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

信学技報
TECHNICAL REPORT OF IEICE.

集約されたメソッドを利用したコードクローンベンチマークの作成

幸　佑亮† 　肥後　芳樹† 　堀田　圭佑† 　楠本　真二†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5
E-mail: †{y-yusuke,higo,k-hotta,kusumoto}@ist.osaka-u.ac.jp

あらまし　コードクローンとは，ソースコード中に存在する互いに類似したコード片のことである．一般的に，コードクローンはソフトウェアの保守性を低下させる原因になるとされている．そのため，コードクローン情報を理解することはソフトウェア保守の観点から重要であり，これまでに多くのコードクローン検出ツールが開発されている．そこで，コードクローン検出ツールがどの程度正確にコードクローンを検出できるかどうかの精度評価が必要となる．コードクローン検出ツールの精度評価では，ベンチマーク（コードクローンの正解集合）を作成し，適合率と再現率の観点から評価することが多い．そのため，コードクローン検出ツールの精度評価は使用するベンチマークに依存する．先行研究で作成されたベンチマークは作成者の主観に依存していることと実際の開発過程で発生したコードクローンではないことが課題である．そこで本研究では，ソースコードリポジトリに蓄えられているソフトウェアの開発履歴情報を利用して集約されたメソッドを検出し，集約前のメソッドを正解のコードクローンと定義することで上記の課題を解決するベンチマークを提案する．このベンチマークはコードクローン検出ツールがどの程度正確に集約すべきコードクローンを検出できるかどうかを評価する際に有用である．

キーワード　コードクローン，コードクローン検出ツール，精度評価，ベンチマーク，集約

1. ま え が き

コードクローン（以降，クローンと表記する）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ある [1]．一般的に，クローンはソフトウェアの保守性を低下させる原因になるとされている．例えば，あるコード片にバグが存在した場合，そのコード片のクローンに対しても同様のバグが存在する可能性があり，同様の変更を検討する必要がある．そのため，クローンがソフトウェア中にどの程度存在しているか，及びどこに存在しているかを理解することはソフトウェア保守の観点から重要であり，これまでに多くのクローン検出ツールが開発されている [1] [2]．そこで，検出ツールがどの程度正確にクローンを検出できるかどうかの精度評価が必要となる．

クローン検出ツールの精度評価では，ベンチマーク（クローンの正解集合）を作成し，適合率と再現率の観点から評価することが多い [3]〜 [5]．適合率は検出ツールの検出結果のうちどの程度が正解のクローン（以降，正解クローンと表記する）かを指し，再現率は正解クローンをどの程度検出できたかを指す．そのため，検出ツールの精度評価は使用するベンチマークに含まれる正解クローンに依存する．ベンチマークに誤りが存在すれば，精度評価の結果に影響を与えることになる．

これまでにいくつかのベンチマークが作成されている．それらは以下のいずれかの方法でベンチマークを作成している．

人間が目視によって判断：クローン検出ツールが検出したクローンのペアや対象の機能を持つ可能性があるコード片の集合に対して，人間がクローンか否かを 1 つずつ目視で判断する [3] [6]．この手法は正解クローンか否かの判断がベンチマーク作成者の主観に依存していることが課題である．ベンチマークに含まれる正解クローンはその作成者のクローンの定義に依存するため，精度評価の客観性が乏しい．また，検出ツールの検出結果を基に正解集合を作成している場合，検出されなかったクローンが正解クローンに含まれないことも課題である．

人工的にクローンを生成：あるソフトウェアに存在するコード片を無作為に抽出し，そのコード片に対して何かしらの変更を加えた後，そのソフトウェアの任意の場所にペーストし，人工的に正解クローンを生成する [7]．この手法は実際の開発過程で発生したクローンではないことが課題である．実際の開発過程でほとんど発生しないようなクローンが正解クローンとされることによって，正確な精度評価ができなかった可能性がある事例が存在した [4]．

上記の手法の課題を解決するために本研究では，実際の開発過程で発生したクローンから成るベンチマークを自動で作成する．具体的にはソースコードリポジトリに蓄えられているソフトウェアの開発履歴情報を利用し，集約されたメソッドを検出する．集約はクローンに対するリファクタリングの 1 つであり，

