Cross-Polytope LSH を用いたコードクローン検出のためのパラメータ決定手法

徳井 翔梧　吉田 則裕　崔 恩瀞　井上 克郎

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部分を持つコード片のことである．横井らが提案したコードクローン検出ツール CCVolti は，情報検索技術である TF-IDF と近似最近傍探索アルゴリズム Cross-Polytope LSH を利用して，従来の手法では困難であった意味的に類似するコードクローンを高速に検出可能とした．しかし，CCVolti は検出時間が Cross-Polytope LSH に大きく依存し，Cross-Polytope LSH によるコードクローンの検出漏れが発生するという問題点がある．本研究では，クローン検出の利用者が与えた再現率の目標値を満たしつつ，できるだけ時間を短縮することを目的として，プロジェクトの規模から適切なパラメータ値を求める線形回帰モデルを構築し，コードクローン検出対象に適した Cross-Polytope LSH に与えるパラメータ値の組を決定する手法を提案する．さらに，20 個のプロジェクトに対して本手法で決定されたパラメータ値を CCVolti に適用し，コードクローン検出する評価実験を実施して本手法の有効性を示す．

A code clone is a code fragment that has identical or similar code fragments to it in the source code. A code clone detector CCVolti has been developed using Cross-Polytope Locality-Sensitive Hashing (LSH). CCVolti can detect not only syntactic clones but also semantic clones, which are difficult to be detected. However, CCVolti has two problems: (1) the detection time depends on Cross-Polytope LSH, and (2) several missed code clones. In this study, we propose an approach to determine Cross-Polytope LSH parameters to obtain a target value of recall given by a user and save as much time as possible. The approach builds a linear regression model that learns suitable parameters based on the size of target projects and then determines appropriate Cross-Polytope LSH parameters for a code clone detection target. Finally, we apply this approach with CCVolti to 20 open source software projects and confirm this approach's effectiveness.

1 まえがき

ソフトウェア保守を困難にする大きな要因の 1 つとしてコードクローンが指摘されている [14]．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部分を持つコード片のことであり，互いにコードクローンであるコード片の組をクローンペアと呼ぶ．コードクローンを保守するために，ソースコード中からコードクローンを識別して管理する必要がある．しかし，ソースコードの規模が大きくなるとソースコード中に含まれるコードクローンも膨大な量となる．手作業でコードクローンを管理することが困難となるため，コードクローンをソースコードから自動的に検出するための手法が提案されている [6] [14]．

横井らが提案したブロッククローン (コードブロック単位のコードクローン) 検出ツール CCVolti[†1] は情報検索技術 [4] と局所性鋭敏型ハッシュ (Locality-Sensitive Hashing，以降 LSH) [3] を利用することによって，従来の手法では困難であった意味的に処理が類似したコードクローンを検出できる [23]．CCVolti におけるコードブロックは，関数と，関数内部の if, for 文等の波括弧で囲まれた部分である．CCVolti は，

Determining Cross-Polytope Locality-Sensitive Hashing Parameters for Code Clone Detection.

Shogo Tokui, Katsuro Inoue, 大阪大学, Osaka University.
Norihiro Yoshida, 名古屋大学, Nagoya University.
Eunjong Choi, 京都工芸繊維大学, Kyoto Institute of Technology.

コンピュータソフトウェア, Vol.38, No.4 (2021), pp.60–82.
[研究論文] 2020 年 11 月 11 日受付.

†1 https://github.com/k-yokoi/CCVolti

入力ソースコードに対して構文解析を行い，コードブロックの抽出を行う．その後，抽出した各コードブロックを情報検索技術の 1 つである TF-IDF (Term Frequency-Inverse Document Frequency) 法 [4] に基づいて特徴ベクトルに変換する．最後に，特徴ベクトル間の類似度が閾値以上の対をコードクローン対として検出する．CCVolti は高速にコードクローン対を検出するために，近似最近傍探索アルゴリズム Cross-Polytope LSH [3] を用いている．Cross-Polytope LSH とは，高次元なベクトル集合を確率的にハッシュ化して最近点を求める近似最近傍探索アルゴリズムである局所性鋭敏型ハッシュ (LSH) の一種である．Cross-Polytope LSH はパラメータ値の与え方によって，精度や実行速度を変化させることができる．CCVolti は，Cross-Polytope LSH を利用するために，LSH ライブラリ FALCONN[†2] を使用した．

CCVolti は，既存のコードクローン検出法と比べて高い精度でコードクローンを検出できる [23]．また，大規模なプロジェクトに対して現実的な計算時間でコードクローン検出可能である．実際，15MLOC の Linux Kernel に対して CCVolti を用いてコードクローン検出した場合，20 分程度で検出が完了し，100MLOC においても 4 時間程度でコードクローンを検出した．

一方，CCVolti には次の 2 つの問題点が挙げられる [18]．1 つ目は，類似度が閾値以上のベクトル対を Cross-Polytope LSH を用いて探索する処理で，約 10% の検出漏れが発生する場合がある点である．2 つ目は，類似度が閾値以上のベクトル対を探索する時間が CCVolti のコードクローン検出時間の約 90% を占めている点である．これらの問題は，CCVolti が Cross-Polytope LSH のパラメータをクローン検出対象プロジェクトに対して調整していないことが原因である．特徴ベクトルの数や次元の大きさはプロジェクトごとに異なるため，ベクトル対の探索の精度と実行速度のトレードオフとなるパラメータはプロジェクトごとに異なる．しかし，FALCONN には 10 を超えるパラメータが存在するため，CCVolti の利用者が Cross-Polytope LSH のアルゴリズムを理解し，クローン検出するたびにクローン検出対象プロジェクトに適したパラメータを調整することは困難である．

この問題を解決するために，本研究では，類似度が閾値以上のベクトル対に対して検出できる割合が目標値を満たし，かつ高速であるための，Cross-Polytope LSH に与えるパラメータ決定手法を提案する．本研究では，類似度が閾値以上のベクトル対の内，検出できたベクトル対の割合を再現率と定義し，クローン検出の利用者が与える再現率の目標値を目標再現率と定義する．本手法は，学習用プロジェクトに対して目標再現率を超える再現率となりできるだけ高速となるパラメータ値の組を学習データとして，クローン検出対象プロジェクトに対するパラメータ値の組を決定する線形回帰モデルを作成する．そして，利用者はクローン検出対象プロジェクトを入力として線形回帰モデルを適用し，目標再現率を超える再現率となりつつ，高速なパラメータ値の組を求める．

評価実験では，10 個の C 言語プロジェクトと 10 個の Java プロジェクトを学習用プロジェクトとして，本手法で決定されたパラメータを使用する CCVolti を用いてコードクローン検出を実施した．実験結果では，多くの場合で再現率が目標再現率を上回ることを確認した．また，本手法で決定されたパラメータは FALCONN のデフォルトのパラメータ値より多くの場合で高速であることを確認した．デフォルトのパラメータ値での再現率と同等の再現率が得られるように目標再現率 0.99 としたときの探索時間と比較して，目標再現率 0.8 での探索時間はおよそ半減できていた．

以降，2 章では，コードクローン検出法 CCVolti，Cross-Polytope LSH とその関連技術について述べる．3 章では，CCVolti の利用者が与えた再現率の目標値を満たし，かつ高速であるための Cross-Polytope LSH に与えるためのパラメータ決定手法を示す．4 章では，本手法の有効性の評価を行う．5 章では，実験結果をふまえた本手法の有効性の範囲と妥当性について考察する．最後に 6 章では，まとめと今後の課題について述べる．

†2 https://falconn-lib.org/

2 コードクローン検出とその関連技術

本章では，局所性鋭敏型ハッシュ (LSH)，近似最近傍探索アルゴリズム Cross-Polytope LSH，コードクローン検出ツール CCVolti について述べる．

2.1 局所性鋭敏型ハッシュ (LSH)

LSH とは，近似最近傍探索問題をハッシュを用いて解くアルゴリズムである [3]．近似最近傍探索問題とは，入力ベクトルに対してベクトル集合であるデータセットの中から近傍ベクトルを近似的に高速に見つける問題であり，最近点問題の一種である．近傍ベクトルとは，入力ベクトルに対して一定の類似度以上のベクトルのことである．

入力ベクトルに最も近いベクトルを求める最も単純な方法は，データセット中のすべてのベクトルとの類似度を計算することであるが，計算に時間がかかる．LSH は，ハッシュを用いて入力ベクトルの近傍ベクトルを求め，近傍ベクトルのみと類似度を計算することによって，高速に最も近いベクトルを求めることができる．LSH のアルゴリズムは，類似度の閾値 θ と近似因数 $c<1$ に対して，ベクトル集合の中に入力ベクトルとの類似度が θ 以上のベクトルが存在するとき，類似度が $c\theta$ 以上のすべてのベクトルを返す [3]．

ベクトルをハッシュ値に変換するハッシュ関数に対して，2 つのベクトル x, y が同じハッシュ値を取ることをハッシュの衝突という．LSH のハッシュ関数は，類似度が高いベクトル同士がハッシュの衝突を起こしやすくなるように定義される．ベクトルのハッシュ値が入力ベクトルのハッシュ値と衝突するとき，そのベクトルは入力ベクトルの近傍ベクトルとなる．2 つのベクトル x, y に対して類似度 $S(x, y)$ が定義された d 次元空間上において，x, y のハッシュ値が衝突する確率を衝突確率と呼ぶ．

データセット中から，ある入力ベクトルに対する近傍ベクトルを LSH を用いて探索する処理の時間計算量は $O(dn^{\rho})$ となる [3]．ここで，d はベクトルの次元数，n はベクトル集合のベクトル数を表し，ρ は式 1 のように表される．

$$\rho = \frac{\log(1/p)}{\log(1/q)} \tag{1}$$

ここで，p は類似度 $S(x, y)$ が θ 以上となる 2 つのベクトルの衝突確率を表し，q は類似度 $S(x, y)$ が $c\theta$ 以下となる 2 つのベクトルの衝突確率を表す．ρ が小さいほど実行時間の計算量のオーダーが小さくなるため，ρ は LSH のアルゴリズムの評価基準として用いられる．

2.2 近似最近傍探索アルゴリズム Cross-Polytope LSH

LSH の一種である Cross-Polytope LSH は，d 次元単位球上のベクトル集合に対して有効性が保証されており，効率的な実装も可能である [3]．コードクローン検出ツール CCVolti が用いる LSH ライブラリ FALCONN は，大規模なベクトル集合の近似最近傍探索問題を解くための実装として Andoni らにより開発された．本節では，Cross-Polytope LSH のアルゴリズム，および Cross-Polytope LSH を用いた類似探索について説明する．

2.2.1 Cross-Polytope LSH のアルゴリズム

Cross-Polytope LSH は，2 つのベクトル x, y に対するユークリッド距離 $d(x, y) = \|x - y\|$ に基づいて近傍ベクトルを探索するアルゴリズムである．コサイン類似度 $C(x, y) = (x \cdot y)/(\|x\| \|y\|)$ は式 $C(x, y) = 1 - d(x, y)^2/2$ によってユークリッド距離と 1 対 1 対応する．本研究では，CCVolti が用いるコサイン類似度に基づいて議論する．

