASU 及び Scorpio が Java のみに対応しているため，本実装ではパターン適用時の必要条件に Java に特化した条件を追加した．以降で追加した条件について述べる．

条件 3:抽出頂点集合が返すべき値が 1 つ以下である

Java では 1 つのメソッドは 1 つ以下の値しか返すことができない．このため，提案手法で特定した抽出頂点集合が新規メソッドとして抽出不可能である可能性がある．そこで本実

装では，3.3.2 で述べた抽出頂点集合特定処理の終了後に処理を追加する．

まず，ある頂点集合 N について，$Bounds(N)$ を式 (16) で定義する．

$$Bounds(N) := \{n \in N : (\exists e \in ForwardEdges(n)(tonode(e) \notin N))\} \quad (16)$$

次に，$NodesContainEdgesOfOther(N, v)$ を式 (17) で定義する．

$$NodesContainEdgesOfOther(N, v) := \{n \in N : \exists e \in ForwardEdges(n) \\ (data(e) \wedge (tonode(e) \notin N) \wedge (var(e) \neq v))\} \quad (17)$$

このとき，次に示す処理を $DiffGroups(m_i)$ に施すことで条件の解決を行う．

Input: $DiffGroups(m_i)$
Output: $DiffGroups(m_i)$ (sartisfying $\forall dg \in DiffGroups(m_i)(|ReturnValues(dg)| <= 1)$)
for all dg such that $dg \in DiffGroups(m_i)$ **do**
  if $(|ReturnValues(dg)| > 1)$ **then**
    $max \leftarrow 0$
    $U \leftarrow \emptyset$
    for all v such that $v \in ReturnValues(dg)$ **do**
      $S \leftarrow dg$
      $T \leftarrow \emptyset$
      repeat
        $T \leftarrow Bounds(S)$
        $S \leftarrow S - NodesContainEdgesOfOther(T, v)$
      until $(|NodesContainEdgesOfOther(T, v)| = 0)$
      if $|S| \geq max$ **then**
        $max = |S|$
        $U \leftarrow S$
      end if
    end for
    $DiffGroups(m_i) \leftarrow DiffGroups(m_i) - dg + U$
    $RN \leftarrow dg - U$
    for all n such that $n \in RN$ **do**
      $tdg \leftarrow search(n, RN)$
      for all $atdg$ such that $(atdg \in DiffGroups(m_i) \wedge \exists a \in tdg(a \in atdg))$ **do**
        $DiffGroups(m_i) \leftarrow DiffGroups(m_i) - atdg + merge(atdg, tdg)$
      end for
    end for
  end if
end for

条件 4:抽出頂点集合がブロック文をまたぎ，かつその一部分のみしか含んでいない

この条件は条件 2 と類似する条件だが，こちらは try 文などの，条件式を含まないブロック文によって生じる問題である．条件式を含まないブロック文の場合は制御依存関係が発生しないため，PDG の情報のみを用いてこの条件を満たすか否かを判別することはできない．

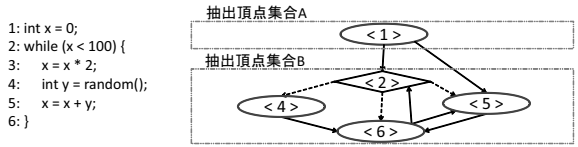

図 4 制約の追加による抽出頂点集合の分離問題
Fig. 4 A Problem of Secession of Extract-Nodes Set by Adding the Block Statement Constraint

そこで本実装では，MASU から得た構文情報を用いて条件の解決を行う．

まずはじめに，3.3.2 において PDG を走査する $search(n, N)$ を実行する際に，$sameBlock(f, t)$ という条件を追加する．すなわち，$search(n, N)$ を以下のように変更する．

Input: n:基点，N:探索可能頂点集合
Output: n から辺を辿って到達可能かつ N に含まれる頂点集合
$T \leftarrow n$
for all e such that $e \in ForwardEdges(n)$ **do**
  if $tonode(e) \in N \wedge sameBlock(fromnode(e), tonode(e))$ **then**
    $T \leftarrow T + search(tonode(e), N)$
  end if
end for
return T

ここで，頂点 n を直接保持するブロックを $ownerblock(n)$ とするとき，2 つの頂点 f，t を保持するブロック文の一致性を判定する式 $sameBlock(f, t)$ を式 (18) に定義する．

$$sameBlock(f, t) := (ownerblock(f) = ownerblock(t)) \quad (18)$$

この制約を追加することで，同じブロック内に含まれる頂点のみが抽出頂点集合として特定されるため，前述の問題点を解決することが可能となる．しかし，この制約を追加することで，抽出頂点集合の大きさが各ブロック文までに制限されてしまい，抽出頂点集合の数が増大してしまうという課題点が生じる．**図 4** の場合，制約の追加前はすべての文が 1 つの抽出頂点集合に含まれるが，制約を追加することでこれらが 2 つに分断される．