表 1　先行研究のベンチマークのまとめ

ベンチマーク	手法	良い点	課題点
Bellon's benchmark	クローン検出ツールの実行と 目視による判断	クローンのベンチマークの先駆け	作成者の主観に依存 クローン検出ツールの検出結果に依存
Mutation and Injection Framework	擬似コピーアンドペースト によるクローン生成	作成者の主観に依存しない 正解クローンを自動で生成	開発者が作成したクローンではない
BigCloneBench	対象機能の検索と 目視による判断	クローン検出ツールの 検出結果に依存しない	作成者の主観に依存 正解クローンの候補が限定

集約前のメソッドを正解クローンと定義することでベンチマークを作成する．また，自動で作成することによって主観に依存しない正解クローンを多量に取得できる．

集約されたメソッドを検出するツールを実装し，実際にメソッドの集約が行われているかどうかを目視によって評価した．その結果，ツールは 19 個のメソッドの集約を検出し，全てが正しい検出であることを確認した．作成したベンチマークはウェブサイト (注1) で公開している．

2. 先行研究のベンチマーク

本章では，クローン検出ツールの精度評価の先行研究にて使用されたベンチマーク（Bellon's benchmark [3]，Mutation and Injection Framework [7]，BigCloneBench [5] [6]）を紹介する．以下では，各ベンチマークの概要を説明する．各ベンチマークの特徴を表 1 にまとめた．

Bellon's benchmark について説明する．このベンチマークは対象ソフトウェアに対してクローン検出ツールを実行し，その検出結果に対してクローンか否かを 1 つずつ目視で判断することで作成された．クローン検出ツールの精度評価で使用するベンチマークとして先駆けて作成された．正解クローンか否かの判断がベンチマーク作成者の主観に依存していることとクローン検出ツールの検出結果に依存していることが課題である．

次に，Mutation and Injection Framework について説明する．このベンチマークは人工的にクローンを生成することで作成された．主観に依存しない正解クローンを自動で生成できる．正解クローンが実際の開発過程で発生したクローンではないことが課題である．

最後に，BigCloneBench について説明する．このベンチマークはバブルソートやファイルコピーなどのソースコード中に頻繁に出現する機能を対象とする．まず，対象の機能を持つ可能性があるコード片の集合を，その機能の特徴を基に対象ソフトウェアから抽出することで作成した．次に，その集合に対してその機能を持つコード片か否かを 1 つずつ目視で判断した．対象の機能を持つと判断されたコード片を正解クローンとすることでベンチマークが作成された．クローン検出ツールを使用せずに正解クローンを取得することができる．対象の機能か否かの判断がベンチマーク作成者の主観に依存していること正解クローンの候補が対象の機能に限定していることが課題である．

以降では，以上 3 つのベンチマークの作成手順と課題点を詳細に説明する．

(注1)：http://sdl.ist.osaka-u.ac.jp/~y-yusuke/2016_clone_references_contain_merged_methods/

2.1 Bellon's benchmark

Bellon's benchmark の作成手順は以下の通りである．

STEP1：Bellon らは比較対象となる 6 つのクローン検出ツールと検出対象となる 8 つのソフトウェアを選択した．

STEP2：Bellon らは各クローン検出ツールの開発者に検出対象に対するクローン検出を依頼し，開発者は検出結果を Bellon らに送付した．

STEP3：Bellon らは検出結果のうち 2%を無作為に抽出し，それらがクローンか否かを 1 つずつ目視によって判断した．

Bellon's benchmark の課題点として以下の 4 つが挙げられる．

- クローンか否かの判断が Bellon らの主観に依存している．正解クローンがベンチマーク作成者に依存するため，精度評価の客観性が乏しくなり，結果に影響を与える．
- クローン検出ツールの検出結果に依存している．正解クローンの候補はベンチマーク作成当時（2002 年）のクローン検出ツールが検出したクローンに限定されているため，検出されなかったクローンは正解クローンに含まれない．
- 現代のツールによって再現不可能である．Bellon らの正解クローンの定義が十分に文書化されていない．

2.2 Mutation and Injection Framework

Mutation and Injection Framework の作成手順は以下の通りである．

STEP1：Svajlenko らは対象ソフトウェアから 250 個のコード片を無作為に抽出した．

STEP2：Svajlenko らは抽出したコード片に対して 15 種類の変更を適用し，トータルで 3,750 個のコード片を生成した．

STEP3：Svajlenko らは抽出元のソフトウェアに対して変更を適用した各コード片を無作為に 10 箇所へ挿入した．

以上の手順を踏むことで，擬似的にコピーアンドペーストを実行し，人工的にクローンを生成した．

Mutation and Injection Framework の課題点を以下に示す．

- 生成したクローンが実際の開発過程で発生したクローンではない．識別子のインスタンス名の変更という実際の開発環境でほとんど存在しないような変更を適用したクローンが正解クローンとされることによって，正確な精度評価ができなかった可能性がある事例が存在した [4]．

