Smith-Waterman アルゴリズムを利用したギャップを含むコードクローン検出

村上 寛明[1,a)]　堀田 圭佑[1,b)]　肥後 芳樹[1,c)]　井垣 宏[1,d)]　楠本 真二[1,e)]

概要：これまでにさまざまなコードクローン検出手法が提案されている．ギャップを含むコードクローンを検出する手法として抽象構文木を用いた手法，プログラム依存グラフを用いた手法，関数メトリクスを用いた手法，LCS アルゴリズムを用いた手法が存在する．しかし，これらの既存手法には検出時間が長い，もしくは検出精度が低いといった課題点がある．そこで本研究は Smith-Waterman アルゴリズムを応用して，上述の課題点を改善したコードクローン検出手法を提案する．提案手法をコードクローン検出ツールとして実装し，Bellon らの評価実験を通じて，提案手法は既存手法の課題点を改善していることを示した．

キーワード：コードクローン，プログラム解析，文字列アルゴリズム

Gapped Code Clone Detection Using The Smith-Waterman Algorithm

Hiroaki MURAKAMI[1,a)]　Keisuke HOTTA[1,b)]　Yoshiki HIGO[1,c)]　Hiroshi IGAKI[1,d)]
Shinji KUSUMOTO[1,e)]

Abstract: A variety of techniques detecting code clones has been proposed before now. In order to detect gapped code clones, AST(Abstract Syntax Tree)-based technique, PDG(Program Dependence Graph)-based technique, metric-based technique and finger print technique using the LCS algorithm have been proposed. However, each of those techniques has limitations such as detection time is long or detection accuracy is not sufficient. This paper proposes a new method that detects gapped code clones using the Smith-Waterman algorithm in order to resolve those limitations. The authors developed the proposed method as a software tool, and confirmed that the proposed method could resolve the limitations by conducting a quantitative evaluation with Bellon' s benchmark.

Keywords: Code Clone, Program Analysis, String Algorithm

1. はじめに

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を指し，その主な生成要因はコピーアンドペーストである [1]．一部のコードクローンはソフトウェア保守に悪影響を与えると考えられており，これまでに多くのコードクローン検出技術および除去技術が提案されている [2]．

またプログラマはコード片をコピーアンドペーストした後，ペーストしたコード片に変更を加えることがある [3]．このことから，コピー元のコード片とペースト後に変更が加えられたコード片の間に，ソースコード上の不一致部分が存在することがある．以降，コードクローンに含まれる不一致部分をギャップと呼ぶ．あらゆるコードクローンを検出するためには，ギャップを含むコードクローンにも対応する必要がある．

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
a) h-murakm@ist.osaka-u.ac.jp
b) k-hotta@ist.osaka-u.ac.jp
c) higo@ist.osaka-u.ac.jp
d) igaki@ist.osaka-u.ac.jp
e) kusumoto@ist.osaka-u.ac.jp

既存のコードクローン検出手法は以下のように分類されている．

- 行単位の検出手法.
- 字句単位の検出手法.
- 抽象構文木 *1 を用いた検出手法.
- プログラム依存グラフ *2 を用いた検出手法.
- 関数メトリクスや LCS アルゴリズムなど，その他の技術を用いた検出手法.

これらの検出手法のうち，AST を用いた検出手法，PDG を用いた検出手法，関数メトリクスを用いた検出手法，LCS アルゴリズムを用いた手法はギャップを含むコードクローンを検出できる．しかし，これらの検出手法には課題点がある．AST や PDG を用いた検出手法は，ソースファイルを AST や PDG といった中間表現への変換し，さらに類似する AST や PDG を探索するため多くの検出時間を要する．関数メトリクスや LCS アルゴリズムを用いた検出手法は関数単位やブロック単位のコードクローンを検出するため，関数やブロックの内部に存在するコードクローンを検出することができない.

これらの課題点を改善するため，著者らは Smith-Waterman アルゴリズム [4] を応用したコードクローン検出手法を提案する．Smith-Waterman アルゴリズムとは，2 つの配列の中から類似する配列のペアを検出するアルゴリズムである．提案手法は，AST や PDG を用いた検出手法に比べて，ギャップを含むコードクローンを短時間で検出できる．その理由は，提案手法は AST や PDG といった中間表現を必要としないからである．さらに，提案手法は関数メトリクスや LCS アルゴリズムを用いた検出手法が検出できないコードクローンを検出できる．その理由は，関数メトリクスや LCS アルゴリズムを用いた検出手法は関数単位もしくはブロック単位のコードクローンを検出するのに対し，提案手法はより粒度の小さい文単位のコードクローンを検出するからである.

著者らは提案手法をコードクローン検出ツールとして実装し，Bellon らの実験 [5] を通じて評価を行った．しかし，Bellon らの実験で使用されているコードクローンには，ギャップの位置情報が含まれていない．そのため，Bellon らのコードクローンをそのまま用いると，正しく評価を行えない可能性がある．そこで，著者らは Bellon らのコードクローンにギャップの位置情報を追加し，それを用いてギャップの位置情報がコードクローンの検出結果に与える影響を調査した．最後に，ギャップの位置情報を追加したコードクローンを使用して，提案手法を実装したツールと既存のコードクローン検出ツールの精度比較を行った.

