修士学位論文

題目

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

指導教員

井上 克郎 教授

報告者

徳井 翔梧

平成 31 年 2 月 6 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻 ソフトウェア工学講座

平成 30 年度 修士学位論文

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

内容梗概

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．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．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様々な手法が研究されている．

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 は情報検索技術を用いることにより，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検出できた．CCVolti におけるコードブロックは，関数と，関数内部の if，for 文等の中括弧で囲まれた部分である．CCVolti は，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 法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ブロッククローンとして検出する．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 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．

しかし，CCVolti には次の 2 つの問題点が挙げられる．1 つ目は，Cross-Polytope LSH を用いた類似度が閾値以上のベクトル対の探索において，約 10%の検出漏れが発生しており，検出漏れの割合を調整できない点である．LSH は類似度が閾値以上であるベクトル対に対して検出漏れを起こす可能性があり，高速化を優先するほど多くの検出漏れを起こす．2 つ目は，ベクトル間の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%を占めており，CCVolti のクローン検出時間が類似度が閾値以上のベクトル対の探索時間に大きく依存している点である．本研究において，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する．

これらの問題を解決するために，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，かつ高速であるための，Cross-Polytope LSH に与えるパラメータの決定手法

を提案する．まず，20 個のオープンソースソフトウェアプロジェクトに対して CCVolti を用いてコードクローンを検出し，Cross-Polytope LSH に与えるパラメータに関する予備実験を行う．その時に計測した類似度が閾値以上のベクトル対の探索時間と Cross-Polytope LSH の精度に関して分析した．分析結果を元に，コードクローン検出対象プロジェクトの規模に対して，目標再現率を上回るパラメータ値の中から高速に類似しているベクトル対を探索するためのパラメータ値を決定する回帰モデルを構築した．その後，構築した回帰モデルに基づいて，コードクローン検出対象プロジェクトの規模と目標再現率から，適したパラメータ値を決定し，そのパラメータ値を用いて類似度が閾値以上のベクトル対の探索を行う．

評価実験では，C 言語で記述された 10 個のプロジェクトと Java で記述された 10 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使用する CCVolti を用いてクローン検出し，再現率の調整ができることを確認した．さらに，目標再現率を 0.9 に設定した場合，Cross-Polytope LSH ライブラリ FALCONN のデフォルトのパラメータ値と比べて，検出時間を平均 46.6%削減することを確認した．

主な用語

コードクローン
ソフトウェア保守
局所性鋭敏型ハッシュ

目 次

1 まえがき **4**

2 関連研究 **6**

2.1 局所性鋭敏型ハッシュ(LSH) 6

2.2 近似最近傍探索アルゴリズム Cross-Polytope LSH 6

2.2.1 Cross-Polytope LSH のアルゴリズム 7

2.2.2 Cross-Polytope LSH を用いた類似探索 7

2.3 コードクローン検出ツール CCVolti 8

2.3.1 コードクローン 8

2.3.2 コードクローン検出 9

2.3.3 コードクローン検出ツール CCVolti のアルゴリズム 10

2.3.4 コードクローン検出ツール CCVolti の問題点 10

3 Cross-Polytope LSH に与えるパラメータ決定手法 **11**

3.1 STEP I パラメータの抽出 12

3.2 STEP II パラメータに関する予備実験と分析 13

3.2.1 Cross-Polytope LSH に与えるパラメータ 13

3.2.2 予備実験 14

3.2.3 パラメータ毎の結果と分析 15

3.3 STEP III パラメータ値の組み合わせの候補を抽出 16

3.4 STEP IV パラメータ決定のための線形回帰モデルの構築 17

4 評価実験 **19**

4.1 実験内容 19

4.2 実験結果と考察 20

5 まとめ **28**

謝辞 **29**

参考文献 **30**

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り，手作業でコードクローンを管理することが困難となる．この問題を解決するために，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 [23, 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[21] は情報検索技術 [5] と局所性鋭敏型ハッシュ(Locality-Sensitive Hashing，以降 LSH)[4] 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．CCVolti におけるコードブロックは，関数と，関数内部の if，for 文等の中括弧で囲まれた部分である．CCVolti は，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 [5] 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ブロッククローンとして検出する．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[4] 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．CCVolti は，Cross-Polytope LSH を利用するために，LSH ライブラリ FALCONN[1]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が検出できる [21]．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．

しかし，CCVolti には次の 2 つの問題点が挙げられる．1 つ目は，Cross-Polytope LSH を用いた類似度が閾値以上のベクトル対の探索において，約 10%の検出漏れが発生しており，検出漏れの割合を調整できない点である．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%を占めており，CCVolti のクローン検出時間が類似度が閾値以上のベクトル対の探索時間に大きく依存している点である．本

[1] https://falconn-lib.org/

研究において，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する．

これらの問題を解決するために，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，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．具体的には，まず，20 個のオープンソースソフトウェアプロジェクトに対して CCVolti を用いてコードクローンを検出し，Cross-Polytope LSH に与えるパラメータに関する予備実験を行う．その時に計測した類似しているベクトル対の探索時間と Cross-Polytope LSH の精度に関して分析した．分析結果を元に，コードクローン検出対象プロジェクトの規模に対して，目標再現率を上回るパラメータ値の中から高速に類似しているベクトル対を探索するためのパラメータ値に決定する回帰モデルを構築した．その後，構築した回帰モデルに基づいて，コードクローン検出対象プロジェクトの規模と目標再現率から適したパラメータ値が決定され，そのパラメータ値を用いて類似しているベクトル対の探索を行う．

評価実験では，C 言語で記述された 10 個のプロジェクトと Java で記述された 10 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を使用する CCVolti を用いてクローン検出し，再現率の調整ができることを確認した．さらに，目標再現率を 0.9 に設定した場合，LSH ライブラリ FALCONN のデフォルトのパラメータ値と比べて，検出時間を平均 46.6% 削減することを確認した．

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，Cross-Polytope LSH に与えるパラメータの調査結果と，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 に与えるパラメータ決定手法について述べる．4 章では，本手法の有効性の評価を行う．最後に 5 章では，まとめと今後の課題について述べる．