2.3 BigCloneBench

BigCloneBench の作成手順は以下の通りである．

STEP1：Svajlenko らはバブルソートやファイルコピーなどの計 43 種類の対象となる機能を選択した．

STEP2：Svajlenko らは対象の機能がどのように実装されているかとその機能の特徴を StackOverflow や API のドキュメントなどで調査した．

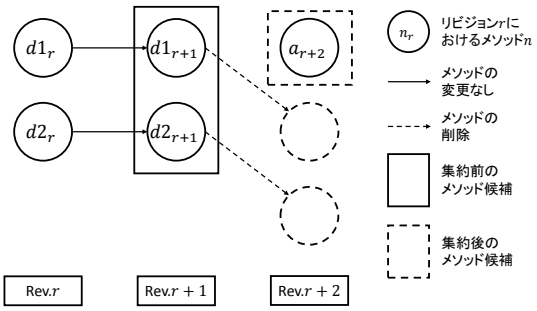


図 1　集約パターン 1

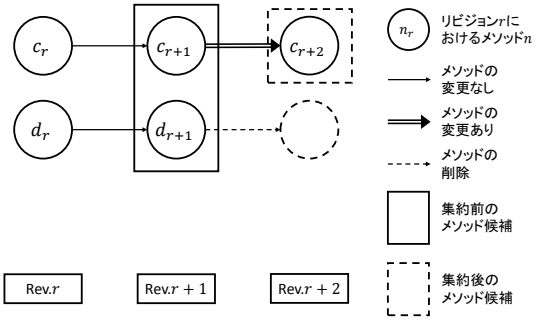


図 2　集約パターン 2

STEP3 : Svajlenko らは調査した情報を基にサンプルコードを作成し，IJaDataSet 2.0 [9] から候補を抽出した．

STEP4 : Svajlenko らは機能の特徴とサンプルコードの情報を基に抽出した候補に対して 1 つずつ目視によって対象の機能か否かを判断した．

STEP5 : Svajlenko らは対象の機能であると判断したコード片とサンプルコードの集合を正解クローンとして，構文の類似度によってタイプ別に仕分け，ベンチマークに追加した．

STEP6 : Svajlenko らは対象の機能ではないと判断したコード片とサンプルコードの集合を不正解クローンとしてベンチマークに追加した．

以上の手順を踏むことで，クローン検出ツールを使用せずに正解クローンを取得した．

BigCloneBench の課題点として以下の 2 つが挙げられる．

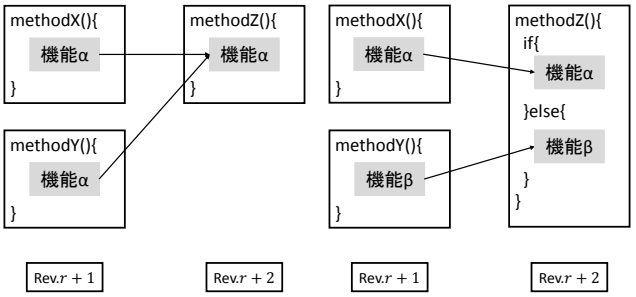
- 対象の機能か否かの判断が Svajlenko らの主観に依存している．これはクローンか否かの判断が主観に依存していることと同義であると考える．
- 正解クローンの候補がソースコード中に頻繁に出現する機能に限定している．

3. 提案するベンチマーク

2. 章で先行研究のベンチマークについて述べたが，共通する課題点として，ベンチマークに客観性がないことと正解クローンが実際の開発過程で発生したクローンではないことが挙げられる．そこで本研究では，これら 2 つの課題点を同時に解決するベンチマークを提案する．