本研究の貢献は以下の通りである.

- 単一のソースファイル対から複数のコードクローン検出が可能となるように，Smith-Waterman アルゴリズムに変更を加えた.
- ギャップの位置情報を利用してコードクローン検出を行うと，それを利用しない場合に比べて検出結果を正しく評価できることを示した.
- 提案手法は既存手法に比べて検出結果の F 値が高い，つまり検出結果の再現率と適合率がバランスよく高いことを示した.

*1 Abstract Syntax Tree，以降 AST と呼ぶ
*2 Program Dependence Graph，以降 PDG と呼ぶ

2. Smith-Waterman アルゴリズム

Smith-Waterman アルゴリズム [4] とは，2 つの配列の中から類似する部分配列のペアを 1 つ検出するアルゴリズムである．Smith-Waterman アルゴリズムは，類似する部分配列の中にいくつかのギャップが含まれていても検出できるという特徴をもつ．図 1 は “TCGATAGCT” と “ACGTACGCA” という 2 つの塩基配列に対して Smith-Waterman アルゴリズムを適用した例である.

Smith-Waterman アルゴリズムの手順を以下に示す.

1. 表の作成: 入力された 2 つの配列が $\langle a_1, a_2, \cdots, a_N \rangle$ と $\langle b_1, b_2, \cdots, b_M \rangle$ のとき，$(N+2) \times (M+2)$ の表を作る.

2. 表の初期化: ステップ A で作られた表の一番上の行と一番左の列を入力された配列で埋める．さらに，2 番目の行と列を 0 で埋める.

3. セルのスコアの計算: 以下の数式にしたがって残りのセルのスコアを計算する.

$$v_{i,j}(2 \leq i, 2 \leq j) = max \begin{cases} v_{i-1,j-1} + s(a_i, b_j), \\ v_{i-1,j} + gap, \\ v_{i,j-1} + gap, \\ 0. \end{cases} \quad (1)$$

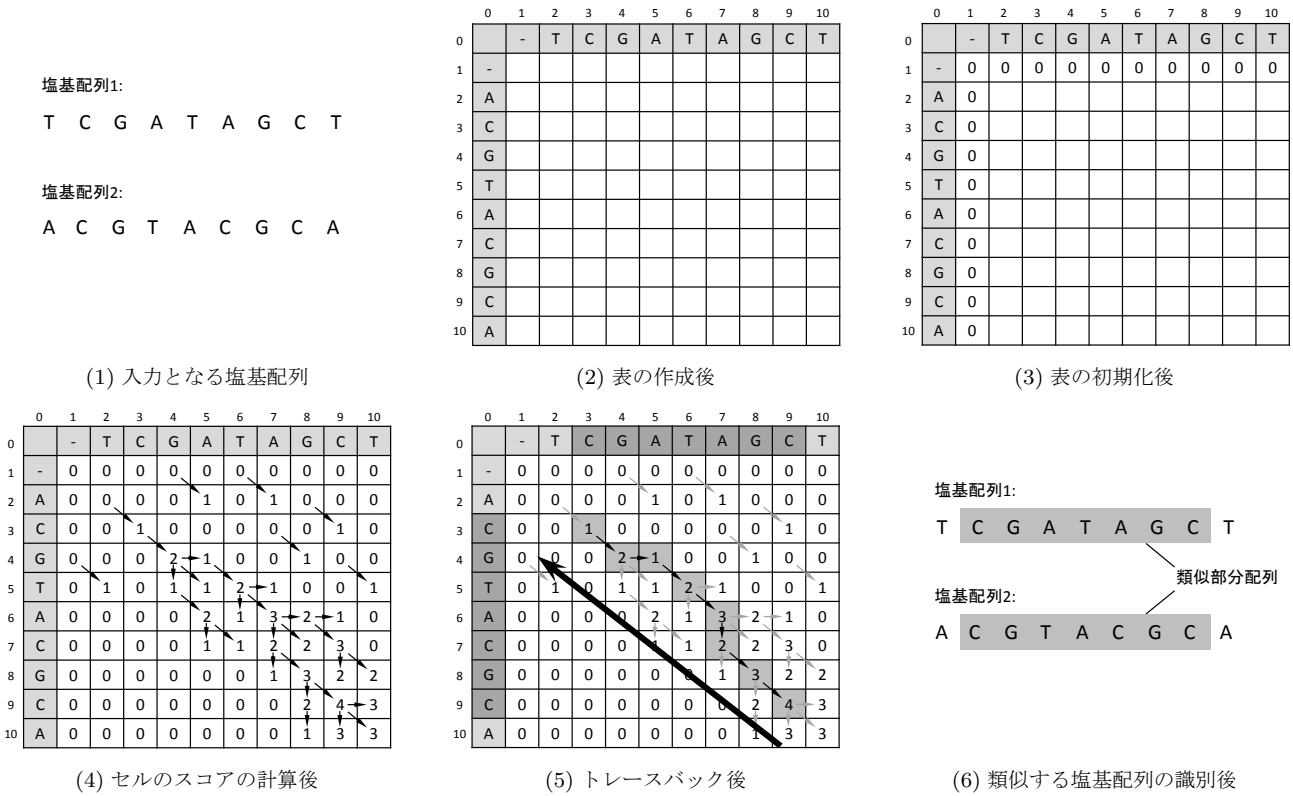
$$s(a_i, b_j) = \begin{cases} match & (a_i = b_j), \\ mismatch & (a_i \neq b_j). \end{cases} \quad (2)$$