2　関連研究

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 [4]．近似最近傍探索問題は最近点問題の一種で，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中から，最も近いベクトルの候補である近傍ベクトルを近似的に高速に見つける問題である．

あ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衝突という．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．LSH では，入力ベクトルに対して，ハッシュの衝突が起こるデータセット中のベクトルを近傍ベクトルとする．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．入力ベクトルは近傍ベクトルのみ類似度を計算するベクトルを絞ることによって，高速に検出を行う．

LSH を用いて最近点を求めるときの時間計算量は $O(dn^{\rho})$ となる [4]．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す．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 p とし，類似度 $S(x, y)$ が $c\theta$ 以下となる 2 つのベクトルを衝突確率を q とすると，ρ は式 1 のように表される．LSH では，時間計算量の評価基準として ρ が用いられ，ρ が小さいほど n が時間計算量に与える影響が小さくなる．

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

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 [4]．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 のアルゴリズムと 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，コサイン類似度において類似したベクトルを探索するアルゴリズムである．2 つのベクトル x, y のコサイン類似度 $C(x, y)$ は式 2 ように表される．

$$C(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \tag{2}$$

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，正規直行基底 $\{\pm e_i\}_{1\leq i\leq d}$ の中で最も距離が近い基底の添え字 $\pm i$ を求める．最後に，x のハッシュ値を $\pm i$ とする．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 [19, 1].

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 3 ように表される [4].

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

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 [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 [4].

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，そのハッシュテーブルにおける 2 つのベクトル x, y の衝突確率 $P_{T,K}$ は式 4 のように表される [3]．

$$P_{T,K} = \prod_{i=1}^{K} P_{T_i} \tag{4}$$

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，2.2.1 節の式 3 で表したハッシュテーブル 1 個当たりの衝突確率 $P_{T,K}$ に対して，L 個のハッシュテーブルでの衝突確率 P_T は式 5 のように表される．

$$P_{T,K,L} = 1 - (1 - P_{T,K})^L \tag{5}$$

2.3 コードクローン検出ツール CCVolti

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．

2.3.1 コードクローン

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 [15]．

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

2.3.2 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多くの手法が研究されている．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 [23].

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 [6].

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 [21].

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単位の手法は，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 3 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 [21]．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．

2.3.3 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 [5] 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 [5]．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．

2.3.4 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 [22]．

- Cross-Polytope LSH を用いたベクトル間の類似度が閾値以上のベクトル対の探索において，約 10%の検出漏れが発生する場合があり，検出漏れの割合が安定していない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起こってしまうのは問題である．

- 2.3.3 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．

3　Cross-Polytope LSH に与えるパラメータ決定手法

本章では，2.3.4 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．CCVolti の利用者が用途に応じて目標再現率を変更できることにより，速度を優先するか精度を優先するかを決めることが可能となる．例えば，リファクタリング支援など，速度を優先するような目的で CCVolti を利用する場合，低い目標再現率を設定することで，高速にクローン検出できる．また，同時修正箇所の検出など，精度を優先するような目的で CCVolti を利用する場合，高い目標再現率を設定することで，高い精度でクローン検出できる．

まず，本研究における再現率 r について定義を行う．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 6 のように表される．ここで $|\cdot|$ は集合の要素数を表す．

$$r = \frac{|U_{\mathrm{lsh}}|}{|U_{\mathrm{all}}|} \tag{6}$$

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．

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は，以下の 4 つのステップで行う．

STEP I 再現率に関係するパラメータを抽出

STEP II 再現率と探索時間に影響を与えるパラメータに関する予備実験と分析

STEP III 再現率が目標再現率を超えるパラメータの組み合わせを，最も高速なパラメータの組み合わせの候補として選択

STEP IV STEP III で抽出したパラメータ値の組み合わせの中から，学習用プロジェクト毎に最も高速な組み合わせを実験結果から求め，与えられた目標再現率とプロジェクトの規模に対して適する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構築

STEP I では，Cross-Polytope LSH に与えるパラメータの内，再現率に影響を与えるパラメータを抽出する．STEP II では，STEP I で抽出したパラメータについて再現率と探索時間に関して実験をするために，パラメータに様々な組み合わせの値を与えた CCVolti を用いて，20 個のプロジェクトに対してクローン検出を行う．そして，パラメータ毎の特徴を再現率と検出時間に関して分析する．STEP III では，再現率が目標再現率を超えるパラメー

タ値の組み合わせを全て，最も高速な組み合わせの候補として抽出する．STEP Ⅱ と同様の実験を n 個のプロジェクトに対して行い，$n-1$ 個のプロジェクトで再現率が目標再現率を超えるパラメータの組み合わせを候補として抽出し，そのパラメータの組み合わせに対してラベル付けを行う．STEP Ⅳ では，STEP Ⅲ で抽出したパラメータの組み合わせの集合とラベルを元に，与えられた目標再現率とプロジェクトの規模に対して最も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構築する．以降の節で，それぞれのステップの詳細について説明する．

3.1 STEP Ⅰ パラメータの抽出

STEP Ⅰ では，再現率が目標再現率以上となるパラメータ値を決定するために，再現率に影響を与えるパラメータを抽出する．定理 1 より，再現率の期待値は衝突確率と一致するため，Cross-Polytope LSH の衝突確率について解析し，衝突確率に影響を与えるパラメータを抽出する．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 5 のように表される．衝突確率 $P_{T,K,L}$ は，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 を用いて表されるため，再現率はこの 3 つのパラメータに依存して変化する．

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 の衝突確率を $P_{T,K,L}$ と表すと，再現率の期待値 E は $P_{T,K,L}$ と一致する．*

$$E = P_{T,K,L} \tag{7}$$

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式 8 ように計算される．

$$E_{\text{lsh}} = |U_{\text{all}}| \times P_{T,K,L} \tag{8}$$

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 $|U_{\text{all}}|$ を用いて式 9 ように表される．

$$E = \frac{E_{\text{lsh}}}{|U_{\text{all}}|} = P_{T,K,L} \tag{9}$$

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 5 から算出した衝突確率が含まれていた． □

表 1: 抽出した Cross-Polytope LSH に与えるパラメータ

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	$K = 1, 2$
ハッシュテーブル数 L	$1 \leq L$

