ソースコードの圧縮性とコードクローンの関係の分析

Analyzing the Relationship Between Compressibility of Source Code and Code Clone

左藤 裕紀, 亀井 靖高, 上野 秀剛, 川口 真司, 門田 暁人, 松本 健一 †

Hiroki SATO, Yasutaka KAMEI, Hidetake UWANO, Shinji KAWAGUCHI, Akito MONDEN, Ken-ichi MATSUMOTO

† 奈良先端科学技術大学院大学情報科学研究科

Graduate School of Information Science, Nara Institute of Science and Technology

hiroki-sa@is.naist.jp

ソフトウェア中に含まれるコードクローン（重複するコード列）の大きさや含有率は，ソフトウェアの冗長性を表す一つの指標であるといえる．一方，ソースコードを文字列とみなして圧縮を行い，その圧縮サイズや圧縮率を計測することで，より直接的に冗長性の計測が可能である．本稿では，ソースコードの圧縮がコードクローン計測の代替となり得るかどうかを明らかとするために，ソースコードの圧縮性（圧縮サイズ，圧縮率）とコードクローンとの関係を実験的に分析する．

1 はじめに

近年，ソフトウェアの大規模化に伴ってソフトウェア保守の重要性が高まっており，保守を困難にする要因の一つとしてコードクローンが注目されている [8]．コードクローンとはソースコード中に含まれる重複，もしくは類似したコード断片のことである．コードクローンはソフトウェアの構造を複雑にし，ソフトウェアの品質を低下させる一因であると言われており，大規模ソフトウェアの少なくない部分（5〜60%）をコードクローンが占めていたことが報告されている [3, 6]．コードクローンを構成するコードの中に不具合が見つかった場合，関係する全てのコードクローンを調査し，それぞれに対して同様の修正を施すことになる可能性が高い．この作業はソフトウェアが大規模になるほど困難となる．従来，コードクローンを検出するための様々な手法が提案されており [1, 2, 3, 6]，そのいくつかはツールとして実装されている [4, 5]．

本稿では，コードクローンとソースコードの冗長性（圧縮性）との関係に着目する．一般に，コードクローンを除去することで，ソースコードがよりコンパクトになることから，コードクローンの大きさや含有率は，ソースコードの冗長性を表す一つの指標となると考えられる．情報量の観点からは，コードクローンを含むソースコードは，コードクローンを含まない同程度の規模のソースコードと比べてより少ない情報量しか持たないともいえる．一方，ソースコードの冗長性や情報量に着目すると，ソースコードを文字列（ビット列）とみなして圧縮を行い，その圧縮サイズや圧縮率を計測することで，より直接的に冗長性や情報量の計測が可能である [10]．ソースコードの圧縮は，コードクローンの計測と比べると，プログラミング言語に依存しないことや，開発途中の不完全なプログラムに対しても実施可能な点が特長である．しかしながら，プログラミング言語の文法が考慮されない点が弱点となる．コードクローンとソースコードの圧縮性の関係は，従来明らかにされておらず，ソースコードを圧縮し，その圧縮サイズや圧縮率を計測するという方法がコードクローン計測の代替となりえるのかも不明である．

本稿では，C 言語で記述された大規模ソフトウェア (Apache2.0.59) を題材とし，ソースコードの圧縮性とコードクローンの関係を実験的に分析した結果について報告する．コードクローン計測には CCfinder[5] を用い，ソースコードの圧縮アルゴリズムには LZMA を用いた．実験では，ソフトウェアを構成する各ファイル（ソースコード）を個別に圧縮し，各ファイルの圧縮率とクローン含有率の関係，及び，圧縮サイズとクローン含有量（トークン数）の関係を調べた．

以降，2 章ではコードクローンの定義と分類について説明し，コードクローンの特性を表すパラメータについて述べる．3 章ではコードクローンとソースコードの圧縮性の関係を調べるためのケーススタディとその結果についての考察を述べる．4 章でまとめと今後の課題について述べる．