Cross-Polytope LSH のハッシュ関数を用いた，d 次元ベクトル x に対するハッシュ値の計算方法について説明する．まず，ベクトル x を正規化し，ランダム行列 $A \in \mathbb{R}^{d \times d}$ を乗算してランダム回転を行い，ベクトル $y = Ax/\|Ax\|$ に変換する．次に，ランダム回転後のベクトル y に対して，正規直交基底の基底ベクトルとそれらの逆ベクトル $\{\pm e_i\}_{1 \le i \le d}$ の中から最も類似度が高いベクトルを求める．最も類似度が高いベクトルの符号と添え字 i によって，x のハッシュ値が $\pm i$ に決定される．すなわち，Cross-Polytope LSH のハッシュ関数は，行列 A を用いて入力ベクトルをランダムに回転させ，回転後のベクトルが d 個に分割された単位球のどの区画に含まれるかを，入力ベクトルのハッシュ値とする．

ベクトルにランダム行列を掛けることにより，ベクトルがランダムに回転し，類似度が高いベクトル対が一定の確率で衝突を起こすようになる．CCVolti が用いる Cross-Polytope LSH ライブラリ FALCONN では，前処理で次元圧縮をしたり，ランダム回転の処理に高速アダマール変換を用いたりするなど，メモリ削減や高速化を行っている [1] [21]．FALCONN には 10 種類のパラメータが存在し，次元圧縮後の次元数などの検出結果に影響を与えるパラメータと，メモリ上のデータ保持方法などメモリや計算速度に影響を与えるパラメータがある [18]．

ある類似度 δ に対して 2 つのベクトル x, y が $C(x,y)=\delta$ をみたすとき，Cross-Polytope LSH の衝突確率 P_T は式 2 のように表される [3]．

$$\ln \frac{1}{P_T} = \frac{1-\delta}{1+\delta} \cdot \ln T + O_\delta(\ln \ln T) \tag{2}$$

T は区画の分割数を表す．$O_\delta(\ln \ln T)$ は δ に依存する誤差項であり，$\ln \ln T$ に比例する．誤差項 $O_\delta(\ln \ln T)$ は，区画の分割数 T が大きくなるほど 0 に近づく [3]．また，式 2 と同様に，$C(x,y)=c\delta$ のときの衝突確率 Q_T を式 3 とする．

$$\ln \frac{1}{Q_T} = \frac{1-c\delta}{1+c\delta} \cdot \ln T + O_{c\delta}(\ln \ln T) \tag{3}$$

Cross-Polytope LSH による近傍ベクトルの検出時間の時間計算量 $O(dn^\rho)$ の ρ は，式 2 と式 3 を用いて，2.1 節の式 1 から式 4 に変形でき，T に依存して決まることが分かる．

$$\rho = \frac{\frac{1-\delta}{1+\delta} \cdot \ln T + O_\delta(\ln \ln T)}{\frac{1-c\delta}{1+c\delta} \cdot \ln T + O_{c\delta}(\ln \ln T)} \tag{4}$$

2.2.2 Cross-Polytope LSH を用いた類似探索

類似探索とは，ベクトル集合から類似度が閾値 θ 以上のベクトル対を探索することである．Cross-Polytope LSH を用いた類似探索のアルゴリズムは，以下の 3 つのステップで構成される [3]．

STEP A ベクトルの集合から L 個のハッシュテーブルを作成

STEP B いずれかのハッシュテーブルで，ハッシュ値が衝突するベクトル対を抽出

STEP C STEP B で抽出したすべてのベクトル対の類似度を計算し，類似度が閾値以上であるベクトル対をクローンペアとして検出する

Cross-Polytope LSH のランダム性から，異なるハッシュ関数を複数用意できる．

ハッシュテーブル 1 つ当たり K 個のハッシュ関数を使用することで，最も近いベクトルの候補を減らし，より高速に最も近いベクトルを検出することができる．K 個のハッシュ関数の衝突確率をそれぞれ，$P_{T_1}, ..., P_{T_K}$ とすると，ハッシュテーブル 1 つ当たりの衝突確率は式 5 のように表される [2]．

$$P_{T,K} = \prod_{i=1}^{K} P_{T_i} \tag{5}$$

ただし，ライブラリ FALCONN では，T の上限 d に対して $P_{T_1}, ..., P_{T_K-1} = P_d$ となるように実装されているため，FALCONN でのハッシュテーブル 1 つ当たりの衝突確率は式 6 のように表される．

$$P_{T,K} = {P_d}^{K-1} \cdot P_T \tag{6}$$

L 個のハッシュテーブルを用意し，いずれかのハッシュテーブルで衝突したベクトル対をクローンペアの候補として抽出する．ハッシュテーブルを増やすことで，探索時間は増加するが，衝突確率を上げて検出漏れを減らすことができる．このとき，式 5 で表したハッシュテーブル 1 つ当たりの衝突確率 $P_{T,K}$ に対して，L 個のハッシュテーブルでの衝突確率 $P_{T,K,L}$ は式 7 のように表される．

$$P_{T,K,L} = 1 - (1 - P_{T,K})^L \tag{7}$$

2.3 コードクローン

本節では，コードクローンの定義，コードクローン検出ツール CCVolti のアルゴリズムと，その問題点について述べる．

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部分を持つコード片のことであり，既存コードのコピーアンドペーストによる再利用等が原因で生じる．ソフトウェア保守を困難にする要因の 1 つとしてコードクローンが指摘されている．

Roy らは，コードクローン間の違いの度合いに基づき，コードクローンを以下の 4 つの定義に分類している [16]．

タイプ 1 空白やタブの有無，コーディングスタイル，コメントの有無などの違いを除き完全に一致

するクローン

タイプ 2 タイプ 1 の違いに加えて，変数名などのユーザー定義名，変数の型などが異なるコードクローン

タイプ 3 タイプ 2 の違いに加えて，文の挿入や削除，変更などが行われているコードクローン

タイプ 4 類似した処理を実行するが，構文上の実装が異なるコードクローン

2.3.1 コードクローン検出

本節では，これまでに提案されているコードクローン検出技術の説明を行う．大規模なソースコード中のコードクローンを手作業で管理することが困難であるため，コードクローンを自動で検出する手法が多数提案されているコードクローン検出手法は，その手法で用いる検出単位によって，行単位の検出，字句単位の検出，抽象構文木を用いた検出，プログラム依存グラフを用いた検出，メトリクスなどその他の技術を用いた検出に分類することができる [6].

行単位の検出では，言語に依存せず検出できるが，タイプ 1 のコードクローンのみ検出可能である．Baker らは，プログラミング言語に依存せず線形時間でコードクローンを検出できる手法を提案した [5].

字句単位の検出では，比較的高速に，タイプ 1 からタイプ 2 のコードクローンを検出可能である．神谷らが提案した字句単位の検出手法は，ユーザ定義名を特殊文字に置き換えるという言語依存の処理をするにもかかわらず，C/C++，Java，COBOL など広く用いられている複数のプログラミング言語に対応している [9].

抽象構文木を用いた検出では，検出の前処理としてソースコードに対して構文解析を行うことで抽象構文木を構築し，抽象構文木上の同形あるいは類似した部分木をコードクローンとして検出する．また，各部分木を特徴ベクトルに変換し，特徴ベクトル間の類似度を求めることによって，ある程度特徴ベクトルに違いがあっても検出でき，タイプ 1 からタイプ 3 までのコードクローンを検出できる．横井らは，情報検索技術を利用することにより，コードブロック単位のコードクローンを検出する手法を提案した [23].

プログラム依存グラフを用いた検出では，プログラムの意味的な処理の類似性に着目しているため，文の並び替えが発生したコードクローンなど，タイプ 4 のコードクローンを検出可能である．Komondoor らは，ソースコード中の文をプログラム依存グラフのノードとすることで，同一のグラフ構造となるコード片をコードクローンとして検出する手法を提案した [10].

メトリクスなどその他の技術を用いた検出では，プログラムのモジュールに対してメトリクスを計測し，それらの類似度を計算することによって，タイプ 1 からタイプ 3 のコードクローンを検出できる．Mayrand らは，関数に対して 21 種類のメトリクスを計測することによってコードクローンを検出する手法を提案した [13].

横井らが提案した抽象構文木を用いた検出ツール CCVolti は，情報検索技術を利用することにより，タイプ 1 からタイプ 4 までのブロッククローン (コードブロック単位のコードクローン) を検出できる [23]. コードブロックとは，if 文や for 文や関数などの波括弧で囲まれたコード片を指す．CCVolti は，既存のコードクローン検出法と比べて高い精度でコードクローンが検出でき，大規模なプロジェクトに対して現実的な計算時間でコードクローン検出可能である．実際，CCVolti を用いて 15MLOC の Linux Kernel のコードクローン検出を行うと，20 分程度で検出が完了し，100MLOC においても 4 時間程度で検出可能であった．

2.3.2 コードクローン検出ツール CCVolti のアルゴリズム

CCVolti は以下のステップで入力ソースコードからコードクローンを検出する．

STEP 1 ソースコードの構文解析を行い，抽象構文木を生成

STEP 2 抽象構文木からワードとコードブロックを抽出

STEP 3 TF-IDF 法 [4] により，コードブロック単位の特徴ベクトルを計算

STEP 4 Cross-Polytope LSH を用いた類似探索を行い，コサイン類似度が閾値 0.9 以上のクローンペアを検出

STEP 1 では，ソースコードの構文解析を行い，抽象構文木を生成する．STEP 2 では，STEP 1 で生成した抽象構文木からワードとコードブロックを抽出する．ワードとは，予約語と識別子名を構成する言語とする．STEP 3 では，TF-IDF 法によりコードブロック単位の特徴ベクトルを計算する．TF-IDF 法とは，ワードの出現頻度によって重み付けを行うベクトル化手法である [4]．STEP 4 では，2.2.2 節で説明した Cross-Polytope LSH を用いた類似探索を行い，コサイン類似度が閾値 0.9 以上のクローンペアを検出する．

2.3.3 コードクローン検出ツール CCVolti の問題点

徳井らは先行研究において，LSH を用いるコードクローン検出法に対して，以下の 2 つの問題点を指摘した [18]．

- Cross-Polytope LSH を用いた類似探索において，検出漏れが発生する場合があることを指摘している．同時修正箇所の検出などの目的で CCVolti を利用する場合，高い精度が求められるにも関わらず，類似探索において検出漏れが多く発生することは問題である．
- 2.3.2 節で説明した CCVolti の Cross-Polytope LSH を用いた類似探索の処理時間が CCVolti のコードクローン検出時間の約 90% を占めており，クローン検出時間が類似探索の処理時間に大きく依存していることを指摘している．

これらの問題は，CCVolti が Cross-Polytope LSH のパラメータをクローン検出対象プロジェクトに対して調整していないことが原因である．類似度が閾値以上のベクトル対の探索の精度と実行速度は Cross-Polytope LSH に与えるパラメータによって調整できる．プロジェクトごとに特徴ベクトルの数や次元の大きさは異なるため，類似度が閾値以上のベクトル対の探索の精度と実行速度のトレードオフとなるパラメータはプロジェクトごとに異なる．しかし，CCVolti の利用者がクローン検出するたびに対象プロジェクトに適した Cross-Polytope LSH のパラメータを調整することは困難である．

