THE IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS (JAPANESE EDITION)

電子情報通信学会 論文誌 D 情報・システム

VOL. J103-D NO. 10
OCTOBER 2020

本PDFの扱いは、電子情報通信学会著作権規定に従うこと。
なお、本PDFは研究教育目的（非営利）に限り、著者が第三者に直接配布することができる。著者以外からの配布は禁じられている。

情報・システムソサイエティ

一般社団法人 電子情報通信学会
THE INFORMATION AND SYSTEMS SOCIETY
THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

類似ソースコード片検索結果に対するクラスタリング技術を用いたフィルタリング手法

嶋利　一真[†a)]　石尾　隆[††]（正員）
井上　克郎[†]（正員：フェロー）

A Clustering-Based Filtering Method for Similar Source Code Fragment Search

Kazumasa SHIMARI[†a)], *Nonmember*, Takashi ISHIO[††], *Member*, and Katsuro INOUE[†], *Fellow*

[†] 大阪大学大学院情報科学研究科，吹田市
Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan
[††] 奈良先端科学技術大学院大学先端科学技術研究科，生駒市
Graduate School of Science and Technology, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan
a) E-mail: k-simari@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2020JDL8009

あらまし　類似ソースコード片検索ツールの出力結果に対してクラスタリングを適用してコード片をグループ化し，正解の可能性が低いコード片を出力から除外するフィルタリングを提案する．バグを含む類似コード片のデータセットを用いて，提案手法の有用性を確認した．

キーワード　ソースコード検索，クラスタリング，フィルタリング，正規圧縮距離

1. まえがき

ソフトウェア開発において，開発者はプロジェクト内でソースコードを再利用し，類似した複数の機能の実装に使用することがある．しかし，再利用元にバグが含まれていた場合，結果的にソースコード内の複数の場所に同一のバグができてしまうことがある [1]．そのため，ソフトウェアにバグが発見された場合，そのバグを含むソースコード片に類似したソースコード片を検索し，修正することが重要となる [2]．

類似ソースコード片を検索するためのツールの一つとして，我々の研究グループは NCDSearch (注1) を開発している．このツールは，クエリとして与えたソースコード片と検索対象ソースコードから順番に切り出されたソースコード片を比較し，類似するソースコード片の位置（ファイル名と行番号）を全て検出する．既存のコードクローン検出ツール CCFinderX [3] 等と比較して，類似したバグを含むソースコード片の見逃しが少ないが，バグを含まないソースコード片も数多く検出される [4]．理想的には全ての検出結果を調査すればバグの修正も完全なものになるが，実際の開発現場では作業時間が限られていることが多く，優先して調査すべきコード片のみに検索結果を絞り込む技術が必要である．

本研究は，NCDSearch の出力結果に対してフィルタリングを行うことで，バグを含まない類似ソースコード片を検索結果から除去することを目指す．フィルタリングには，Yamaguchi が提案したコード片のクラスタリング手法 [5] を応用する．この手法は，ソフトウェアの脆弱性の調査を行う際に，他のコード片と類似していない特異なコードを調査するべき候補として取り出すというものである．本研究ではこの発想に基づき，検索によって得られた類似ソースコード片のうち，類似バグを含む可能性が高いものには類似するが，類似バグを含む可能性が低いものとは類似しないようなソースコード片のみを調査対象として抽出する．

2. 提案手法

提案手法は NCDSearch がクエリ q に対して検出した類似ソースコード片の集合 F に対するフィルタリングである．NCDSearch は正規圧縮距離 [6] を検索に使用しており，距離しきい値 th に対して各コード片 $f \in F$ とクエリの距離が $d(q, f) \leq th$ を満たすようなコード片を対象ソースコードから検出する．

提案手法は，近傍探索を用いることでフィルタリングを実現する．具体的には，検索結果から得られたソースコード片をしきい値 d_1 $(d_1 < th)$ によって，クエリに強く類似するものと，そうでないものに分類し，後者のグループはバグを含む可能性が低いと考える．このコード片の集合を F' として以下のように定義する．

$$F' = \{f' \in F \mid d(q, f') > d_1\} \tag{1}$$

F' に含まれるコード片のいずれかから距離 d_2 以内にあるコードは，たとえクエリと類似していても，特異なコードではないと考え，検索結果 R から以下のようにして除外する（クエリと完全一致するコードを R から除外しないように $d_2 < d_1$ とする）．

$$R = \{f \in F \backslash F' \mid \forall f' \in F'.\, d(f, f') > d_2\} \tag{2}$$

