Vol.2013-DPS-155 No.8
Vol.2013-MBL-66 No.8
2013/5/23

プログラム演習における 複雑度を用いたトークンベースの不正コピー検出手法

岩本 舞[1,a)]　小島 俊輔[2,b)]　中嶋 卓雄[3,c)]

概要：大学等におけるプログラミング系科目の演習課題において，他人のソースコードを流用し提出する行為が問題となっている．そのため，教育機関において，このような不正コピーを自動で検出するアルゴリズムが望まれている．従来の研究では，トークン列長を判定基準とする手法が提案されているが，単純な長さをしきい値としており，print 文の羅列のような単純なプログラム，あるいはトークン列がステートメントの途中だった場合に誤検知となる．そこで，本稿ではプログラムの複雑度と完全トークン列を用いた検出手法を提案する．実験の結果，学生が提出した課題プログラムにおいて，複雑度の採用により再現率 R が上昇し，また，完全トークン列の採用により適合率 P が上昇した．

Mai Iwamoto[1,a)]　Shunsuke Oshima[2,b)]　Takuo Nakashima[3,c)]

1. はじめに

近年，大学等のプログラミング系講義において，学生が他の学生によって作成されたソースコードを提出する行為が問題となっている．このような行為は正当な評価を妨げるだけでなく，教員による到達レベルの把握を困難にし，ひいては学生の能力向上を阻害する．本稿では，学生が提出するプログラムをソースコード，学生間のソースコードのコピーを不正コピーと表記する．ソースコード間のコピーを発見する手法には，コードクローン検出技術がある．コードクローンとは，ソースコード中に存在する互いに一致または類似したコード片を指す．コードクローンはバグの温床となることから，これまで多くの発見手法が提案されてきた．しかし既存の研究の多くは主に産業界でのコードクローンを対象としており，学生間の不正コピーの検出にはいくつかの問題があった．そこで本研究は，学生の提出する不正コピーに特化したアルゴリズムの開発を行う．

[1] 熊本高等専門学校 技術・教育支援センター
866–8501 熊本県八代市平山新町 2,627
Technical Practice Center, Kumamoto National College of Technology, 2,627, Hirayama-Shinmachi, Yatsushiro, Kumamoto 866–8501, Japan
[2] 熊本高等専門学校 ICT 活用学習支援センター
ICT Center for Learning Support, Kumamoto National College of Technology
[3] 862–8652 熊本県熊本市東区渡鹿 9-1-1
東海大学 基礎工学部 電気電子情報工学科
Dep. of Electronics Engineering and Computer Science, Tokai University, 9-1-1, Toroku, Kumamoto 862–8652, Japan
[a)] m-iwamoto@kumamoto-nct.ac.jp
[b)] oshima@kumamoto-nct.ac.jp
[c)] taku@ktmail.tokai-u.jp

2. コードクローン検出

本研究で取り扱う不正コピーを検出する手法には，コードクローン検出手法がある．ここでは，コードクローン分野の先行研究について述べ，関連研究における提案手法の位置づけを明らかにする．

2.1 産業界におけるコードクローン検出

従来のコードクローン研究では，主に産業界の大規模なコードクローンを検出する手法が提案されてきた．これらのコードクローン検出手法は，木やグラフを基礎とした構文レベルの手法と，テキストやトークンを比較する字句レベルの手法に分類することができる．文献 [13] では，さまざまな手法によるコードクローン検出方法を比較・検証している．

構文レベルの手法として，文献 [6] では，依存グラフによりコードクローンを同定する．文献 [2] では，抽象構文木を用いたアルゴリズムによりコードクローンを発見する．コードクローンはプログラムのまとまりとして検出されるため，機械的な手法で調整が可能である．