この問題を解決するために，ブロック文の集約処理をさらに追加する．以降このブロック文集約処理について述べる．

あるブロック文 s について，そのブロックを構成する頂点集合を $BlockNodes(s)$ とする．このとき，ブロック文の全頂点が最頂点集合に含まれるかを判定する式 $allDN(s, m_i)$ を式 (19) に定義する．

$$allDN(s) := \forall n \in BlockNodes(s)(n \in DiffNodes(m_i)) \quad (19)$$

また，$BlockStatements(m)$ をメソッド m のブロック文の集合とする．以上の定義を用いて，3.3.2 の処理を行う直前に以下に述べる処理を追加する．

for all s such that $BlockStatements(m_i)$ **do**
  if $allDN(s, m_i)$ **then**
    $n \leftarrow mergeNode(BlockNodes(s))$
    $DiffNodes(m_i) \leftarrow DiffNodes(m_i) - BlockNodes(s) + n$
  end if
end for

ここで，$owner(N)$ を頂点集合 N を直接保持するブロック文とするとき，$mergeNode(N)$ は頂点集合 N を基に以下の条件を満たす単一頂点 n を生成する処理である．

$$BackwardEdges(n) = \{e \in Edges(N) : ((fromnode(e) \in N) \wedge (tonode(e) \notin N))\} \quad (20)$$
$$ForwardEdges(n) = \{e \in Edges(N) : ((fromnode(e) \notin N) \wedge (tonode(e) \in N))\} \quad (21)$$
$$ownerblock(n) = owner(BlockNodes(owner(N))) \quad (22)$$

4.3 候補の提示

実装したツールは適用対象の PDG とソースコードの提示機能を有している．ソースコード表示では，共通部分をハイライト表示し，差異部分については 1 つのメソッド内に抽出すべき文を同じ色枠で囲んで表示することで集約の支援を行う．

また，対象メソッドペアにメソッド間の類似度などのメトリクスをいくつか定義し，利用者がそれらのメトリクスの閾値を自由に設定できる機能を実装した．これは，利用者によって集約したいと考える条件が異なると考えられるため，利用者が自由に候補を選定可能にすることで，集約したいと思える候補がより発見しやすくなると考えたためである．

5. 評　　価

提案手法を実装したツールを用いて，Apache Ant に対して適用実験を行った．Apache Ant の規模及びツールの実行に要した時間を**表 1** に示す．

適用実験の結果，282 の適用候補が検出された．**図 5** に検出された候補と適用例を示す．この例では変数 *op* と *image* という，ユーザ定義名の異なる変数が使用されており，既存手法ではこれらの変数に関わる return 文を共通部分とみなすことができない．そのため，既存手法では return 文を新規メソッドとして抽出する箇所に含める必要があり，パターンの適用に複雑な処理を要する．しかし，提案手法ではこの return 文を親クラスに引き上げることができ，適用後のコードの複雑さが既存手法と比較して軽減されていると考えられる．

また，Apache Ant に付属しているテストケースを実行し，この修正の有無によって挙動が変化していないことを確認した．さらに，メソッドペアの各メソッドの頂点総数に占める共通頂点の割合を表す類似度 $(2 * |CloneNodePairs|/|Nodes(m_0)| + |Nodes(m_1))|$ が 0.8〜0.9 という制約を設けて得られた候補のうち 27 の候補に対してパターンの適用を行ったところ，いずれの候補についても適用前後で挙動に変化がないことが確認できた．しかし，可視性の調整等で作成したツールが示した抽出頂点集合以外の箇所を新規メソッドとして抽出する必要がある場合や，反対にツールが抽出頂点集合と示した箇所を共通部分として親クラスに引き上げることができる場合がいくつか存在した．

表 1　実験対象と所要時間
Table 1　Target Software System and Execution Time

行数	ファイル数	所要時間と実験環境
212,401	829	110s (CPU: Xeon 2.67GHz(4 core)，メモリ: 4GB)

6. 関 連 研 究

提案手法と既存手法を比較した表を**表 2** に示す．Juillerat らは，ソースコードから抽象構文木 (Abstract Syntax Tree，以降 AST) を構築し，それをもとに Template Method パターンの適用を自動化する手法を提案している[2]．Juillerat らの手法では 2 つのメソッドの AST を比較し，同形部分木となっている箇所を共通部分とみなすことで Template Method パターン適用の自動化を実現している．この手法の特長として，適用後のソースコード例を提示することができるという点と，実行に要する時間的，空間的コストが小さいという点が挙げられる．しかし，各頂点が表すコード片の文字列が一致するか否かで AST の頂点を比較しており，ユーザ定義名の違いを吸収することはできず，また AST を用いているためコードの順序の違いや表現の違いを吸収することができないという課題点が存在する．

政井らは，Juillerat らの手法と同様に AST を用いて Template Method パターンの適用を支援する手法を提案している[3]．政井らの手法は適用後のソースコードを提示する機能は

