開発者のIDE操作履歴に基づくソースコード著者推定

大森 隆行[1,a)]　桑原 寛明[2]　西垣 正勝[1]

概要：ソースコードの著者推定は知的財産権保護，剽窃や盗用の検出・防止等の観点から重要であるとされている．これまで，ソースコード自体やバイナリコード等を用いて著者推定を行う手法が提案されてきたが，コードの執筆過程に着目した研究はあまり行われていない．本論文では，統合開発環境上で記録された操作履歴がソースコードの著者推定に有用であることを示す．実験の結果，操作履歴のうちメニュー操作列のみを利用することで，高精度な著者推定が可能であることが確認された．

Source Code Author Estimation based on Developers' IDE Operation Histories

1. はじめに

ソースコードの著者推定手法は，ソースコード盗用の検出等の知的財産権の侵害の問題，プログラミング課題における剽窃の検出や抑止の問題，マルウェア等の著者を明らかにするソフトウェア・フォレンジクスの問題といった文脈で非常に重要であるとされている [1,2].

既存の手法の多くは，ソースコードから得られる各種のメトリクス値等を分析し，「その著者らしさ」（各著者のコードに表れやすい特徴）を明らかにすることでコードの真著者を推定する．近年の機械学習を用いた手法を使用すれば，高い精度で著者を推定することも可能となりつつある．しかしながら，高精度を得られる既存の手法には，多くのソースコードが必要であったり，学習済みモデル構築のための手間がかかる等の問題がある．

本研究では，著者らがこれまでに行ってきた統合開発環境 (IDE) の操作履歴に関する研究をソースコード著者識別に応用し，簡便な手法で高精度にコードの著者を推定できることを示す．

ここで，本研究における著者推定では，以下の条件を前提としている．

(1) 少なくとも真著者は IDE で操作履歴を記録している．
操作履歴の記録はソフトウェア開発において必須ではないため，悪意のある偽著者が操作履歴を記録し，なおかつその操作履歴を正直に (改竄せずに) 提出することは期待できない．コード盗用が発生したときに，真著者が自らの操作履歴を使ってコードの真著者が自分であると示せるようにすることで，真著者の権利を守ることを目指している．そのために，真著者が持つ履歴がその著者らしさを十分に含んでおり，他の開発者の履歴と何らかの方法で弁別可能であることが前提となる．後述の通り，本研究では同一開発者の履歴は高い確率で類似していることを確認した．

(2) 操作履歴とソースコードの紐付けは可能．
本研究で扱う操作履歴には，ソースコード執筆時の編集履歴がすべて含まれているため，コード執筆時に記録された操作履歴に含まれる編集操作を適用した結果得られるコードは，完成したソースコードと一致する．

(3) ソースコードの著者は明らかではない．
ソースコード中にコメントにより著者を明示することは一般的に行われており，また，版管理システムを使用する場合，コミットしたユーザの情報も記録される．しかし，これらは容易に偽造でき，コードをコピーせずに自ら書いた証明ともなり得ないため，真著者を示す確かな情報とは言えない．

偽著者が盗用対象のソースコードを得られた場合，履歴の適用結果がそのコードと完全に一致するように操作履歴を作成することは不可能ではない．ゆえに，上記 (2) の条

[1] 静岡大学
Shizuoka University, Hamamatsu, Shizuoka, 432–8011, Japan
[2] 南山大学
Nanzan University, Nagoya, Aichi, 466–8673, Japan
[a)] tomori@inf.shizuoka.ac.jp

件下でも，あるコードに真の操作履歴と偽の操作履歴が紐付けされる恐れがある．よって，真著者が持つ操作履歴が本当にその著者らしさを示すかどうかを確かめることが有効となる．

本研究では，Eclipse 上で行われた操作の履歴を記録するツール OperationRecorder [3] により記録された履歴を用いる．この履歴には，ソースコード編集に関する情報 (文字列の挿入・削除) の他，ファイル操作に関する情報 (開く・閉じる・セーブ等)，メニュー操作 (以下 MO(menu operation)) の使用が含まれている．ここでの MO とは，Eclipse においてコマンド (拡張ポイント `org.eclipse.ui.commands` を拡張して実装されるもの) を起動する操作を意味する．ツールバーやショートカットキーを使用してコマンドを呼び出した場合もメニュー項目から呼び出した場合と同様に記録される．コピー・ペーストのように文字列の変更を伴うコマンドの場合，コマンドの起動を示す MO に続いて，文字列変更内容を示す操作が別途記録される．また，本論文における MO の種類とは，各コマンドが持つ ID を意味する．例えば，ファイルセーブ操作の ID は `org.eclipse.ui.file.save` となる．

最新の OperationRecorder は，履歴ファイルの内容から生成されるハッシュ値を同ファイル内に付与するため，書き出された履歴を後から改変した場合の検知が可能である．ただし，履歴ファイル内のユーザ名や時刻は OS 上での設定変更により容易に偽装が可能であることに注意する必要がある．

本研究では，操作履歴のうち MO のみを使用して真著者を推定することができるかどうかを調査した．結果として，MO 履歴 (MO のみから成る履歴) から，一定以上の長さの履歴を使用すること，適切な変換関数を適用することにより，高い精度で真著者を推定できることを確認した．

本論文の貢献を以下に示す．

- 開発時の IDE 操作履歴に含まれる MO 履歴から，真著者を高い精度で推定できることを示す．
- 高い精度を得るための履歴断片長や変換関数の設定に関する実験結果を示す．
- MO 履歴比較の性質を調べるための試行や考察を述べる．

本論文の構成を示す．2 章では関連研究についてまとめる．3 章では，本研究で用いたデータセットについて述べる．4 章では著者推定の方法と実験の結果について述べる．5 章では 4 章の結果を受けて行った追加実験について述べる．6 章では考察を述べる．7 章ではまとめと今後の課題を述べる．