また字句レベルの手法として，文献 [7] は，判定に 21 種

類の関数メトリクスを使用しており，それらを 4 つに分類して比較・検討している．また，文献 [1][5] では，巨大なソフトウェアにおいて，コードクローンを高速に発見することを目的としたツールが開発されている．比較前にユーザ定義名を特殊文字に置き換えるため，定義名異なる場合でもコードクローンとして検出が可能である．行単位での比較には接尾辞木検索アルゴリズムが用いられており，線形時間でコードクローンが検出可能である．文献 [11] では，動的計画法に基づいた手法により，以前のテキスト比較ツールより高精度に 2 つのプログラム間のコードクローンを検出することができる．文献 [12] では，ソフトウェア間の最大クローン長と部分類似度に着目し，どの程度の値であれば流用があると言えるかを実験的に導出している．

さらに，文献 [3] では，文献 [1][2][5][6] など 6 つのコードクローン検出方法を，8 つの C や Java で記述されたプログラムを用いて検証している．その結果，字句レベルの技術 [1][5] はかなり高い再現率となることを示した．

本研究で提案するアルゴリズムは，文献 [5][7] で用いられたトークンベースのコードクローン検出手法に分類される．トークンベースの手法を選択した理由は，不正コピーにおける特徴を検出する機能を有しているためである．しかし，これまでの字句レベル手法の研究は，類似するコード片を発見するものであり，不正コピーか否かについては言及しない．例えば，学生が課題演習で提出するソースコードにおいて，単純なコードは類似する可能性が高く，偶然似たものをコピーと判定してしまう．また，学生は不正コピーを隠すため，ステートメントの入れ替えなどの細工を施す場合があり，従来の手法では不正コピーを見逃す可能性があった．さらに，制御ロジックが変化する行の入れ替えをしただけのソースコードを提出する場合もあり，接尾辞木や抽象構文木を利用した文献 [2][5] においても，正しい判定が困難となる．

2.2 不正コピーの特徴

学生間の不正コピーには，産業界におけるコードクローンとは異なる特徴がある．ここでは，大学等のプログラム演習課題において学生が提出する不正コピーの特徴について説明する．

今回の実験には熊本高等専門学校八代キャンパス 3 年次の学生が課題として提出したソースコードを使用した．これらのソースコードを目視で比較した結果，学生による不正コピーには，大きく分けて以下のような 5 つの特徴があることが分かった．

TypeV　ユーザ定義名，数値定数，文字列等の付け替え
TypeC　コメント行の追加・削除・変更
TypeI　インデントの編集
TypeS　関数・ステートメント行の入れ替え
TypeD　行の入れ替えによりロジックが変更されたコピー

また，学生が提出する演習課題においては，以下のような，コードクローンとして検出されるものの，不正コピーではない特徴もある．

TypeT　教員が雛形ソースコードを提供するタイプの課題
TypeP　単純なステートメントの繰り返し

そこで本稿では，TypeV, TypeC, TypeI, TypeS, TypeD を検出し，TypeT や TypeP の特徴を持つソースコードを不正コピーとしないために，構文レベルおよび字句レベルでのアルゴリズムの改良を行う．これによって再現率や適合率を改善し，F 値の改善を試みる．

3. 従来手法

ここでは，不正コピーを検知するための手法について説明する．検知までの大まかな処理の流れを以下に示す．

Step1　前処理
ソースコードのコメントおよびインデントを削除し，雛形プログラムとの差分を抽出する．差分抽出にはレーベンシュタイン距離 [10] を用いる．さらに抽出した差分について，トークン解析を行う．例えば，図 2 のソースコードはトークン解析によって図 3 のようにトークンに分割される．またこのとき，ユーザ定義名は ID，数値定数は NUM，文字列は STR に抽象化する．

Step2　同一トークン列の取り出し
不正コピーであるか否かを判定したい 2 つのソースコードのトークン列を用意する．ここでは，$\mathbb{P}$ をコピー元，$\mathbb{Q}$ をコピー先のプログラムのトークン列とする．ここで $p_i \in \mathbb{P}$，$q_{j.} \in \mathbb{Q}$ はそれぞれソースコー

```
printf("名前を入力してください\n");
scanf("%s",name);
printf("年齢を入力してください\n");
scanf("%d",&age);
printf("電話番号を入力してください\n");
scanf("%s",phone);
```