ここで，$c_{i,j}$ は i 行 j 列に位置するセル，$v_{i,j}$ は $c_{i,j}$ のスコア，a_i は一方の入力配列 $\langle a_1, a_2, \cdots, a_N \rangle$ の i 番目の要素，b_j はもう一方の入力配列 $\langle b_1, b_2, \cdots, b_M \rangle$ の j 番目の要素，$s(a_i, b_j)$ は a_i と b_j の類似度を表す．また，$match$，$mismatch$，gap はスコアパラメータを表している.

スコアパラメータの $match$，$mismatch$，gap には任意の値を設定できる．図 1 においては，説明を簡単にするため，($match$, $mismatch$, gap) にそれぞれ (1, -1, -1) を設定している.

各セルのスコアを計算している間，$v_{i,j}$ を求めるのに使用されたセルから $c_{i,j}$ へのポインタが作られる．例えば図 1(4) において，$v_{7,6}(=3)$ は $v_{6,5}(=2)$ と $s(v_{0,6}, v_{7,0})(=1)$ の和である．この場合，$c_{6,5}$ から $c_{7,6}$ へのポインタが作られる.

4. 表のトレースバック: トレースバックとは，ステップ C で作ったポインタを順に遡る操作を意味する．トレース



(1) 入力となる塩基配列　(2) 表の作成後　(3) 表の初期化後

(4) セルのスコアの計算後　(5) トレースバック後　(6) 類似する塩基配列の識別後

図 1　Smith-Waterman アルゴリズムの適用例

Fig. 1　An Example of the Smith-Waterman Algorithm.

バックは表中の最大のスコアをもつセルから始まり，0 のスコアをもつセルで終わる．

5. 類似配列の識別: トレースバックのパスが指している要素が類似配列として識別される．図 1 において，数字が格納されたグレーのセルはトレースバックのパスを表している．このパスは “CGATAGC” と “CGTACGC” を指しているため，“CGATAGC” と “CGTACGC” が類似配列として識別される．

3. 提案手法

提案手法の入力は以下の通りである．

- ソースファイル．
- 最小クローン長（字句数）．
- 最大ギャップ率（検出された字句数に対するギャップとなるトークン数の割合）．
- スコアパラメータ（*match*，*mismatch*，*gap*）．

スコアパラメータ（*match*，*mismatch*，*gap*）は，5 章で記述する予備実験にて求めた値を使用する．提案手法の出力は，コードクローンのペアのリストである．

提案手法の手順を以下に示す．

ステップ 1: 字句解析および正規化．
ステップ 2: すべての文に対するハッシュ値の生成．
ステップ 3: 類似するハッシュ値列の識別．
ステップ 4: ギャップとなる字句の識別．
ステップ 5: 出力整形処理．