3.2 STEP Ⅱ パラメータに関する予備実験と分析

STEP Ⅱ では，STEP I で抽出した 3 つ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，およびハッシュテーブル数 L について再現率と検出時間に関する予備実験と分析を行う．予備実験では，様々な組み合わせを CCVolti に適用してクローン検出を行い，パラメータ毎の特徴を再現率と検出時間に関して分析する．以降，抽出したパラメータの性質について述べ，予備実験内容と結果について述べる．

3.2.1 Cross-Polytope LSH に与えるパラメータ

表 1 は STEP I で抽出したパラメータと，そのパラメータに与えられる値の範囲を示す．

区画の分割数 T

区画の分割数 T は，単位球をいくつの区画に分割するかを表す．Cross-Polytope LSH のアルゴリズムの性質上，T に与えることができる値はベクトルの次元数以下の自然数である．2.2.1 節の式 3 から，類似度 δ に対して，$C(x, y) > \delta$ をみたす 2 つのベクトル x, y の Cross-Polytope LSH における衝突確率 $P(x, y)$ は T の対数に比例し，T を増加させると衝突確率 $P(x, y)$ は減少する．

2.1 節で説明した Cross-Polytope LSH の時間計算量 $O(dn^{\rho})$ の ρ は，2.1 節の式 1 と 2.2.1 節の式 3 から，式 10 ように表され，T に依存して決まることが分かる．Andoni らは Cross-Polytope LSH における ρ の下限について考察を行っており，区画の分割数 T を増加させると時間計算量の評価基準 ρ は単調減少し，$T \to \infty$ のとき $\rho \to 1/7$ となることを示した [4, 2].

$$\rho = \frac{\frac{1-\delta}{1+\delta} \cdot \ln T + O_{\delta}(\ln \ln T)}{\frac{c^2(1-\delta)}{2-c^2(1-\delta)} \cdot \ln T + O_{c\delta}(\ln \ln T)} \tag{10}$$

ハッシュ関数の数 K

ハッシュ関数の数 K は，1 つのハッシュテーブルに対するハッシュ関数の数を表す．Cross-Polytope LSH において，ハッシュ関数の数 $K \geq 2$ のとき，$K - 1$ 個のハッシュ関数には区画の分割数 T に最大値を与え，残り 1 個のハッシュ関数には区画の分割数 T に指定したパ

ラメータ値を与える．このように定義することによって，区画の分割数 T を最大数以上の擬似的な分割が可能となる．ベクトルの次元数を d とおくと，区画の分割数 T の最大値を d と表せるので，式 5 より，衝突確率は式 11 のように表される．

$$P_{T,K,L} = 1 - (1 - {P_d}^{K-1} \cdot P_T)^L \tag{11}$$

また，式 3 と式 11 より，誤差項 $O_\delta(\ln \ln T)$ を無視した場合，ハッシュテーブル 1 個当たりの衝突確率 ${P_d}^{K-1} \cdot P_T$ に関して，式 12 が成り立つ．

$$\ln({P_d}^{K-1} \cdot P_T) = -\frac{1-\delta}{1+\delta}((K-1)\ln d + \ln T) \tag{12}$$

ここで，T に与える値は $1 \leq T \leq d$ をみたす 2 の累乗であることから，パラメータ値の組 (T, K) に対して，$(K-1)\ln d + \ln T$ は一意に定まる．

ハッシュテーブル数 L

ハッシュテーブル数 L は，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する．2.2.2 節の式 5 において，$0 \leq P_{T,K} \leq 1$ だから，ハッシュテーブル数 L の増加に伴って衝突確率 $P_{T,K,L}$ は増加する．類似するベクトル対のすべての候補に対して類似度の計算を行うため，ハッシュテーブル数 L の増加に伴って探索時間は線形に増加する．

3.2.2 予備実験

STEP I で抽出した各パラメータについて，再現率と検出時間に関して予備実験を行う．実験は，様々なパラメータの組み合わせを CCVolti に適用してクローン検出を行い，再現率と検出時間を計測し，各パラメータが与える影響について調べる．検出対象のプロジェクトには，約 15MLOC の Linux Kernel 4.19 を用いる．

CCVolti でのコードクローン検出後の再現率の計測は以下の手順で行う．

STEP i 2.3.3 節で述べた STEP 4 の類似探索において LSH を利用せずすべてのベクトル対の類似度を計算し，類似度が閾値 0.9 以上のベクトル対の数 $|U_{\text{all}}|$ をプロジェクト毎に計測

STEP ii 2.3.3 節で述べた STEP 4 の類似探索において LSH を用いて，類似度が閾値 0.9 以上のベクトル対の数 $|U_{\text{lsh}}|$ を計測

STEP iii 式 6 に従って再現率を計算

類似探索の検出時間は，2.3.3 節で述べた STEP 4 の類似探索にかかる時間を計測する．U_{lsh} は包含関係 $U_{\text{lsh}} \subseteq U_{\text{all}}$ を常にみたすため，適合率は常に 1 となる．実験環境は，CCVolti の評価実験と同じ環境である，CPU Intel Xeon 2.80GHz，メモリ 32.0GB，OS Windows 10 64bit. Java 仮想マシンのヒープ領域 15GB とした．