図 1　単純なステートメントの繰り返しの例

```
for(i=0;i<10;i++){
    sum += i;
    printf("%d\n",sum);
}
```

図 2　トークン解析前のソースコード

```
1: FOR      10: ;        19: printf
2: (        11: ID:i     20: (
3: ID:i     12: ++       21: STR
4: =        13: )        22: ,
5: NUM      14: {        23: ID:sum
6: ;        15: ID:sum   24: )
7: ID:i     16: +=       25: ;
8: <        17: ID:i     26: }
9: NUM      18: ;
```

図 3　トークン解析後のトークン列

ドの先頭から i 番目，j 番目のトークンを示し，写像関数 τ によって $\mathbb{T}$ の該当するトークンに写像される．$\mathbb{T}$ へ写像した結果が同一となる元の要素が $\mathbb{P}$，$\mathbb{Q}$ の双方に存在するとき，その 2 つの要素を同一トークンと表記する．ただしこの段階では，ユーザ定義名や数値の付け替えである TypeV の不正コピーに対応するため，ユーザ定義名は同じトークンであると見なし，数値についても同様に扱う．本研究では，このような同一トークンが連続したトークン列を同一トークン列と表記する．このとき，

$$\mathbb{P} = \{p_1, p_2, \cdots, p_m\} \tag{1}$$

$$\mathbb{Q} = \{q_1, q_2, \cdots, q_n\} \tag{2}$$

$$\mathbb{T} = \{\ \text{すべてのトークン}\ \} \tag{3}$$

$$\tau : \mathbb{P}, \mathbb{Q} \rightarrow \mathbb{T} \tag{4}$$

とすれば，$\mathbb{P}$ の i 番目，$\mathbb{Q}$ の j 番目のトークンを起点とする同一トークン列 $\mathbb{C}_{ij}$ は次のように定義できる．

$$\mathbb{C}_{ij} = \{q_{j+r} | 0 \leq r < \min\{l | \tau(p_{i+l}) \neq \tau(q_{j+l})\}\} \tag{5}$$

例を図 4 に示す．ここで，図中の黒点はそれぞれのトークン列で同一トークンが使われていることを示す．このとき，黒点が斜めに連なっている部分が同一トークン列である．

Step3　コピートークン集合の作成

Step2 で取り出した各々の同一トークン列について，コピーされたものかどうかを判定する．本研究では，同一トークン列 $\mathbb{C}_{ij}$ のうち，コピーであると判定された同一トークン列をコピートークン列，すべてのコピートークン列の和集合をコピートークン集合 $\mathbb{C}$ と表記する．$\mathbb{C}_{ij}$ がコピートークン列であるか否かの判定は，トークン列長に基づく手法 (Token Length Medthod:TLM) で行う．

すなわち，$|\mathbb{C}_{ij}|$ を $\mathbb{C}_{ij}$ の要素数としたとき，コピートークン集合 $\mathbb{C}$ を次式のように定義する．

$$\mathbb{C} = \bigcup_{|\mathbb{C}_{ij}| \geq \lambda_t} \mathbb{C}_{ij} \tag{6}$$

しきい値 λ_t が大きいほど，誤ってコピートークン列と検出される可能性は小さくなる．

Step4　コピー率による不正コピー判定

Step3 で作成したコピートークン集合 $\mathbb{C}$ が全体のトークン数に占める割合がしきい値 $\lambda_c (0 \leq \lambda_c \leq 1)$ を越えたとき，そのプログラムは不正コピーであったと判定する．すなわち，プログラムのコピー率 $CR(\mathbb{C}, \mathbb{Q})$ を式 (7) のように定義したとき，その値が式 (8) を満たせば，そのプログラムは不正コピーであると判定する．λ_c が 1 に近づくことは，プログラムを不正コピーと見なす条件が厳しくなることを意味する．