図 2 は提案手法による検出処理の例を示している．以降，各ステップについて詳細に説明する．

ステップ 1: 字句解析および正規化

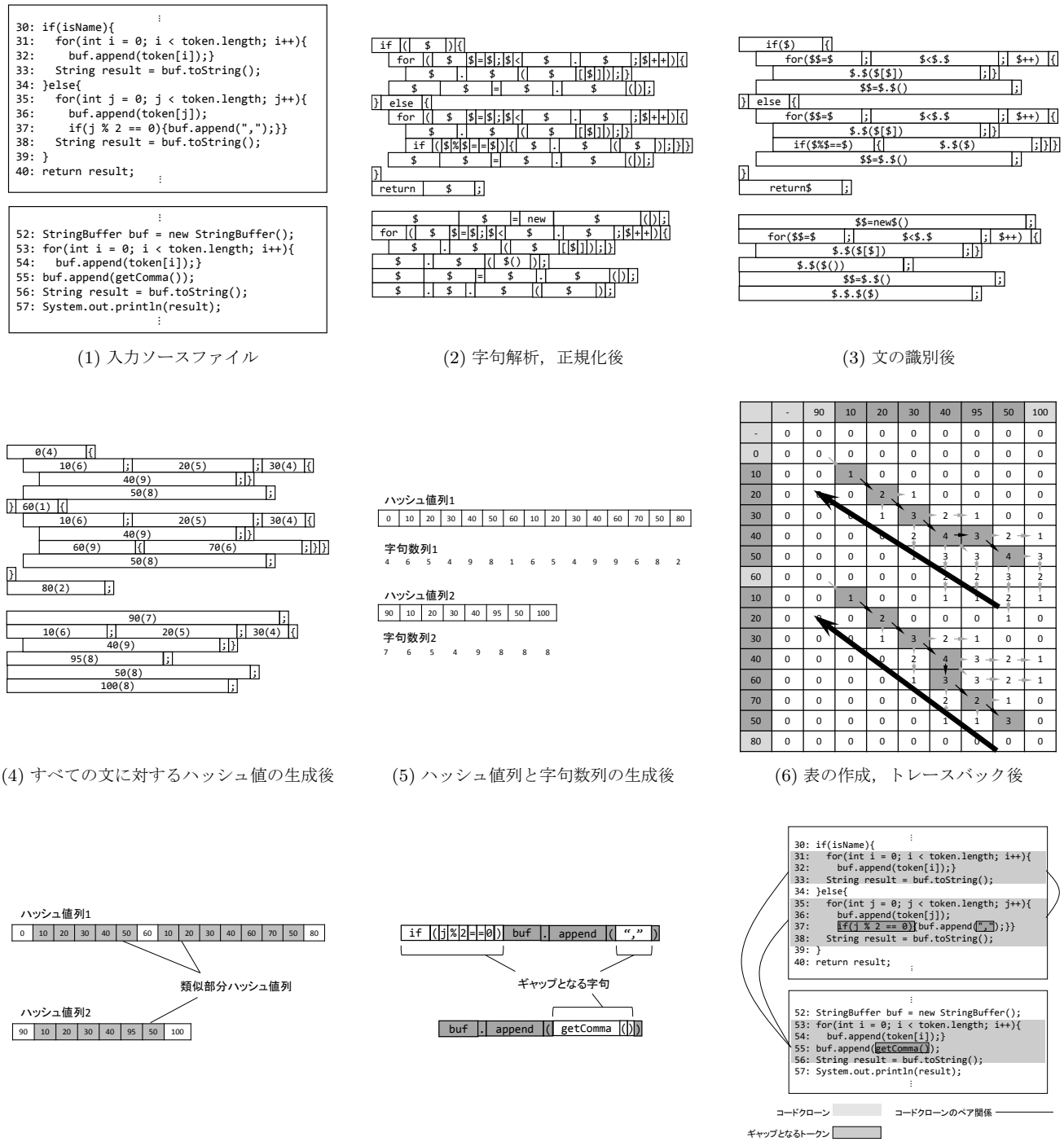
入力されたソースファイルを字句列に変換する．変数名や関数名が異なるコードクローンを検出するために，ユーザ定義名はすべて特殊文字に置換される．また同様の理由で，すべての修飾子は削除される．

ステップ 2: すべての文に対するハッシュ値の生成

字句列の中に存在するすべての文に対してハッシュ値を生成する．ここで，文はセミコロン（“;”）や中括弧（“{”，“}”）で区切られた字句列とする．また，文に含まれる字句の数も保存しておく．ステップ 2 までの過程で，1 つのソースファイルから 1 つのハッシュ値列が生成される．

ステップ 3: 類似するハッシュ値列の検出

Smith-Waterman アルゴリズムを用いて，ステップ 2 で得られたハッシュ値列から類似するハッシュ値列を検出する．ハッシュ値列のすべてのペアが 1 つの表を作成するため，1 つの表は 2 つのソースファイルに含まれるコードクローンを検出するために使用される．複数の表を用いることで，入力ソースファイルに含まれるすべてのコードクローンを検出することができる．ここで，2 章で説明した手順「4. 表のトレースバック」に対して以下に示す 2 つの



(1) 入力ソースファイル　(2) 字句解析，正規化後　(3) 文の識別後

(4) すべての文に対するハッシュ値の生成後　(5) ハッシュ値列と字句数列の生成後　(6) 表の作成，トレースバック後

(7) 類似するハッシュ値列の識別後　(8) ギャップとなる字句の識別後　(9) 出力整形処理後のソースファイル

図 2　提案手法による検出処理の例

Fig. 2　An Example of Detection Process Using the Proposed Method.

変更を加える．

- 2 つのソースファイルに含まれるすべてのコードクローンを見つけるために，トレースバックの開始点となるセルを複数選択する．そのセルは表の右下から左上へ順に探索され，以下に示す 2 つの条件をもつセルをトレースバックの開始点とする．
  - $v_{i,j} > 0$.
  - $v_{i,0} = v_{0,j}$.

さらに，トレースバックの開始点を $c_{i,j}$，終了点を $c_{k,l}$ $(k \leq i, l \leq j)$ とするとき，以下の集合 S に含まれるセルを次回以降のトレースバックにおいて探索の対象外とする．

$$S = \{c_{m,n} \mid k \leq m \leq i \wedge l \leq n \leq j\}. \qquad (3)$$

トレースバックの探索範囲を狭める理由は，ソースコード上の近い位置において，いくつものコードクローン

Algorithm 1 トレースバックの開始点となるセルを選ぶ

Input: 全てのセルのスコアを計算した $(N+2) \times (M+2)$ の表
Output: 表中におけるトレースバックの開始点

```
for i = N + 2 to 1 do
  for j = M + 2 to 1 do
    if v_{i,j} > 0 and v_{i,0} = v_{0,j} and (c_{i,j} が探索の対象外でない)
    then
      c_{i,j} をトレースバックの開始点とする
    end if
  end for
end for
```