2 コードクローンの定義と分類

2.1 コードクローンとは

コードクローンとはソースコード中の重複したコード列のことであり，ソフトウェアの開発者や保守作業者がコード列をコピー＆ペーストするといったいくつかの原因によって作られる．コピー＆ペースト以外の原因としては，コードジェネレータによって生成されたコード，特定のコーディングスタイルによるもの，パフォーマンスを向上させるための意図的な繰り返し，偶然の一致がある [3]．コピー＆ペーストによって作られたコードクローンの場合には，コード列に部分的な変更を加えることも多く，そのような部分的に異なる類似コード列の組もコードクローンと見なすのが一般的である．

このようなコードクローンの多くはデバッグや機能拡張の際に同時に更新しなければならず，プログラムの保守性を低下させるためリファクタリングなどによって除去されることが望ましい．ただし，どのようなコード列がコードクローンと見なすかの正確な定義はクローン検出方法やツールごとに異なる [1, 2, 3, 8]．

2.2 トークンベースコードクローン検出手法

本稿では，神谷らが提案したトークンベースのコードクローン検出方法を採用する [8, 9]．この方法はプログラミング言語の構文規則に基づき，ソースコードをトークン別に変換することで，コード列中の空白やコメント，インデントが異なったり，あるいは変数名等が書き換えられたりした場合でも，コードクローンを抽出することができる．この手法を実装したツール CCFinder は COBOL や C++等のプログラミング言語で記述されたソフトウェアへの適用が可能である．トークンベースによるコードクローン摘出手順の概要を次に示す．

1. 字句解析
   入力として与えられたソフトウェアに含まれる全てのソースコードを，プログラミング言語の字句規則に従ってトークンに分割する．ソースコード中の空白やコメントは無視される．

2. トークン変換
   型，変数，定数に属するトークンはそれぞれ同一のトークン (e.g. *type*， *variable*， *invariable*) に置き換えられる．この置き換えにより，たとえば変数名だけが異なるコード列の組をコードクローンとして検出することができる．

3. マッチングおよびフォーマッティング
   変換後のトークン列に含まれる全ての部分列の集合から，同一の部分列の組を探し出してコードクローンとして検出する．検出にあたっては Suffix Tree マッチングアルゴリズム [7] を用いることで，ソフトウェアのサイズ n に対して $O(n)$ の計算量で検出できる．最後に，検出されたトークン列の位置情報を元のソースコード上の行番号に変換し，出力する．

2.3 コードクローンの分類

本稿ではコードクローンとして検出された類似する 2 つのコード列の組を「コードクローンペア」と呼ぶ．コードクローンペアは，2 つのコード列が含まれているソースコードの位置によって次の 2 つに分類される [11]．

1. In-module クローンペア
   コードクローンペアを構成する 2 つのコード列の両方が同一のソースコードに存在する．

2. Inter-module クローンペア
   コードクローンペアを構成する 2 つのコード列がそれぞれ異なるソースコードに存在する．

どちらのコードクローンペアもプログラムの保守性を下げる要因となるが，特に Inter-module クローンペアは類似の処理を行うコード断片が 2 つのモジュールにまたがるため，モジュール間の結合度を増大させる可能性がある．

2.4 パラメータ

本節では各モジュール及び検出されたコードクローンを評価するための尺度を定義する．

(1)LEN(length)
対象とするコード列の長さをトークン数で表したもの．この値が大きいほどコード列が大きいことを表す．本稿では以下の 4 つを表現するために用いる．

a) ソースコード長：対象とするソースコードに含まれるトークンの総数．
b) 最小コードクローン長:コードクローンとみなすコード列の最小トークン数．この値を超える重複コー

ド列をコードクローンとする．
c) コードクローン長:検出したコードクローン１つに含まれるトークン数．
d) ソースコード内コードクローン長:ソースコード内に存在するコードクローンのトークン数の和．