$$CR(\mathbb{C}, \mathbb{Q}) = \frac{|\mathbb{C}|}{|\mathbb{Q}|} \tag{7}$$

$$CR(\mathbb{C}, \mathbb{Q}) \geq \lambda_c \tag{8}$$

上記のアルゴリズムにより，TypeV，TypeC，TypeI，TypeS，TypeD といった学生による不正コピーの特徴を検出し，TypeT の特徴を持つソースコードを不正コピーとしない判定が可能である [4]．しかしこの方法では，トークン列長をコピー判定のしきい値としており，図 1 のような誰が書いても同じ書式となる単純なプログラムも，トークン列長がしきい値以上であれば検出され，誤検知が増える要因となっていた．

そこで本稿では，TypeV，TypeC，TypeI，TypeS，TypeD を不正コピーとして検出しつつ，TypeT や TypeP を不正コピーとみなさない検出アルゴリズムを提案する．

4. 提案手法

従来手法には，同じステートメントを連続して記述した単純なプログラムをコピーと見なしてしまう問題があった．そこで本研究では，単純なプログラムを除外する構文レベルの同一トークン列の判定手法を提案する．

また，同一トークン列の検出において，検出される同一トークン列がステートメントの途中で開始または終了する場合があり，誤検知の原因となっていた．そこで同一トークン列から不完全なステートメントを取り除く字句レベルの手法を提案する．

4.1 複雑度を用いた同一トークン列のコピー判定手法

同一トークン列がコピートークン列であるか否かを判定する手法として，従来の研究では，TLM を用いて判定している．しかし，print 文の羅列のような単純なプログラム

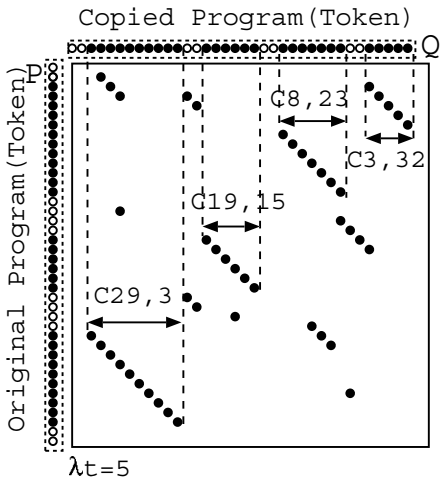

図 4　同一トークン列の検出

は，プログラムは意図的にコピーしなくても似てしまう可能性が高いため，不正コピーと判定されないことが望ましい (TypeP)．しかし，TLM では print 文の連続のような単純なプログラムであっても，同一トークン列の長さがしきい値 λ_t を越えればコピートークン列と判定する．そこで，本研究では，トークン列の複雑度に基づく手法 (Token Complexity Method:TCM) を提案する．TCM では，このような単純な構造のプログラムをコピーと判定しないために，プログラムの複雑度という概念を導入し，このような単純な構造のプログラムをコピーと判定しないようにした．

ソースコードの複雑さについて，文献 [9] では，ルーチンのディシジョンポイントの個数で計測できるとしている．また文献 [8] では，ソースコードの複雑さを実行可能なパスの数としている．これらの文献では，プログラムの難読化を防ぐ目的で複雑さを評価しているため，本研究における複雑度の概念とは意味合いが異なる．そこで本研究では，複雑度を次式のように定義する．

$$\begin{aligned}\mathrm{Complexity}(\mathbb{C}_{ij}) =& \left|\{\tau(q)|q \in \mathbb{C}_{ij}\}\right| \\ &+ K\left|\{q \in \mathbb{C}_{ij}|\tau(q) \in \mathbb{R}\right|\end{aligned} \tag{9}$$

ここで $\mathbb{R}$ は繰り返しトークンの集合であり，C 言語では $\mathbb{R}$ ={for, while} となる．第 1 項は使用されたトークンの種類数，第 2 項は繰り返しトークンの総数である．複雑度の式では，定義名の違うユーザ定義名を別のトークンとして数える．例えば，

```
a=a*10+b;
```