3 Cross-Polytope LSH に与えるパラメータ決定手法

本章では，2.3.3 節で述べたコードクローン検出ツール CCVolti の問題点を解決するために，CCVolti の利用者が与えた目標再現率を超える再現率となり，かつ高速であるための Cross-Polytope LSH に与えるパラメータ決定手法を提案する．本手法は，学習用プロジェクトの規模に対する適切なパラメータ値の組を学習データとして，プロジェクトの規模に対するパラメータ値の組を決定する線形回帰モデルを作成し，コードクローン検出対象プロジェクトを入力として線形回帰モデルを適用することで，目標再現率を超える再現率となりつつ，高速なパラメータ値の組を求める．本研究における再現率は，類似度が閾値以上のベクトル対の数に対して Cross-Polytope LSH が検出したベクトル対の数の割合を指す．

本手法の目的は CCVolti の利用者が指定した目標再現率を超えつつできるだけ高速なパラメータを決定することである．コードクローン検出の目的に応じて目標再現率を設定し，速度を優先することや精度を優先することができる．例えば，再現率を落とすことで比較的高速にコードクローン検出すると，コードクローンがどの程度存在するかの予備分析を低コストで実行できる．また，精度を優先したパラメータでコードクローン検出すると，より正確に同時修正すべきクローンペアを検出し，保守作業としてコードクローンをリファクタリングできる．しかし，リファクタリングはコスト削減を目的としているため，リファクタリングに投入できる時間に限界がある．そのため，本手法は精度を優先しつつできるだけ高速にコードクローン検出を行うパラメータを決定する．

本手法をコードクローン管理支援ツールに適用した例を 2 つ挙げる．本田らは，コードクローンの同時修正を支援するために，ソフトウェア進化の可視化システム CCEvovis を提案した [7]．CCEvovis は，開発者がソースコードに変更を加えるたびに，変更前のバージョンに存在していたコードクローン集合 (すべてのペアがコードクローンである集合) に変更が加えられたかどうかを分類して可視化することで，開発者の

コードクローン管理を促進する．しかし，CCEvovis は複数のバージョンに対してコードクローン検出するため時間がかかる．そこで，CCEvovis が利用している CCVolti のパラメータを本手法で速度を優先して決定することで，低コストでコードクローンに対する変更有無の予備分析を実行できる．徳井らはコードクローン変更管理システム Clone Notifer を提案した [19]．Clone Notifier は，2 バージョン間の差分を検出し，コードクローン集合のコードクローンが追加/削除/変更されたかどうかを分類し，特に変更されたコードクローンと変更がないコードクローンが同時に含まれるコードクローン集合 (Inconsistent changed clone set) を開発者に知らせ，コードクローンの同時修正を促す．しかし，Clone Notifier は CCVolti を利用しているため，一部のコードクローンを取りこぼしている．そこで，本手法で精度を優先して決定したパラメータを与えた CCVolti を用いることで，任意の 2 バージョン間の Incosistent changed clone set をより正確に検出することができる．

本手法は，利用者が決定した目標再現率に対して学習用プロジェクトを用いて線形回帰モデルを作成する学習プロセスと，クローン検出対象プロジェクトの規模に対して線形回帰モデルを用いてパラメータ値を決定する適用プロセスからなる．学習プロセスでは，学習用プロジェクトの規模に対する適切なパラメータ値の組を学習データとして，クローン検出対象プロジェクトの規模に対するパラメータ値の組を決定する線形回帰モデルを作成する．適用プロセスでは，線形回帰モデルを用いて，クローン検出対象プロジェクトを入力として，目標再現率を超える再現率となりつつ，高速なパラメータ値を決定する．

本手法で作成する線形回帰モデルはプロジェクトの規模とパラメータ値の関係を推論するモデルである．利用者は学習プロセスで作成した線形回帰モデルを適用プロセスで再利用可能であり，さらに，未学習のプロジェクトに対して適用可能である．しかし，Java 言語と C 言語のプロジェクトで学習した線形回帰モデルを，スクリプト言語や関数型言語などの異なるドメインのプロジェクトに適用する場合，プロジェクトの規模とパラメータ値の関係が異なると考えられる．したがって，利用者はコードクローン検出対象プロジェクトと同じドメインのプロジェクトを用意して学習プロセスを実施する必要がある．

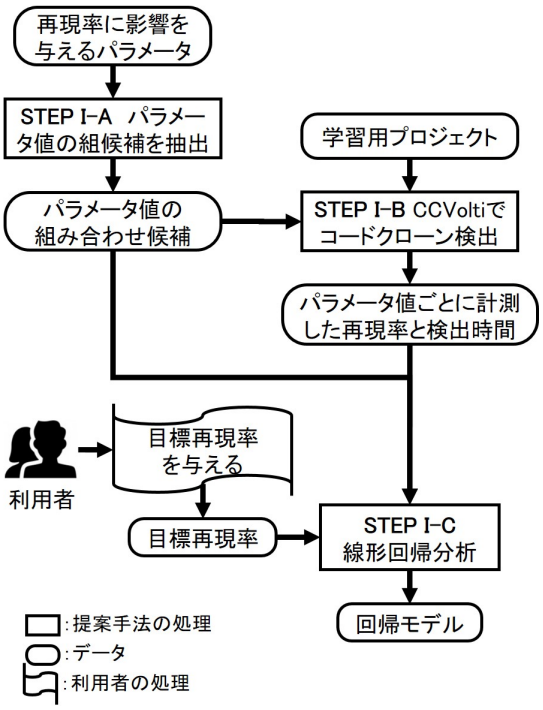

図 1　提案手法 STEP I の学習プロセス

学習プロセスでは，学習データ作成と線形回帰分析に時間を要するため，利用者が求める目標再現率に対応する線形回帰モデルを事前に作成する必要がある．一方で，作成した線形回帰モデルは文法が類似する言語のプロジェクトに対して汎用的に利用可能である．したがって，コードクローン検出利用者だけではなく開発現場の環境整備の担当者や CCVolti の開発者が学習プロセスをあらかじめ実施し，利用者に線形回帰モデルを提供することが可能である．

図 1 は STEP I の学習プロセスを示す．学習プロセスでは，学習用プロジェクトを用いて線形回帰モデルを以下の 3 つのステップで作成する．

STEP I-A　Cross-Polytope LSH に与えるパラメータから，再現率に影響を与えるパラメータを抽出する．抽出したパラメータが再現率と探索時間に与える影響を分析し，パラメータ値の組み合わせの候補を抽出する．

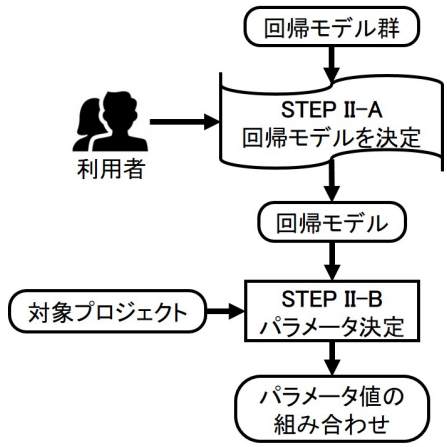

図 2 提案手法 STEP Ⅱ の適用プロセス

STEP I-B 抽出したパラメータ値の組み合わせの候補を与えた Cross-Polytope LSH を用いて CCVolti で学習用プロジェクトに対してコードクローンを検出し，パラメータごとの再現率と類似探索の探索時間を計測する．類似探索の探索時間とは 2.3.2 節で述べた STEP 4 の類似探索にかかる時間である．

STEP I-C 学習用プロジェクトの実験結果を用いて，プロジェクトの規模に対して目標再現率を超えつつ高速なパラメータ値の組み合わせを決定する線形回帰モデルを作成する．

図 2 は STEP Ⅱ の適用プロセスを示す．適用プロセスでは，学習プロセスで生成した 1 つ以上の線形回帰モデルとクローン検出対象プロジェクトを入力として以下の 2 つのステップでパラメータを決定する．

STEP Ⅱ-A STEP I で作成した線形回帰モデル群から利用者が目標再現率に対応する線形回帰モデルを選択する．

STEP Ⅱ-B クローン検出対象プロジェクトを入力として，選択した線形回帰モデルを用いてパラメータ値を決定する．

ただし，CCVolti が用いる LSH ライブラリ FALCONN は複数のパラメータを持ち，高速化のためのパラメータなどのパラメータが存在する．そこで，Cross-Polytope LSH を用いた類似探索の再現率に影響するパラメータを 3.1 節で示す．3.2 節以降，提案手法の各ステップの詳細を示す．

3.1 類似探索の再現率に関係するパラメータ

本節では，類似探索の再現率に影響を与えるパラメータを過不足なく抽出するために，本手法における再現率の定義と算出方法を示し，定義した再現率の期待値が Cross-Polytope LSH の衝突確率 $P_{T,K,L}$ に一致することを示す．本節では，類似探索の再現率を目標再現率以上の値とし，できるだけ高速となるために，提案手法では，再現率に影響を与える 3 つのパラメータ T, K, L を決定することを示す．

最初に，本研究における再現率 r を定義する．ベクトル集合に対して，U_{all} を閾値 θ 以上の類似度であるすべてのベクトル対の集合，U_{lsh} を LSH を用いた類似探索により検出したベクトル対の集合として，再現率 r は式 8 のように表される．ここで $|\cdot|$ は集合の要素数を表す．

$$r = \frac{|U_{\mathrm{lsh}}|}{|U_{\mathrm{all}}|} \tag{8}$$

LSH を用いた類似探索は，U_{all} に含まれる可能性があるベクトル対を LSH を用いて探索し，LSH で取得した全てのベクトル対の類似度を計算し，閾値 θ 以上の類似度であるベクトル対を得る．そのため，U_{lsh} は包含関係 $U_{\mathrm{lsh}} \subseteq U_{\mathrm{all}}$ を常にみたす．類似探索の適合率を，LSH を用いて検出したベクトル対集合 U_{lsh} に対して閾値 θ 以上であるベクトル対の割合とすると，包含関係 $U_{\mathrm{lsh}} \subseteq U_{\mathrm{all}}$ だから，類似探索の適合率は常に 1 となる．

あるプロジェクトに対して閾値を θ として類似探索を用いたコードクローン検出を実行したとき，類似探索の再現率は以下の手順で算出される．

STEP i LSH を利用せずすべてのベクトル対の類似度を計算して，類似度が閾値 0.9 以上のベクトル対を求め，ベクトル対の数 $|U_{\mathrm{all}}|$ を計測する．

STEP ii LSH を用いた類似探索を実行し，類似度が閾値 0.9 以上のベクトル対の数 $|U_{\mathrm{lsh}}|$ を計測する．

STEP iii 2 つの値を式 8 に代入して再現率 r を算出する．

次に，Cross-Polytope LSH の衝突確率と CCVolti

の類似探索の再現率の期待値が一致することを示す．2.2 節で説明した Cross-Polytope LSH を用いた類似探索において，衝突確率は式 7 のように表される．本研究における再現率の定義と，Cross-Polytope LSH の衝突確率の定義から以下の定理を導くことができる．

