機械学習に基づく有用なコードクローンの自動特定手法

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

概要：これまでにソースコード中からコードクローン (互いに類似するコード片) を自動的に検出するツールが多数開発されている．これらの検出ツールはコードクローンを多数検出するが，それらすべてがソフトウェア保守に有用であるとは限らない．さらに，あるコードクローンが有用であるか否かの判別基準は，個々の開発者により，あるいはコードクローン情報の使用目的などにより異なる可能性が高い．そこで本稿では，機械学習を用いた有用なコードクローンの自動特定手法を提案する．提案手法は，ツールによって検出されたコードクローンの一部を利用者に有用であるか否かに分類してもらい，それらを学習データとして残りのコードクローンからその利用者にとって有用なコードクローンを自動的に特定する．また，提案手法を実装し，被験者を用いて評価実験を行った．その結果，有用であるか否かの判断が利用者によって異なることを確認するとともに，提案手法が全体の 20%程度のデータを学習させることで，およそ 70%以上，最良の場合約 90.1%の精度で予測ができることを確認した．

A Method for Identifying Useful Code Clones based on Machine Learning Techniques

Jiachen Yang[1,a)]　Keisuke Hotta[1,b)]　Yoshiki Higo[1,c)]　Hiroshi Igaki[1,d)]　Shinji Kusumoto[1,e)]

Abstract: Currently, there is a variety of code clone detection tools. However, results from clone detectors contain plentiful useless code clones. Thus, users of clone detectors need to identify useful code clones from a flood of code clones, which requires much efforts. In addition, judging whether a code clone is useful or not varies from user to user and from purpose to purpose. Therefore, it is difficult to define what code clones are useful for software maintenance. This paper proposes a filtering method for code clones based on machine learning techniques. The proposed method has its users mark on some of code clones whether they are useful or useless. Firstly, the proposed method learns the judgement of the individual user, and then predicts unmarked code clones whether they are useful or not. We have implemented the proposed method as a web-based system, and conducted a case study with subjects. As a result, we confirmed that the judgement of whether a code clone is useful or not varies from user to user. Moreover, we confirmed that the proposed method can predict the judgement of code clones by users with over 70% accuracy, in some cases it can predict the judgement with over 90% accuracy.

1. まえがき

近年，ソフトウェア工学における研究対象としてコードクローンが注目されている．コードクローンとは，ソースコード中の同一，あるいは類似するコード片のことを指す．一般的にコードクローンの存在はソフトウェアの保守に悪影響を与えると考えられており，これまでにコードクローンの検出技術，及びその除去技術が盛んに研究されている．また，コードクローンをソースコード中から自動的に検出するツールも多数開発，公開されている [1], [2], [3], [4].

コードクローン検出ツールは対象となるソフトウェア中から特定したコードクローンをすべて提示するため，提示されるコードクローンの量は膨大になる．しかし，これらのコードクローンすべてがソフトウェア保守にとって有用であるとは限らない．このため，開発者は検出ツールによって提示された膨大なコードクローンの中から検出目的に即したコードクローンを特定する必要があるが，この作業は非常に大きな作業量を要する．さらに，あるコード

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

クローンが有用か否かを判別する基準は，個々の開発者によって，あるいはコードクローンの検出目的によって異なる．したがって，どのようなコードクローンがソフトウェア保守に有用かを一般化することは難しい．

これまでにメトリクスなどを用いてコードクローンを分類する，あるいは有用でないものを除外する手法が提案されている [5], [6]．しかし，利用者ごと，あるいは利用目的ごとの違いを考慮したコードクローンのフィルタリング手法はこれまでに提案されていない．

本稿では，開発者がソフトウェア保守に有用なコードクローンを特定する作業を支援するために，機械学習を用いたコードクローンの自動フィルタリング手法を提案する．提案する手法では，まず利用者に一部のコードクローンを “有用” か “有用でない” かに分類してもらう．その後，それらを学習データとして，残りのコードクローンを “有用” または “有用でない” に自動的に分類し，利用者に提示する．提案手法を用いることで，利用者は一部のコードクローンに対して自身が有用と思うか否かを入力するのみで，膨大な検出結果から自身にとって有用なコードクローンのみを抽出することが可能となる．これにより，有用なコードクローンの特定に要するコストを低減させることができる．さらに，利用者ごとに学習データを作成するため，利用者や利用状況による有用か否かの判別基準の違いにも対応できる．