というステートメントには，ID:a，=，*，NUM，+，ID:b，; の 7 種類のトークンがある．

繰り返しトークンは他のトークンに比べ複雑度を増加させる傾向が強いと考えられるため，重み付けの定数 K を 0 より大きな値とする．本研究では予備実験の結果，$K = 2$ がもっとも良好な結果を示したため，以降では $K = 2$ として実験を行う．

TCM では，複雑度のしきい値を λ_t とし，コピートークン集合 $\mathbb{C}$ を次式のように定義する．TCM は，TLM 同様，λ_t が大きいほど，同一トークン列を誤ってコピートークン列とする可能性は小さくなる．

$$\mathbb{C} = \bigcup_{\mathrm{Complexity}(\mathbb{C}_{ij}) \geq \lambda_t} \mathbb{C}_{ij} \tag{10}$$

4.2 完全トークン列手法

編集距離を用いた同一トークン列の発見手法では，図 5 のように，同一トークン列がステートメントの途中から開始または終了する場合がある．しかし，実際にはステートメントが不完全な状態でコピーされるとは考えにくい．そこで，ステートメントの途中から始まる同一トークン列や，ステートメントの途中で終わる同一トークン列を削除し，ステートメントとして成立している部分だけを判定の対象とする完全トークン列手法 (Completely Token Sequence Method:CTSM) を提案する．完全トークン列のみを判定の対象とすることで，不完全なトークン列によって複雑度がしきい値 λ_t を越えることを防ぎ，誤検知を減らす狙いがある．例えば，図 5 に示すような同一トークン列は，CTSM を適用することで図 6 になる．

```
ID:a - NUM ) printf ( STR ) ;
}while ( ID:no != ID:a && ID:b < ID:a ) ;
printf(STR,
```

図 5 同一トークン列の例

$\mathbb{E}$ はステートメントの区切り記号の集合である．今回の実験では，C 言語で記述されたソースコードを対象としたため，$\mathbb{E}$ ={';', '}', ')'} とした．ここで，$\mathbb{K}$ は区切り記号の位置の集合であり，以下のように定義する．

$$\mathbb{K} = \{k|\tau(q_k) \in \mathbb{E} \wedge (q_{k+1} \in \mathbb{C}_{ij} \vee q_k \in \mathbb{C}_{ij})\} \tag{11}$$

このとき，完全トークン列 $\mathbb{C}'_{ij}$ を次式で定義する．

$$\mathbb{C}'_{ij} = \{q_k \in \mathbb{C}_{ij}|\min \mathbb{K} < k \leq \max \mathbb{K}\} \tag{12}$$

このようにして作成した完全同一トークン列 $\mathbb{C}'_{ij}$ について，コピートークン列か否か判定する．

5. 実験手法

5.1 評価基準

コードクローン研究では，一般的に誤検知を用いた評価基準が使用される．ここでは，誤検知である False-Positive(以下 FP)，False-Negative(以下 FN) で実験の評価を行う．

ここで，FP，FN を客観的に評価するための一般的な尺度となる再現率 Recall (以下 R)，適合率 Precision(以下 P)，F-measure(以下 F 値) を導入する．R，P，F 値は，一般に次式で定義される．

$$R = \frac{tp}{tp + fn} \tag{13}$$

$$P = \frac{tp}{tp + fp} \tag{14}$$

$$F\text{ 値} = \frac{1}{\frac{1}{2}(\frac{1}{R} + \frac{1}{P})} \tag{15}$$

ここで，tp，fn，fp はそれぞれ TP，FN，FP の数である．R，P，F 値はそれぞれ 0 以上 1 以下の値をとり，1 に近いほど判定が正確であったことを意味する．そこで，本研究では不正コピー検出の性能評価に F 値を用いる．

```
printf ( STR ) ;
}while ( ID:no != ID:a && ID:b < ID:a ) ;
```