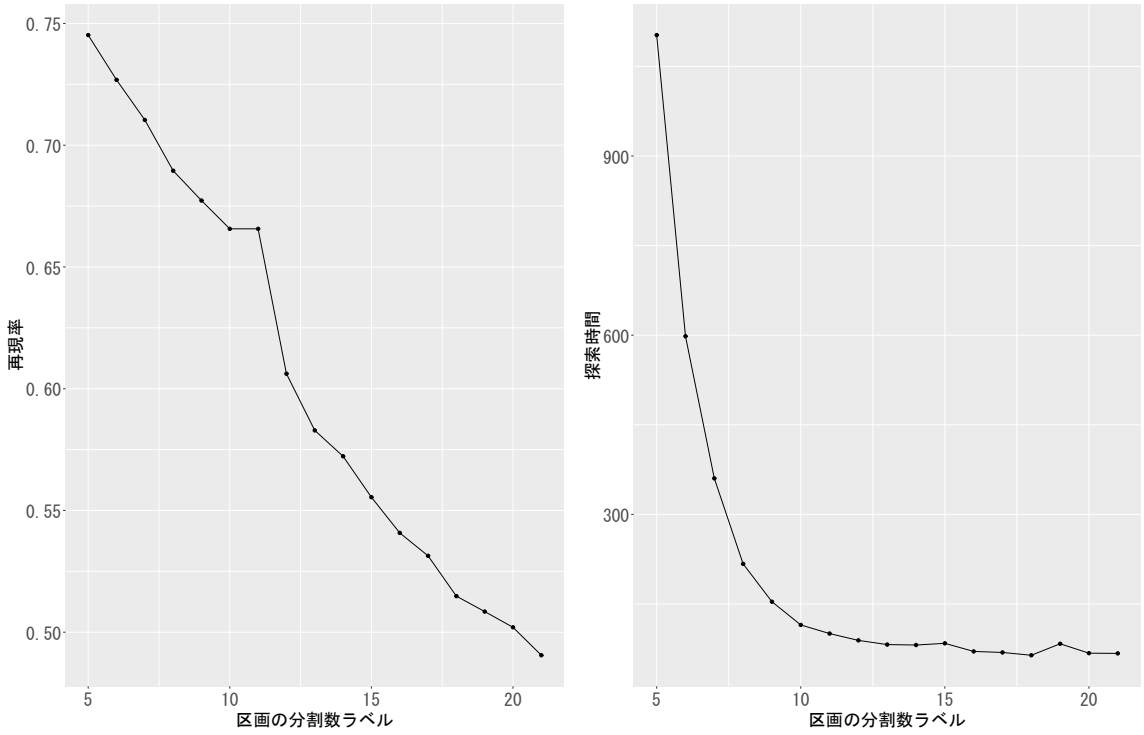

図 1: 区画の分割数 T と再現率や探索時間の関係

3.2.3 パラメータ毎の結果と分析

本節では，3.2.2 節で説明した予備実験結果をパラメータ毎に分析する．

区画の分割数 T とハッシュ関数の数 K

区画の分割数 T とは，単位球をいくつの区画に分割するかを表し，ハッシュ関数の数 K は 1 つのハッシュテーブルに対するハッシュ関数の数を表す．区画の分割数 T とハッシュ関数の数 K に関する実験結果を図 1 に示す．図 1 の横軸のラベルとは $(K-1)\ln d + \ln T$ であり，値が大きいほど区画の分割数が多いことを表す．探索時間は対数グラフで表している．グラフより，再現率に関してほぼ線形に減少している．また，探索時間に関して 10 以下のラベルでは大幅に減少しており，10 以上のラベルでは大きな変化がない．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ことが原因だと考えられる．つまり，ラベルの選択の仕方によって，探索時間が大幅に変化すると考えられる．

ハッシュテーブル数 L

ハッシュテーブル数 L に関する実験結果を図 2 に示す．区画の分割数を表すラベルは 10 とした．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．図 2 に示したグラフは L を変化させたときの類

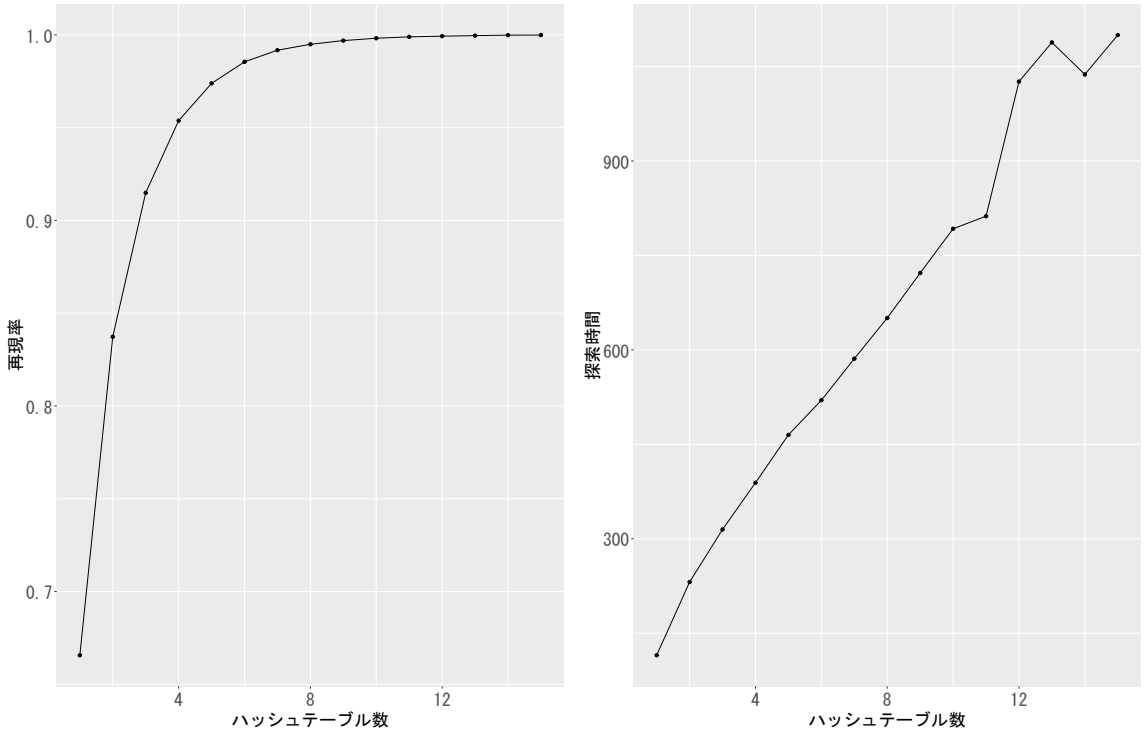

図 2: ハッシュテーブル数 L と再現率や探索時間の関係

似探索の時間の変化を計測した結果である．このグラフより，実際に L を増加させると探索時間は線形に増加する．そのため，類似探索の時間計算量は $O(dn^{\rho}) \times L = O(Ldn^{\rho})$ と表せる．つまり，探索時間に与える影響は，ハッシュテーブル数の方が区画の分割数より小さい．