(2)RSI (ratio of similarity within the file)
対象とするソースコードが，ソースコード内コードクローンによって占められている割合（%）．RSI は以下の式で表される．

$$RSI = \frac{\text{ソースコード内コードクローン長}}{\text{ソースコード長}}$$

この値が大きいソースコードは，同じような関数やメソッドが並んでいる可能性がある．

3 ケーススタディ

本ケーススタディでは，Web サーバ (HTTP Server) の 1 つである Apache を対象とし，ソースコードの圧縮性とコードクローンの関係を分析する．今回のケーススタディでは，Inter-module クローンペアのコードクローンを扱わず，In-module クローンペアのコードクローンのみを扱う．ソースコードの圧縮率と RSI の関係，圧縮されたファイルサイズとソースコード内コードクローン長の関係の 2 つの視点で関係を調べる．

なお，ソースコードから検出されるコードクローンは，設定する最小コードクローン長によって異なる．ケーススタディでは，最小コードクローン長を 10 と 50 に設定し，ソースコードの圧縮性との関係を分析する．

本ケーススタディでは，コードクローンを検出するツールとして CCFinder[5] を，ソースコードを圧縮するツールとして 7-zip[12] を用いた．コードクローンを検出する際にはコードクローン内のトークンの最小種類数を 12 とし，ソースコードの圧縮アルゴリズムには LZ77 アルゴリズムを元に改良された LZMA を用いた．

3.1 対象データ

対象データとして，Web サーバの１つである Apache（httpd）のバージョン 2.0.59 のソースコードを用いた．本ケーススタディでは，Apache に含まれる C 言語で記述されたソースコード（拡張子が".c" のファイル）707 個のうち，以下のソースコードを除外した 490 個を用いた．

- 1KB 以下のソースコード

  LZMA を用いた圧縮では，頻繁に出現する単語を短い単語に置き換えるための辞書をファイルに追加する．ファイルサイズが小さすぎる場合，辞書の追加によって圧縮後のファイルサイズが元のファイルサイズより大きくなる場合がある．そこで本ケーススタディでは，ファイルサイズが 1KB 以下のソースコードを対象から除外する．除外対象となったソースコードは 21 個である．

- /home/srclib/apr-iconv/ccs ディレクトリ以下のソースコード

  本ディレクトリ以下のソースコードは，90%以上の部分が変数定義（コードクローンの検出対象外）である．そこで，本ケーススタディでは，/home/srclib/apr-iconv/ccs ディレクトリ以下のソースコードを対象から除外する．除外対象となったソースコードは 196 個である．

3.2 分析手順

検出する最小コードクローン長を 10 と 50 に設定し，ソースコードの圧縮性とコードクローンの関係を以下の手順により分析した．

1. ソースコードの圧縮

   各ソースコードの圧縮性を調べるために，ソースコード 1 つずつを 7-zip によって圧縮し，圧縮率と圧縮されたファイルサイズを求める．本稿では圧縮率を（圧縮後のファイルサイズ）/（圧縮前のファイルサイズ）の意味で用いる．

2. コードクローンの検出

   各ソースコードに含まれるコードクローンの割合 (RSI) を調べるために，CCFinder によってソースコード 1 つずつからコードクローンを検出し，ソースコード内コードクローン長と RSI を求める．