図 6 完全トークン列の例

5.2 実験データ

2012 年 6 月に熊本高等専門学校八代キャンパスの 3 年次のクラスにて実施したプログラム演習において，学生が提出した 119 件のプログラムを実験データとした．演習は，教員が与えた簡単な数字当てゲームの雛型ソースコードを自由に拡張するものであった．そのため，学生が作成したプログラムには雛型ソースコードが多く含まれていたが，その内容はさまざまであった．なお，学生に提示したソースコードは 29 行であり，学生が追加または変更した行数の平均は 30.7 行，ソースコード総行数の平均は 53.7 行であった．今回，全ての組合せ $n \times (n-1)$ 通り ($n = 119$ では 14042 通り) について，目視検査によりコピーが疑われた組み合わせ 27 件を正解データセットとして実験を行った．

6. 実験結果

実験では λ_t をパラメータとして λ_c を 0.1 から 0.9 まで変化させたときの R，P，F 値を測定した．全ての表は λ_t を固定した場合に F 値が最大となる λ_c，R，P を併記した．また図には，λ_t を固定し λ_c を 0.1 から 0.9 まで変化させた場合の R，P，F 値を示した．

6.1 同一トークン列のコピー判定手法

同一トークン列のコピー判定に従来手法である TLM を使用した場合の結果を表 1 および図 7 に，提案手法である TCM を使用した場合の結果を表 2 および図 8 に示す．表 1 より，TLM では，$\lambda_t = 18$，$\lambda_c = 0.69$ で $R = 0.67$，$P = 0.64$ となり，F 値は最大の 0.65 である．これは 27 件の不正コピーのうち 18 件を正確に検出し，10 件を誤検知したことを意味する．表 2 より，TCM では，$\lambda_t = 14$，$\lambda_c = 0.56$ で $R = 0.78$，$P = 0.57$ となり，F 値は最大の

表 1 TLM を使用した場合の F 値の最大値

λ_t	λ_c	$F_{\max}$	R	P
10	0.82	0.60	0.48	0.81
14	0.77	0.64	0.52	0.82
18	0.69	0.65	0.67	0.64
22	0.61	0.56	0.52	0.61
26	0.57	0.64	0.52	0.82
30	0.43	0.51	0.74	0.39

表 2 TCM を使用した場合の F 値の最大値

λ_t	λ_c	$F_{\max}$	R	P
10	0.76	0.56	0.44	0.75
12	0.58	0.54	0.93	0.38
14	0.56	0.66	0.78	0.57
16	0.57	0.62	0.52	0.78
18	0.49	0.50	0.44	0.57
20	0.43	0.52	0.59	0.47

表 3 CTSM と TCM を併用した場合の F 値の最大値

λ_t	λ_c	$F_{\max}$	R	P
10	0.65	0.65	0.74	0.57
12	0.55	0.70	1.00	0.54
14	0.55	0.78	0.74	0.83
16	0.55	0.62	0.48	0.87
18	0.43	0.58	0.59	0.57
20	0.43	0.58	0.52	0.67

