Vol.2022-SE-210 No.4
2022/3/11

クラウド知識を用いたデバッグ支援システムの検討

森内 海[1,a)]　奥野 拓[1,b)]

概要：デバッグを行う際，Web 上のクラウド知識を用いてバグの解決策を探すことがある．クラウド知識とは，様々な人の知識の集合体のことであり，StackOverflow や teratail などから得ることができる．クラウド知識は年々増加しており，ユーザが求めている解決策を探す際の負担が大きい．そこで本研究では，ユーザがデバッグを行う際の負担軽減を目的とし，ユーザに適した解決策を提示するシステムの検討を行う．本システムではまず，ユーザにソースコードの中から，バグが発生していると思われるコード片を選択してもらう．次に，選択したコード片とクラウド知識上のコード片を用いてコードクローン検出を行う．続いて，多くの投稿から解決策となりうる投稿に絞り込むために，クラウド知識上のコード片に対して網羅率を算出し閾値以下のコード片を除外する．網羅率とは，クラウド知識の投稿から抽出したコード片全体に対して，コードクローンが検出される行数の割合のことである．そして，コード片が含まれる投稿のテキストを用いてトピック抽出を行い，ユーザに求めている情報に近しいトピックを選択してもらう．最後に，選択されたトピックが含まれる投稿に対し生起確率でのスコア付けを行い，ユーザに提示する．生起確率とは，トピックに関連する単語がそのトピックにおいて生起する確率のことを指す．StackOveflow の投稿に含まれるコード片をもとにコードクローン検出を行い，網羅率の算出を行なった結果，多くのコード片の網羅率が 7%未満となっていた．しかし，今回の検証のように網羅率 7%未満のコード片を除外した場合，解決策となりうるコード片が除外される可能性があるため，様々な種類のバグが発生しているコード片を用いた検証を行う必要がある．

A Study on a Debugging Support System Using Crowd Knowledge

1. はじめに

ソフトウェア開発では，書籍や Web サイトなどから必要な知識を得ることがある．中でも Web サイトは多くの人が利用しやすく，知識を得る際に多く利用される [1]．デバッグを行う際に Web 上のクラウド知識を用いてバグの解決策を探すことがある．クラウド知識とは様々な人の知識の集合体であり，クラウド知識群から得ることができる．プログラミングにおけるクラウド知識群には，StackOverflow (以下，SO とする)[*1]，GitHub[*2]，Qiita[*3]，teratail[*4]などが含まれる．クラウド知識の特徴として，構文エラーやライブラリの使用ミスなどのような既出のバグに対して有効であることが挙げられる．クラウド知識を利用する際には，キーワード，コード片，エラー文の一部などの要素を用いて検索する．各要素を用いて検索することで，投稿の絞り込みを行うことができる．しかし，言語特有のキーワード，コード片の長さ，エラー文のどの部分を検索に用いるかなど，投稿の絞り込みには各要素の利用に精通している必要がある．また，膨大な投稿の中から自分の状況にあった投稿を選択するために本文に目を通す必要がある．これらのことにより，クラウド知識を利用する際のユーザの負担が大きいという問題がある．そこで本研究では，クラウド知識を利用する際のユーザの負担軽減を目的としたシステムの検討を行う．本システムではまず，コード片をもとにクラウド知識上で解決策となりうる投稿をコードクローン検出を用いて検索する．次に，投稿のテキストを用いてトピックモデルを構築し，ユーザに求めている情報に近しいトピックを選択してもらう．最後に，選択されたトピックを用いて各投稿に対してスコア付けを行い，ユーザに提示する．

[1] 公立はこだて未来大学
Future Uniersity Hakodate
[a)] g2121053@fun.ac.jp
[b)] okuno@fun.ac.jp
[*1] https://stackoverflow.com/
[*2] https://github.co.jp/
[*3] https://qiita.com/
[*4] https://teratail.com/