3.3 STEP Ⅲ パラメータ値の組み合わせの候補を抽出

STEP Ⅲ では，再現率が目標再現率以上となるパラメータ値の組み合わせを最も高速なパラメータの組み合わせの候補として抽出する．パラメータ値の組み合わせの総当たりを Cross-Polytope LSH に与え，学習用プロジェクトに対し CCVolti を用いてクローン検出を行い，探索時間と再現率の計測を行う．再現率が目標再現率を超えるパラメータ値の組み合わせのうち，区画の分割数のラベルが同じ組み合わせは，ハッシュテーブル数が最も小さい組み合わせのみを抽出する．

例えば，目標再現率 0.9 に対するパラメータ値の組み合わせは図 3 の赤いセルのように抽出される．白いセルは再現率が目標再現率を超えないパラメータの組み合わせである．青いセルは再現率が目標再現率を超えるが，同じ区画の分割数でより少ないハッシュテーブルの数のパラメータの組み合わせがすでにあるため，抽出されないパラメータの組である．このように抽出することにより，区画の分割数のラベルに対して，一意にハッシュテーブル数と

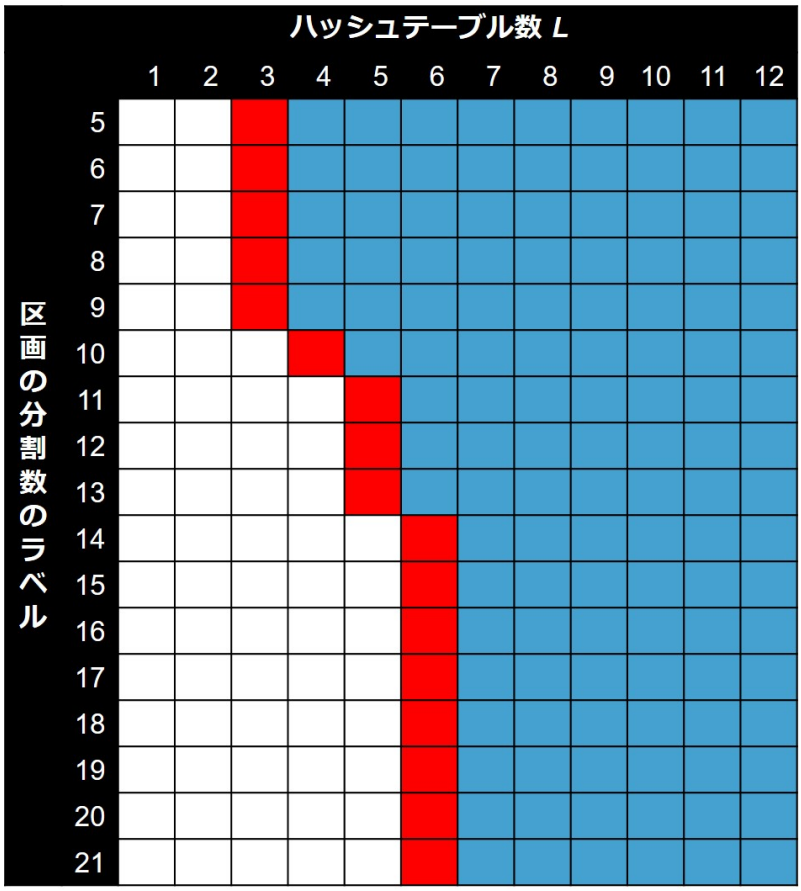

図 3: 目標再現率 0.9 に対して抽出されるパラメータの組み合わせ

の組み合わせが決定できる．

3.4 STEP IV パラメータ決定のための線形回帰モデルの構築

STEP IV では，与えられた目標再現率と検出対象プロジェクトの規模に対して，STEP III で抽出したパラメータ値の組み合わせの中から適する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構築する．まず，学習用プロジェクト毎に，与えられた目標再現率に対して STEP III で抽出したパラメータ値の組み合わせの中から，最も高速な組み合わせを実験結果から求める．次に，学習用プロジェクトのコードブロック数と，そのパラメータ値の組み合わせにおける区画の分割数のラベルを対応付け，線形回帰モデルを生成する．そして，生成した線形回帰モデルを元に，検出対象プロジェクトのコードブロック数に対して，適するパラメータ値の組み合わせを決定する．

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

数を用いる理由は，CCVolti のおいて特徴ベクトルの数とコードブロック数が一致するからである．与えられた目標再現率に対して STEP Ⅲ で抽出したパラメータ値の組み合わせに対し，最も高速なものの区画の分割数のラベルと，特徴ベクトルの数の間には強い相関があった．例えば，20 個のプロジェクトに対して，目標再現率 0.9 における最も高速なパラメータ値のラベルとコードブロック数の間の相関係数は 0.94 であった．一方，プロジェクトの規模を表すメトリクスが複数あるのに対し，コードブロック数のみ回帰モデルの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に悪影響を及ぼすと判断したからである．

4　評価実験

本研究では，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い，本手法の有効性の評価を行った．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で調整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．

表 2 に FALCONN のデフォルトのパラメータ値を示す．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．区画分割数 T とハッシュ関数の数 K は類似探索の対象ベクトル集合のベクトル数とベクトルの次元数に対して計算される．これは，クエリベクトルに対してデータセット中から最も近いベクトルが高確率で検出できるように，ベクトル集合の密度に対して区画の大きさを調整している．また，ハッシュテーブルを複数用意すると，ほぼ確実に最も近いベクトルを探索できるため，ハッシュテーブル数 L は 10 と固定されている．