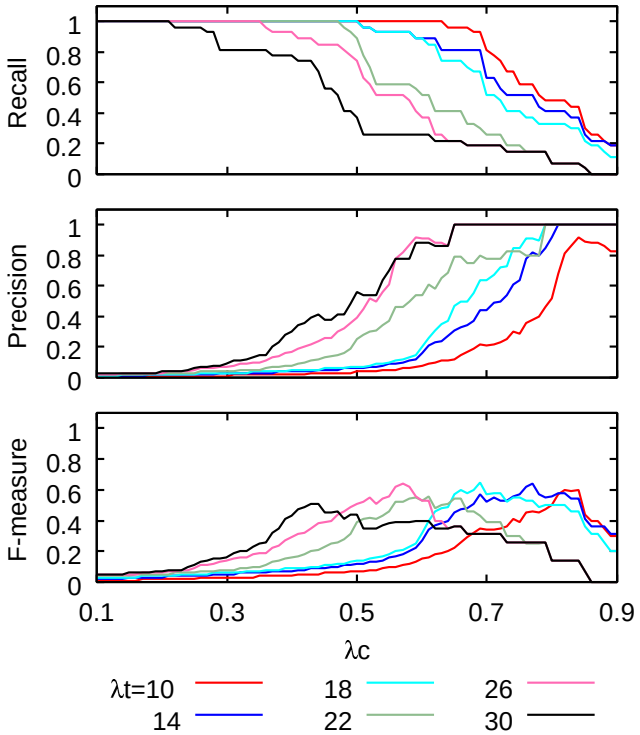

図 7 TLM を使用した場合の R，P，F 値

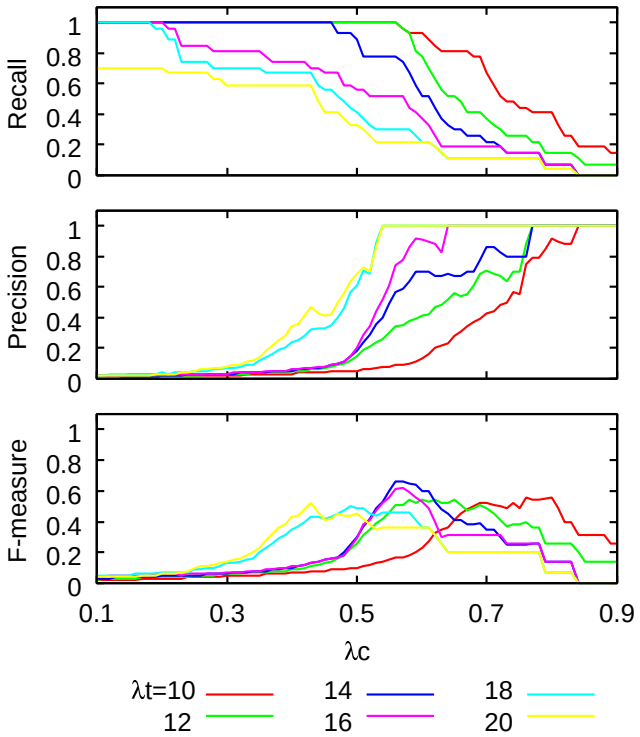

図 8 TCM を使用した場合の R，P，F 値

0.66 である．これは 27 件中 21 件を正確に検出し，16 件を誤検知したことを意味する．図 7，図 8 より，TCM では TLM より F 値が最大になる λ_c が小さくなる．これは単純なプログラムがコピーと誤判定されず，結果として P が上昇し始める λ_c が TLM に比べ小さくなるためである．λ_c が小さくなることで，一部のみをコピーしたソースコードであっても不正コピーとして検出されるようになり，R が上昇した．

6.2 同一トークン列の使用範囲

同一トークン列のコピー判定に提案手法である CTSM と TCM を併用した場合の結果を表 3 および図 9 に示す．表 3 より，$\lambda_t = 14$，$\lambda_c = 0.55$ で $R = 0.74$，$P = 0.83$ となり，F 値は最大の 0.78 である．すなわち，27 件中 20 件の不正コピーを正確に発見し，誤検知は 4 件である．図 8 と図 9 を比べると，R にはあまり変化がないのに対し，λ_c の変化に対する P の上昇率が大きくなっている．ただし F 値が最大値をとる範囲が小さくなり，λ_c の設定が難しくなる．

7. おわりに

学生間の不正コピーは，多くのコードクローン研究が対象とする産業界でのコードクローンとは異なる特徴があり，本研究では，これを検出する手法の開発を行った．従来手法では，学生間の不正コピー検出にコードクローン検出手法を用いていたが，単純なプログラムや不完全な同一トークン列が誤検知の原因となっていた．そこで，本稿では構文レベルにおいて，TCM を用いた同一トークン列のコピー判定により，R の改善を試みた．さらに，字句レベルでは，完全トークン列を用いた CTSM により，P の改善を試みた．