表 2　既存手法との比較
Table 2　Comparison between Existing Methods and Proposed Method

	Juillrat らの手法[2]	政井らの手法[3]	提案手法
用いるデータ構造	AST	AST	PDG
コード順序の違いの吸収	×	○	○
ユーザ定義名の違いの吸収	×	×	○
表現の違いの吸収	×	×	○
実行コスト	◎	○	△

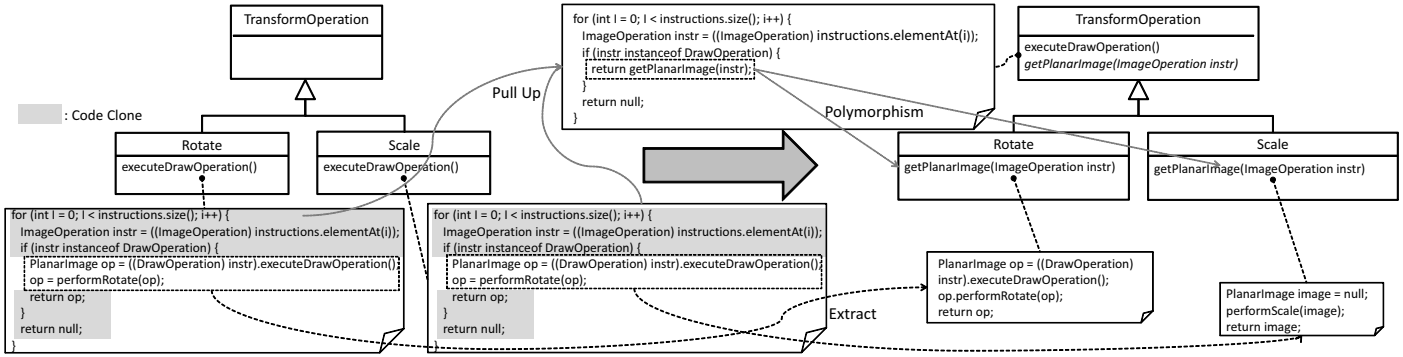

図 5 適用例
Fig. 5 An Example of Application of Template Method Pattern with Proposed Method

持たないが，AST の比較時に特殊な処理を施すことで，Juillerat らの手法の課題点として挙げられているコードの順序の違いを吸収することに成功している．また，適用可能な候補に対し，適用時に新たにメソッドとして抽出すべき箇所の戻り値の型や引数等の検査を行うことで適用容易性を判定し，提示する機能を備えている．ただし，AST の各頂点の比較を値の比較で行っており，変数名のように頂点の値が文字列である場合，文字列の一致性によって頂点の比較を行っている．このため，Juillerat らの手法と同様にユーザ定義名の違いを吸収することができず，また表現の違いの吸収についても対応していない．

これらの既存手法と比較すると，提案手法の最大の長所としてユーザ定義名や表現の違いを吸収可能であるという点が挙げられる．また，提案手法を実装したツールでは適用可能な候補の特定を併せて行っており，適用対象を容易に発見できるという利点がある．しかし，提案手法は事前に PDG を構築する必要があるため実行コストが大きく，また適用例の提示機能や，抽出容易性の提示機能を備えていないという点も課題点として挙げられる．

7. あとがき

本稿では，PDG を用いて Template Method パターンの適用が可能な候補を特定し，親クラスに引き上げるべき箇所と子クラスに残すべき箇所を利用者に提示することでコードクローンの集約を支援する手法を提案した．今後の課題として，パターン適用後のソースコード例の提示，及び 3 つ以上の類似メソッドへの拡張などが挙げられる．

謝辞 本研究は，日本学術振興会科学研究費補助金基盤研究 (C)(課題番号：20500033)，文部科学省科学研究費補助金若手研究 (B)(課題番号：22700031) の助成を得た．

参 考 文 献

1) 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌， Vol.J91-D, No.6, pp.1465–1481 (2008).
2) Juillerat, N. and Hirsbrunner, B.: Toward an Implementation of the ”Form Template Method Refactoring”, *7th IEEE International Working Conference on Source Code Analysis and Manipulation*, pp.81–90 (2007).
3) 政井智雄，吉田則裕，松下 誠，井上克郎：類似メソッド集約のための差分抽出支援，電子情報通信学会技術報告， Vol.110, No.60, pp.45–50 (2010).
4) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.M.: *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional (1995).
5) Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional (1999).
6) Ferrante, J., Ottenstein, K.J. and Warren, J.D.: The Program Dependence Graph and Its Use in Optimization, *ACM Transactions on Programming Languages and Systems*, Vol.9, No.3 (1987).
7) 肥後芳樹，楠本真二：プログラム依存グラフを用いたコードクローン検出法の改善と評価，情報処理学会論文誌， Vol.51, No.12, pp.2149–2168 (2010).
8) 三宅達也，肥後芳樹，楠本真二，井上克郎：多言語対応メトリクス計測プラグイン開発基盤 MASU の開発，電子情報通信学会論文誌 D， Vol.J92-D, No.9, pp.1518–1531 (2009).