2. 提案手法

本節では，提案するコードクローンのフィルタリング手法について述べる．

2.1 入出力

提案手法が入力として必要とするものは以下の通り．

(1) 対象ソフトウェアのソースコード
(2) 対象ソフトウェア中に存在する全クローンセットの位置情報
(3) 学習データ

ここでクローンセットとは，互いにコードクローン関係にあるコード片の集合を指す．また (3) の学習データとは，対象ソフトウェアから検出されたクローンセットの一部であり，利用者によって “有用” あるいは “有用でない” のいずれかの分類が付与されている．

2.2 処理の流れ

提案手法は与えられた入力情報をもとに，利用者によって分類されていないすべてのクローンセットについて，それらが “有用” か “有用でない” かを自動的に判別し，提示する．**図 1** に提案手法を用いたフィルタリング処理の流れを示す．提案手法を利用する際，準備として以下の (1), (2) の作業が必要となる．

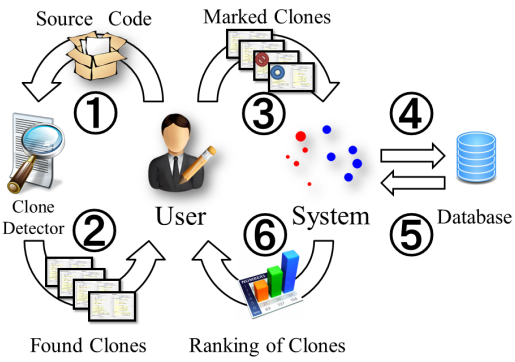


図 1 提案手法を用いたフィルタリング処理の流れ
Fig. 1 Overall Workflow of Filtering with the Proposed Method

(1) 利用者が対象システムのソースコードを検出ツールに入力する．
(2) 検出ツールは入力されたソースコードからクローンセットを検出する．

個々の利用者が提案手法を利用する手順は以下の通り．

(3) 検出ツールにより検出されたクローンセットの中のいくつかを，利用者がその有用性に応じて “有用” か “有用でない” かに分類し，登録する．
(4) 提案システムは利用者によって分類されたクローンセット (学習データ) をデータベースに格納する．
(5) 提案システムは学習データを分析し，機械学習を用いてそれらの特徴を抽出する．
(6) 提案システムは (5) で抽出した特徴をもとに，学習データに含まれていないクローンセットすべてに対し，それらの有用性を判別する．

2.3 機械学習手法

提案手法における機械学習手法は，クローンセット間の類似度の算出，並びにそれを用いた有用性の判別の 2 つのステップから成る．以降それぞれについて述べる．

2.3.1 クローンセット間の類似度の算出

提案手法では，各クローンセットを構成するコード片のトークン型列 (トークンの種類をアルファベットとする文字列) をもとにクローンセット間の類似度を算出する．コード片のトークン型列を用いることで，細かな文字列の違いを吸収することが可能となり，算出するクローンセット間の類似度，及び最終的な予測精度の向上が期待できる．**図 2** にトークン型列の例を示す．

提案手法では，類似度の算出に TF-IDF 値 (Term Frequency - Inverse Document Frequency)[7] を用いる．以降の説明では，1 つのクローンセットを d，クローンセットの集合を D と表記する．

TF-IDF 値の算出に際し，まず各クローンセット d を構成するコード片のトークン型列から N-Gram を抽出する．

```
struct blob *lookup_blob(struct object *obj)
{
   if (!obj->type)
      obj->type = OBJ_BLOB;
   if (obj->type != OBJ_BLOB)
      return NULL;
   return (struct blob *) obj;
}
```