定理 1. 閾値 θ に対して類似探索を行うとき，$S(x,y) \geq \theta$ である 2 つのベクトル x, y に対する *LSH* の衝突確率 $P_{T,K,L}$ と，再現率の期待値 E は一致する．

Proof. 2.2.2 節より，確率 $P_{T,K,L}$ は L 個のハッシュテーブルに対する衝突確率を表し，ある 1 つの類似ペアが検出できる確率といえる．すべてのベクトル対集合 U_{all} のベクトルは，それぞれ確率 $P_{T,K,L}$ で衝突するから，衝突するベクトル対の数は二項分布に従う．従って，検出できるベクトル対の数の期待値 E_{lsh} は，ベクトル対の数 $|U_{\text{all}}|$ と 1 つのベクトル対が衝突する確率を用いて $E_{\text{lsh}} = |U_{\text{all}}| \times P_{T,K,L}$ ように計算される．また，再現率の期待値 E は，検出できるベクトル対の数の期待値 E_{lsh} とベクトル対の数 $|U_{\text{all}}|$ を用いて $E = \frac{E_{\text{lsh}}}{|U_{\text{all}}|} = P_{T,K,L}$ と表される．

よって，再現率の期待値と衝突確率は一致する．実際に，いくつかのパラメータ値の組み合わせにおいて 20 プロジェクトに対して実験を行い再現率を計測したところ，再現率の信頼区間に式 7 から算出した衝突確率が含まれていた． □

定理 1 より，再現率の期待値は衝突確率と一致する．Cross-Polytope LSH の衝突確率 $P_{T,K,L}$ は，Cross-Polytope LSH のパラメータである，区画の分割数 T，ハッシュ関数の数 K，ハッシュテーブル数 L に依存して増減する．従って，本手法は，再現率が目標再現率を超えつつ，できるだけ高速となるために，類似探索の再現率に影響を与える 3 つのパラメータ T, K, L の値を決定する．

3.2 STEP I-A パラメータ値の組候補を抽出

3 つのパラメータ T, K, L はそれぞれが独立に再現率と探索時間に影響を与えるため，再現率が目標再現率を超えつつ，できるだけ高速となるための T, K, L のパラメータ値を同時に決定する必要がある．3 つのパラメータ T, K, L の値を同時に決定するために，本節では，線形回帰モデルの目標変数とするラベルを示す．ラベルとは，T の上限の値を d とするとき，区画の分割数 T とハッシュ関数 K の値の組 (T, K) と 1 対 1 対応する式 $(K-1)\ln d + \ln T$ の値とする．

区画の分割数 T とハッシュ関数 K の値の組 (T, K) は，1 対 1 対応する式をハッシュテーブル 1 個当たりの衝突確率の式 5 から導出できる．式 6 の両辺の逆数に対して底 2 の対数を取り，右辺の各項に式 2 を誤差項を無視して代入し，式 9 を導く．

$$\ln \frac{1}{P_{T,K}} = \frac{1-\delta}{1+\delta}((K-1)\ln d + \ln T) \tag{9}$$

右辺に現れた式 $(K-1)\ln d + \ln T$ について，T, K が自然数であり d は T の上限であることから，任意の整数 $K_1 > K_2$ に対して $(K_1-1)\ln d + \ln T > (K_2-1)\ln d + \ln T$ である．従って，T と K の値の組合わせと $(K-1)\ln d + \ln T$ の値は 1 対 1 に対応している．これ以降，ハッシュ関数の数 K と区画の分割数 T の組を，ラベル $(K-1)\ln d + \ln T$ として表す．

ラベルと L の性質を調査するため，Linux Kernel を対象に予備実験をした．予備実験の内容と結果を付録 A に示す．実験結果から，K と T の組を表すラベルとハッシュテーブル数 L について，以下の性質を確認した．

- ラベルを増加すると，再現率は減少し，探索時間は短縮される．
- ハッシュテーブルの数を増加すると，再現率は増加し，探索時間は線形に増加する．
- さらに，探索時間に与える影響は，ラベルの方がハッシュテーブル数より大きい．

1, 2 番目の性質から，ラベルが小さいと再現率は高くなり，ハッシュテーブルの数が大きいと再現率は高くなるといえる．3 番目の性質から，ラベルのほうが探索時間に大きな影響を与えるため，ラベルの値をハッシュテーブルより先に決定することで，探索時間をより短くできると考えられる．ラベルの値に対して，ハッシュテーブル数 L は再現率を目標再現率以上になるために十分な値を決定する必要がある．できるだけ高速なラベルを決定するための線形回帰モデルと，ラベルから L を一意に決定する方法を STEP I-C に示す．したがって，本手法はラベルの

値を決定することで，再現率が目標再現率を超えつつ，できるだけ高速となるための 3 つのパラメータ T, K, L を自動的に決定する．

パラメータ T, K, L が取りうる値の組の候補として，ラベルとハッシュテーブル数に取りうる値の範囲の総当たりが考えられる．ラベル $label = (K-1)\ln d + \ln T$ に 1 以上の整数値を 1 刻みで与えるような値を区画の分割数 T とハッシュ関数の数 K に与える．ハッシュテーブル数 L には 1 以上の整数値を 1 刻みで与える．これらの総当たりのパラメータ値の組の候補に対して学習データを生成する．

3.3 STEP I-B 学習データの生成

本節では，学習用プロジェクトを用いて，線形回帰モデルを作成するための学習データを作成する．学習データは，学習用プロジェクトのコードブロック数，Cross-Polytope LSH に与えるパラメータ値の組に対する類似探索の再現率と探索時間とする．

STEP I で抽出したパラメータ T, K, L の組の候補を Cross-Polytope LSH に与え，CCVolti を用いて学習用プロジェクトに対するコードクローン検出を実行し，コードブロック数，類似探索の再現率，探索時間を計測する．計測結果を含めた 4 つのデータ，Cross-Polytope LSH に与えたパラメータ値の組，コードブロック数，類似探索の再現率，探索時間，の組を学習データとする．学習プロセスで作成した線形回帰モデルを適用プロセスで，再利用可能，あるいは未学習のプロジェクトに対して適用可能とするために，コードクローン検出対象プロジェクトと同じドメインの学習用プロジェクトを用意する必要がある．また，本手法が生成する線形回帰モデルはコードブロック数とパラメータ値の関係を示すため，コードブロック数が異なるプロジェクトを複数用意することが必要である．本評価実験では，コードブロック数が 2,000 以上ある OSS プロジェクトを収集し学習用データとして利用した．

3.4 STEP I-C 線形回帰分析

本節では，STEP I-B で作成した学習データを用いて，コードクローン検出対象プロジェクトのコードブロック数に対して，目標再現率を超えつつできるだけ探索時間を短縮するためのパラメータ値を推論する線形回帰モデルを作成する．STEP Ⅱ では STEP I で作成した線形回帰モデルからパラメータ決定に利用するモデルを選択する．そのため，STEP I の実施者である利用者あるいは開発現場の環境整備の担当者や我々は，CCVolti の利用者が求める目標再現率に対応する線形回帰モデル群を STEP I で作成する．

STEP I-C では，STEP I-B で取得した学習用プロジェクトに対する再現率の計測結果から，すべてのプロジェクトの再現率が目標再現率を超えるパラメータ組 $(label, L)$ をすべて抽出する．抽出されたパラメータ組 $(label, L)$ は，すべての学習用プロジェクトで再現率が目標再現率を超えるため，任意のプロジェクトで再現率の期待値が目標再現率を超える．そのため，線形回帰モデルによりラベルの値を決定すると一意にハッシュテーブル数を決定する．

再現率に基づいて抽出したパラメータ組に対して，STEP I-B で取得した学習用プロジェクトに対する探索時間の計測結果から，プロジェクトごとに探索時間が最も短いパラメータ組を抽出する．学習用プロジェクトごとに学習データから抽出したパラメータ組を用いて，コードブロック数を説明変数とし，ラベルを目標変数とする線形回帰モデルを作成する．線形回帰モデルの説明変数としてプロジェクトの規模を表すメトリクスにコードブロック数を用いた理由は，CCVolti がコードブロック単位のコードクローン検出手法だからである．一方，プロジェクトの規模を表すメトリクスは複数あるのに対し，コードブロック数のみを説明変数としたのは，行数やメソッド数などのメトリクス同士の相関が強く，多重共線性により結果を偏らせると判断したからである．

線形回帰モデルにコードクローン検出対象プロジェクトのコードブロック数を入力すると，ラベルの値が決定される．ラベルの値が決定されると，ラベルを表す式 $label = (K-1)\ln d + \ln T$ から，ラベルの値に対応する 2 つのパラメータ T, K を求めることができる．また，すべての学習用プロジェクトに対して抽出した目標再現率を超える再現率であるパラメータ組を用いて，ラベルの値に対して探索時間が最短の L

を決定する．L は小さくなるほど探索時間が線形に短くなるため，目標再現率を超える再現率であるパラメータ組に対して線形回帰モデルで決定されたラベルを含むパラメータ組の内，最も小さい L に決定する．従って，作成した線形回帰モデルは，コードブロック数を説明変数として，Cross-Polytope LSH に与える 3 つのパラメータ T, K, L を決定することができる．

3.5 STEP Ⅱ 線形回帰モデルを用いたパラメータ決定

本節では，学習プロセスで作成した線形回帰モデルを用いて，本手法の利用者がパラメータを決定する手順 STEP Ⅱ-A, Ⅱ-B を示す，

STEP Ⅱ-A では，STEP I で作成された線形回帰モデル群から，利用者が目標再現率に対応する線形回帰モデルを選択する．線形回帰モデル群は，利用者あるいは開発現場の環境整備の担当者や我々によって，目標再現率ごとに STEP I に従って事前に作成される．利用者は，作成された線形回帰モデル群から，速度を優先する目標再現率に対応する線形回帰モデルや，精度を優先する目標再現率に対応する線形回帰モデルを選択する．

STEP Ⅱ-B では，STEP Ⅱ-A で選択した線形回帰モデルを用いて，コードクローン検出対象プロジェクトを入力として，区画の分割数 T，ハッシュ関数の数 K，ハッシュテーブル数 L の値を決定する．STEP I-A で示した通り，他のパラメータは再現率に影響を与えないため，任意のパラメータ値を用いてよい．選択した線形回帰モデルにプロジェクトのコードブロック数を入力すると，ラベルの値が決定され，自動的にパラメータ組 (T, K, L) が決定される．ラベルと K, T に関する式 $label = (K-1)\ln d + \ln T$ から，ラベルの値に対してパラメータ T, K が一意に決まる．また，STEP I-C で示したようにラベルの値に対して探索時間が最短の L を一意に決定できる．線形回帰モデルを用いて，コードクローン検出対象プロジェクトのコードブロック数から，目標再現率を超えつつ，できるだけ高速なパラメータ値の組 (T, K, L) を決定する．

表 1　FALCONN のデフォルトのパラメータ値

パラメータ名	値
区画分割数 T	$2^{(r-1)}$ ($r = d \bmod \log_2 n$)
ハッシュ関数の数 K	$(\log_2 n - 1)/d$
ハッシュテーブル数 L	10