2. 関連研究

本章では，先行研究の概要およびそれらと本研究との違いについて述べる．

2.1 著者推定と盗用検出に関する研究

これまでに，ソースコードの著者推定やソースコード盗用に関する研究は数多く行われている．He ら [1] は自然言語文書も含めた著者推定手法の分類法を考案した． これによると，著者推定手法はテキストベースとコードベースの手法に分類される．テキストベースの手法は，語彙的，構文的，意味的，N-gram，内容特化，適用特化に分類される．コードベースの手法は，スタイロメトリック，グラフベース，N-gram，振る舞いに関する手法に分類される．例えば，スタイロメトリックモデルには，フィンガープリント等，各著者のコーディングスタイルに着目した手法が含まれる．Kalgutkar ら [2] はセキュリティに着目して著者推定手法の文献調査を行っており，既存の手法を語彙的，構文的，意味的，振る舞い，アプリケーション依存に分類している．これらの論文では，ソースコードやバイナリコードといった成果物，プログラムの振る舞いに基づく著者推定手法を紹介しているが，それらの作成過程に着目した研究には言及していない．

ソースコード著者推定の手法としては，ソースコードメトリクスに着目した研究が数多く行われてきた．[4] では，推定のために使用できるメトリクスの集合を特定している．[5] では，メトリクス集合に基づく著者推定をソフトウェア・フォレンジクスに応用している．[6] や [7] では，バイトレベルやトークンレベルの N-gram を用いた著者推定を行っている．これらの手法では，適切な N の値を設定することで推定の精度を高められる可能性があるが，スケーラビリティの問題も指摘されている [8]．

近年ではニューラルネットワークや大規模言語モデルを用いた著者推定手法が提案されている [9–13]．これらの手法により，高精度な著者推定も可能となっているが，ニューラルネットワークの構築や学習が必要であったり，既存のモデルに依存するという制約がある．これらの手法と比べると，本研究の著者推定手法は MO 履歴から生成したベクトルの類似度計算のみで著者推定が可能であり，簡便であるといえる．

2.2 操作履歴を使用した剽窃検出に関する研究

2.1 節で述べたように，多くの既存手法ではソースコード自体を中心とした成果物を利用して著者推定を行っている．剽窃検出を目的とした手法においても，ソースコードを用いる手法が中心である．例えば，[14] や [15] では，ソースコード (コーディングスタイル) から得られるメトリクスをプログラミング課題の剽窃検出に応用している．一方で，以下に挙げる 3 つの剽窃検出手法では，開発者のソースコード記述時の操作履歴を利用している．

Ljubovic ら [16] は，リポジトリにコミットされるコードだけでは編集過程の解像度が不十分であるとして，自動的に編集中のコードがリポジトリにコミットされる開発環境

を構築し，キーストロークレベルまで編集過程を再現可能としている．そのうえで，ソースコードから得られる特徴 (コード行数，数値の数，各演算子の数等) [17] と編集履歴から得られる情報 (追加行数，ペースト行数，コンパイル回数，作業時間等) を入力としてニューラルネットワークにより学習を行い，著者を推定する手法を提案している．

Meier らの手法 [18] ではプログラミング学習者の開発環境上で行われたアクションのログを記録し，実行回数やペースト率等，種々の情報から剽窃を自動的に判定したり，教員が記録された情報を確認可能な環境を構築している．

Schneider らの手法 [19] では，Eclipse 上の操作を記録するツール Fluorite [20] が出力する操作履歴を使用して剽窃検出を行っている．プログラミング課題提出時に操作履歴も同時に提出することを想定しており，剽窃者が元のログを流用，一部改変して提出する場合や，IDE 操作をシミュレートして捏造するケース等を想定している．前者の場合は，流用元の履歴と極端に類似することが，後者の場合は，他の履歴と極端に異なることが想定される．このことから，外れ値 (outlier) となる履歴を剽窃候補として特定する．

上記の通り，操作履歴を使用する先行研究では，いずれもプログラミング学習における剽窃検出を目的としており，剽窃検出以外の文脈に適用できない場合がある．例えば，[18] で設定されている閾値は剽窃検出を前提としたものである．また，本研究のような一般に広く使用される Eclipse とは異なり，[16] や [18] は専用の学習用環境を利用している．

[19] では，履歴の類似性を判定する際に，本研究と同様に MO(コマンド) に着目しており，かつ類似度 (相関係数) 計算の際に MO ヒストグラムに Box-Cox 変換を適用する点も本研究と類似している．しかしながら，[19] は剽窃検出を目的としているため，同じ開発者の操作履歴同士が類似しているかについて議論していない．対して，本研究では，同じ開発者の履歴断片は適切な長さと変換関数適用という条件のもとで，非常に高い確率で類似することを明らかにしている．さらに，本研究では高精度な著者推定のための様々な工夫について，実験の結果と考察を述べる．

ここで説明してきた通り，先行研究では，多くがソースコード等の成果物から様々な情報を取得することで著者推定を行っている．対して，本研究では，履歴とコードの紐付けの際に編集操作の情報を使用するものの，その後の著者推定の過程では MO のみを使用して高精度な著者推定を実現している．このことから，先行研究で用いられていたソースコード由来の特徴量等と比べて，操作履歴中の MO の情報が特に高い本人性を持つ可能性が高いと考えられる．ここでの本人性とは，ソースコードが実際にその著者によって書かれたものであることを示せる性質を意味する．コードの情報に頼らずとも著者推定が可能であることを示したことも本研究の特徴である．

3. データセット

公開された操作履歴のデータセットは存在しないため，本実験では著者らがこれまでの研究で収集した操作履歴データを用いた．本実験で用いた操作履歴データセットに含まれる開発プロジェクト (PJ) の情報を表 1 に示す．「MO 数」は履歴に含まれる MO の数，「時間」は開発全体に要した時間（1 時間以上の無操作時間を除く）を示す．すべてのプロジェクトが Eclipse を使用して行われた Java プログラムの開発であり，それぞれ開発者 1 人で行われたものである．データセット全体に含まれるプロジェクト数は，開発者 1 のものが 9 個，開発者 2 は 6 個，開発者 3〜10 は各 1 個となっている．開発者 1 と 2 はソフトウェア工学分野を専門とする大学教員，開発者 3〜10 は著者らの所属大学の学部生・院生である．データセット全体では 300 種類の MO が含まれていた．