(a) ソースコード

```
STRUCT ID TIMES ID LPAREN STRUCT ID TIMES ID RPAREN
LBRACE
  IF LPAREN LNOT ID RPAREN
    ID ARROW ID EQUALS ID SEMI
  IF LPAREN ID ARROW ID NE ID RPAREN
    RETURN ID SEMI
  RETURN LPAREN STRUCT ID TIMES RPAREN ID SEMI
RBRACE
```

(b) トークン型列

図 2 トークン型列の例

Fig. 2 An Example of Token Types Seqneuce

例えば，図 2(a) のソースコードから得られるトークン型列 (図 2(b)) に対して $N = 3$ とすると，以下のような N-Gram が得られる．

```
STRUCT   ID     TIMES
         ID     TIMES    ID
         TIMES  ID       LPAREN
                ID       LPAREN    STRUCT
                         LPAREN    STRUCT    ID
                                   STRUCT    ID      TIMES
                                             ...
```

このようにして得られた N-Gram のうちの 1 つを t と表記する．次に，得られたすべての N-Gram に対して `term frequency` $tf(t,d)$ を式 (1) を用いて算出する．

$$tf(t,d) = \frac{|t : t \in d|}{|d|} \tag{1}$$

$tf(t,d)$ はあるクローンセット d 中に N-Gram t がどの程度の割合で出現するのかを表している．図 2 の場合，すべての N-Gram t に対して $tf(t,d)$ を算出すると以下のようになる．

```
STRUCT ID TIMES      :  0.06818181818181818
ID TIMES ID          :  0.045454545454545456
TIMES ID LPAREN      :  0.022727272727272728
ID LPAREN STRUCT     :  0.022727272727272728
...
```

続いて，`inverse document frequency` $idf(t,D)$ を式 (2) を用いて算出する．$idf(t,D)$ は t がどれだけ多くのクローンセット内に出現するのかを表す指標であり，多くのクローンセット内に出現している場合にその値が小さくなる．

$$idf(t,D) = \log \frac{|D|}{1 + |d \in D : t \in d|} \tag{2}$$

次に，式 (1) 及び式 (2) を用いて，$tfidf$ を算出する (式 (3)，式 (4))．

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D) \tag{3}$$

$$\overrightarrow{tfidf(d,D)} = [tfidf(t,d,D) \quad \forall t \in d] \tag{4}$$

$tfidf(t,d,D)$ は個々の N-Gram に対する TF-IDF 値を表しており，$\overrightarrow{tfidf(d,D)}$ は $tfidf(t,d,D)$ を連結して作成したベクトルである．

この $\overrightarrow{tfidf(d,D)}$ を用いて，2 つのクローンセット a，b 間の類似度 $nsim(a,b,D)$ を以下のように算出する．

$$sim(a,b,D) = \overrightarrow{tfidf(a,D)} \cdot \overrightarrow{tfidf(b,D)} \tag{5}$$

$$nsim(a,b,D) = \begin{cases} 0 & , sim(a,b,D) = 0 \\ \frac{sim(a,b,D)}{|sim(a,b,D)|} & , \text{otherwise} \end{cases} \tag{6}$$

2.3.2 有用性の判別

本小節では，学習データとして与えられた分類済みのクローンセット集合を用いて，未分類のクローンセットが “有用” か否かを判別する方法について述べる．

学習データとなるクローンセットの集合を M，判別の対象となる未分類のクローンセットを $t(t \notin M)$ と表記する．また，M に含まれるクローンセットのうち，“有用” と分類されているクローンセットの集合を M_i，“有用でない” と分類されているクローンセットの集合を M_n とする．仮定より，$M_i \cap M_n = \emptyset$，$M_i \cup M_n = M$ である．このとき，M_i，M_n のそれぞれに対し，式 (7) で定義する $poss(t, M_x)(M_x = M_i \text{ or } M_n)$ を算出する．

$$poss(t, M_x) = \begin{cases} 1 & , |M_x| = 0 \\ \frac{\sum_{\forall m \in M_x} nsim(t,m,M_x)}{|M_x|} & , \text{otherwise} \end{cases} \tag{7}$$