を検出することを防ぐためである．トレースバックの開始点となるセルを選ぶアルゴリズムを Algorithm 1 に示す．

- トレースバックの最中に字句数とギャップとなる文に含まれる字句数を数える．その理由は，字句数が最小クローン長を超え，かつギャップの割合が最大ギャップ率未満であるコードクローンを検出するためである．なお，後述する実験では最小クローン長は 30，最大ギャップ率は 0.5 を設定している．また，トレースバックの最中にギャップとなる文が識別される．ギャップとなる文を構成する字句列はステップ 4 で用いられる．

ステップ 4: ギャップとなる字句の識別

ステップ 3 で得られたギャップとなる字句列に対して，LCS アルゴリズム [*3] を適用する．LCS アルゴリズムを用いることで，字句単位のギャップを検出する．

ステップ 5: 出力整形処理

ステップ 3 とステップ 4 で得られた類似部分ハッシュ値列をコードクローンを表す情報（ソースファイル名，開始行，終了行，ギャップとなる行）に整形する．ギャップとなる行は，ギャップとなる字句を含む行を表す．

4. 実験の概要

4.1 調査項目

3 章で述べた提案手法をコードクローン検出ツール CDSW として実装した．まず，Smith-Waterman アルゴリズムで用いる 3 つのパラメータ ($match$, $mismatch$, gap) の適した組み合わせを求めるために予備実験を行う．次に，以下に示す項目を調査するために実験 A と実験 B を行う．

調査項目 1: コードクローンの開始行と終了行だけではなく，ギャップの情報も用いることで検出結果の F 値は向上するのか．

調査項目 2: 提案手法は既存手法に比べて高精度か．

調査項目 3: 提案手法は大規模ソフトウェアに対して短時間でコードクローンを検出できるのか．

[*3] Longest Common Subsequence アルゴリズム．2 つの列の中から共通な部分列を検出する．

予備実験は，実験 A および実験 B で用いる Smith-Waterman アルゴリズムのパラメータを調査する．実験 A は調査項目 1 を，実験 B は調査項目 2 と調査項目 3 をそれぞれ調査する．各実験の詳細は，5 章，6 章，7 章でそれぞれ述べる．

4.2 実験対象，実験環境

コードクローン検出ツールの精度を求めるためには，正解となるコードクローンが必要である．本研究では，Bellon らの実験で使用されたコードクローン群 [6] を正解として扱う．このコードクローン群は，8 つの対象ソフトウェアから複数のコードクローン検出ツールを用いて検出し，さらに検出結果から Bellon らの目視確認によって抽出されたコードクローンの集合である [*4]．表 1 に 8 つの対象ソフトウェアの概要を示す．

本実験で用いた計算機の CPU は 2.27GHz Intel Xeon CPU，メモリサイズは 16.0GB である．

4.3 用語

本論文では以下に示す用語を用いる．

クローン候補: 各ツールが検出したコードクローン．

正解クローン: Bellon らが抽出したコードクローン．

ツールが正解クローンを検出できているか否かは ok 値 [5] を用いて判断する．ok 値の定義を以下に示す．

Definition 4.1 (ok 値)**.** あるコード片（f_1）が他のコード片（f_2）に含まれている程度を以下の式で表す．なお，$lines(f)$ はコード片 f に含まれる行の集合を表し，$|A|$ は集合 A の要素数を表す．

$$contain(f_1, f_2) = \frac{|lines(f_1) \cap lines(f_2)|}{|lines(f_1)|}. \qquad (4)$$

このとき，コードクローンのペア同士（p_1 と p_2）の ok 値は以下の式で定義される．以下の式において，$p.f_1$ はコードクローンのペア p の一方のコード片，$p.f_2$ はもう一方のコード片を表している．

表 1 対象ソフトウェア
Table 1 Target Software Systems.

名前	言語	総行数	名前	言語	総行数
netbeans	Java	14,360	weltab	C	11,460
ant	Java	34,744	cook	C	70,008
jdtcore	Java	147,634	snns	C	93,867
swing	Java	204,037	postgresql	C	201,686

[*4] ツールによって検出されたコードクローンをそのまま抽出する場合もあるが，検出されたコードクローンを一部加工 (周囲のコード片の追加・削除) して抽出する場合もある．

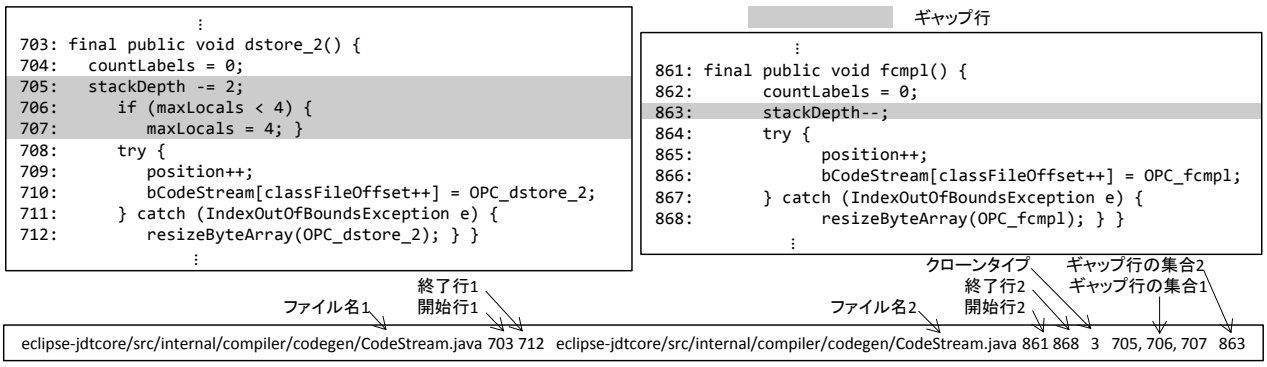


図 3 ギャップの情報を追加した正解クローンの例（正解クローン No. 5771）
Fig. 3 An Example of the Clone Reference the Authors Remade (Reference No. 5771).