以降，本章では，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　実験内容

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 2 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．表 2 において，d はベクトルの次元数を表し，n はベクトルの再現率の計測方法は，3.2.2 節で説明した方法と同様の手順で行う．よって，U_{lsh} は包含関係 $U_{\mathrm{lsh}} \subseteq U_{\mathrm{all}}$ を常に満たすため，適合率は常に 1 となる．実験環境は，予備実験と同じく，CPU Intel Xeon 2.80GHz，メモリ 32.0GB，OS Windows 10 64bit. Java 仮想マシンのヒープ領域 15GB とした．

表 2: 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

実験は，目標再現率に 0.8〜0.99 を 0.01 刻みで与え，本手法に基づいて決定されたパラメータ値と，FALCONN のデフォルトのパラメータ値を用いて行った．それらのパラメータ値を与えた Cross-Polytope LSH を用いて，CCVolti によってクローン検出を行い，再現率と探索時間を計測する．

実験対象のプロジェクトは，3.2.2 節の予備実験のとき使用した，表 3 に記されている 20 個のプロジェクトである．実験では 10 分割交差検証を用いて，実験対象の 20 個のプロジェクトを，18 個の学習用プロジェクトと 2 個の検出対象プロジェクトに分割して実験を行う．クローンペアとするコサイン類似度の閾値 θ は，クローン検出法 CCVolti がデフォルトとする 0.9 とした．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 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．

また，表 3 は実験対象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し，表 4 は各プロジェクトの参考 URL と実験に使用したバージョンのリリース月を示す．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 1000 以上あるプロジェクトを選択した [7, 8, 11, 12, 14, 16, 17, 18, 20, 21]．

4.2 実験結果と考察

本節では，本手法の有効性を示すために，実験結果の説明を行い，以下の 2 つの RQ に対して考察を行う．

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の増減率は減少しているか？

実験結果を表 5，図 4，図 5，図 6 に示す．表 5 には，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 4 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 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，FALCONN のデフォルトのパラメータ値での

表 3: 対象プロジェクト

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

探索時間と比較した増減率を目標再現率毎に箱ひげ図で示した．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 6 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．

表 4: プロジェクトの取得元

プロジェクト	URL	リリース
Antlr	https://github.com/antlr/antlr4	2017/12/10
SNNS	https://github.com/mwri/snns	2015/10/28
Maven	https://maven.apache.org/	2018/6/21
Ant	https://ant.apache.org/	2018/7/13
zfs-linux	https://launchpad.net/ubuntu/+source/git	2018/10/30
HTTPD	https://httpd.apache.org/	2018/9/22
ArgoUML	http://argouml-downloads.tigris.org/	2011/12/15
Python	https://www.python.org/	2018/9/26
heimdal	https://launchpad.net/ubuntu/+source/git	2018/10/30
Pig	https://pig.apache.org/	2017/6/19
Tomcat	http://tomcat.apache.org/	2018/9/10
Jackrabbit	http://jackrabbit.apache.org/jcr/	2018/8/3
WildFly	https://github.com/wildfly/wildfly	2018/9/5
PostgreSQL	https://www.postgresql.org/	2018/8/6
Camel	http://camel.apache.org/	2018/9/7
gcc	http://ftp.tsukuba.wide.ad.jp/software/gcc/	2018/7/26
JDK	https://jdk.java.net/11/	2018/8/22
FireFox	https://hg.mozilla.org/	2018/4/30
Linux Kernel	https://www.kernel.org/	2018/10/7
FreeBSD	https://github.com/freebsd/freebsd	2018/9/28

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？

図 4 からどの目標再現率についても，75%以上のプロジェクトで再現率が目標再現率を超えていることが分かる．従って，多くの場合で本手法は再現率を調整できている．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，本手法によって再現率を調整する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

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

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．

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の増減率は減少しているか？

図 5 から，目標再現率が 0.96 以下では，75%以上のプロジェクトが 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 0.97 以上の場合でも，50%以上のプロジェクトで 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