この $poss(t, M_x)$ は，クローンセット t とクローンセット集合 M_x 中のクローンセットの類似度を表しており，この値が高いほど t が M_x に属する確率が高いといえる．提案手法では，$poss(t, M_i)$ と $poss(t, M_n)$ の値を比較し，$poss(t, M_i) \geq poss(t, M_n)$ の場合に t は “有用” と判定し，それ以外の場合は t は “有用でない” と判定する．

3. 評価実験

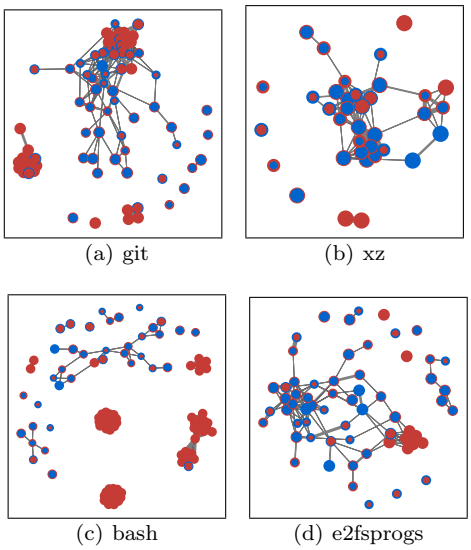
提案手法を実装したツール Fica を用い，評価実験を行った．本実験における評価項目は以下の通り．

- 利用者による “有用” か “有用でない” かの判別基準の違い
- 提案手法による予測の精度

以降，実験方法，実験結果，及び考察について述べる．

3.1 実験方法

提案手法の予測精度を評価するため，4 つのオープンソースソフトウェアを対象とした評価実験を行った．実験対象のソフトウェアを**表 1** に示す．これらのプロジェクトに含まれるファイルのうち，C 言語のソースファイルのみを実験の対象とした．また，表中の CSs はそのプロジェクトから検出されたクローンセットの総数を，Len はコードクローン検出時の最小一致トークン数をそれぞれ表している．



(a) git (b) xz

(c) bash (d) e2fsprogs

図 3 クローンセット間の類似度を表すグラフ
Fig. 3 Force-Directed Graphs Showing Code Clone Similarity

ここで最小一致トークン数とは，コードクローンとして検出されるコード片の最小の大きさ (トークン数) である．本実験では `e2fsprogs` に対してのみ異なる最小一致トークン数を用いているが，これは最小一致トークン数 48 でコードクローンの検出を行った際に多くのクローンセットが検出され，実験に過度な負担がかかると判断したためである．さらに本実験では，ソフトウェア開発やコードクローン検出技術について異なる知識，経験を持つ 8 人の被験者に協力を依頼し，検出された全 281 のクローンセットを “有用” もしくは “有用でない” のいずれかに分類させた．

3.2 利用者による判別基準の違いについて

図 3 にクローンセット間の類似関係を force-directed アルゴリズム [8] を用いてグラフ化したものを示す．グラフ中の 1 つの頂点が 1 つのクローンセットを表しており，青に塗られた頂点は被験者によって “有用” と分類されたクローンセットを，赤に塗られた頂点は “有用でない” と分類されたクローンセットをそれぞれ表している．また，2 つの色が共に塗られている頂点は被験者によって分類結果が異なるクローンセットを表しており，各色の面積比がそれぞれの分類を行った被験者の比率に対応している．

クローンセット間の類似度 (式 (6)) が一定値以上の場合に，クローンセット間に辺が存在する．辺が存在する基準値として，今回は 0.028 を設定している．類似度の高さがクローンセット間の結びつきの強さに対応しており，類似するクローンセットほどそれらの距離が短くなっている．

図 3 より，“有用でない” クローンセットは “有用” なクローンセットと比較して互いに類似度が高く，1 つの群として密集している．これは，“有用でない” コードクローンはそのトークン型列の特徴によっていくつかのカテゴリに分類できる可能性を示唆している．一方，“有用” と分類されたコードクローンは密集しておらず，カテゴリに分類することが難しい．すなわち，他のコードクローンとの類似度が小さい “有用な” コードクローンの判別には利用者の判断が重要であると考えられる．