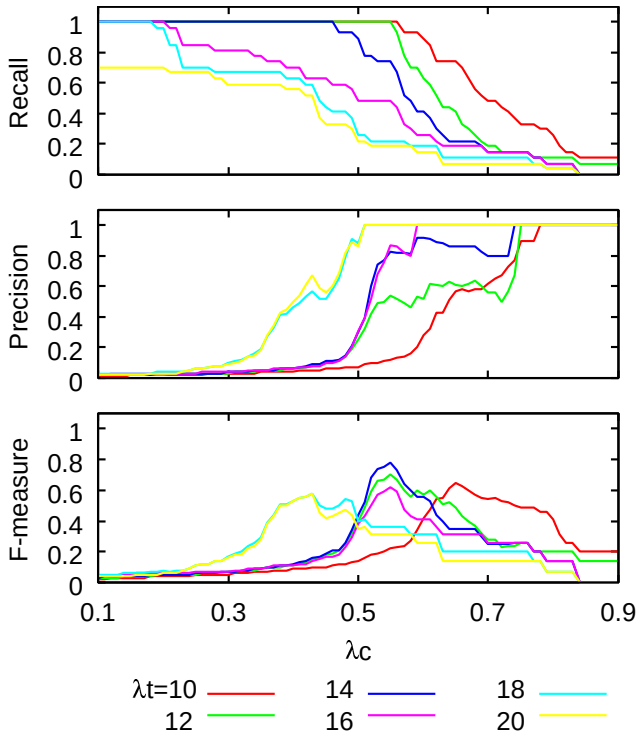

図 9 CTSM と TCM を併用した場合の R，P，F 値

今回提案した手法は，C 言語で記述された演習課題において，TLM を TCM に変更することにより，R が 0.67 から 0.78 に向上した．また CTSM では，P を 0.57 から 0.83 に向上させることができた．CTSM と TCM の併用により，R と P の双方が向上し，F 値が 0.65 から 0.78 に改善した．以上より，本研究で開発したアルゴリズムは，学生間の不正コピーの特徴検出に有用である．

参考文献

[1] Baker, B. S.: On finding duplication and near-duplication in large software system, *WCRE*, IEEE Computer Society, pp. 86–95 (1995).
[2] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S. and Bier, L.: Clone Detection Using Abstract Syntax Trees (1998).
[3] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of Clone Detection Tools, *IEEE Transactions on Software Engineering*, Vol. 33, pp. 577–591 (2007).
[4] Iwamoto, M., Oshima, S. and Nakashima, T.: Token-based Code Clone Detection Technique in a Student's Programming Exercise, *BWCCA*, IEEE, pp. 650–655 (2012).
[5] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, Vol. 28, pp. 654–670 (2002).
[6] Krinke, J.: Identifying Similar Code with Program Dependence Graphs, *8th Working Conference On Reverse Engineering*, Stuttgart, Germany, IEEE, pp. 301–309 (online), available from ⟨http://www.dcs.kcl.ac.uk/staff/krinke/publications.php⟩ (2001).
[7] Mayrand, J., Leblanc, C. and Merlo, E. M.: Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics, *ICSM*, IEEE Computer Society, pp. 244–253 (1996).
[8] McClure, C.: *The three Rs of software automation: re-engineering, repository, reusability*, Prentice Hall (1992).
[9] McCONNELL, S.: *CODE COMPLETE*, Microsoft Press (1993).
[10] Navarro, G.: A guided tour to approximate string matching, *ACM Computer Surveys(CSUR)*, Vol. 33, No. 1, pp. 31–88 (2001).
[11] Yang, W.: Identifying Syntactic Differences Between Two Programs, *Software - Practice and Experience*, Vol. 21, pp. 739–755 (1991).
[12] 岡原 聖，真鍋雄貴，山内寛己，門田暁人，松本健一：ソースコード流用のコードクローンメトリクスに基づく検出手法 (ソフトウェア解析)，電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学， Vol. 109, No. 307, pp. 73–78 (2009).
[13] 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. J91-D, No. 6, pp. 1465–1481 (2008).