2. 関連研究

2.1 クラウド知識を用いたデバッグ支援

Chen らは，SO の投稿内に含まれるコード片を利用したデバッグ支援システムの構築を行っている [2]．Chen らのシステムではドメインに依存しないバグを対象としている．ドメインに依存しているバグとは，解決にプログラミング以外の知識を必要とするバグを指す．ドメインに依存しているバグには，銀行の金利計算において金利が付与されないバグなどが含まれる．この種類のバグはユーザに大きく依存し，クラウド知識上で議論されることが少ないため対象外としている．このシステムでは，入力されたソースコードと SO の質問内のコード片を用いてコードクローン検出を行う．コードクローンとは，ソースコード中で一致あるいは類似しているコード片を指す．システムに入力されたソースコード内でコードクローンの検出された箇所が，バグとなりうる箇所として提示される．バグに対する解決策として，コードクローンが検出されたコード片が含まれる質問に対する回答のコード，回答の要約，質問の URL が提示される．Chen らは，オープンソースプロジェクトのソースコードを用いて，構築したシステムと他の静的分析ツールとで検出可能なバグの数の比較を行なっている．静的分析ツールでは 6 個のバグとなりうる箇所を検出することができたが，このシステムでは 181 個のバグとなりうる箇所を検出できることが示された．しかし，Chen らのシステムではソースコード全体にコードクローン検出を行うため，ユーザの求めている投稿以外も多く提示される．そのため，ユーザが求めている解決策となりうる投稿が埋もれる可能性がある．そこで本研究では，ユーザが選択した部分のコード片をもとにコードクローン検出を行い，解決策となりうる投稿を探す．

2.2 SO における投稿分析

Allamanis らは，トピック分析手法を用いて SO の投稿の分析を行っている [3]．Allamanis らの手法では，トピック分析手法である Latent Dirichlet Allocation（以下，LDA とする）[4] を用いてトピックモデルを構築し，以下の 3 つのテキストデータを用いてモデルの学習を行っている．最初に SO の投稿のテキストのみを用いてモデルの学習を行っている．その結果，特定の技術についての言及や，何らかのものが動かないといったトピックを抽出できることが判明している．次にテキストに加えてコードスニペットを用いてモデルの学習を行っている．その結果，テストや音楽再生などタスク固有のトピックや，バージョンについてのトピックを抽出できることが判明している．以上の 2 つの方法では問題の原因までは特定できなかったため，質問者が何をしたいかを表す動詞句に着目し，名詞句を削除してモデルの学習を行っている．その結果，「API の使用方法について」，「実装方法がわからない」のような問題の原因ごとのカテゴリーに分類できることが判明している．しかし，名詞句を削除してモデルの学習を行った場合，質問で使用されている API 名や関数名が含まれていないため，ユーザがトピックを選択する際，質問者が利用している API や関数がわからない．そこで本研究では Allamanis らの手法を参考に，コードクローンが検出された投稿のテキストを用いて，API 名や関数名が含まれる句で分かち書きをしてトピックモデルを構築し，トピックの抽出を行う．