表 1 操作履歴データ

開発者	PJ	MO 数	時間	開発者	PJ	MO 数	時間
1	A	8867	33	2	M	40597	130
1	B	7459	41	2	N	11817	37
1	C	14294	37	2	O	19005	29
1	D	187030	1315	3	P	9166	53
1	E	31434	208	4	Q	11678	73
1	F	16349	119	5	R	9317	79
1	G	57942	596	6	S	3878	48
1	H	7284	19	7	T	4658	92
1	I	4068	64	8	U	3080	54
2	J	5436	21	9	V	3438	69
2	K	54683	161	10	W	4142	45
2	L	39214	111				

※ H と R(パズルゲーム)，Q と T(ロールプレイングゲーム)，U と V と W(シミュレータ) は同一の要求仕様に基づいて開発された．
※開発時間の単位は「時 (hour)」である．

図 1 に各 MO の出現数を棒グラフで示す．出現数は第 1 縦軸に示す通り対数表示となっている．図中の赤線は出現数の累積値であり，第 2 縦軸に対応している．横軸は MO の種類に対応している．MO のうち出現数が多い 10 種を表 2 に示す．最も出現数が多い MO はファイルセーブ操作であり，66850 回行われている一方で，10 回以下の MO が半数以上を占めており，少数の MO に出現が偏っていることがわかる．

4. 著者推定実験

本章では，MO 履歴が著者推定に有用であるかを調査する実験について説明する．

4.1 実験手順

図 2 に，本実験における類似度計算の手順を示す．以下，

表 2　データセット中の MO 上位 10 種

MO ラベル	出現数
org.eclipse.ui.file.save	66850
org.eclipse.debug.ui.commands.StepOver	63364
org.eclipse.ui.edit.text.goto.lineEnd	40776
org.eclipse.ui.edit.text.deletePrevious	35037
org.eclipse.ui.edit.text.contentAssist.proposals	30580
org.eclipse.ui.edit.text.goto.lineDown	25067
org.eclipse.ui.edit.delete	22974
org.eclipse.ui.edit.paste	19735
org.eclipse.ui.edit.text.goto.lineUp	17753
org.eclipse.ui.edit.text.delete.line	15985

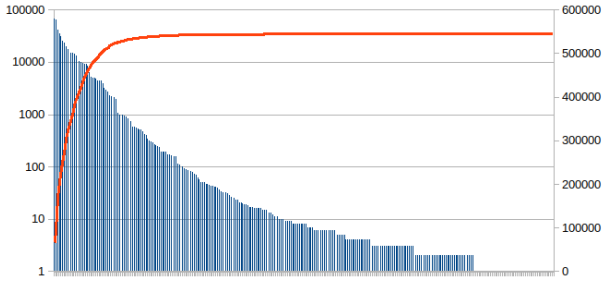

図 1　MO 種類ごとの出現数

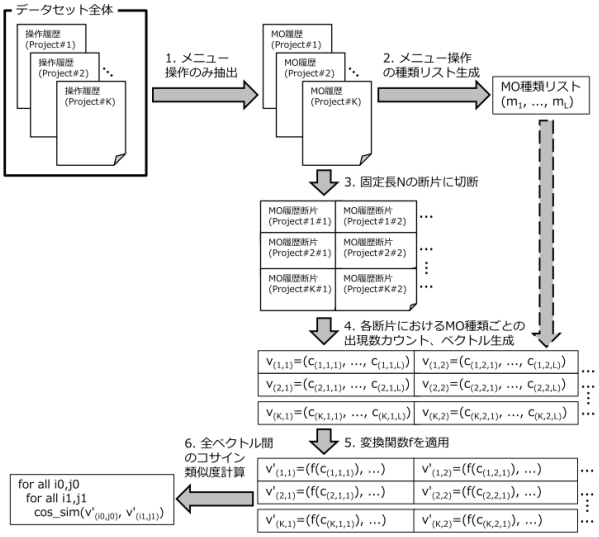

図 2　類似度計算の手順

各手順について説明する．

(1) データセット全体に含まれる各プロジェクトの操作履歴から MO のみを抽出する．

(2) 全プロジェクトの MO の情報から，出現する MO の種類のリストを生成する (ここで，L 種類の MO が存在するとする)．

(3) (1) で得られた各プロジェクトの MO 履歴をある一定の長さ N ごとの断片に切り分ける．履歴末尾の端数分は今後の処理では使用しない．

(4) (3) で得られた各断片に含まれる各 MO の数を数え，L 次元のベクトルを生成する．

(5) (4) で生成したベクトルを変換関数 f により変換する．変換関数はベクトルを構成する各元に対して適用する．

(6) 変換後のベクトル同士のコサイン類似度を計算する．a_i と b_i をそれぞれベクトル $\mathbf{a}$，$\mathbf{b}$ の元とするとき，コサイン類似度は以下のように計算される．

$$sim(\mathbf{a},\mathbf{b}) = \frac{\sum_{i=1}^{L} a_i b_i}{\sqrt{\sum_{i=1}^{L} a_i^2}\sqrt{\sum_{i=1}^{L} b_i^2}}$$

コサイン類似度算出後，次のようにして著者推定の精度を計算する．比較基準となる断片 1 つを選択し，その基準断片と同著者の断片を比較した場合の最小類似度 $sameMin$ と異著者の断片を比較した場合の最大類似度 $diffMax$ を得る．それぞれの比較におけるコサイン類似度を順番に見ていき，(i) 同一断片の比較の場合は無視する．比較対象のどちらかが，全体で断片が 1 つしか存在しない著者の断片の場合も無視する[*1]．(ii) 同著者断片の比較の場合，類似度が $diffMax$ を上回る場合，推定成功とみなす．(iii) 異著者断片の比較の場合，類似度が $sameMin$ を下回る場合，推定成功とみなす．

すべての断片についてこれを繰り返し，推定成功件数を比較件数で割った値が「精度」となる．つまり，基準断片と同著者の断片の比較によって得られたコサイン類似度が，必ずその基準断片と異著者の断片比較によって得られたコサイン類似度を上回る場合，精度 1.0 となる．