3.1 キーアイデア

先行研究におけるベンチマークでは，対象ソフトウェアの単一のリビジョンのみを対象に 1. 章で述べた 2 つの手法を適用することで作成したが，本研究におけるベンチマークでは，ソースコードリポジトリに蓄えられているソフトウェアの開発履歴情報を利用することで作成する．具体的には全リビジョンを対象にリビジョン間の情報を利用し，集約されたメソッドを検出することでベンチマークを作成する．集約はクローンに対するリファクタリングの 1 つであり，集約前のメソッドを正解クローンと定義する．集約されたメソッドを自動で検出することで主観に依存しない正解クローンを多量に取得できる．



(a) クローンの集約の例　　(b) クローンの集約ではない例

図 3　構文の類似度を利用したクローンの集約かどうかの判断

3.2 概　　要

本研究では，メソッドの集約が行われるパターンを 2 つ定義する．各集約パターンを検出した後，構文の類似度とメソッドの呼び出し関係の情報を利用して誤検出を減らす．

まず，各集約パターンについて以下に説明する．

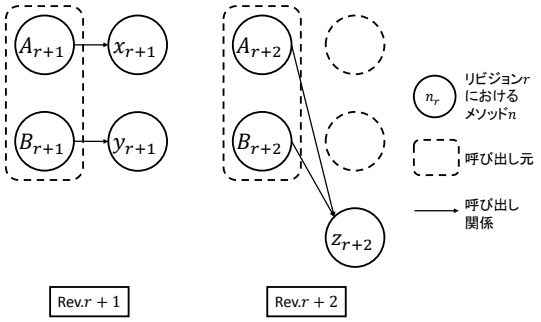
集約パターン 1 : 複数のメソッドが削除されて，追加されたメソッドに集約された場合である．例を図 1 に示す．

リビジョン r でメソッド $d1$，メソッド $d2$ が存在し，リビジョン $r+1$ でも存在していたとする．そして，リビジョン $r+1$ から $r+2$ へのコミットでメソッド $d1$，メソッド $d2$ が削除されたとする．また，メソッド a が追加されたとする．ここでリビジョン $r+1$ におけるメソッド $d1$，メソッド $d2$ を集約前のメソッド候補，リビジョン $r+2$ におけるメソッド a を集約後のメソッド候補と仮定する．

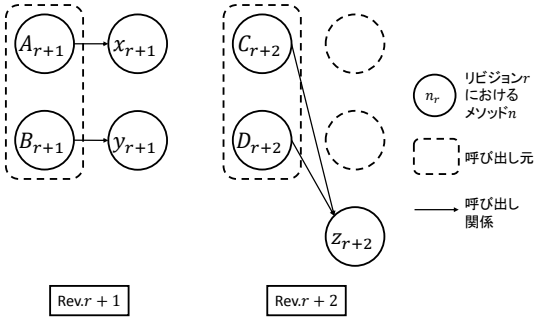
集約パターン 2 : 複数のメソッドが削除されて，既存のメソッドに集約された場合である．例を図 2 に示す．

リビジョン r でメソッド c，メソッド d が存在し，リビジョン $r+1$ でも存在していたとする．そして，リビジョン $r+1$ から $r+2$ へのコミットでメソッド c に何かしらの変更が行われたとする．また，メソッド d が削除されたとする．ここでリビジョン $r+1$ におけるメソッド c，メソッド d を集約前のメソッド候補，リビジョン $r+2$ におけるメソッド c を集約後のメソッド候補と仮定する．変更されたメソッドを対象とした理由は，メソッドの集約が行われる際，引数の変更や条件分岐の追加などの変更が行われることがあるためである．

2 つの集約パターンの検出に加えて，構文の類似度とメソッドの呼び出し関係の情報を利用する．構文の類似度を利用する理由は，2 つの集約パターンの検出だけでは集約されたメソッドとは限らないためである．例を図 3 に示す．図 3(a) に示すように，リビジョン $r+1$ におけるメソッド x とメソッド y が同じ機能 α を持つメソッドであるとする．そして，リビジョン $r+1$ から $r+2$ へのコミットで機能 α をメソッド z がまとめて保持するように集約する場合，クローンの集約である．し



(a) クローンの集約の例



(b) クローンの集約ではない例

図 4　メソッド呼び出し関係を利用したクローンの集約かどうかの判断

かし，図 3(b) に示すように，メソッド x とメソッド y が全く異なる機能 α，β を持つメソッドであるとして，その 2 つの機能をメソッド z がまとめて保持するように集約する場合，クローンの集約ではない．そこで，クローンの集約ではない場合を除くためにメソッド x とメソッド z，メソッド y とメソッド z の構文の類似度を算出する．