このようにして得られる R には，検索クエリとなったコード片と類似しており，かつ，バグを含んでいる可能性が低いコード片とは異なるようなコード片だけが含まれる．R に含まれるコード片をクエリに対する距離 $d(q, f)$ の順番で出力したものが，提案手法の最終的な出力である．

（注1）：https://github.com/takashi-ishio/NCDSearch

3. 評価実験

提案手法が NCDSearch の検索結果を改善する度合いを，OSS におけるバグ修正事例のデータセット [7] を用いて評価する．表 1 にデータセットの規模やバグの数を示す．このデータセットは PostgreSQL，Git，Linux において，一つの修正が複数の類似ソースコードに対して行われた事例を集めたものである．クエリとして与えられたコード片で検索を行い，同時に修正されるべきコード片をどれだけ検出できるかを評価する．

提案手法は d_1，d_2 の二つのパラメータをもつ．既存研究の実験 [4] で使用したときの設定（$th = 0.50$）を使用し，提案手法で述べた $d_2 < d_1 < th$ という関係を満たす範囲で d_1 と d_2 をそれぞれ 0.05 から 0.45 まで 0.05 ずつ変化させ，フィルタリング結果 R を求める．また，評価のベースラインには，NCDSearch の結果に対して単純に距離のしきい値 d_1 のみを用いてフィルタリングした結果 $R' = \{f \in F \mid d(q, f) \leq d_1\}$ を用いる．これは NCDSearch のしきい値 th を変化させた場合に相当する．ベースラインの結果は，d_1 を 0.01 から 0.50 まで 0.01 ずつ変化させたものを用いる．ツールの利用者が全クエリの実行結果を全て確認する場合を想定し，クエリごとに得られる R の和集合を提案手法の検索結果，クエリごとに得られる R' の和集合をベースラインの検索結果とする．評価実験において使用した NCDSearch のバージョンは v0.3.5 (注2) である．

評価に用いるのは以下の指標である．

適合率． コード片集合出力結果のコード片のうち，バグを含むコード片の割合．

再現率． バグを含むコード片のうち，検索結果に含まれるものの割合．

削減率 r． $th = 0.50$ での検索結果を基準としたソースコード片の減少の割合（$r = 1 - \frac{|R|}{|F|}$ または $1 - \frac{|R'|}{|F|}$）．この値が高いほど利用者が確認するコード片の数が少ない．

再現率と削減率の調和平均 h． 出力される結果を減らすほど見逃しの可能性が上がるため，再現率と削減率はトレードオフの関係にある．再現率を高く保ちながら報告されるコード片の数を大きく減らすような手法を高く評価するために，この調和平均を用いる．

表 1　類似ソースコード片のデータセット
Table 1　Dataset for similar source code fragments.

プロジェクト	クエリ	バグ	ファイル数の中央値	行数の中央値
PostgreSQL	14	39	1,058	277,959
Git	5	8	261	67,028
Linux	34	41	22,181	6,931,715
合計	53	88	792,432	241,074,652

(注2)：commitID:5b76c37193741edfd41fbd7b865d04194cb3ddfa

表 2 に調和平均 h に基づく提案手法の最良の結果とベースラインの結果を示す．ベースライン ($d_1 = 0.50$) の出力コード片の数は合計で 4,866 であり，その中には 88 個のバグが全て検出されている（再現率 1.000）．提案手法は最良の場合で，出力結果を 80.6% 削減しながら，バグのうち 88.6% を出力結果に含んでいる．提案手法は削減率が同程度のベースライン ($d_1 = 0.32$) と比較した場合に高い再現率を示しており，また，再現率で近い値をもつベースライン ($d_1 = 0.35$) の場合と比べると，出力されるコード片の数を 1,176 から 942 へと約 20% 削減している．したがって，ベースラインの手法で単純に距離のしきい値 d_1 を変化させるよりも効率的な調査が可能である．

提案手法へのパラメータの影響を図 1 に示す．各点の X 座標は提案手法及びベースラインにおける d_1 であり，Y 座標はそのときの調和平均 h である．提案手

表 2　提案手法とベースラインのフィルタリングの精度
Table 2　Filtering accuracy of the proposed method and baseline.