上記の手順では，履歴断片長 N と変換関数 f が可変となっている．本研究では様々な N と f の組み合わせを試し，精度がどのように変化するかを調査した．

本研究で試した N は 1000, 2000, ..., 10000 の 10 通りである．ここで，N を変化させると，利用できる断片数が変化することに注意されたい．表 1 に示した通り，プロジェクトによって MO 数は大きく異なる．N の値が大きい場合，使用されないプロジェクトが発生する．例えば，N が 1000 のときは全 23 個のプロジェクト (全 10 名の著者による) が使用され，比較対象となる履歴断片・ベクトルの数は 545 個であるが，N が 10000 になるとプロジェクトは 11 個 (著者は 1,2,4 の 3 名)，履歴断片・ベクトルの数は 43 個となる．全 10 名の著者の履歴が使用されるのは N が 3000 以下のときである．

また，本研究では次の 5 種類の変換関数 f を検討した．

(1) **Raw**: プロジェクト i の j 番目の断片の k 種類目の MO 数 $c_{(i,j,k)}$(以下同様に表記) をそのままベクトルの元として使用する．

(2) **Power**: 引数 p を取る．$c_{(i,j,k)}{}^p$ を元とする．p を 1 未満とすることで，出現数の少ない MO の影響を残しつつ出現数の多い MO の影響を低減することを狙っている．

*1　全体で断片が 1 つしか存在しない著者の断片も，$diffMax$ の計算には影響を与える．

(3) **Log**: 指数関数的に増加するデータの分布を改善するために一般的に用いられる対数変換を適用する．$\log(c_{(i,j,k)}+1)$ を元とする[*2]．

(4) **Boxcox**: データを正規分布に近付けるために用いられる Box-Cox 変換 [21] を適用する．以下の数式により元を計算する．ただし，Box-Cox 変換は入力値として 0 を扱えないため，$c_{(i,j,k)}$ に 1 を加えている．Box-Cox 変換では最尤法により正規性が最も高くなるように λ を決定する． 本実験では SciPy[*3]により λ を求めた．

$$\begin{cases} \dfrac{(c_{(i,j,k)}+1)^{\lambda}-1}{\lambda} & (\lambda \neq 0) \\ \log(c_{(i,j,k)}+1) & (\lambda = 0) \end{cases}$$

(5) **PBoxcox**: Box-Cox 変換における λ を一定の範囲で可変とし，本研究の著者識別の精度を最も高くする λ を求めることとした疑似的な Box-Cox 変換を適用する．本実験では，λ を-1.0〜0.4 の範囲[*4]で 0.1 ずつ変化させている．

ここで，Raw は Power(1.00)[*5]と同じである．また，Log は PBoxcox(0.0) と同じである．Boxcox における λ 値は断片長 1000〜10000 のとき，それぞれ-3.64, -2.69, -2.26, -2.01, -1.83, -1.71, -1.62, -1.53, -1.46, -1.39 となっている．後述の PBoxcox の結果を見ると，これらの λ 値は最高の精度を得られる範囲から大きくずれていることがわかる．

4.2 実験結果

図 3 に断片長 N と変換関数 f およびその引数を変化させた場合の著者推定精度を示す．上段は精度，下段は Min-Max 値 (「同著者比較におけるコサイン類似度の最小値」から「異著者比較におけるコサイン類似度の最大値」を引いた値) を示している．ここで，Min-Max 値はデータセット全体からコサイン類似度の最小値，最大値を求めて計算する．一方で，精度計算における $diffMax$，$sameMin$ は断片ごとに計算していることに注意されたい．精度が 1.0 であっても，Min-Max 値が負数となる可能性がある．

図中では値の大小比較を行いやすいよう，精度については 1.00 を緑，0.99 を黄，0.70 を赤，Min-max 値については最大値を緑，0 を黄，最小値を赤とした 3 色スケールで色付けしている (以降の図も同様)．Power の引数はここでは-0.05〜1.00 の範囲で 0.05 ずつ変化させているが，精度が低下する 0.70 以上は Raw と同じ結果となる 1.00 を除き省略している．また，Log は PBoxcox(0.0) と同じであり，Boxcox は PBoxcox に-1.0 より小さな引数を与えた結果と同じになることに注意されたい．

全体的な傾向としては，変換関数の種類を問わず，断片長 N が大きいほど精度は高くなる傾向にある．Power や PBoxcox の引数は大きすぎても小さすぎても精度が低下する傾向にある．Power(1.00) と同一となる Raw は他の変換関数と比較して精度が大きく劣る．Power と PBoxcox を比較すると，全体的に PBoxcox の方が高い精度を得られることがわかる．精度 1.0 を得るために必要な最小断片長は，Power の場合 5000 ($p = [0.20, 0.30]$ のとき) であるのに対し，PBoxcox では 4000 ($\lambda = 0.1$ のとき) となった．

下段の Min-Max 値は，0 以上のとき，データセット全体で同著者かどうかを判定するための類似度の閾値を定められることを意味している．つまり，Min-Max 値が負のときは，どのような閾値を設定しても，コサイン類似度のみから同著者かどうかを判定できない (判定を誤る事例が存在する) ことを意味する．

精度が 1.0 でなければこの閾値について議論する意味がないため，図 3 では精度 1.0 の場合のみ Min-Max 値を提示している．非負の Min-Max 値を得るために必要な最小断片長は，Power，PBoxcox ともに 6000 であった．Power，PBoxcox ともに，より小さな N で高精度を得るための引数よりも，Min-Max 値を最大とする引数はより大きな値になる傾向があることがわかった．

なお，開発者が 1 時間あたりに行う MO の平均は，表 1 より約 162 回である．MO 数 4000 の履歴断片は約 24.7 時間，MO 数 6000 の履歴断片は約 37.0 時間の開発に相当する．

Q MO 履歴は著者推定に有用か？
A 一定以上の MO 履歴断片長と適切な変換関数適用という条件下で，MO 履歴から生成したベクトルのコサイン類似度を計算することで，高精度な著者推定が可能である．変換関数として PBoxcox を選択した場合が最も高精度となった．

5. 追加実験

ここでは，MO に基づく著者推定手法が持つ特徴を調べるために行った追加実験について説明する．

5.1 履歴断片乱択による λ 値変動調査

4 章の実験で，PBoxcox に最適な引数 λ を与えることで高精度に著者推定が可能であることがわかった．この λ 値にどの程度普遍性があるか調べるために，ここではデータセットの一部のみを使用すると最適な λ 値がどの程度変動するかを確かめた．最適な λ 値とは，最も高い精度となる場合の λ 値である．複数の λ 値で精度が 1.0 となる場合，Min-Max 値が最も大きくなるものを採用した．精度が 1.0 でなく同一の場合は，より小さい λ 値を優先して採用した．