メソッドの呼び出し関係を利用する理由は，集約前のメソッド候補が構文上は類似しているが，全く異なる機能を持つメソッドである場合が存在するためである．クローンの集約の場合，同じ機能を持つ複数のメソッドが 1 つのメソッドに集約される．その集約の過程において，削除されたメソッドを呼び出している場所は追加あるいは変更されたメソッドを代わりに呼び出す．例を図 4 に示す．集約されたメソッド候補を呼び出している場所（以降，呼び出し元と表記する）を，そのメソッドを呼び出しているメソッドと定義する．リビジョン $r+1$ から $r+2$ へのコミットでメソッド x とメソッド y が削除されて，メソッド z が追加されたとする．図 4(a) に示すように，リビジョン $r+1$ におけるメソッド x とメソッド y の呼び出し元とリビジョン $r+2$ におけるメソッド z の呼び出し元が一致する場合，クローンの集約である．しかし，図 4(b) に示すように，メソッド x とメソッド y の呼び出し元とメソッド z の呼び出し元が一致しない場合，クローンの集約ではない．そこで，クローンの集約ではない場合を除くためにメソッドの呼び出し関係の情報を利用する．

3.3 作成手順

提案するベンチマークの作成手順について説明する．入力はソースコードリポジトリに蓄えられているソフトウェアの開発履歴情報であり，出力は集約されたメソッドの情報である．

STEP1：リビジョン間で追加，削除，変更されたメソッドを検出する．

STEP2：STEP1 で検出されたメソッドを対象に 2 つの集約パターンから集約前と集約後のメソッド候補を対応付ける．

STEP3：STEP2 で検出された集約前のメソッド候補と集約後のメソッド候補の類似度を用いて集約前のメソッド候補を絞る．

STEP4：STEP3 で絞り込まれたメソッドの呼び出し元の情報を利用して集約されたメソッドを検出し，正解クローンとする．

以上の STEP をソースコードリポジトリの全リビジョン間に対して実行する．以降，各 STEP の詳細について説明する．

3.4 STEP1

ECTEC [10] を使用してソースコードリポジトリに蓄えられているソフトウェアの開発履歴情報を解析する．ECTEC の入力は Java で記述されており，バージョン管理システム Subversion 及び Git で管理されたソフトウェアである．出力はリビジョン間の追加，削除，変更の情報が保存されたデータベースである．クラス，メソッド，ブロック単位でリビジョン間の追加，削除，変更を検出することができる．本研究では，ECTEC を使用してリビジョン間で追加，削除，変更されたメソッドを検出する．

先述の各集約パターンを例として説明する．以降の説明では，リビジョン $r+1$ から $r+2$ へのコミットで追加されたメソッドの集合を A_{r+2}，削除されたメソッドの集合を D_{r+2}，変更前のメソッドの集合を $C^{(b)}_{r+2}$，変更後のメソッドの集合を $C^{(a)}_{r+2}$ とする．また，リビジョン $r+1$ におけるメソッド n を n_{r+1} と表記する．

集約パターン 1 の場合：リビジョン r+1 から r+2 へのコミットでメソッド $d1$ とメソッド $d2$ が削除された．また，メソッド a が追加された．式 (1) で表記することができる．

$$d1_{r+1}, d2_{r+1} \in D_{r+2},\ a_{r+2} \in A_{r+2} \tag{1}$$

集約パターン 2 の場合：リビジョン r+1 から r+2 へのコミットでメソッド d が削除された．また，メソッド c が変更された．式 (2) で表記することができる．

$$d_{r+1} \in D_{r+2},\ c_{r+1} \in C^{(b)}_{r+2},\ c_{r+2} \in C^{(a)}_{r+2} \tag{2}$$

3.5 STEP2

この STEP では，STEP1 で検出されたリビジョン間で追加，削除，変更されたメソッドを対象に先述の 2 つの集約パターンを利用して集約前と集約後のメソッド候補を対応付ける．先述の各集約パターンを例として説明する．以降の説明では，リビジョン $r+1$ から $r+2$ へのコミットで集約された可能性があるメソッドの集合に対して，集約前のメソッド候補を $I^{(b)}_{r+2}$，集約後のメソッド候補を $I^{(a)}_{r+2}$ とする．