4 評価実験

本実験では，本手法の有効性を評価するために，LSH ライブラリ FALCONN のデフォルトのパラメータ値に対して，本手法に基づいて決定されたパラメータ値を比較する実験を行った．LSH ライブラリ FALCONN は，CCVolti が用いる Cross-Polytope LSH の 1 つの実装である．Cross-Polytope LSH の振舞いをパラメータ値により変更できるという利点があるため，評価実験においてもライブラリ FALCONN を用いる．本研究で実施する評価実験は，類似探索の探索時間と再現率という 2 つの観点で行う．

本手法の目的は，クローン検出の利用者が与えた目標再現率を満たし，かつ高速であるための Cross-Polytope LSH に与えるパラメータ値を決定することである．そこで本実験では，本手法で決定したパラメータ値と，表 1 に示した FALCONN のデフォルトのパラメータ値との比較実験を行い，類似探索の探索時間と再現率という 2 つの観点で有効性の評価を行う．そして，本手法の有効性を示すために，以下の 2 つの RQ に対して実験結果に基づいて考察を行う．

RQ1　本手法で決定したパラメータ値での再現率は目標再現率を超えているか？

RQ2　FALCONN のデフォルトのパラメータ値での探索時間に対して，本手法で決定したパラメータ値での探索時間は減少しているか？

以降，評価実験の詳細と結果，そこから得られる考察について述べる．

4.1 実験対象

本節では，本実験の実験対象プロジェクトと，比較対象とする FALCONN のデフォルトのパラメータ値について述べる．

表 2 学習用プロジェクト

プロジェクト	言語	コードブロック数	類似度 0.9 以上のクローンペア数	行数
Antlr 4.7.1	Java	2,787	3,566	92,976
SNNS 4.2	C	3,113	2,640	133,968
Maven 3.5.4	Java	3,468	3,448	133,238
Ant 1.10.5	Java	5,619	1,785	273,631
zfs-linux 2.19.1	C	6,806	1,119	259,771
HTTPD 2.4.35	C	7,626	1,501	255,468
ArgoUML 0.34	Java	8,696	5,038	391,837
Python 3.7.1	C	9,685	2,223	400,916
heimdal 2.19.1	C	12,083	2,335	549,880
Pig 0.17.0	Java	12,259	16,462	398,130
Tomcat 9.0.12	Java	13,488	9,043	562,549
Jackrabbit 2.16.3	Java	15,591	7,930	617,459
WildFly 14.0.1	Java	19,026	11,394	906,776
PostgreSQL 10.1	C	25,596	12,108	1,314,890
Camel 2.22.0	Java	50,515	508,298	1,953,433
gcc 8.2.0	C	93,104	847,841	4,079,924
OpenJDK 11.28	Java	110,364	53,347	4,766,529
FireFox 59.0.3	C	182,233	92,757	7,046,826
Linux Kernel 4.19	C	363,935	108,932	15,000,647
FreeBSD 11.2	C	379,014	196,714	15,694,482

本実験では，実験対象のプロジェクトとして，過去にコードクローン検出器の評価の実験対象にされたことがある 20 個のプロジェクトを用意した．表 2 は学習用プロジェクトの言語，コードブロック数，類似度が 0.9 以上のクローンペア数，および行数を示す．プロジェクトの順はコードブロック数によって並びかえた．クローン検出の対象とするプロジェクトは，C 言語で記述されたプロジェクトと Java で記述されたプロジェクトがそれぞれ 10 個ずつある．これらは，コードクローンに関する論文の評価実験等で用いられたプロジェクトから収集し，類似度が 0.9 以上のベクトル対集合 U_{all} が 1,000 以上あるプロジェクトを選択した [8] [11] [12] [14] [15] [17] [20] [22] [23].

これらの学習用プロジェクトに対して，CCVolti を用いてプロジェクトごとにコードクローンを検出し，Cross-Polytope LSH に与えるパラメータごとに再現率と類似探索の探索時間を計測する．クローンペアの基準として類似度の閾値 θ を CCVolti がデフォルトとする 0.9 を用いる．CCVolti が用いる Cross-Polytope LSH のパラメータの内，STEP I で抽出したパラメータ以外のパラメータ値は，ライブラリ FALCONN のデフォルトのパラメータ値に統一した．STEP I で抽出したパラメータには，ラベルとハッシュテーブル数に取りうる値の範囲の組み合わせを

表 3 再現率に影響を与えるパラメータ値の入力可能範囲

パラメータ名	値の範囲
区画の分割数 T	$1 \leq T \leq 1024$
ハッシュ関数の数 K	$K = 1, 2, 3$
ハッシュテーブル数 L	$1 \leq L$

総当たりを与える．ライブラリ FALCONN の 3 つのパラメータ T, K, L に入力可能な値は，表 3 に示す範囲の整数値である．ラベル $label = (K-1)\ln d + \ln T$ に $1 \leq label \leq 20$ の範囲で 1 刻みで値を与えるように，区画の分割数 T とハッシュ関数の数 K に値を与える．ハッシュテーブル数 L には，$1 \leq L \leq 30$ の範囲で 1 刻みで値を与える．

FALCONN のデフォルトのパラメータ値を表 1 に示す．FALCONN のデフォルトのパラメータ値は，最近点を高確率で検出できる値を規模に応じて与えられる．d はベクトル集合の各ベクトルの次元数を表し，n はベクトル集合に含まれるベクトルの数を示す．つまり，区画の分割数 T とハッシュ関数の数 K のデフォルトのパラメータ値は，ベクトル集合のベクトル数とベクトルの次元数から算出している．この算出方法の理由は開発者によって明示されていないが，ベクトル集合の密度に対して区画の大きさを調整することによって，クエリベクトルに対して最も近いベクトルを高確率で検出するためだと考えられる．また，ハッシュテーブル数 L は 10 に固定されている．ハッシュテーブルが 10 個あれば，限りなく 1 に近い確率で最も近いベクトルを探索できると考えられる．

4.2 線形回帰モデル作成

本節では，表 2 の実験対象プロジェクトを 20 個の学習用プロジェクトとして，目標再現率 0.9 に対して生成される線形回帰モデルを示す．また，生成した線形回帰モデルを各実験対象プロジェクトに適用して得られるパラメータの値を示す．

表 2 の実験対象プロジェクトを 20 個の学習用プロジェクトとして，目標再現率 0.9 に対する線形回帰モデルを作成する．STEP I-B に従って，各学習用プロジェクトに対してコードクローン検出を行い，パラメータごとの再現率と類似探索の探索時間を計測し，学習データを作成する．さらに，STEP I-C に従って，各学習用プロジェクトに対して目標再現率を超えつつ探索時間が短いパラメータ組を 1 つずつ抽出し，各プロジェクトのコードブロック数を説明変数とし，抽出したラベルを目標変数として線形回帰分析を行い，線形回帰モデルを作成する．ラベルを目標変数，プロジェクトのコードブロック数を説明変数として線形回帰分析すると，回帰係数は 1% 水準で統計的に有意であった．

目標再現率 0.9 に対して生成された線形回帰モデルの回帰係数は 1.87×10^{-7} であり，切片は 9.79 である．作成した線形回帰モデルは，学習データとして 2 つの言語の異なる規模のプロジェクトを用いており，C 言語や Java 言語などの手続き型言語に対して汎用的に利用可能な線形回帰モデルであると考えられる．本手法の評価実験を踏まえた汎用性に関する考察を 5.2 節で述べる．

生成された線形回帰モデルに対象プロジェクトのコードブロック数を入力すると，ラベルの値が決定される．ラベルを表す式は $label = (K-1)\ln d + \ln T$ であり，表 3 から $d = 1024$, $1 \leq T \leq 1024$ だから，決定されたラベルの値に対して 2 つのパラメータ T, K を算出できる．例えば，$label = 14$ の場合，$K = 2, T = 16$ となる．ラベルの値に対して STEP I-C1 で取り出されるパラメータ組の中で最も小さい L を決定し，探索時間が最短の L を得る．従って，作成した線形回帰モデルを用いて，対象プロジェクトのための Cross-Polytope LSH に与える 3 つのパラメータ T, K, L を決定できる．

4.3 実験方法

本実験では，本手法の有効性を示すために，0.8 以上 1 未満の 20 個の目標再現率に対して表 2 の 20 個のプロジェクトを用いて 10 分割交差検証を実施する．20 個のプロジェクトを 18 個の学習用プロジェクトと 2 個のコードクローン検出対象プロジェクトに分割する．18 個の学習用プロジェクトを用いて，0.8 以上 1 未満の 0.01 刻みの 20 個の目標再現率の各値に対して STEP I を実行し，各目標再現率に対する線形回帰モデルを作成する．作成した線形回帰モデル群と 2 個のコードクローン検出対象プロジェクトに対して STEP II を実行しパラメータ値の組を決定する．決定したパラメータを与えた Cross-Polytope LSH を用いて，CCVolti によってコードクローン検出を行い，再現率と探索時間を計測する．類似探索の再現率の計測は，3.1 節で説明した方法と同様の手順で行

表 4 目標再現率 0.9 に対する線形回帰モデルで決定される各プロジェクトのパラメータ値

プロジェクト	T	K	L
Antlr 4.7.1	1024	1	3
SNNS 4.2	1024	1	3
Maven 3.5.4	1024	1	3
Ant 1.10.5	512	1	3
zfs-linux 2.19.1	1024	1	3
HTTPD 2.4.35	1024	1	3
ArgoUML 0.34	1024	1	3
Python 3.7.1	1024	1	3
heimdal 2.19.1	1024	1	3
Pig 0.17.0	1024	1	3
Tomcat 9.0.12	1024	1	3
Jackrabbit 2.16.3	1024	1	3
WildFly 14.0.1	1024	1	3
PostgreSQL 10.1	1024	1	3
Camel 2.22.0	1024	1	3
gcc 8.2.0	128	1	3
OpenJDK 11.28	1024	1	3
FireFox 59.0.3	1024	1	3
Linux Kernel 4.19	16	2	5
FreeBSD 11.2	16	2	5

表 5 FALCONN のデフォルトのパラメータ値での実験結果

プロジェクト名	再現率	探索時間 [ms]
Antlr	1.000	1,411
SNNS	0.998	1,566
Maven	0.999	1,696
Ant	0.999	2,737
zfs-linux	0.990	3,363
HTTPD	1.000	3,806
ArgoUML	0.997	6,634
Python	0.996	7,171
heimdal	0.997	9,197
Pig	0.999	11,010
Tomcat	0.997	10,514
Jackrabbit	0.994	12,167
WildFly	0.991	15,457
PostgreSQL	0.995	20,011
Camel	0.986	2,158,552
gcc	0.996	5,755,016
OpenJDK	0.983	93,318
FireFox	0.976	183,147
Linux Kernel	0.974	480,423
FreeBSD	0.980	1,054,177

う．よって，適合率は常に 1 となる．10 分割交差検証のために，学習用プロジェクトとコードクローン検出対象プロジェクトを入れ替え，各目標再現率に対してすべての実験対象プロジェクトのためのパラメータ値を決定し，CCVolti によるコードクローン検出の再現率と探索時間を計測する．