さらに図 3 より，被験者によって “有用” か “有用でない” かの判別が分かれたクローンセットが多数存在することがわかる．このことは “有用” なコードクローンの特定を行う際に利用者ごとの判断基準の違いを反映させることの必要性を示唆している．

表 1 実験対象ソフトウェア
Table 1 Open Source Projects Used in Experiments

Project	CSs	Len	LOC	Tokens	Files
git	78	48	153,388	829,930	315
xz	36	48	25,873	113,894	113
bash	105	48	133,547	494,248	248
e2fsprogs	62	64	99,129	442,978	274
Total	281		411,937	1,881,050	950

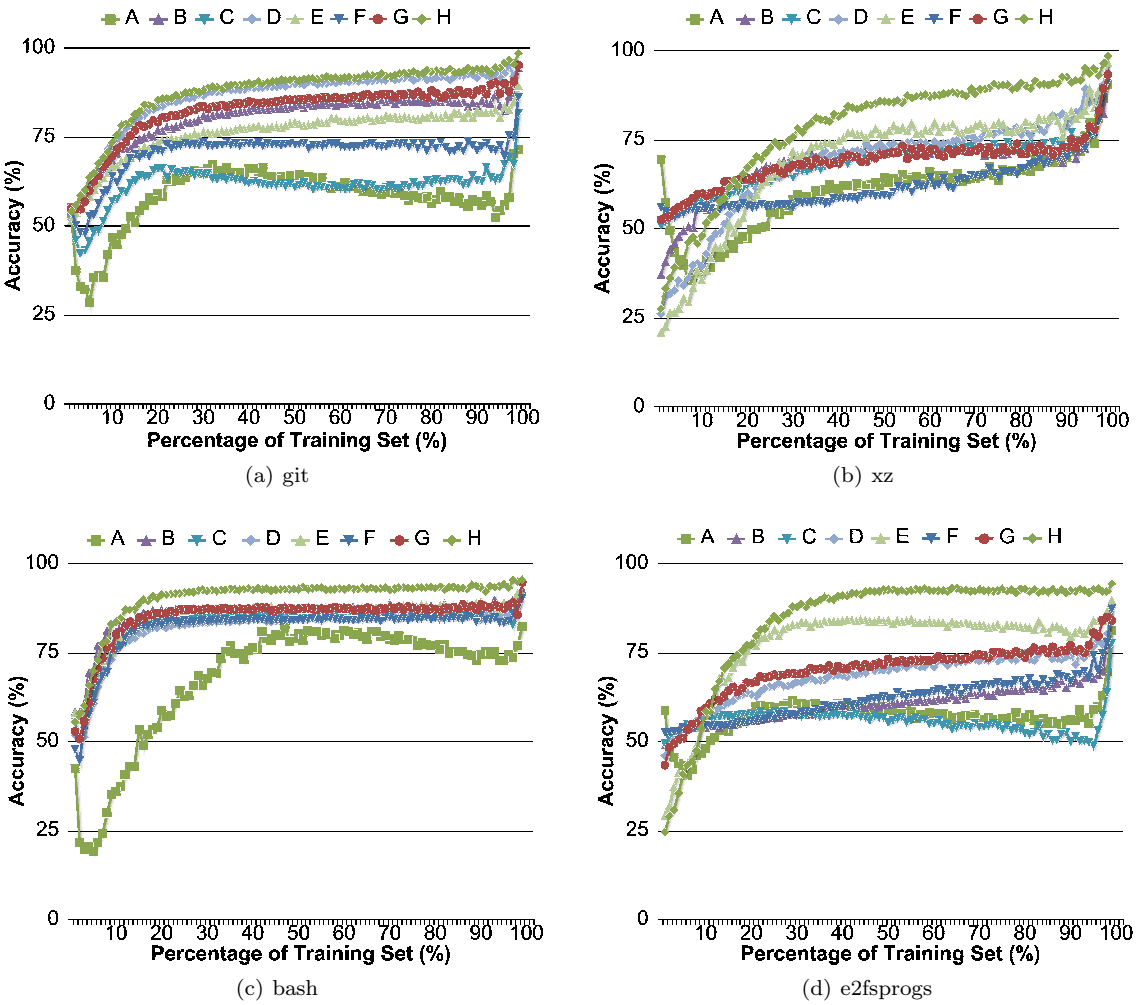
3.3 提案手法による予測の精度について

それぞれのプロジェクトについて，以下の手順で予測精度の評価を行った．

(1) 被験者によって分類されたクローンセットのうち，一定数をランダムに抽出し，学習データとして Fica に登録する．
(2) それぞれの被験者について，学習データとして登録されていないクローンセットの分類を Fica に予測させる．
(3) Fica の予測結果と，実際に被験者が分類を行った結果がどの程度一致しているのかを調査する．
(4) (1)〜(3) を 256 回繰り返し，予測精度の平均値を算出する．
(5) 学習データとして使用するクローンセットの数を変更し，(1)〜(4) の手順を繰り返す．

図 4 に実験結果を示す．グラフの X 軸が全クローンセットのうち学習データとして使用したクローンセットの割合を表しており，Y 軸が Fica の予測結果と実際の被験者の分類結果が一致していた割合の平均値を表している．図 4 より，Fica の予測精度は被験者により，またプロジェクトにより大きく異なっていることがわかる．すべてのプロジェクトに共通してみられる傾向として，グラフの右側ほど，すなわち学習データとして与えるクローンセットの割合を増やすほど，予測精度が向上している．

4 つのプロジェクトのうち，`bash` に対する予測精度が最も高く，被験者 A を除くすべての被験者に対して 15%以上のクローンセットを学習データとして与えることで 80%以上の精度で予測を行えた．`bash` から検出されたクローンセット数は 105 であるため，16 のクローンセットに対して分類を行うことで 80%以上の精度で予測を行えるという結



(a) git　(b) xz

(c) bash　(d) e2fsprogs

図 4　予測精度 (プロジェクトごと)

Fig. 4　Result of Machine Learning (for Each Project)

果となった.

3.4　考察