集約パターン 1 の場合：集約前のメソッド候補である全ての削除されたメソッドを，集約後のメソッド候補である追加されたメソッドと対応付ける．式 (3) で表記することができる．

$$I^{(b)}_{r+2} = D_{r+2},\ I^{(a)}_{r+2} = A_{r+2} \tag{3}$$

集約パターン 2 の場合：集約前のメソッド候補である全ての

表 2　対象ソフトウェアと検出結果

ソフトウェア	対象ディレクトリ	開始リビジョン（日付）	終了リビジョン（日付）	リビジョン数	検出数
Ant	/ant/core/trunk/main	r267549（2000/01/13）	r1240680（2012/02/05）	6,022	7
ArgoUML	/trunk/src	r1（1998/01/27）	r19893（2012/07/10）	3,925	10
jEdit	/jEdit/trunk	r3791（2001/09/02）	r22016（2012/08/17）	5,168	2

削除されたメソッドと変更前のメソッドを，集約後のメソッド候補である変更後のメソッドと対応付ける．式 (4) で表記することができる．

$$I^{(b)}_{r+2} = D_{r+2} \cup C^{(b)}_{r+2},\ I^{(a)}_{r+2} = C^{(a)}_{r+2} \tag{4}$$

以降では，説明の簡略化のため，$I^{(a)}_{r+2}$ に含まれるメソッドのうち 1 つを対象に説明する．集約パターン 1 では，追加されたメソッド a_{r+2}（$a_{r+2} \in I^{(a)}_{r+2}$），集約パターン 2 では，変更されたメソッド c_{r+2}（$c_{r+2} \in I^{(a)}_{r+2}$）を対象とする．$I^{(a)}_{r+2}$ に含まれる各メソッドに対して STEP3 以降に説明する処理を行う．

3.6　STEP3

この STEP では，STEP2 で検出された集約前のメソッド候補と集約後のメソッド候補の類似度を算出し，集約前のメソッド候補を絞り込む．先述の各集約パターンを例として説明する．

集約パターン 1 の場合：式 (5) に集約前のメソッド候補を絞り込むための条件を示す．$similarity(p, q)$ は，メソッド p とメソッド q の構文の類似度を算出する関数とする．$I^{(b)}_{r+2}$ に含まれるメソッドと集約後のメソッド候補である a_{r+2} の類似度を算出し，類似度が閾値 θ 以上のメソッドをこの STEP で絞り込まれた集約前のメソッド候補 $I^{'(b)}_{r+2}$ とする．

$$I^{'(b)}_{r+2} = \{m | similarity(m, a_{r+2}) \geqq \theta, m \in I^{(b)}_{r+2}\} \tag{5}$$

$|I^{'(b)}_{r+2}| < 2$ の場合，a_{r+2} に集約されたメソッドは存在しないとして，この STEP で処理を終了する．

集約パターン 2 の場合：式 (6) に集約前のメソッド候補を絞り込むための条件を示す．$I^{(b)}_{r+2}$ に含まれるメソッドと集約後のメソッド候補である c_{r+2} の類似度を算出し，類似度が閾値 θ 以上のメソッドをこの STEP で絞り込まれた集約前のメソッド候補 $I^{'(b)}_{r+2}$ とする．

$$I^{'(b)}_{r+2} = \{m | similarity(m, c_{r+2}) \geqq \theta, m \in I^{(b)}_{r+2}\} \tag{6}$$

$|I^{'(b)}_{r+2}| < 2$ の場合，c_{r+2} に集約されたメソッドは存在しないとして，この STEP で処理を終了する．また，$c_{r+1} \notin I^{'(b)}_{r+2}$ の場合，c_{r+2} の変更前である c_{r+1} が集約前のメソッド候補に含まれていないため，c_{r+2} に集約されたメソッドは存在しないとして，この STEP で処理を終了する．

3.7　STEP4

この STEP では，STEP3 で絞り込まれた集約前のメソッド候補を対象にそのメソッドの呼び出し元（そのメソッドを呼び出しているメソッド）の情報を抽出し，その情報を利用して集約されたメソッドを検出する．先述の各集約パターンを例として説明する．以降では，E_m をメソッド m の呼び出し元の集合，O_m をメソッド m に集約されたメソッドの集合とする．