$$ok(p_1,p_2) = min(\ max(\ contain(p_1.f_1,p_2.f_1),\ contain(p_2.f_1,p_1.f_1)),\ max(\ contain(p_1.f_2,p_2.f_2),\ contain(p_2.f_2,p_1.f_2))). \quad (5)$$

本論文では Bellon らの実験と同様，ok 値の閾値として 0.7 を用いる．クローン候補の集合を S_{cand}，正解クローンの集合を S_{refs}，S_{cand} と S_{refs} に共通して含まれているコードクローンの集合を S としたとき，再現率（$Recall$），適合率（$Precision$）および F 値（$F\text{-}measure$）は，それぞれ以下の式で表される．

$$Recall = \frac{|S|}{|S_{refs}|}. \quad (6)$$

$$Precision = \frac{|S|}{|S_{cand}|}. \quad (7)$$

$$F\text{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision}. \quad (8)$$

5. 予備実験

予備実験の目的は，Smith-Waterman アルゴリズムで用いる 3 つのパラメータ ($match$, $mismatch$, gap) の適した組み合わせを求めることである．本実験では，以下の範囲のパラメータについて調査を行った．なお，式 (9)，(10)，(11) において $\mathbb{Z}$ は整数の集合を表す．

$$match = \{x \in \mathbb{Z} \mid 1 \leq x \leq 4\}. \quad (9)$$

$$mismatch = \{y \in \mathbb{Z} \mid -4 \leq y \leq -1\}. \quad (10)$$

$$gap = \{z \in \mathbb{Z} \mid -4 \leq z \leq -1\}. \quad (11)$$

予備実験の手順を以下に示す．

手順 1: 64 ($= 4 \times 4 \times 4$) 通りのパラメータ設定のもとで，各対象ソフトウェアについて CDSW を適用し，F 値を求めた．

手順 2: 手順 1 で求めた各対象ソフトウェアにおける F 値に対して中央値を算出し，その中央値が最も高くなる ($match$, $mismatch$, gap) の組を求めた．

予備実験の結果，($match$, $mismatch$, gap) が (2, -2, -1) のときに F 値が最大になることが分かった．したがって，実験 A および実験 B において ($match$, $mismatch$, gap) の値は (2, -2, -1) を用いる．

6. 実験 A

実験 A の目的は，ギャップの情報がコードクローン検出精度の評価に影響を与えるか否かを調べることである．Bellon らの実験においてコードクローン検出ツールの精度比較が行われているが，ツールが正解クローンを検出できたか否かはコードクローンの開始行と終了行の情報のみで判断されている．したがって，本来コードクローンではないギャップの部分も精度評価の対象となってしまっている．そのため，検出精度の評価が正しく行えていない可能性がある．そこで著者らは，Bellon らの正解クローンにギャップの情報を追加することで，正解クローンを作り直した．ギャップの位置情報を追加した正解クローンを用いることで，本当にコードクローンである行の情報を用いて精度評価を行うことができる．作り直した正解クローンは Web 上で公開している [*5]．

図 3 はギャップの情報を追加した正解クローンの例である．左のソースファイルは 705-707 行目がギャップであるのに対し，右のソースファイルは 863 行目がギャップとなっている．ギャップの情報を追加した正解クローンを用いることで，検出結果の F 値の評価をより正しく行えると著者らは考えている．

Bellon らの正解クローンと著者らが作り直した正解クローンを用いて F 値を求めた．図 4 は両方の正解クローンを用いて算出した F 値を示している．すべてのソフトウェアについて，著者らが作り直した正解クローンを用いることで F 値が向上している．最大では 3.8%，最低では 0.43% 向上した．

調査項目 1 について次のように回答する．コードクロー

[*5] http://sdl.ist.osaka-u.ac.jp/~h-murakm/2013_clone_references_with_gaps/

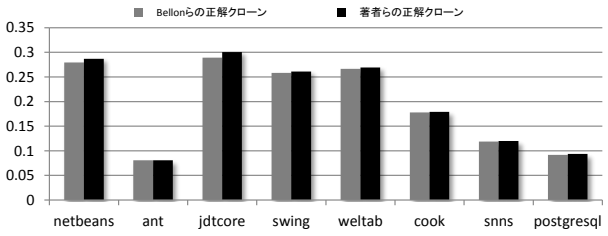


図 4 両方の正解クローンを用いて算出した F 値
Fig. 4 *F-measure* of CDSW Using Both the Clone References.