評価実験の結果，Fica は概ね 70%〜90%の精度で予測できることが確認できた．この結果は，適切に予測できなかった事例がいくつか存在することを意味している．

図 5 にそのような事例の 1 つを示す．この例は **xz** から得られた例である．この例には以下の 2 つのクローンセットが存在する．

α :　(a)130-142 & (b)61-72

β :　(a)136-145 & (c)475-484

被験者実験の結果，8 人の被験者のうち 7 人が，α を “有用” と分類している．他方，β は 8 人中 6 人が “有用でない” と分類している．しかしながら，α と β の間の類似度を式 (5) を用いて算出すると，43%という値が得られる．この値は他のクローンセット同士の組み合わせから得られる類似度と比較して高い値である．このため，Fica は α が “有用” であるという学習データが存在した場合，β も同様に “有用” と判別する可能性が高い．

この問題点は，他のメトリクスを組み合わせて類似度を算出することによって改善が見込まれる．

4.　関連研究

Higo らは複数のメトリクスを用いてリファクタリングしやすいコードクローンを特定する手法を提案している [5]．また，Koschke は機械学習技術を用いてライセンスに違反して流用されているソースコードの特定を支援する手法を提案している [9]．これらの手法はある特定の目的に特化しているのに対し，提案手法は様々な使用用途に対応することが可能であるため，汎用性においてこれらの手法に勝るといえる．

Jiang らは CCFinder によって得られた検出結果に対してデータマイニング手法を適用することでコードクローンをフィルタリングする手法を提案している [10]．Jiang らの手法は利用者による手動の分類を必要としない．提案手法は Jiang らの手法と比較して利用者の負担が大きくなるものの，利用者ごとの判断基準の違いを繊細に反映させることができる．

```
130     }
131     return LZMA_PROG_ERROR;
132 }
133 static void
134 block_encorder_end(lzma_coder *coder, ... )
135 {
136     lzma_next_end(&coder->next, allocator);
137     lzma_free(coder, allocator);
138     return;
139 }
140 static lzma_ret
141 block_encorder_update(lzma_coder *coder, ... ,
142             const lzma_filter *filters,
143             const lzma_filter * reversed_filters)
144 {
145     if (coder->sequence != SEQ_CODE)
```