実験環境は，CPU Intel Xeon 2.80 GHz，メモリ 32.0 GB，OS Windows 10 64 bit．Java 仮想マシンのヒープ領域 15 GB とした．クローンペアとするコサイン類似度の閾値 θ は，コードクローン検出法 CCVolti がデフォルトとする 0.9 とした．また，CCVolti が用いる Cross-Polytope LSH ライブラリ FALCONN に与えるパラメータの内，本手法の STEP I-A で抽出したパラメータ以外のパラメータ値は，FALCONN のデフォルトのパラメータ値に統一した．

4.4 実験結果

実験結果を表 4，表 5，図 3，図 4，図 5，付録 B に示す．表 4 は，目標再現率を 0.9 とした場合に本手法によって得られたパラメータ値を示す．表 5 は，FALCONN のデフォルトのパラメータ値での再現率と探索時間を計測した結果を示す．図 3 は，目標再現率毎に本手法で決定したパラメータ値と，FALCONN のデフォルトのパラメータ値に対して，再現率の比較を箱ひげ図で示した．このグラフの横軸は目標再現率の値を表し，縦軸はその目標再現率のときの本手法で決定したパラメータ値での実験を行ったときの再現率を表す．ただし，最も右の列は FALCONN のデフォルトのパラメータ値での結果を表している．図 4 は，本手法で決定したパラメータ値での探索時間に関して，FALCONN のデフォルトのパラメータ値での探索時間と比較した増減率を目標再現率毎に箱ひげ

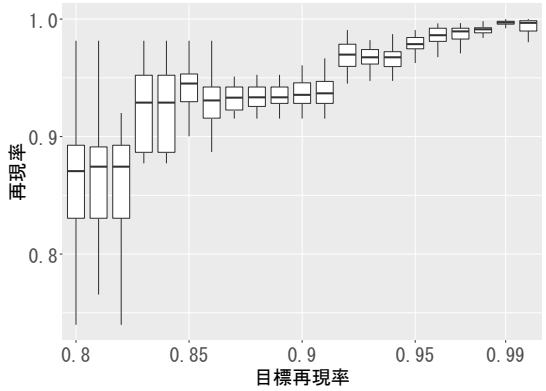

図 3　再現率比較

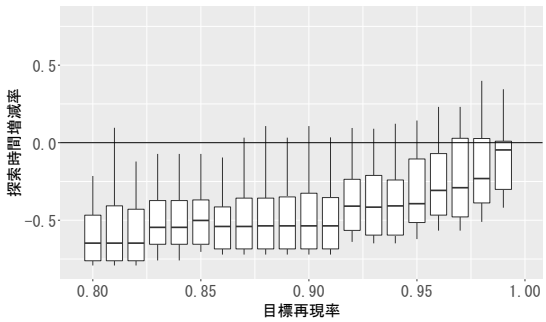

図 4　目標再現率毎の検出時間増減率

図で示した．この箱ひげ図の横軸は目標再現率の値を表し，縦軸はその目標再現率に対して本手法で決定したパラメータ値で実験を行ったときの類似探索の時間に関して FALCONN のデフォルトのパラメータ値での探索時間と比較した増減率を表している．図 5 は，本手法で決定したパラメータ値での探索時間に関して，目標再現率の変化に伴うプロジェクト毎の探索時間の増減率をパラメータ毎に調査した結果である．このグラフの横軸は目標再現率の値を表し，縦軸はその目標再現率のときの本手法で決定したパラメータ値でそのプロジェクトの探索時間の増減率を表しており，折れ線グラフは各プロジェクトの増減率の変化を表している．付録 B は，プロジェクトごとに各目標再現率と，FALCONN のデフォルトのパラメータ値に対して，CCVolti の探索時間を計測した結果を示す．

4.4.1　RQ1

図 3 から，75% 以上のプロジェクトにおいてどの目標再現率についても再現率が目標再現率を超えていることが分かる．これにより，あるプロジェクトに対して CCVolti によってコードクローン検出する際，本手法は多くの場合で再現率の目標値を満たすことができるといえる．

しかし，いくつかのプロジェクトで再現率が目標再現率を下回る場合を確認し，また，目標再現率 0.9 以下を与えているにも関わらず，再現率が 0.95 付近になる場合を確認した．再現率にばらつきが起こる原因は，クローンセット (互いにクローンペアとなるコードクローンの集合) 内のコード片の数の平均に差があることや，閾値に近い類似度であるクローンペアの数が多いことだと考える．また，FALCONN のデフォルトのパラメータ値での再現率はすべて 0.97 以上である．本手法で決定したパラメータ値が FALCONN のデフォルトのパラメータ値と同程度の再現率を得るためには，目標再現率を 0.98 以上にする必要がある．

RQ1 の答え

多くの場合で本手法は再現率の目標値を満たしている．ただし，再現率が目標再現率を下回るプロジェクトがいくつかあることを確認した．

4.4.2　RQ2

図 4 から，目標再現率が 0.96 以下では，16 プロジェクトが FALCONN のデフォルトのパラメータ値より高速であり，目標再現率が 0.98 以上の場合でも，14 プロジェクトで FALCONN のデフォルトのパラメータ値より高速であることが分かる．さらに，目標再現率が 0.91 以下の場合，18 プロジェクトに関して，デフォルトの探索時間からの増減率が −0.3 を下回っている．このように，多くの場合で FALCONN のデフォルトのパラメータ値より高速である．これにより，あるプロジェクトに対して CCVolti によりコードクローン検出する際，他の OSS を学習して生成した回帰モデルを用いて高速なパラメータを選択できるといえる．

また，図 5 から多くのプロジェクトに関して，目標再現率を下げることによって探索時間削減できて

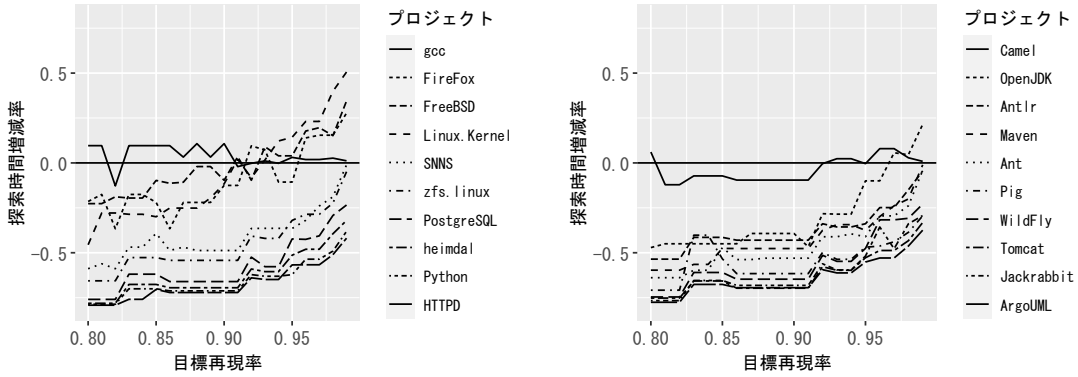

図 5 プロジェクト毎の検出時間 (左：C プロジェクト，右：Java プロジェクト)

いること分かる．特に，gcc と Camel を除く 18 プロジェクトにおいて，目標再現率 0.8 での探索時間は，目標再現率 0.99 のときの探索時間からおよそ半減できている．これにより，CCVolti の利用者が速度を優先したい場合，低めの目標再現率を設定することで，CCVolti の利用者が許容する再現率を満たし，かつ高速化できる．

図 5 と 3.4 節の表 2 から，クローンペア数が多いプロジェクトであるほど，FALCONN のデフォルトのパラメータ値との探索時間の増減率が変化しないことがわかる．特に，gcc と Camel の 2 つのプロジェクトに関しては，目標再現率 0.80 の場合に FALCONN のデフォルトのパラメータ値と比べて探索時間が増加している．この原因について考察するために，2.3.2 節で述べた CCVolti の検出の STEP 4 におけるクローンペアの検出の手順を，クローンペアの類似探索とフィルタリングに分割する．クローンペアのフィルタリングでは，クローンペアの重複を排除したり，被覆関係のクローンペアを排除する．FALCONN のデフォルトのパラメータ値での，クローンペアの類似探索とフィルタリングに分割してそれぞれの時間を計測した結果を表 6 に示す．FALCONN のデフォルトのパラメータ値の場合，クローンペア数が最も少ない zfs-linux では，フィルタリング時間が類似探索の約 0.04 倍であるのに対し，gcc では 81 倍，Camel では 56 倍もの時間をフィルタリングにかけている．クローンペアが多い場合，フィルタリングの時間が支配的になり，類似探索の高速化だけでは検出時間全体の時間短縮ができない．今後は，クローンペアのフィルタリングの改善が必要だと考えられる．

RQ2 の答え

多くの場合で本手法は FALCONN のデフォルトのパラメータ値より高速である．特に，gcc と Camel を除く 18 プロジェクトにおいて，目標再現率 0.8 での探索時間は，目標再現率 0.99 での探索時間からおよそ半減している．

5 考察

本章では，本手法を適用した CCVolti の有用性と本手法の妥当性の脅威を示す．

5.1 本手法を適用した CCVolti の有用性

本手法を適用した CCVolti の有用性について考察する．本実験では，CCVolti の再現率，適合率，F 値を計測していないが，類似探索の再現率を低く設定したときの CCVolti の精度について CCVolti の評価実験の結果に基づいて，類似探索の再現率が 0.8 のときの CCVolti の検出結果を推定し議論する．CCVolti の再現率とは，正解集合とするコードクローンに対して実際に検出された割合を指す．適合率とは，検出結果に対して正しかったコードクローンの割合を指す．F 値とは，再現率と適合率の調和平均によって表される値である．

表 6　類似探索時間とクローンペアのフィルタリング時間の比較

プロジェクト名	類似探索 [ms]	フィルタリング [ms]	フィルタリング / 類似探索
Antlr	1,188	223	0.188
SNNS	1,386	180	0.13
Maven	1,484	212	0.143
Ant	2,567	169	0.066
zfs-linux	3,223	141	0.044
HTTPD	3,634	172	0.047
ArgoUML	6,302	332	0.053
Python	6,992	180	0.026
heimdal	9,025	172	0.019
Pig	9,049	1,961	0.217
Tomcat	9,690	824	0.085
Jackrabbit	11,444	723	0.063
WildFly	13,836	1,622	0.117
PostgreSQL	18,604	1,407	0.076
Camel	37,529	2,121,023	56.517
gcc	69,973	5,685,043	81.246
OpenJDK	79,453	13,865	0.175
FireFox	133,360	49,787	0.373
Linux Kernel	264,757	215,666	0.815
FreeBSD	277,871	776,306	2.794

類似探索の再現率に対してクローン検出の適合率が一定であると仮定するとき，クローン検出の再現率と類似探索の再現率の関係を図 6 に示す．類似探索の再現率が r のとき，CCVolti が検出したクローンペアの数は r 倍となる．さらに適合率が一定である仮定から，CCVolti が検出した正解クローンの数も r 倍となるため，クローン検出の再現率も r 倍となる．例えば，類似探索の再現率が 1 のときクローン検出の再現率が 0.8 とすると，類似探索の再現率が 0.8 のときの CCVolti の再現率は 0.56 となる．したがって，類似探索の再現率が r のとき CCVolti の再現率は，類似探索の再現率が 1 の場合での再現率の r 倍となる．