[*2] 対数の底はコサイン類似度の計算に影響を与えない．
[*3] https://scipy.org/
[*4] この範囲は実験を繰り返すことで経験的に得たものである．
[*5] 本論文では，関数名の後に丸括弧付きで実引数を付記することで，変換関数およびその引数を表現する．

・精度

変換関数/	Power														PBoxcox																
N \ param	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	1.00	-1.0	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
1000	0.97112	0.97454	0.97441	0.97104	0.96387	0.95215	0.93671	0.91798	0.89563	0.87240	0.85072	0.82990	0.81007	0.70490	0.97956	0.97971	0.97956	0.97903	0.97833	0.97719	0.97541	0.97287	0.96904	0.96365	0.95625	0.94581	0.93031	0.90668	0.87675
2000	0.98466	0.99139	0.99491	0.99592	0.99591	0.99487	0.99171	0.98675	0.97783	0.96585	0.95076	0.93318	0.91352	0.79290	0.99498	0.99554	0.99608	0.99661	0.99702	0.99731	0.99760	0.99781	0.99798	0.99794	0.99781	0.99697	0.99438	0.98811	0.97549
3000	0.99440	0.99666	0.99812	0.99861	0.99871	0.99781	0.99641	0.99360	0.98817	0.97870	0.96505	0.95106	0.93303	0.81939	0.99847	0.99882	0.99892	0.99934	0.99962	0.99972	0.99965	0.99972	0.99976	0.99976	0.99955	0.99927	0.99788	0.99530	0.98771
4000	0.99056	0.99633	0.99869	0.99941	0.99954	0.99941	0.99858	0.99639	0.99364	0.98866	0.98282	0.97286	0.96217	0.86146	0.99810	0.99882	0.99928	0.99928	0.99934	0.99934	0.99961	0.99980	0.99987	0.99987	0.99987	1.00000	0.99974	0.99757	0.99338
5000	0.99104	0.99698	0.99933	1.00000	1.00000	1.00000	0.99966	0.99922	0.99866	0.99832	0.99742	0.99536	0.99295	0.96056	0.99854	0.99933	0.99955	0.99978	0.99978	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99966	0.99866
6000	0.99915	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99985	0.99963	0.99761	0.99573	0.99402	0.96429	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99985
7000	0.99976	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99976	0.99856	0.99615	0.99471	0.95601	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
8000	0.99966	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99933	0.99832	0.99731	0.96465	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
9000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99956	0.99867	0.96897	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
10000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99942	0.98722	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

・Min-Max値

変換関数/	Power														PBoxcox																
N \ param	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	0.55	0.60	0.65	1.00	-1.0	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
1000																													
2000																													
3000																													
4000																										-0.04843			
5000				-0.06364	-0.04775	-0.05909														-0.09922	-0.07841	-0.06593	-0.04986	-0.04029	-0.04302	-0.05679	-0.07726		
6000		-0.15458	-0.10900	-0.06097	-0.01600	0.01153	-0.03152	-0.09459							-0.18070	-0.17253	-0.16236	-0.14971	-0.13400	-0.11630	-0.09565	-0.07843	-0.06079	-0.03802	-0.01007	0.02192	-0.00126	-0.05396	
7000		-0.12329	-0.08032	-0.03228	0.00634	0.01938	-0.01015	-0.05930	-0.11052						-0.14391	-0.13812	-0.13095	-0.12202	-0.11085	-0.09685	-0.07936	-0.05920	-0.04167	-0.02041	0.00637	0.03768	0.03724	-0.02333	-0.10027
8000		-0.13935	-0.09598	-0.04808	-0.00037	0.03295	0.00929	-0.03552	-0.07990	-0.12333					-0.16185	-0.15598	-0.14629	-0.13954	-0.12930	-0.11623	-0.09958	-0.07855	-0.05247	-0.02342	0.00318	0.03485	0.05501	-0.00554	-0.07336
9000	-0.13803	-0.10355	-0.05967	-0.02461	0.01091	0.04584	0.05914	0.02780	-0.00738	-0.04184	-0.07470				-0.12428?	-0.11828	-0.11089	-0.10176	-0.09043	-0.07637	-0.06719	-0.05510	-0.03880	-0.01727	0.01002	0.04240	0.06360	0.04228	-0.00546
10000	-0.16309	-0.11771	-0.06845	-0.02770	0.00955	0.04637	0.08010	0.08538	0.06711	0.04490	0.01967	-0.00767			-0.14172?	-0.13412	-0.12491	-0.11373	-0.10015	-0.08679	-0.07757	-0.06504	-0.04809	-0.02558	0.00317	0.03768	0.07559	0.11008	0.07954

図 3　断片長と変換関数を変化させたときの著者推定精度

ここでは，断片長 N は 3000 に固定している．

乱択条件 1 類似度計算手順 (3) で MO 履歴を断片に切断した後，各プロジェクトの断片のうち半分をランダムに選択し (ただし，各プロジェクトから最低 1 個は必ず選ぶ)，全断片の比較を 100 回繰り返した．結果として，最適な λ 値の平均 μ は-0.154，標準偏差 σ は 0.186 となった．なお，$\mu \pm 2\sigma$ 範囲内の平均精度は 0.99779[*6]以上であった．

乱択条件 2 MO 履歴の断片切断後，データセットに含まれる全断片のうち半分をランダムに選択し，全断片の比較を 100 回繰り返した場合，最適な λ 値の平均 μ は-0.329，標準偏差 σ は 0.268 となった．なお，$\mu \pm 2\sigma$ 範囲内の平均精度は 0.99704 以上であった．

乱択条件 2 では，データセットの多くを占める開発者 1 と 2 の断片が多く選択される確率が高い．上記の結果より，データセット全体を満遍なく使用するより，開発者 1 と 2 の情報を多く使用する方が最適な λ 値は小さくなる傾向があるといえる．すなわち，使用するデータによって，最適な λ 値は変動するといえる．ただし，λ 値を $\mu \pm 2\sigma$ の範囲内としたときの精度は両条件とも高水準を維持している．このことから，過去に得た最適な λ 値を，同じ開発者の他の断片の比較の際に再利用しても十分高精度となる可能性が高いと考えられる．