ンの精度評価において，開始行と終了行だけではなくギャップの位置情報も追加することで，F 値は向上する．

7. 実験 B

実験 B の目的は CDSW が既存ツールに比べて高精度か否かを調べること，また CDSW は高速にコードクローンの検出を行えるか否かを調べることである．本実験では，比較対象として表 2 に示したコードクローン検出ツールを用いる．NiCad と DECKARD を除くツールは Bellon らの実験においても用いられている．すべてのツールはデフォルトの設定を用いてコードクローンを検出している．特に NiCad はブロック単位あるいは関数単位の検出を行うことができるが，本実験ではブロック単位の検出を行った．

3 章で CDSW はコードクローンに含まれるギャップの行番号を出力すると記述したが，本実験において CDSW が出力したギャップの行番号は用いない．その理由は，他のツールはギャップの行番号を出力していないため，公平な比較ができないからである．したがって，コードクローンの開始行と終了行のみを用いて精度の比較を行った．

作り直した正解クローンを用いてすべてのツールの F 値を求めた．図 5 はすべてのコードクローン検出ツールの F 値を示している．図 5 より，CDSW の F 値の中央値は最も高いことが分かる．したがって，CDSW は再現率と適合率がバランスよく高いことを意味している．この精度評価において，ツールが正解クローンを検出できているか否かは ok 値を用いて判断している．したがって，クローン候補が完全に正解クローンに一致していない場合でも，ok 値

表 2 比較対象とするコードクローン検出ツール
Table 2 Clone Detectors Used for Comparison.

開発者	ツール名	検出手法
Baxter	CloneDR	抽象構文木を用いた検出
Merlo	CLAN	関数メトリクスを用いた検出
Kamiya	CCFinder	字句単位の検出
Baker	Dup	字句単位の検出
Rieger	Duploc	行単位の検出
Krinke	Duplix	プログラム依存グラフを用いた検出
Roy	NiCad	LCS アルゴリズムを用いた検出
Jiang	DECKARD	抽象構文木を用いた検出

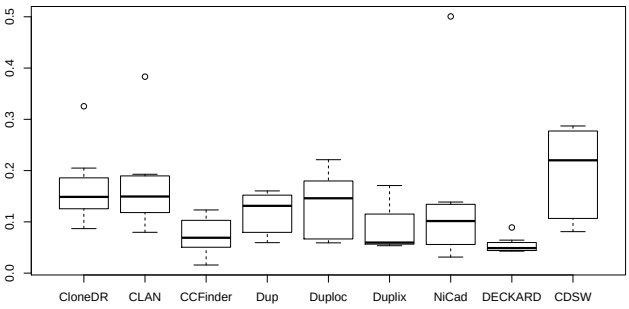


図 5 すべてのコードクローン検出ツールの F 値
Fig. 5 *F-measure* of All the Clone Detectors.

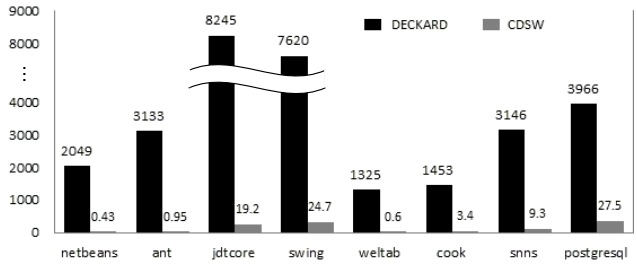


図 6 DECKARD と CDSW の実行時間
Fig. 6 Exection Time of DECKARD and CDSW.

が閾値である 0.7 を超えていれば検出できたとみなされていることに注意されたい．

次に CDSW と DECKARD の実行時間を図 6 に示す．図 6 より，CDSW は数秒から 30 秒以内に検出処理を終えているのに対し，DECKARD は最短でも 20 分，最長では 2 時間 17 分も要していることが分かる．

調査項目 2 について次のように回答する．CDSW は比較対象ツールの中で最も高い F 値を示した．これは，CDSW の再現率と適合率がバランスよく高いことを意味する．

調査項目 3 について次のように回答する．CDSW は大規模ソフトウェアに対して短時間でコードクローンを検出できた．具体的には，既存手法は約 20 万行のソフトウェアに対して約 2 時間で検出処理を終えるのに対し，提案手法は同じソフトウェアに対して 30 秒以内に検出処理を終えることができた．

8. 妥当性への脅威

本実験では，Bellon らが作成した正解クローンを基にギャップの位置情報を追加し，それを用いて CDSW と既存ツールの精度比較を行った．しかし，Bellon らの正解クローンは対象ソフトウェアに含まれるすべてのコードクローンから抽出されたものではない．したがって，対象ソフトウェアに含まれるすべてのコードクローンを対象にして精度の評価を行うと，異なる結果が得られる可能性がある．しかし，対象ソフトウェアに含まれるすべてのコードクローンを対象にすることは現実的ではない．

NiCad はブロック単位や関数単位でコードクローンを検

出する．しかし，Bellon らの正解クローンはそれらより粒度の小さいコードクローンを多く含んでいる．そのため，Bellon らの正解クローンを用いて評価を行った場合，NiCad の検出精度が過度に悪くなる可能性がある．