3. 圧縮性とコードクローンの比較

   手順 1，2 で求めた，圧縮率と RSI の関係，圧縮されたファイルサイズとソースコード内コードクローン長に相関関係があるか分析する．

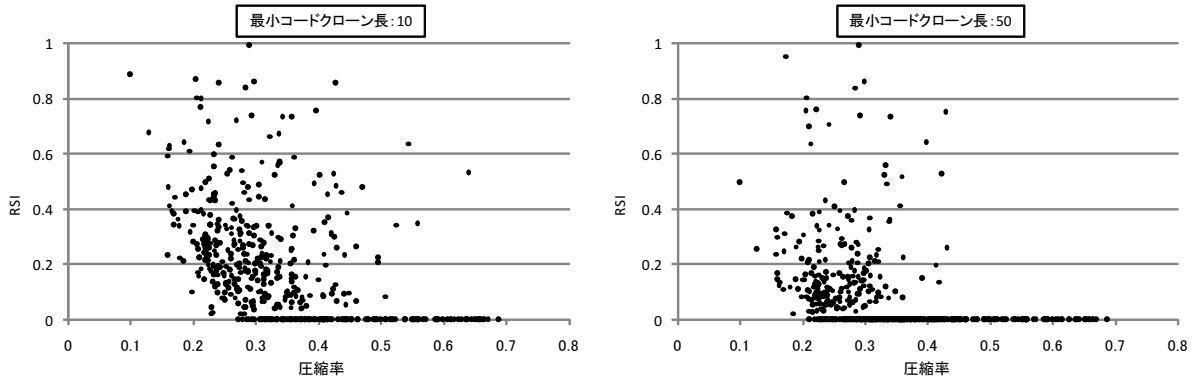

図 1: 圧縮率と RSI の関係

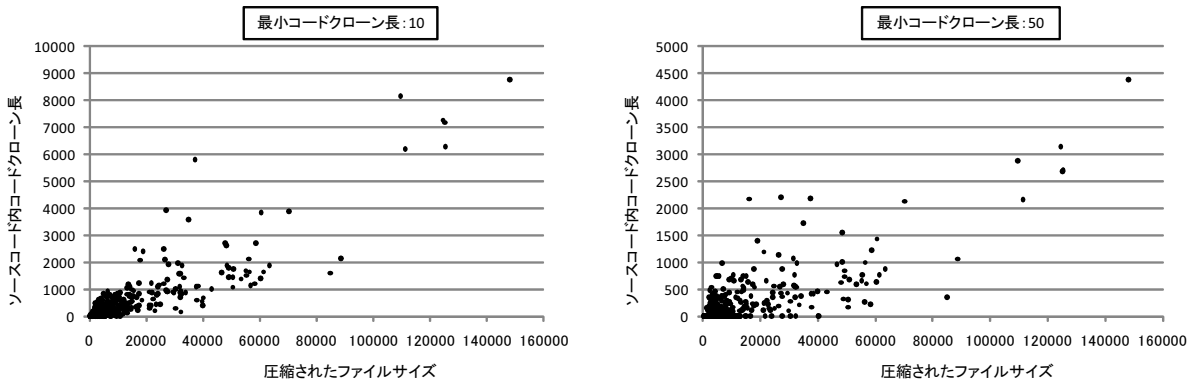

図 2: 圧縮されたファイルサイズとソースコード内コードクローン長の関係

3.3 ケーススタディの結果と考察

圧縮率と RSI の関係を図 1 に，圧縮されたファイルサイズとソースコード内クローン長の関係を図 2 に示す．

図 1 より，ソースコード圧縮率とコードクローン含有率 (RSI) との間には，弱い関係が見られた．図 1 左（最小コードクローン長が 10）を見ると，グラフ上の点は広範に分布しており，強い関係は見られないが，圧縮率が 0 に近い（＝極めて圧縮性が高い）付近では，RSI の小さなファイルが存在しないことが見て取れる．この傾向は，図 1 右（最小コードクローン長が 50）でも同様である．このことから，極めて圧縮性の高いファイルは，例外なくコードクローンを含んでいるといえる．ただし，圧縮率が一定以上（図 1 左で 0.25 以上，図 1 右で 0.2 以上）になると，コードクローン含有率との明確な関係はみられない．

図 1 右（最小コードクローン長が 50）の圧縮率が 0.5 以上のファイルでは，RSI が 0 より大きいファイルが存在しないことが見て取れる．これは，圧縮性の悪いファイルは，一般にサイズが小さく，トークン数も少ないため，コードクローンが検出されなかったためと考えられる．今後の課題として，ファイルサイズを考慮した分析を行っていくことが重要となる．