集約パターン 1 の場合：式 (7) に条件を示す．$I^{'(b)}_{r+2}$ に含まれるメソッドと集約後のメソッド候補である a_{r+2} の呼び出し元を比較し，少なくとも 1 箇所が一致するようなメソッドを集約されたメソッドとする．

$$O_{a_{r+2}} = \{m | \exists e \in E_m [e \in E_{a_{r+2}}], m \in I^{'(b)}_{r+2}\} \tag{7}$$

ただし，$|O_{a_{r+2}}| < 2$ の場合，$O_{a_{r+2}} = \emptyset$ とする．

集約パターン 2 の場合：式 (8) に条件を示す．$I^{'(b)}_{r+2}$ に含まれるメソッドの呼び出し元と，集約後のメソッド候補である c_{r+2} とその変更前である c_{r+1} の呼び出し元の差分を比較し，少なくとも 1 箇所が一致するようなメソッドを集約されたメソッドとする．$I^{(b)}_{r+2}$ に含まれるメソッドのうち c_{r+1} は，呼び出し元の比較をせず，集約されたメソッドとする．

$$O_{c_{r+2}} = \{c_{r+1}, m | \exists e \in E_m [e \in (E_{c_{r+2}} - E_{c_{r+1}})], m \in (I^{'(b)}_{r+2} - c_{r+1})\} \tag{8}$$

ただし，$|O_{c_{r+2}}| < 2$ の場合，$O_{c_{r+2}} = \emptyset$ とする．

上記の STEP を実行することで，a_{r+2} もしくは c_{r+2} に集約されたメソッドが検出できる．$a_{r+2}, c_{r+2} \in I^{(a)}_{r+2}$ であり，他にもリビジョン $r+2$ において集約後のメソッド候補は存在する．また，ソースコードリポジトリの全リビジョン間に対して各 STEP を実行するため，ベンチマークとなる正解クローンの集合 O は，以下の式 (9) によって表記できる．リポジトリの開始リビジョンを $start$，終了リビジョンを end とする．

$$O = \{O_a | a \in \bigcup_k A_k, start + 1 \leqq k \leqq end\} \cup \{O_c | c \in \bigcup_k C^{(a)}_k, start + 1 \leqq k \leqq end\} \tag{9}$$

4.　評　　価

4.1　準　　備

3. 章で述べたベンチマークの作成手順を基に集約されたメソッドを検出するツールを実装した．実装したツールは，Java で記述されており，かつ Subversion を使用して管理されているソフトウェアのみを対象に検出可能である．検出対象のメソッドの制限として，6 行以上，かつ 50 トークン以上という条件を設けた．また，構文の類似度を算出する際の閾値を 70%に設定した．これらの条件は先行研究 [3] [6] を参考にした．

本研究では，3 つのオープンソースソフトウェアを対象に実装したツールを実行した．それぞれのソフトウェアの概要と検出結果を表 2 に示す．表中の“リビジョン数”は，検出対象となったリビジョンの数，つまりソースファイルに追加，削除，変更が加えられたリビジョンの数を示している．“検出数”は，ツールが検出した集約されたメソッドのグループ数である．

これらのソフトウェアを選定した基準は以下の通りである．

- Java で記述されており，かつ Subversion を使用して管理されていること．
- 長い開発期間を有し，かつ広く使用されているソフトウェアであること．

4.2 実験結果

実装したツールが集約されたメソッドをどの程度正確に検出できているかを目視により評価した．その結果，検出された 19 個のメソッドの集約は全て正しい検出であることを確認した．

4.3 妥当性の評価

評価手法：本研究では，実装したツールが集約されたメソッドをどの程度正確に検出できているかを目視によって評価した．しかし，著者らは本研究で対象としたソフトウェアの開発者ではないため，判断に誤りが混入している可能性がある．

対象ソフトウェア：本研究では，3 つのオープンソースソフトウェアを対象にしている．しかし，他のソフトウェアに対して実装したツールを実行し，再度ツールの検出精度を評価した場合，誤検出が検出される可能性がある．

5. 作成したベンチマークの特徴

5.1 ベンチマークの使用方法

本研究におけるベンチマークでは，検出対象期間内の全リビジョンを対象に作成したため，正解クローンが存在するリビジョンの全 Java ソースファイルに対してクローン検出ツールを実行する必要がある．そして，その検出結果を基に適合率や再現率を評価する．

5.2 ベンチマークの有用性

このベンチマークの有用性をクローン検出ツールが集約すべきクローンを検出できているかを評価する場合と検出すべきクローンを検出できているかを評価する場合に分けて説明する．