9. 議論

9.1 Smith-Waterman アルゴリズムと LCS アルゴリズムの比較

Smith-Waterman アルゴリズムと LCS アルゴリズムは共に類似文字列検出アルゴリズムであるが，その最大の違いは Smith-Waterman アルゴリズムは *mismatch* と *gap* のパラメータを用いているのに対し，LCS アルゴリズムはそれらを用いていない点である．したがって，Smith-Waterman アルゴリズムは類似文字列に含まれるギャップの大きさを考慮して検出できる．さらに提案手法は Smith-Waterman アルゴリズムに 2 つの変更を加えている．これらの変更により，提案手法は 2 つの文字列から複数の類似文字列を検出できる．一方の入力配列を $\langle a_1, a_2, \cdots, a_n \rangle$，もう一方の入力配列を $\langle b_1, b_2, \cdots, b_m \rangle$ とすると，単純な Smith-Waterman アルゴリズムは $O(mn)$ の時間計算量および $O(mn)$ の空間計算量を要する．LCS アルゴリズムも同様の時間計算量および空間計算量を要する．

9.2 関連研究

日比らは Smith-Waterman アルゴリズムを用いて剽窃ソースファイルを検出する手法を提案している [7]．日比らの手法と著者らの手法の違いは 2 つある．1 つ目は検出の粒度である．日比らの手法はソースファイル単位の類似度を求めているのに対し，著者らの手法はコード片単位の類似度を求めている．2 つ目は Smith-Waterman アルゴリズムに加える変更点である．著者らの手法はコードクローン検出に特化した変更を加えているのに対し，日比らの手法はコーディングスタイルにも考慮する変更を加えている．

肥後らはギャップを含むコードクローンを検出できないコードクローン検出手法の出力結果に対する後処理を行うことで，ギャップを含むコードクローンの情報を生成する手法を提案している [8]．肥後らの手法は，コードクローン検出ツールの出力結果のみを用いているため，対象ソースファイルを直接解析する必要がないという利点を持つ一方で，ギャップの前後のコード片がコードクローンとして検出されていなければならないという欠点を持つ．それに対して著者らの手法は，ギャップの前後のコード片がコードクローンとして検出されないほど小さい場合でも，それらをまとめてコードクローンとして検出することができる．

10. おわりに

本論文では，Smith-Waterman アルゴリズムを応用してギャップを含むコードクローン検出手法を提案し，提案手法をコードクローン検出ツール CSDW として実装した．著者らは 8 つのオープンソースソフトウェアに対して実験を行い，Smith-Waterman アルゴリズムで用いる 3 つのパラメータ (*match*, *mismatch*, *gap*) の適した値を求めた．さらに著者らは，Bellon らの正解クローンに対して，ギャップの位置情報を加えることで新しく正解クローンを作り直した．作り直した正解クローンを用いて，CSDW と既存のコードクローン検出ツールの精度比較を行った．その結果，CSDW は既存のコードクローン検出ツールに比べて，最も F 値が高いことを明らかにした．

7 章で述べたように，本実験では既存のコードクローン検出ツールとの精度比較の際，CDSW が出力するギャップの位置情報を用いていない．その理由は，既存のコードクローン検出ツールはギャップの位置情報を出力していないため，公平な比較を行えないからである．今後は，ギャップの位置情報も考慮して精度比較実験を行いたい．ギャップの位置情報を用いることで，より正確な実験結果を得られると考えられる．

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 (S)(課題番号 : 25220003)，挑戦的萌芽研究 (課題番号 : 24650011)，および文部科学省科学研究費補助金若手研究 (A)(課題番号 : 24680002)，独立行政法人情報処理推進機構 技術本部　ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けた．

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 D，Vol. 91-D, No. 6, pp. 1465–1481 (2008).

[2] Clone Detection Literature: . `http://www.cis.uab.edu/tairasr/clones/literature/`.

[3] Kim, M., Bergman, L., Lau, T. and Notkin, D.: An Ethnographic Study of Copy and Paste Programming Practices in OOPL, *Proc. of the 2004 International Symposium on Empirical Software Engineering*, pp. 83–92 (2004).

[4] Smith, T. F. and Waterman, M. S.: Identification of common molecular subsequences, *Journal of Molecular Biology*, Vol. 147, No. 1, pp. 195–197 (1981).

[5] Bellon, S., Koschke, R., Antniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Trans. on Software Engineering*, Vol. 31, No. 10, pp. 804–818 (2007).

[6] Detection of Software Clones: . `http://bauhaus-stuttgart.de/clones/`.

[7] 日比健太，雲居玄道，三川健太，後藤正幸：特徴語に注目した Smith-Waterman アルゴリズムに基づく剽窃ソースコードの自動検出手法，電子情報通信学会技術研究報告，Vol. 112, No. 319, pp. 1–6 (2012).

[8] 肥後芳樹，宮崎宏海，楠本真二，井上克郎：グラフマイニングアルゴリズムを用いたギャップを含むコードクローン情報の生成，電子情報通信学会論文誌 D，Vol. 93-D, No. 9, pp. 1727–1735 (2010).