Q PBoxcox の最適な λ は履歴が変わるとどの程度変わるのか？
A 著者が変わると λ も変動するが，同著者の履歴を同程度使用したときの λ 値の変動は著者推定が十分高精度に行える範囲にとどまる可能性が高い．

5.2　特定著者除去

本研究のデータセットは特定の開発者の履歴に偏った構成となっているため，結果の信頼性に懸念が残る．そこで，一部の開発者を除外したときの精度を計測した．結果を図 4 に示す．ここでは，断片長 N は 3000 に固定している．

図 4 の通り，開発者 1 と 2 両方を除外したとき，

[*6] $\mu \pm 2\sigma$ を包含する最小範囲内の各 λ 値について 100 回の試行で得られた精度の平均値のうち最小のもの．

・精度

f \ 除外対象	1,2	3-10	1	2	3	4	5	6	7	8	9	10
-1	1.00000	1.00000	1.00000	0.99976	0.99841	1.00000	0.99841	0.99847	0.99841	0.99847	0.99847	0.99847
-0.9	1.00000	1.00000	1.00000	0.99745	0.99877	1.00000	0.99877	0.99882	0.99882	0.99882	0.99882	0.99882
-0.8	1.00000	1.00000	1.00000	0.99768	0.99888	1.00000	0.99888	0.99892	0.99892	0.99892	0.99892	0.99892
-0.7	1.00000	1.00000	1.00000	0.99858	0.99931	1.00000	0.99931	0.99934	0.99934	0.99934	0.99934	0.99934
-0.6	1.00000	1.00000	1.00000	0.99918	0.99960	1.00000	0.99960	0.99962	0.99962	0.99962	0.99962	0.99962
-0.5	1.00000	1.00000	1.00000	0.99940	0.99971	1.00000	0.99971	0.99972	0.99972	0.99972	0.99972	0.99972
-0.4	1.00000	1.00000	1.00000	0.99925	0.99964	0.99993	0.99971	0.99965	0.99965	0.99965	0.99965	0.99965
-0.3	1.00000	1.00000	1.00000	0.99940	0.99971	0.99993	0.99978	0.99972	0.99972	0.99972	0.99972	0.99972
-0.2	1.00000	1.00000	1.00000	0.99948	0.99975	0.99986	0.99986	0.99976	0.99976	0.99976	0.99976	0.99976
-0.1	1.00000	1.00000	1.00000	0.99948	0.99975	0.99986	0.99986	0.99976	0.99976	0.99976	0.99976	0.99976
0	1.00000	1.00000	1.00000	0.99943	0.99975	0.99971	0.99968	0.99955	0.99955	0.99955	0.99955	0.99955
0.1	1.00000	1.00000	1.00000	0.99843	0.99935	0.99949	0.99949	0.99927	0.99927	0.99927	0.99927	0.99927
0.2	0.94444	1.00000	0.99898	0.99563	0.99816	0.99827	0.99867	0.99788	0.99788	0.99788	0.99788	0.99788
0.3	0.94444	1.00000	0.99898	0.98983	0.99582	0.99614	0.99711	0.99530	0.99530	0.99530	0.99530	0.99530
0.4	0.88889	1.00000	0.99795	0.97354	0.98853	0.98883	0.99455	0.98782	0.98771	0.98771	0.98771	0.98771

図 4　特定の著者を除外した場合の精度

・精度

N \ param	PBoxcox														
	-1.0	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
1000	0.98963	0.99010	0.99074	0.99115	0.99142	0.99165	0.99163	0.99094	0.98910	0.98665	0.98254	0.97576	0.96909	0.96113	
2000	0.99552	0.99592	0.99628	0.99667	0.99703	0.99736	0.99768	0.99811	0.99825	0.99840	0.99798	0.99682	0.99140	0.98402	0.97684
3000	0.99778	0.99809	0.99837	0.99854	0.99882	0.99921	0.99922	0.99939	0.99948	1.00000	1.00000	1.00000	0.99865	0.99467	0.98771
4000	0.99853	0.99901	0.99874	0.99884	0.99916	0.99937	0.99941	1.00000	1.00000	1.00000	1.00000	1.00000	0.99947	0.99674	0.99001
5000	0.99963	0.99963	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99968
6000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99972	0.99941
7000	0.99956	0.99956	0.99956	0.99956	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
8000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
9000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
10000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

図 5　デバッグ操作を除外した場合の精度

$\lambda = [-1.0, 0.1]$ の範囲において，精度 1.0 となった．開発者 1 または 2 を除外したときの精度の最大値はそれぞれ 1.0, 0.99948 であり，$N = 3000$ におけるデータセット全体を使用したときの最大精度 0.99976 と比べて，遜色ない結果となった．このことから，データセットの約 2/3 のプロジェクト数を占める両開発者のデータが必ずしも精度の向上に寄与しているわけではないことが明らかになった．

なお，開発者 3〜10 を除外したとき (すなわち，開発者 1,2 のみを比較対象としたとき)，今回試した λ 値の範囲ではすべて精度 1.0 となった．

Q 本実験のデータセットの多くを占める開発者 1 と 2 のデータを除去しても著者推定できるか？逆に開発者 1 と 2 以外を除去した場合は？
A 開発者 1 や 2 のデータを除去した場合，開発者 1 と 2 以外のデータを除去した場合ともに，高精度な著者推定が可能であった．

5.3　デバッグ操作除去

ここまでで，同一開発者の履歴断片は類似する可能性が高いという結果が得られたが，当然，どのような開発タスクを実施するかによって，同じ開発者でも使用する MO には違いが発生する．

本研究の実験において，著者推定に失敗する事例を調べたところ，断片の大半をデバッグに関する操作が占めており，同一開発者の断片との類似度が低下する事例を

発見した．特に，デバッガ使用時にステップ実行を行う `org.eclipse.debug.ui.commands.StepOver` は何度も繰り返される傾向があり，これが断片中の MO の大半を占めると，著者推定が困難になる．