図 2 左（最小コードクローン長が 10）より，圧縮サイズとコードクローン長との間には，正の相関が見られた．たくさんの分量を圧縮できたファイルほど，より大きなコードクローンを含んでいたといえる．ただし，圧縮サイズが大きくてもコードクローン長がさほど大きくないファイルが存在し，反対に，圧縮サイズが比較的小さいファイルの中にコードクローン長が大きいファイルも存在することが見て取れる．前者については，配列の定義を行うコードにおける重複など，CCFinder ではコードクローンとして検出されない重複コードを多く含むファイルである可能性がある．後者については，現時点では理由は不明であり，今後，ファイルの中身を分析していくことが課題となる．

なお，図 2 右（最小コードクローン長が 50）を見ると，図 2 左（最小コードクローン長が 10）と同様の関係は見られるものの，グラフ上の点の分布がやや拡散しており，関係は弱まっている．これは，50 トークン未満の短い重複コード列がコードクローンとして検出されなくなったためと考えられる．

4 おわりに

本稿ではコードクローンとソースコードの圧縮性の関係を分析した．Web サーバ (HTTP Server) の 1 つである Apache を対象に実験を行い，In-module クローンに含まれるコードクローンの占める割合 (RSI) と圧縮率に負の相関関係があること，ソースコード内コードクローン長と圧縮されたファイルサイズに正の相関関係があることを確認した．

今後の課題としてはファイル間コードクローン (Inter-module クローン）の調査が挙げられる．本稿で行った調査は In-module クローンのみだったので，Inter-module クローンとソースコードの圧縮性の関係を明らかにするために，Inter-module クローンと，Inter-module クローンの存在する複数のソースコードの圧縮率にどのような関係があるのかを調査する．

謝辞

本研究の一部は，文部科学省科学研究補助金 (基盤研究 (B), 課題番号:17300007) の補助を受けた．また，本研究の一部は，文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．

参考文献

[1] B. S. Baker, A program for identifying duplicated code. *24th Symposium on the Interface: Computing Science and Statistics*, 1992, pp. 49–57.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. A. Kontogiannis, Measuring clone based reengineering opportunities. In *Proc. 6th IEEE Int'l Symposium on Softw. Metrics(METRICS'99)*, 1999, pp. 292–303.

[3] I. D. Baxter , A. Yahin, L. Moura, M. Sant'Anna and L. Bier, Clone detection using abstract syntax trees. In *Proc. IEEE Int'l Conf. on Softw. Maintenance*, IEEE, 1998, pp. 368–377.

[4] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools. *IEEE Trans. Softw. Eng.*, Vol.33, No.9(2007), pp. 577–591.

[5] CCfinder, http://www.ccfinder.net/index-j.html.

[6] S. R. Chidamber and C. F. Kemerer, A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, Vol.20, No. 6(1994), pp. 652–686.

[7] D. Gusfield, Algorithms on Strings,Trees and Sequences. Cambridge University Press, 1997, pp. 89–180.

[8] T. Kamiya, S. Kusumoto and K. Inoue, CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, Vol.28, No.7(2001), pp. 654–670.

[9] T. Kamiya, F. Ohara, K. Kondou, S. Kusumoto and K. Inoue, Maintenance support tools for Java programs:CCFinder and JAAT. In *Proc. 23th Int'l Conf. on Softw. Engineering(ICSE2001)*, 2001, pp. 837–838.

[10] D. Mertz, データ圧縮入門 データ表現の理論と戦略, http://www-06.ibm.com/jp/developerworks/linux/010720/j_l-compr.html.

[11] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一, コードクローンに基づくレガシーソフトウェアの品質の分析. 情報処理学会論文誌, Vol. 44, No. 8 (2003), pp. 2178–2188.

[12] 7-zip, http://www.7-zip.org/ja/.