集約すべきクローンを検出できているかを評価する場合：この場合において正解クローンは集約すべきクローンである．本研究の正解クローンは集約されたクローンであり，集約すべきクローンは集約されたクローンを包含する．よって，正解クローンの定義に主観は入っていないため，客観的なクローン検出ツールの評価が可能である．

検出すべきクローンを検出できているかを評価する場合：この場合において正解クローンは検出すべきクローンである．先行研究の正解クローンはベンチマーク作成者が検出すべきと考えたクローンであり，本研究の正解クローンも先行研究と同様に我々が検出すべきと考えたクローンである．その点は先行研究と同様である．しかし，先行研究と異なる点は，クローンか否かの判断に主観が入っていない点と正解クローンが実際の開発過程で発生したクローンである点である．よって，先行研究よりも客観的なクローン検出ツールの評価が可能である．

5.3 ベンチマークの制限

このベンチマークの制限として以下の 3 点が挙げられる．

- 正解クローンを集約されたクローンに限定している．集約されたクローン以外にも検出すべきクローンは存在し，クローン検出ツールは集約されたクローン以外のクローンを検出する可能性がある．しかし，その検出結果に誤りが存在していたとしても，クローンではないコード片が誤ってクローンと検出されたのか，それとも集約されたクローン以外の検出すべきクローンが検出されたのかをこのベンチマークで判断することは不可能である．また，検出結果のうち正解クローンがどの程度含まれているかを評価する適合率の算出は現時点では難しい．
- 正解クローンの粒度がメソッド単位に限定している．クローンの粒度は行単位や字句単位など様々な粒度が存在する．集約されたクローンの粒度も多岐に亘るため，様々な粒度に対応する必要があるが，現時点では対応できていない．
- 数が少ない．ベンチマークに含まれる正解クローンの数が少ないため，クローン検出ツールの精度評価の結果は一般性に欠ける可能性がある．

6. あとがき

本研究では，ソースコードリポジトリに蓄えられている開発履歴情報を利用して集約されたメソッドを検出し，集約前のメソッドを正解のクローンと定義することでベンチマークを作成した．提案するベンチマークは以下の利点を持つ．

- ベンチマークに客観性がある．
- 正解クローンが実際の開発過程で発生したものである．

集約されたメソッドを検出するツールを実装し，実際にメソッドの集約が行われているかどうかを評価した．その結果，ツールは 19 個のメソッドの集約を検出し，全てが正しい検出であることを確認した．

このベンチマークはクローン検出ツールがどの程度正確に集約すべきクローンを検出できるかを評価する際に有用である．

本研究の今後の課題として，他のソフトウェアに対するツールの実行，ブロック単位の集約の検出，集約以外のクローンに対するリファクタリングの検出が挙げられる．最終的には，本研究のベンチマークを使用してクローン検出ツールの精度評価を行う予定である．

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（S）（課題番号：25220003），および文部科学省研究費補助金若手研究（A）（課題番号：24680002）の助成を得て行われた．

文　献

[1] 肥後，楠本，井上，“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.91-D，No.6，pp.1465-1481，2008.
[2] 神谷，肥後，吉田，“コードクローン検出技術の展開”，コンピュータソフトウェア，Vol.28，No.3，pp.28-42，2011.
[3] S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo. “Comparison and evaluation of clone detection tools”, IEEE TSE, Vol.31, No.10, pp.804-818, 2007.
[4] J. Svajlenko and C. K. Roy. “Evaluating modern clone detection tools,” in ICSME, 2014.
[5] J. Svajlenko and C. K. Roy. “Evaluating Clone Detection Tools with BigCloneBench,” in ICSME, 2015.
[6] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. Mia. “Towards a big data curated benchmark of inter-project code clones,” in ICSME, 2014.
[7] J. Svajlenko, C. K. Roy, and J. R. Cordy.“A mutation analysis based benchmarking framework for clone detectors,”in IWSC, 2013.
[8] S. Bellon. “Detection of software clones,” Technical Report, Institute for Software Technology, Universityof Stuttugart, 2003. available at `http://www.bauhaus-stuttgart.de/clones/`.
[9] Ambient Software Evoluton Group.“IJaDataset 2.0,”`http://secold.org/projects/seclone`, 2013.
[10] Y. Higo, K. Hotta, and S. Kusumoto. “Enhancement of crd-based clone tracking,” in IWPSE, 2013.