現時点で，タスクの影響を受けて明確に MO 使用の傾向が変わることが確認できているのはデバッグ時のみであるため，デバッグ操作を除去した場合に著者推定精度がどのように変わるかを確かめた．具体的には，MO の種類 (ラベル) に `debug` を含むものをあらかじめ除外した履歴を用いて実験を行った．

結果を図 5 に示す．全データを用いたときの結果 (図 3) と比べると，精度が低下している箇所もあるが，全体的には精度は改善している．断片の大半を特定の MO が占めることが原因で推定に失敗する場合，原因となっている操作をあらかじめ除外することで精度を改善できる可能性があることがわかった．ただし，何割を占める場合に除外することが適切なのか，デバッグ操作以外に同様の事例が存在するか等については現時点で判明していない．

Q デバッグ操作除外の著者推定精度への影響は？
A 精度は向上する傾向にある．

6. 議論

6.1 実際の著者識別への応用

本研究により MO を用いた著者識別の可能性が示せたと考えるが，現実的な著者識別の要求に応えるためには，具体的な文脈に応じた著者識別手法が必要となる．例えば，複数人で開発しているプロジェクトで，全員が操作履歴を記録する場合，コードは共有するが履歴の内容は互いに知られないように注意する必要がある．この時，あるコードに関して係争が起きた場合は，全員の履歴を利害関係のない第三者が集めて本論文の実験のように類似度を調べることで真著者を推定することができる．第三者が信頼のおける存在でなければならない点に注意が必要である．

真著者と偽著者の履歴のみが入手可能な事例においては，偽著者の履歴が改竄されている可能性が考えられる．この場合でも，真著者の履歴に関しては，最も類似したベクトルは同じ著者のものになる可能性が高い (履歴の偽造に対する耐性が高いと考えられる) が，偽著者が履歴の傾向を変化させて攻撃した場合に確実に真著者を推定できるかについては今後実験により確認する予定である．

最適な λ 値を計算する際，本研究ではデータセット全体で，最も精度が高くなるように λ 値を設定した．最適な λ 値は，その定義より，複数の開発者の履歴を収集しなければ決定できないが，現実の著者推定において，多くの開発者の履歴が得られるとは限らないこと，特に，偽著者の真正な履歴が得られるとは限らないことから，状況によっては PBoxcox を利用できないという問題がある．

6.2 操作履歴とコードの紐付け

1 章で述べた通り，本研究では操作履歴とソースコードの紐付けが可能であることを前提としている．操作履歴ファイルには操作対象のソースファイル名が明記され，編集の内容からコードとの一致も確認可能であるが，複数人で記述したソースファイルの場合は，編集を行った全員の履歴を集めなければコードの一致を確認できないという問題が発生する．なお，全員分の真正な操作履歴が得られた場合，履歴データから最終編集者を調べることで文字ごとに著者を特定することが可能である．

なお，本論文で使用したデータセットでは，各プロジェクトが一人の開発者によって行われたため，ソースファイルごとに履歴との紐付けが可能となっている．

6.3 既存手法との比較実験

容易に利用できる，近年に提案された著者推定ツールは，著者らの知る限り存在せず，既存手法との直接の比較実験は現時点で行えていない．推定精度そのものは 1.0 やそれに近いものを多く得られたが，他手法を本研究のデータセットに適用した場合と比較して優位な結果であるかは定かではない．

6.4 開発者の習熟の影響

開発者が IDE の機能に徐々に習熟することで MO 使用頻度が変化し，同一著者であっても開発初期と末期で特徴が大きく異なる可能性が考えられる．今回使用したデータセットでは，開発者 1,2 については，履歴収集期間はそれぞれ 12 年間，10 年間となっているが，実験結果から，当該期間の初期と末期の違いは，他著者の履歴との違いを凌駕するほど大きくはなっていないとわかる．ただし，初学者の場合，コピーや保存といった基本的な MO が占める割合が大きく，元々の履歴が類似しがちになること，および，新しく学習する内容が相対的に多く，利用形態が時間の経過とともに大きく変わる可能性があることから，推定精度が低下する可能性がある．データセットに含まれる学生の履歴を調べたところ，当該学生の履歴から作成したベクトルの重心からの距離の分散が他の開発者よりも極端に大きいケースが見られた．

6.5 IDE の設定の影響

本論文のデータセットでは，開発者 2 のみ，デフォルト設定から変更して Emacs キーバインドを使用している．この影響を考慮し，関連する MO(`org.eclipse.ui.edit.text.goto.columnNext/Previous` 等) を除外した実験を行った．結果を図 6 に示す．$N = 4000$，$\lambda = [-0.4, 0.1]$ のときに精度 1.0 となっ

ており，除外しない場合よりも高精度となった．一方，$N = 1000$ や $N = 2000$ の場合には精度の悪化が目立つ結果となった．今回の事例では，IDE の設定の影響は精度を向上・低下させるかについて，明確な結果は得られなかった．

・精度

	PBoxcox														
N \ param	-1.0	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
1000	0.96441	0.96491	0.96504	0.96497	0.96412	0.96289	0.96041	0.95705	0.95114	0.94220	0.92876	0.90858	0.87516	0.82548	0.76948
2000	0.99548	0.99588	0.99636	0.99656	0.99678	0.99686	0.99721	0.99696	0.99705	0.99692	0.99573	0.99335	0.98644	0.97141	0.93968
3000	0.99783	0.99798	0.99876	0.99891	0.99915	0.99915	0.99884	0.99907	0.99899	0.99891	0.99829	0.99705	0.99418	0.98432	0.96251
4000	0.99609	0.99654	0.99804	0.99925	0.99970	0.99970	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99970	0.99714	0.98976
5000	0.99842	0.99894	0.99947	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.99947	0.99836	0.99336
6000	0.99957	0.99957	0.99957	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
7000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
8000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
9000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
10000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

図 6　Emacs キーバインドに由来する MO を除外した場合の精度

6.6　開発者ベクトルの活用

提案手法ではすべての断片の長さを同一であると仮定したが，開発者ごとにあらかじめ多量の MO を集めておき，開発者ごとの基準となる「開発者ベクトル」を作成する手法も考えられる．この場合，著者推定対象の履歴断片がより小規模であっても高精度となる可能性がある[*7]．