表 7 は，CCVolti の評価実験の結果と，推定した類似探索の再現率が 0.8 のときの CCVolti の検出結果を示す [23]．表 7 から，クローン検出の再現率と F 値に関して，類似探索の再現率が 0.8 のときの推定

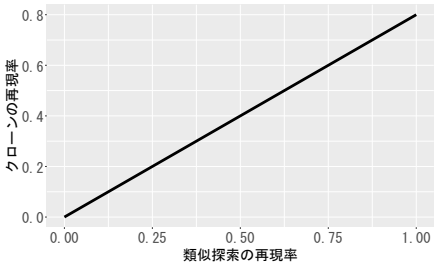

図 6　クローン検出の適合率が一定であるときの類似探索の再現率とクローン検出の再現率の関係

した結果が，関数クローン検出法，CCFinder，粗粒度クローン検出法のいずれよりも優れていることが分かる．従って，類似探索の再現率を 0.8 まで下げたとしても，既存手法である関数クローン検出法と CCFinder と同程度の精度になる．

本手法の実験結果と，本手法を適用した CCVolti

表 7 検出精度の比較

検出手法	クローン検出の適合率	クローン検出の再現率	F 値
CCVolti	0.68	0.70	0.69
関数クローン検出法	0.67	0.47	0.55
CCFinder	0.57	0.52	0.54
粗粒度クローン検出法	0.91	0.24	0.38
CCVolti (類似探索の再現率 0.8 の場合の推定値)	0.68	0.56	0.63

の精度の推定から，目標再現率を決定するときに以下のような指標が得られる．目標再現率を 0.98 以上とすると，FALCONN のデフォルトのパラメータ値と同程度の再現率を得られるパラメータ値の組が決定されることが考えられる．目標再現率 0.98 以上での実験結果では，70% のプロジェクトでデフォルトのパラメータより類似探索の時間を短縮できた．目標再現率を 0.8 とすると，既存手法と同程度の再現率や F 値となり，デフォルトのパラメータ値の半分の探索時間で検出できるパラメータ値の組が決定されることが考えられる．

5.2 妥当性の脅威

本手法によって作成される線形回帰モデルの信頼性について考察する．本手法の評価実験では，10 分割交差検証を行い，目標再現率ごとに 10 個の線形回帰モデルを作成した．10 個の線形回帰モデルの差異について調査した．目標再現率ごとのモデルごとに，回帰係数の平均と分散，切片の平均と分散を計算した．目標再現率毎に，10 分割交差検証に使用した 10 個の線形回帰モデルの回帰係数と切片の平均を，20 個のプロジェクトを学習して作成した線形回帰モデルの回帰係数と切片との差分を計算した．すべての目標再現率に対して，回帰係数の平均の差分は 10^{-9} 以下で，分散は 10^{-14} 以下だった．また切片の平均の差分は 10^{-2} 以下，分散は 10^{-1} 以下だった．どの目標再現率に対しても，10 分割交差検証で作成した線形回帰モデルは，20 個のプロジェクトを学習して作成した線形回帰モデルと大きな差異がないと言える．

次に，本手法の汎用性について考察する．本手法は C 言語と Java 言語のプロジェクトを用いて線形回帰モデルを作成する．評価実験では，10 分割交差検証で C 言語や Java 言語のプロジェクトに適したパラメータを決定し，多くの場合で目標再現率を超え，多くの場合でデフォルトのパラメータ値より高速なパラメータ値を決定した．しかし，HTML や Cobol など C 言語や Java 言語と特徴が大きく異なる言語で記述されたプロジェクトに対しては，本手法で作成した線形回帰モデルを適用して，コードブロック数から正しくラベルの値を決定できないと考えられる．従って，本手法の利用者は対象言語と似たプロジェクトを学習データとして本手法の STEP I に基づいて線形回帰モデルを作成する必要がある．

6 まとめと今後の課題

本研究では，コードクローン検出ツール CCVolti が用いる Cross-Polytope LSH に与えるパラメータ値を，クローン検出の利用者が与えた目標再現率を満たし，かつ高速になる値に決定する方法を提案した．そして，本手法を CCVolti が用いる Cross-Polytope LSH に適用し，異なる規模の 20 のプロジェクトに対してコードクローン検出を行った．その結果，本手法で決定されたパラメータ値は多くの場合でデフォルトのパラメータ値より高速であり，多くの場合で再現率が目標値を超えることを確認した．また，特に目標再現率が 0.91 以下の場合，75% のプロジェクトに関して本手法で決定されたパラメータ値はデフォルトのパラメータ値と比べて探索時間が 30% 以上下回っており，目標再現率を下げることによる高速化を確認した．

今後の課題として，gcc や camel などのクローンペアが多いプロジェクトの場合はクローンペアのフィルタリング時間が支配的となり，LSH の高速化だけでは

検出時間全体の時間短縮ができないため，クローンペアのフィルタリングの速度改善が挙げられる．また，本手法による検出速度を活かした応用として，本手法を適用した Cross-Polytope LSH を用いた CCVolti と他のコードクローン検出法に対して検出精度や検出時間を比較することや，CCVolti をリファクタリング支援ツールに適用することが挙げられる．さらに，限られた時間の中で CCVolti を用いたクローン検出を行う CCVolti 利用者のために，本手法と同様に目標検出時間を与え，検出時間が目標値を上回りつつ，できるだけ再現率が高く維持できるようなパラメータ値を決定する方法への利用を考えることが挙げられる．

謝辞 本研究は JSPS 科研費 JP18H04094, JP19K20240, JP20K11745 の助成を受けた．

参考文献

[1] Ailon, N. and Chazelle, B.: The fast Johnson-Lindenstrauss transform and approximate nearest neighbors, *SIAM Journal on computing*, 2009.
[2] Andoni, A. and Piotr, I.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, *Foundations of Computer Science*, Vol. 51, No. 1 (2006), pp. 117–122.
[3] Andoni, A., Piotr, I., Thijs, L., Ilya, R., and Ludwig, S.: Practical and Optimal LSH for Angular Distance, in *Proc. of NIPS*, 2015, pp. 1225–1233.
[4] Baeza-Yates, R. and Ribeiro-Neto, B.: *Modern Information Retrieval: The Concepts and Technology behind Search*, Addison-Wesley, 2011.
[5] Baker, B. S.: Parameterized duplication in strings: Algorithms and an application to software maintenance, *SIAM Journal on Computing*, Vol. 26, No. 5 (1997), pp. 1343–1362.
[6] 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, No. 6 (2008), pp. 1465–1481.
[7] Honda, H., Tokui, S., Yokoi, K., Choi, E., Yoshida, N., and Inoue, K.: CCEvovis: A clone evolution visualization system for software maintenance, in *Proc. of ICPC*, 2019, pp. 122–125.
[8] Jiang, L., Misherghi, G., Su, Z., and Glondu, S.: Deckard: Scalable and accurate tree-based detection of code clones, in *Proc. of ICSE*, 2007.
[9] 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. 18, No. 7 (2002), pp. 654–670.
[10] Komondoor, R. and Horwitz, S.: Using slicing to identify duplication in source code, in *Proc. of International Static Analysis Symposium*, Springer, 2001, pp. 40–56.
[11] Koschke, R. and Bazrafshan, S.: Software-Clone Rates in Open-Source Programs Written in C or C++, in *Proc. of SANER*, 2016.
[12] Li, Z., Lu, S., Myagmar, S., and Zhou, Y.: CP-Miner: Finding copy-paste and related bugs in large-scale software code., *IEEE Transactions on software Engineering*, 2006.
[13] Mayrand, J., Leblanc, C., and Merlo, E.: Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics., in *Proc. of ICSM*, Vol. 96, 1996, pp. 244.
[14] Rattan, D., Bhatia, R., and Singh, M.: Software clone detection: A systematic review, *Information and Software Technology*, Vol. 55, No. 7 (2013), pp. 1165–1199.
[15] Roy, C. K. and Cordy, J. R.: A survey on software clone detection research, Technical Report of Queen's University, School of Computing, No. 2007-541, 2007.
[16] Roy, C. K., Cordy, J. R., and Koschke, R.: Comparison and evaluation of code clone detection techniques and tools, *Science of Computer Programming*, Vol. 74, No. 7 (2009), pp. 470–495.
[17] Thongtanunam, P., Shang, W., and Hassan, A. E.: Will this clone be short-lived? Towards a better understanding of the characteristics of short-lived clones, 2018.
[18] 徳井翔梧，吉田則裕，崔恩瀞，井上克郎: 局所性鋭敏型ハッシュを用いたコードクローン検出のためのパラメータ決定手法, 電子情報通信学会技術研究報告信学技報, Vol. 117, No. 477, 2018, pp. 57–62.
[19] Tokui, S., Yoshida, N., Choi, E., and Inoue, K.: Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones, in *Proc. of SANER*, 2020, pp. 642–646.
[20] Wang, P., Svajlenko, J., Wu, Y., Xu, Y., and Roy, C. K.: CCAligner: a token based large-gap clone detector, in *Proc. of ICSE*, 2018.
[21] Weinberger, K. Q., Dasgupta, A., Langford, J., Smola, A. J., and Attenberg., J.: Feature hashing for large scale multitask learning, in *Proc. of ICML*, 2009.
[22] Xie, S., Khomh, F., and Zou, Y.: An empirical study of the fault-proneness of clone mutation and clone migration, in *Proc. of MSR*, 2013.
[23] 横井一輝, 崔恩瀞, 吉田則裕, 井上克郎: 情報検索技術に基づく細粒度ブロッククローン検出, コンピュータソフトウェア, Vol. 35, No. 4 (2018), pp. 16–36.