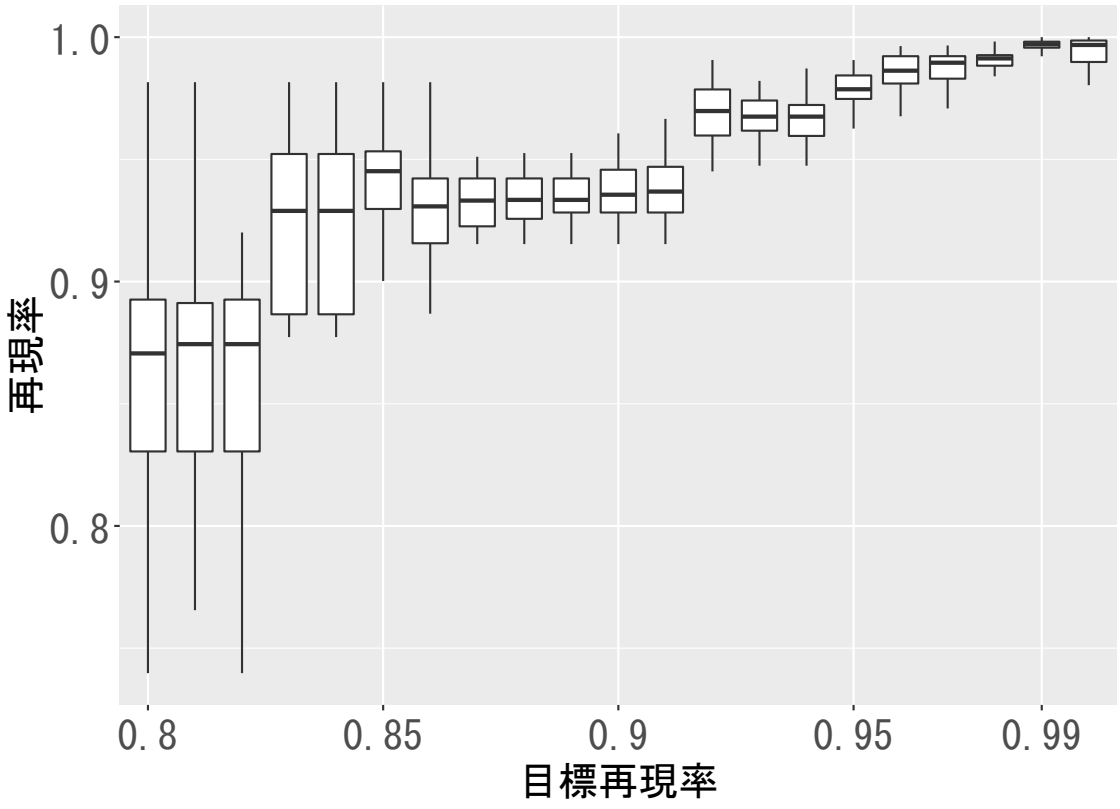

図 4: 再現率比較

標再現率が 0.91 以下の場合，75%以上の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．従って，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 6 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いること分かる．特に，gcc と Camel を除く 18 個の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 6 と 3.2 節の表 3 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，

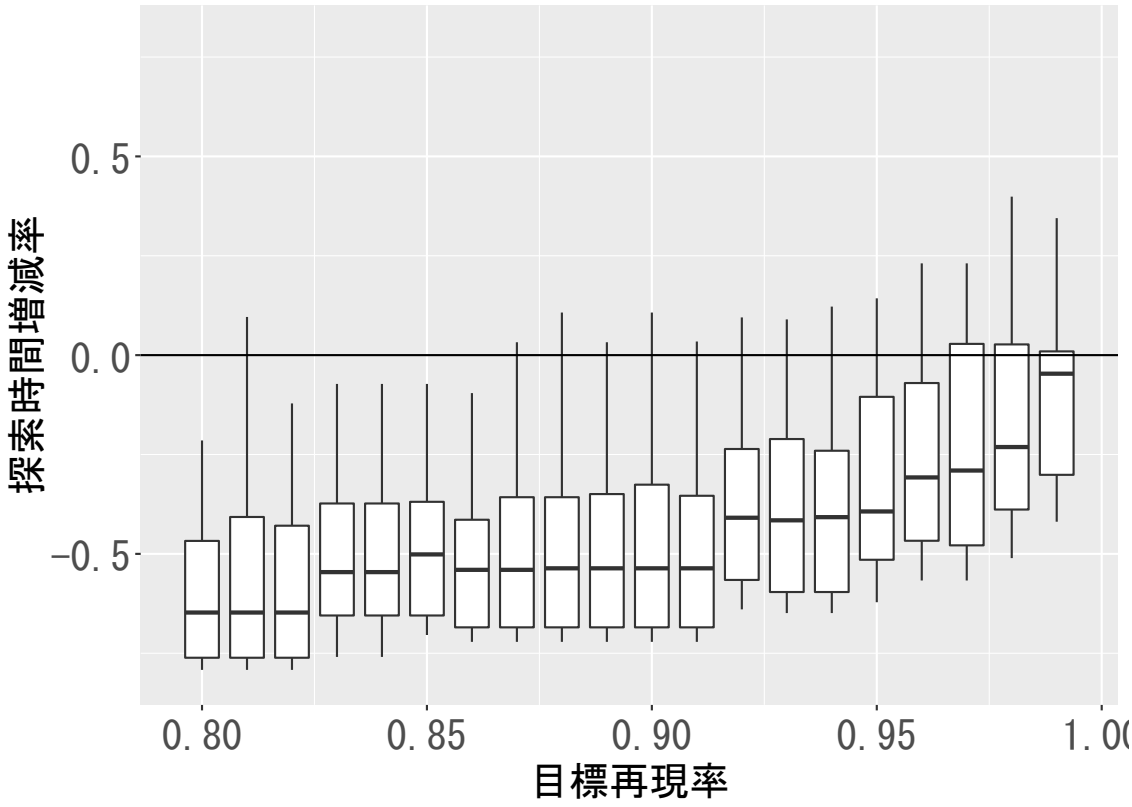

図 5: 目標再現率毎の検出時間増減率

2.3.3 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の排除を行う．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．