(a) src/liblzma/common/block_encoder.c

```
61     }
62     return LZMA_OK;
63 }
64 static void
65 alone_encoder_end(lzma_coder *coder, ... )
66 {
67     lzma_next_end(&coder->next, allocator);
68     lzma_free(coder, allocator);
69     return;
70 }
71 static lzma_ret
72 alone_encoder_init(lzma_next_coder *next, ... ,
73           const lzma_options_lzma *options)
```

(b) src/liblzma/common/alone_encoder.c

```
474     else
475             lzma_free(coder->lz.coder, allocator);
476     lzma_free(coder, allocator);
477     return;
478 }
479 static lzma_ret
480 lz_encoder_update(lzma_coder *coder, ... ,
481         const lzma_filter * filters_null,
482         const lzma_filter * reversed_filters)
483 {
484     if (coder->lz.options_update == NULL)
```

(c) src/liblzma/lz/lz_encoder.c

図 5 ソースコード (xz)
Fig. 5 Example of Source Code (xz)

5. あとがき

本稿では，機械学習を用いたコードクローンの自動フィルタリング手法を提案した．提案手法はその利用者がいくつかのコードクローンに対して与えた “有用” か否かの分類情報を学習することで，未分類のコードクローンが “有用” か否かを自動的に判別する．提案手法を実装し，評価実験を行った．その結果，提案手法の予測精度は概ね 70%を超え，被験者やプロジェクトによっては 90%以上の精度で予測ができることを確認した．さらに，あるコードクローンが “有用” か否かの判断基準が利用者によって異なるということを確認した．

今後の課題として，類似度以外のメトリクス値を組み合わせることで予測精度を向上させることが挙げられる．また，現在は “有用” あるいは “有用でない” の二択による学習データ作成を行っているが，これを拡張して，“有用さ” の度合いによる順位付けが可能な分類方式や，タグを用いた分類方式などに対応することを考えている．さらに，被験者のプログラミングに関する経験，知識の違いや，被験者ごとの’有用である” という判断基準の違いを加味した評価を行いたいと考えている．

謝辞 本研究は日本学術振興会科学研究費補助金基盤研究 (A)(課題番号：21240002)，挑戦的萌芽研究 (課題番号：23650014)，及び若手研究 (A)(課題番号：24680002) の助成を得て行われた．

参考文献

[1] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌， Vol. J91-D, No. 6, pp. 1465–1481 (2008).
[2] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア， Vol. 28, No. 3, pp. 28–42 (2011).
[3] Bellon, S., Koschke, R., Antniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of Clone Detection Tools, *IEEE Transactions on Software Engineering*, Vol. 31, No. 10, pp. 804–818 (2007).
[4] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multi-linguistic token-based code clone detection system for large scale source code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670 (2002).
[5] Higo, Y., Kusumoto, S. and Inoue, K.: A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 20, No. 6, pp. 435–461 (2008).
[6] Tairas, R. and Gray, J.: An information retrieval process to aid in the analysis of code clones, *Empirical Software Engineering*, Vol. 14, pp. 33–56 (2009).
[7] Jones, K.: A statistical interpretation of term specificity and its application in retrieval, *Journal of documentation*, Vol. 28, No. 1, pp. 11–21 (1972).
[8] Fruchterman, T. and Reingold, E.: Graph drawing by force-directed placement, *Software: Practice and experience*, Vol. 21, No. 11, pp. 1129–1164 (1991).
[9] Koschke, R.: Large-Scale Inter-System Clone Detection Using Suffix Trees, *Proc. of the 16th European Conference on Software Maintenance and Reengineering*, pp. 309–318 (2012).
[10] Jiang, Z. and Hassan, A.: A framework for studying clones in large software systems, *Proc. of the 7th International Working Conference on Source Code Analysis and Manipulation*, pp. 203–212 (2007).