6.7　類似 MO の除去

一般的な機械学習においては，多重共線性の問題を回避するため，高い相関を持つ変数をあらかじめ除去することが行われる．そこで，全 MO 種類の出現数の相関係数を計算し，相関が高いものを除去する実験を行った．相関係数が 1.0 となった組について，1 つのみを残し，他を除去した結果，全 300 次元のうち 20 次元を削減できた．次元削減は精度への影響はほぼなかった．

6.8　盗用の抑止効果

一般に，操作履歴の捏造は，ソースコード等の成果物の捏造よりも困難である．ゆえに，操作履歴に基づく著者推定が可能であるという事実は，盗用・知的財産権侵害に対する抑止のためにも有効であると考える．もし既存の技術が無効化されたとしても，追加の著者推定手法が存在すれば，不正利用を難しくし，偽著者の動機を低減させられるものと考える．

6.9　異なる条件での実験

本論文で議論した著者推定の精度は，データセットの性質に強く影響を受ける．極端に多くの開発者が含まれるデータセットの場合，類似する開発者が存在する可能性が上がり，精度が低下する恐れがある．

原則，MO に基づく著者推定手法は IDE 依存である．十分な次元数を持つベクトルが MO 履歴から生成でき，比較対象の開発者が同一の IDE を利用していることが前提となる．また，推定手法自体はプログラミング言語非依存ではあるが，言語が変わると開発者が使用する MO が変化する可能性がある．

[*7] 我々が実施した予備実験では，開発者ベクトルの断片長 9000，著者推定対象の各断片の長さを 1000 としたときに精度 1.0 を得た．ただし，本論文のデータセットとは異なる．

7.　おわりに

本論文では，開発者が IDE 上で行った操作の履歴のうち，特にメニュー操作の情報を利用することで，高精度な著者推定が可能であることを確認した．一定以上の操作数を持つ履歴断片からベクトルを生成すること，ベクトルを構成する各元を適切な関数により変換することで精度を高められることを示した．今後の課題としては，現実的な著者推定の場面に応用が可能であることの確認，より多様・多量なデータセットを使用した場合の結果の確認等が挙げられる．

謝辞　本研究の一部は JSPS 科研費 25K15054 の支援による．

参考文献

[1] X. He, A.H. Lashkari, N. Vombatkere, and D.P. Sharma. “Authorship attribution methods, challenges, and future research directions: A Comprehensive Survey,” Information, Vol.15, No.3, Article 131, 2024.
[2] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina. “Code authorship attribution: Methods and Challenges,” ACM Comput. Surv., Vol.52, No.1, pp.1–36, 2019.
[3] 大森隆行，丸山勝久．開発者による編集操作に基づくソースコード変更抽出．情報処理学会論文誌，Vol.49, No.7, pp.2349–2359, 2008.
[4] H. Ding. “Extraction of Java program fingerprints for software authorship identification,” Master’s thesis, Oklahoma State University, 2002.
[5] R. Lange and S. Mancoridis, “Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics,” Proc. GECCO, pp.2082–2089, 2007.
[6] G. Frantzeskou, E. Stamatatos, S. Gritzalis, C.E. Chaski, and B.S. Howald, “Identifying Authorship by Byte-Level N-Grams: The Source Code Author Profile (SCAP) Method,” IJDE, Vol.6, Issue 1, pp.1–18, 2007.
[7] S.D. Burrows, “Source Code Authorship Attribution,” Ph.D. Thesis, RMIT University, 2010.
[8] S. Burrows, A.L. Uitdenbogerd, and A. Turpin, “Comparing techniques for authorship attribution of source code,” Software — Practice and Experience, 44:1–32, 2014.
[9] M. Abuhamad, T. Abuhmed, D. Mohaisen, and D. Nyang. “Large-scale and Robust Code Authorship Identification with Deep Feature Learning,” Privacy and Security, Vol.24, No.4, Article 23, pp.1–35, 2021.
[10] A. Bogdanova and V. Romanov, “Explainable Source Code Authorship Attribution Algorithm,” J. Physics: Conference Series, ITTCS, Vol.2134, pp.1–10, 2021.
[11] Z. Li, G. Chen, C. Chen, Y. Zou, and S. Xu “RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation,” Proc. ICSE, pp.1906–1918, 2022.
[12] A. Suljic and M.S. Hossain, “Towards Performance Im-

provement of Authorship Attribution," IEEE Access, Vol.12, pp.77054–77064, 2024.
[13] S. Choi, Y.K. Tan, M.H. Meng, et al. "I Can Find You in Seconds! Leveraging Large Language Models for Code Authorship Attribution," arxiv.org/pdf/2501.08165, 2025
[14] 武田隆之，牛窓朋義，山内寛己 ら．コーディングスタイルの特徴量とソースコード盗用との関係の分析．情処研報，Vol.2010-SE-167，No.8，pp.1–8，2010．
[15] 福本大，北川裕基，玉田春昭．コーディングスタイルを基にした初心者プログラマ向けの著者解析．情処研報，Vol.2015-SE-187，No.16，pp.1–7，2015．
[16] V. Ljubovic and E. Pajic, "Plagiarism Detection in Computer Programming Using Feature Extraction From Ultra-Fine-Grained Repositories," IEEE Access, Vol.8, pp.96505–96514, 2020.
[17] E. Pajic and V. Ljubovic, "Improving Plagiarism Detection Using Genetic Algorithm," Proc. MIPRO, pp.571–576, 2019.
[18] H. Meier, M. Lepp, and R. Kütt, "Plagiarism Detection Tool Based on Programming Activity Logs," Proc. EDUCON, pp.1–7, 2024.
[19] J. Schneider, A. Bernstein, J. Vom Brocke, K. Damevski, and D.C. Shepherd. "Detecting plagiarism based on the creation process," IEEE Trans. Learning Technologies, Vol.11, No.3, pp.348–361, 2017.
[20] Y. Yoon and B.A. Myers, "Capturing and Analyzing Low-level Events from the Code Editor," Proc. PLATEAU, pp.25–30, 2011.
[21] G.E.P. Box and D.R. Cox. "An Analysis of Transformations," Journal of the Royal Statistical Society. Series B (Methodological), Vol.26, No.2, pp.211–252, 1964.