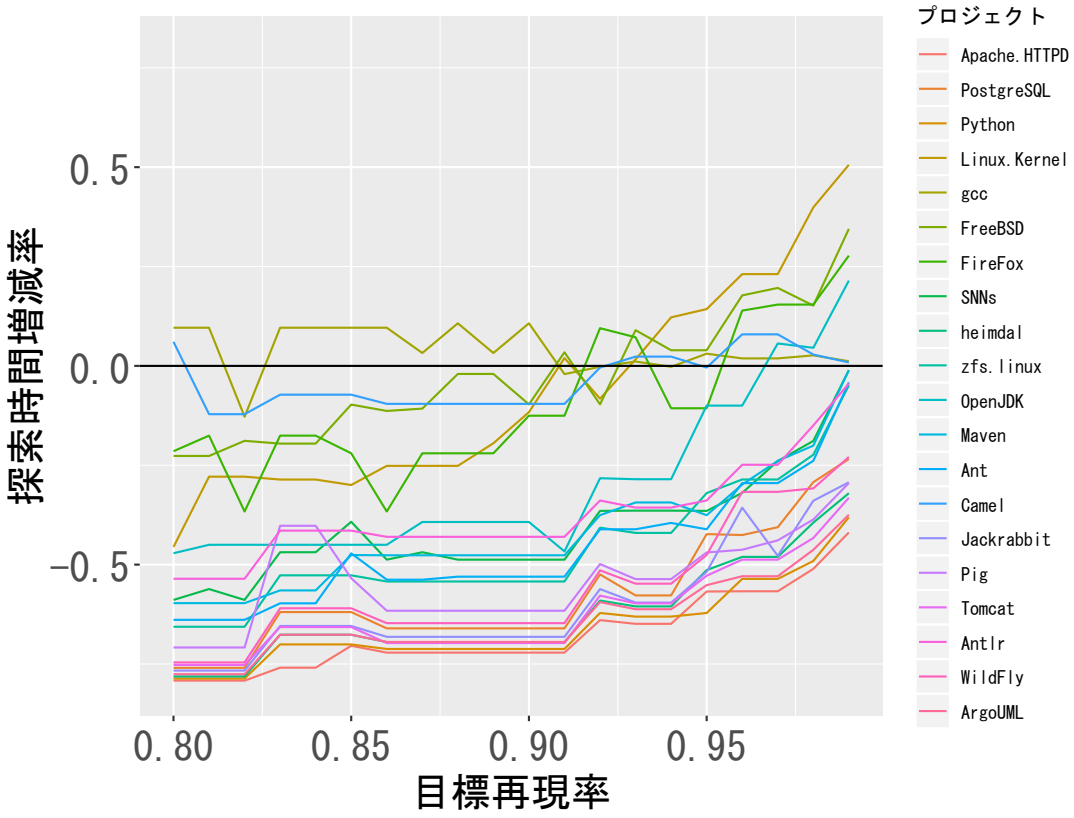

図 6: プロジェクト毎の検出時間

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]	$\frac{\text{フィルタリング}}{\text{類似探索}}$
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

5　まとめ

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，ほとんどの場合で再現率が目標再現率以上となることを確認し，70%のプロジェクトで Cross-Polytope LSH ライブラリ FALCONN のデフォルトのパラメータ値と比べて高速となることを確認した．また，目標再現率 0.91 以下の場合，75%のプロジェクトに関して，FALCONN のデフォルトのパラメータ値と比べて探索時間が 30%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できた．

一方で，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．今後は，クローンペアのフィルタリングの速度改善が必要だと考えられる．また，再現率調整が可能となった CCVolti と他のコードクローン検出法との精度に関する比較を行い，CCVolti をリファクタリング支援ツールに適用するなど，検出速度を活かした応用を考案する必要があると考える．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上克郎教授には，研究に関する御指導および御助言を賜りました．井上教授の御指導のおかげで本論文を完成させることができました．井上教授に心より深く感謝いたします．

名古屋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吉田則裕准教授には，直接の御指導および御助言を賜りました．本研究，ならびに研究室生活，研究への心構えなど，常に適切な指導を頂いたおかげで，本論文を完成させることができました．吉田准教授に心より深く感謝いたします．

奈良先端科学技術大学院大学先端科学技術研究科情報科学領域崔恩瀞助教には，直接の御指導および御助言を賜りました．本研究，ならびに研究室生活，研究への心構えなど，適切な指導を頂き，相談に乗っていただいたおかげで，本論文を完成させることができました．崔助教には心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻松下誠准教授には，分かりやすく説明するための御助言を頂き，論文執筆の助けとなりました．松下准教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻春名修介特任教授には適切な御指導および御助言を賜りました．春名特任教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻神田哲也助教には，日頃から親身に研究の相談や精神面の支えをしてくださいました．神田助教には心より深く感謝いたします．

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に心より深く感謝いたします．

参考文献

[1] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *Society for Industrial and Applied Mathematics*, Vol. 39, No. 1, pp. 302–322, 2009.

[2] Andoni Alexandr, Laarhoven Thijs, Razenshteyn Ilya, and Waingarten Erik. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proc. of SODA 2017*, pp. 47–66, 2017.

[3] Alexandr Andoni and Indyk Piotr. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Foundations of Computer Science*, Vol. 51, No. 1, pp. 117–122, 2006.

[4] Alexandr Andoni, Indyk Piotr, Laarhoven Thijs, Razenshteyn Ilya, and Schmidt Ludwig. Practical and optimal LSH for angular distance. In *Proc. of NIPS*, pp. 1225–1233, 2015.

[5] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley, 2011.

[6] Brenda S Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, Vol. 26, No. 5, pp. 1343–1362, 1997.

[7] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*, pp. 96–105. IEEE Computer Society, 2007.

[8] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. A token-based code clone detection tool-ccfinder and its empirical evaluation. *Techinal report, Osaka University, Department of Information and Computer Scineces, Inoue Laboratory*, 2000.

[9] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *Annual report of Osaka University: academic achievement*, Vol. 2001, pp. 22–25, 2002.

[10] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In *International static analysis symposium*, pp. 40–56. Springer, 2001.

[11] R. Koschke and S. Bazrafshan. Software-clone rates in open-source programs written in c or c++. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 3, pp. 1–7, March 2016.

[12] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering 32.3*, Vol. 32, No. 3, pp. 176–192, March 2006.

[13] Jean Mayrand, Claude Leblanc, and Ettore Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *icsm*, Vol. 96, p. 244, 1996.

[14] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh. Software clone detection: A systematic review. *Information and Software Technology*, Vol. 55, No. 7, pp. 1165–1199, 2013.

[15] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[16] James R. ROY, Chanchal Kumar; CORDY. A survey on software clone detection research. Vol. 541, No. 115, pp. 64–68, 2007.

[17] Patanamon Thongtanunam, Weiyi Shang, and Ahmed E. Hassan. Will this clone be short-lived? towards a better understanding of the characteristics of short-lived clones. 2018.

[18] Pengcheng Wang, Jeffrey Svajlenko, Yanzhao Wu, Yun Xu, and Chanchal K Roy. Ccaligner: a token based large-gap clone detector. In *Proceedings of the 40th International Conference on Software Engineering*, pp. 1066–1077. ACM, 2018.

[19] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proc. of ICML*, pp. 1113–1120, 2009.

[20] S. Xie, F. Khomh, and Y. Zou. An empirical study of the fault-proneness of clone mutation and clone migration. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 149–158, May 2013.

[21] 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, pp. 16–36, 2018.

[22] 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, Vol. 117, No. 477, pp. 57–62, 2018.

[23] 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, pp. 1465–1481, 2008.