付録 A パラメータ分析のための調査

本付録では，3.2 節でラベルとハッシュテーブル数 L がそれぞれ再現率と探索時間に与える影響を調べるために実施した予備実験の内容と結果を示す．

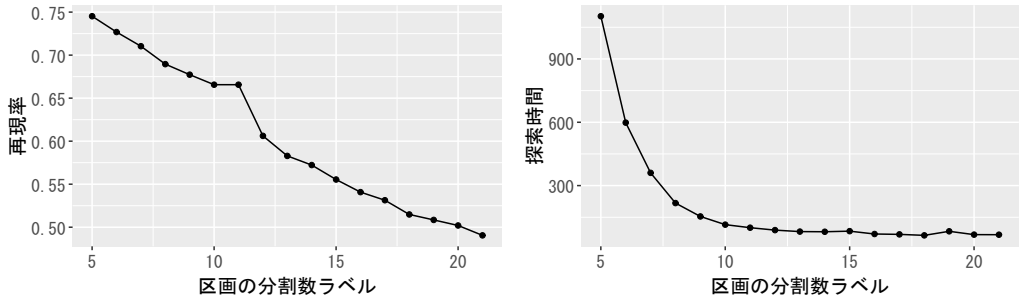

図 7 ラベル $(K-1)\ln d+\ln T$ と再現率や探索時間 [s] の関係

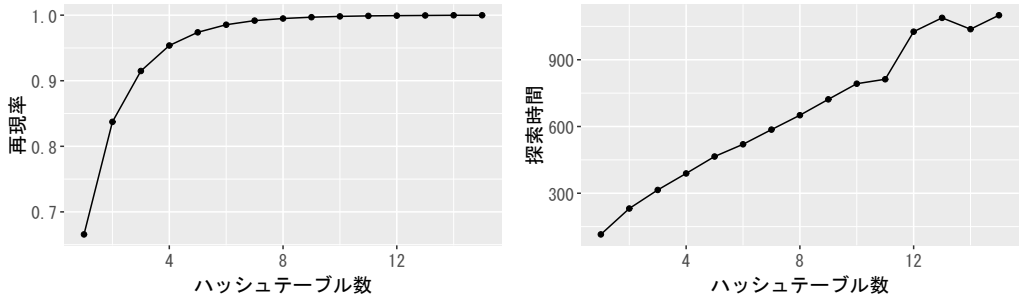

図 8 ハッシュテーブル数 L と再現率や探索時間 [s] の関係

実験環境は本手法の評価実験と同じ環境であり，約 15MLOC の Linux Kernel 4.19 をコードクローン検出対象とした．LSH ライブラリ FALCONN のパラメータの値を一定間隔で与え，CCVolti に適用してコードクローン検出を行い，パラメータごとの再現率と類似探索の探索時間を計測した．

ラベル $(K-1)\ln d+\ln T$

3.1 節で説明した通り，ハッシュ関数の数 K は区画の分割数 T の組をラベル $(K-1)\ln d+\ln T$ として扱う．ハッシュテーブルの数は $L=1$ に固定し，ラベルの値を取りうる値の範囲で 1 刻みで変化させて実験を行った．ラベルが再現率と探索時間に与える影響を表すグラフを図 7 に示す．横軸はラベルの値を表す．再現率のグラフより，ラベルを増加させると再現率が減少していることがわかる．これは，L を固定しているとき，ラベルを増加すると式 7 より Cross-Polytope LSH の衝突確率が単調に減少し，それによって定理 1 から再現率の期待値が減少することと一致する．また，探索時間のグラフより，ラベルの値が 10 以下のとき探索時間が大幅に減少しており，10 以上のラベルでは探索時間の大きな変化がない．これは，区画の分割数が少ないとき，類似するベクトル対の候補が多く探索され，類似度を計算するベクトル対の数が大幅に増えるからだと考えられる．つまり，探索時間はラベルの値に依存して大幅に変化すると考えられる．

ハッシュテーブル数 L

ベクトル対が L 個のハッシュテーブルの内，いずれかのハッシュテーブルで衝突するとき，類似するベクトル対の候補となる．ラベルの値が 10 となるように $K=1, T=1024$ とし，ハッシュテーブルの数 L には 30 以下の自然数を与えて実験を行った．

ハッシュテーブル数 L が再現率と探索時間に与える影響を表すグラフを図 8 に示す．横軸はハッシュテーブルの個数を表す．再現率のグラフより，ハッシュテーブルの増加に従って，再現率が増加していることがわかる．これは，ラベルを固定しているとき，ハッシュテーブルを増加すると式 7 より Cross-Polytope LSH の衝突確率が単調に増加し，それによって定理 1 から再現率の期待値が増加することと一

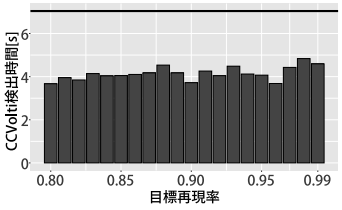

図 9　Antlr

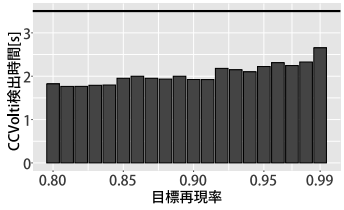

図 10　SNNs

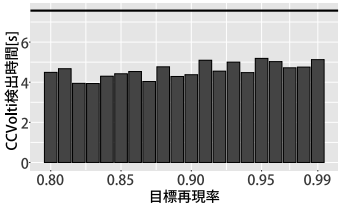

図 11　Maven

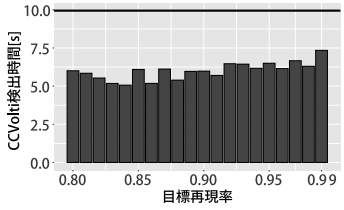

図 12　Ant

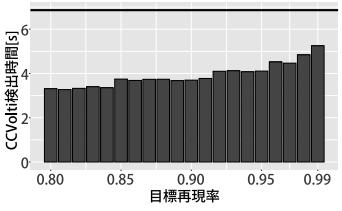

図 13　zfs-linux

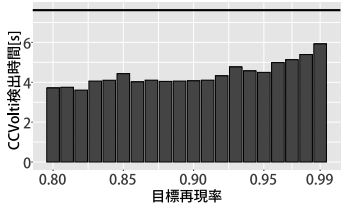

図 14　HTTPD

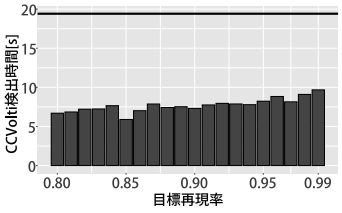

図 15　ArgoUML

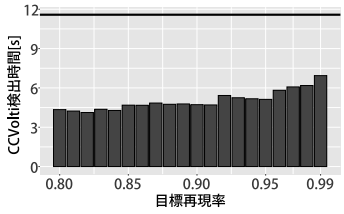

図 16　Python

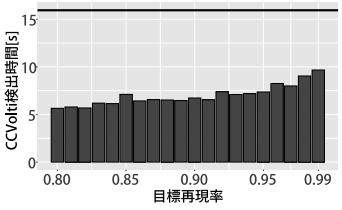

図 17　heimdal

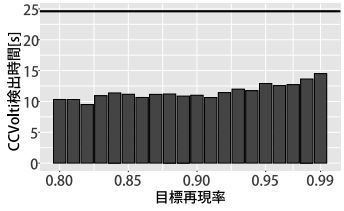

図 18　Pig

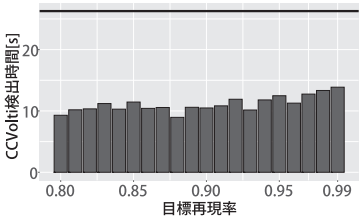

図 19 Tomcat

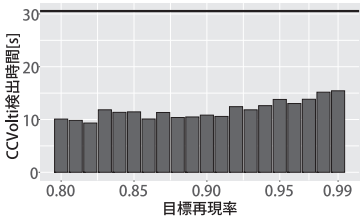

図 20 Jackrabbit

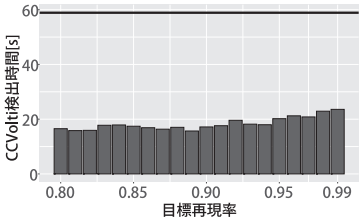

図 21 WildFly

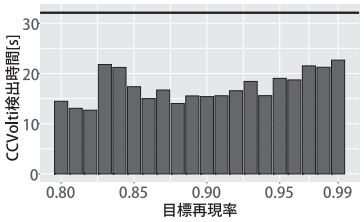

図 22 PostgreSQL

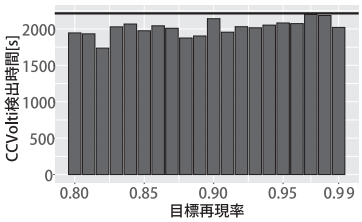

図 23 Camel

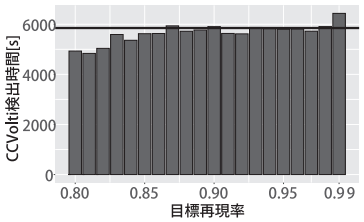

図 24 gcc

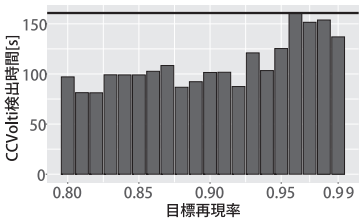

図 25 OpenJDK

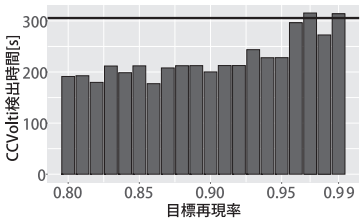

図 26 FireFox

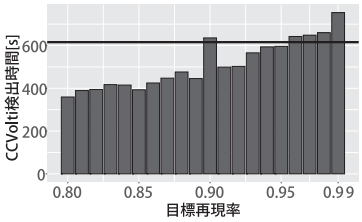

図 27 Linux Kernel

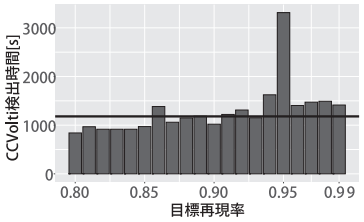

図 28 FreeBSD

致する．また，探索時間のグラフより，ハッシュテーブルの数を増加すると探索時間は線形に増加する．これは，ハッシュテーブルごとに，ハッシュ値の計算と衝突の判定処理をそれぞれ実行しているからだと考えられる．どのハッシュテーブルにも同じ T と K の値が与えられるため，式 5 より同じ衝突確率 $P_{T,K}$ での類似探索が実行される．つまり，L は他のパラメータに依存せず，探索時間を線形に増加させると言える．

付録 B　本手法によるクローン検出時間の削減

本付録では，本手法を用いて決定したパラメータを用いた場合，実際の CColti でのコードクローン検出時間について調査するために，表 2 に示した実験対象である各プロジェクトごとに，各目標再現率に対してコードクローン検出時間を計測したグラフをそれぞれ付録 A の図 9 から図 28 に示す．実験環境は 4.3 節に示した環境である．

グラフの横軸は 0.8 以上 0.99 以下の 0.01 刻みで与えた目標再現率の値を表し，縦軸は CCVolti のクローン検出時間 [s] を表す．棒グラフは目標再現率に対する CCVolti のクローン検出時間を示し，直線は，デフォルトのパラメータで CCVolti を実行した場合のクローン検出時間を示す．

徳 井 翔 梧

2019 年大阪大学大学院情報科学研究科博士前期課程修了．現在，同大学大学院情報科学研究科博士後期課程．および株式会社富士通研究所研究員．コードクローン管理手法の研究に従事．

吉 田 則 裕

2009 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員 (PD)．2010 年奈良先端科学技術大学院大学情報科学研究科助教．2014 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．2017 年より同大学大学院情報学研究科附属組込みシステム研究センター准教授 (改組による)．博士 (情報科学)．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

崔　恩瀞

2015 年大阪大学大学院情報科学研究科博士後期課程修了．同年同大学大学院国際公共政策研究科助教．2016 年奈良先端科学技術大学院大学情報科学研究科助教．2018 年より同大学先端科学技術研究科助教 (改組による)．2018 年より京都工芸繊維大学情報工学・人間科学系助教．博士 (情報科学)．コードクローン管理やリファクタリング支援手法に関する研究に従事．

井 上 克 郎

1984 年大阪大学大学院基礎工学研究科博士後期課程修了 (工学博士)．同年大阪大学基礎工学部情報工学科助手．1984 年～1986 年，ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 年大阪大学基礎工学部助教授．1995 年同学部教授．2002 年より大阪大学大学院情報科学研究科教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．