	適合率	再現率	削減率	調和平均
提案手法 ($d_1 = 0.35, d_2 = 0.25$)	0.083	0.886	0.806	0.844
ベースライン ($d_1 = 0.30$)	0.081	0.773	0.827	0.799
ベースライン ($d_1 = 0.31$)	0.082	0.830	0.816	0.823
ベースライン ($d_1 = 0.32$)	0.077	0.841	0.802	0.821
ベースライン ($d_1 = 0.35$)	0.067	0.898	0.758	0.822
ベースライン ($d_1 = 0.50$)	0.018	1.000	0.000	0.000

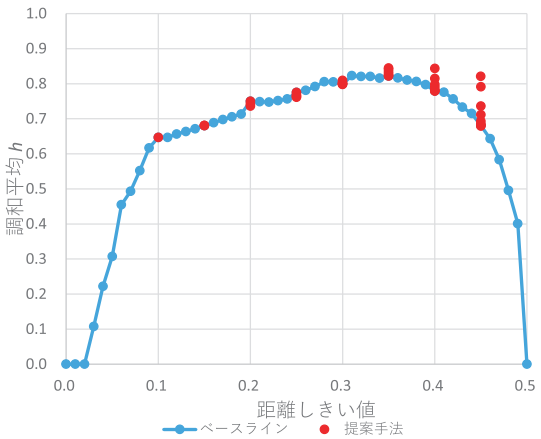

図 1　提案手法とベースラインの調和平均 h
Fig. 1　Harmonic mean of the proposed method and baseline.

法については d_2 として複数の値を使用しているため，それらの結果を全てプロットしている．この図から，調和平均 h の値は安定してベースラインと同じ程度か，良い結果を示しており，提案手法はソースコード検索結果の改善に有効であることが分かる．

なお，提案手法によるフィルタリングはクエリ 1 回あたり平均で 1 秒未満であった．NCDSearch の実行時間はクエリ 1 回あたり平均で 8.5 分であり，それに比べて軽微なコストで出力結果を改善している．

4. むすび

本研究では，特異なコードに注目するという Yamaguchi の脆弱性調査の手法を類似ソースコード検索に応用し，クエリから相対的に遠いソースコード片とは異なるようなソースコード片だけを抽出するフィルタリング手法を定義した．その結果，単純なフィルタリングよりも再現率を高く保ちながら検索結果の個数を削減可能であることを示した．

実験で用いたデータセットは C 言語の 3 プロジェクトの事例に限られているため，新たな OSS の修正事例や，NCDSearch を利用しているソフトウェア企業環境 [4] での効果を調査することが今後の課題である．

謝辞 本研究は科学技術研究費 JP18H03221，JP18KT0013，JP18H04094 の助成を受けて行われた

文　献

[1] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, “An empirical study of operating systems errors,” Proc. SOSP, pp.73–88, Oct. 2001.

[2] Y. Dang, D. Zhang, S. Ge, R. Huang, C. Chu, and T. Xie, “Transferring code-clone detection and analysis to practice,” Proc. ICSE, pp.53–62, May 2017.

[3] T. Kamiya, AIST CCFinderX, http://www.ccfinder.net/ccfinderxos.html

[4] T. Ishio, N. Maeda, K. Shibuya, and K. Inoue, “Cloned buggy code detection in practice using normalized compression distance,” Proc. ICSME, pp.591–594, Sept. 2018.

[5] F. Yamaguchi, “Pattern-based vulnerability discovery,” Ph.D. Thesis, University of Göttingen, 2015.

[6] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitanyi, “The similarity metric,” IEEE Trans. Inf. Theory, vol.50, no.12, pp.3250–3264, Dec. 2004.

[7] J. Li and M.D. Ernst, “CBCD: Cloned buggy code detector,” Proc. ICSE, pp.310–320, June 2012.

（2020 年 4 月 23 日受付，5 月 29 日再受付，
6 月 22 日早期公開）