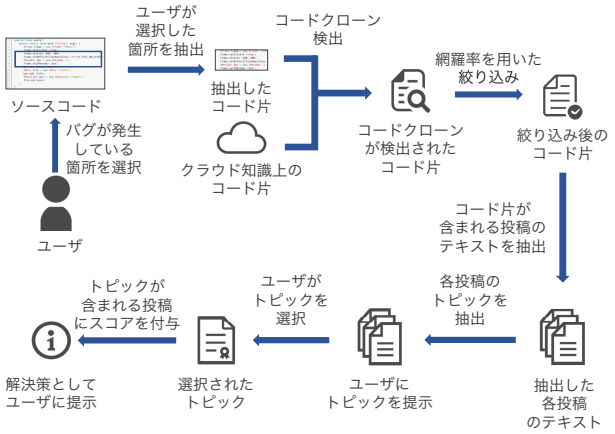

図 1 解決策をユーザに提示するまでの流れ
Fig. 1 The process of presenting a solution to a user

3. 構築するシステム

本研究で構築するシステムの処理の流れを**図 1** に示す．本システムは，統合開発環境の拡張機能として構築する．本システムでは，まず，ユーザにバグが発生していると思われる箇所をソースコードから選択してもらう．ユーザが選択したコード片とクラウド知識の投稿から抽出したコード片を用いてコードクローン検出を行う．コードクローンが検出された投稿から解決策となりうる投稿を探すため，網羅率を用いた投稿の絞り込みを行う．より解決策に近い投稿を推薦するために投稿を分類する．そのため，投稿を象徴するようなトピックを抽出し，ユーザに提示する．ユーザは提示されたトピックから，知りたい情報に近いトピックを選択する．ユーザが選択したトピックが含まれる投稿は多く存在するため，スコア付けを行いユーザに提示する．スコア付けはトピック内の単語の出現率を用いて算出する．以降の節では，それぞれの手法について詳しく述べる．

3.1 コードクローン検出

本研究では，クラウド知識上に存在するコード片を元にユーザの状況に適した解決策となりうる投稿を探すため，コードクローン検出手法を用いる．Bellon らはコードクローンを以下の 3 つのタイプに分類している [5]．

```
1  public class swing {
2      public static void main (String[] args) {
3          JFrame frame = new JFrame ("menu");
4          frame.setVisible (true);
5          frame.setSize (400, 400);
6          frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
7          JMenuBar bar = new JMenuBar ();
8          frame.setJMenuBar (bar);
9          bar.setVisible (true);
10         JMenu file = new JMenu ("File");
11         bar.add (file);
12         JMenuItem open = new JMenuItem ("open");
13         file.add(open);
14     }
15 }
```

コード片全体の行数：15
コードクローンが検出された行数：8

$$\text{網羅率} = \frac{8}{15} \times 100 \fallingdotseq 53\%$$

図 2 網羅率算出の例
Fig. 2 Example of code clone detection results

タイプ 1 空白の有無，コメントの有無などを除いて完全に一致するコードクローン

タイプ 2 タイプ 1 の違いに加えて変数名や関数名，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，削除，修正が行われたコードクローン

Chen らの構築したシステムでは，タイプ 2 までのコードクローンを検出可能な手法を用いている．タイプ 3 までを対象とすると，類似性の低いコードクローンまで検出されるため，タイプ 2 までのコードクローンを対象としている．しかし，網羅率を用いた絞り込みやテキストを用いた分類で関係のない投稿を除外することができると考えられる．そのため，本研究ではタイプ 3 までのコードクローンを対象とした検出を行う．タイプ 3 までのコードクローンを検出可能なツールとして CloneGear がある [6]．CloneGear は，文の挿入・削除がされているなどコードの不一致を考慮した字句単位での検出を行う．字句単位での検出では，ソースコードに対して字句解析を行い，トークンサイズ以上連続して一致している部分をコードクローンとして検出する．従来の字句単位での検出ではタイプ 2 までしか検出ができないが，CloneGear では Smith Waterman アルゴリズム [7] を採用することでタイプ 3 までの検出を可能にしている．Smith Waterman アルゴリズムとは，2 つの配列のペアから一致または類似する部分配列を検出するアルゴリズムである．字句単位の検出では，コードの依存関係を考慮した非連続コードクローンの検出はできない．依存関係を考慮した検出はプログラム依存グラフを用いることで可能になるが，計算に膨大な時間を要するのに加え，検出の精度が低い [8]．また，クラウド知識上のコード片の長さはさまざまであり，プログラム依存グラフが構築できず，コードクローン検出を行うことができない場合がある．字句単位での検出は，プログラム依存グラフを用いた検出に比べて高速かつ精度の高い検出が可能である．そのため，本研究では字句単位での検出を行う CloneGear を用いてコードクローン検出を行う．

表 1 SO の投稿を用いて構築したトピックモデルの代表的な単語
Table 1 Representative words for the topic model created using the SO posts

トピック	代表的な単語
トピック 1	list, help, methods
トピック 2	Java, function, case
トピック 3	request, classes, User

3.2 網羅率

本研究ではタイプ 3 までのコードクローンを対象としているため，多くの投稿のコード片からコードクローンが検出される．そのため，Schramm らの提案するシステムで用いられている網羅率を用いて投稿の絞り込みを行う [9]．網羅率とは，クラウド知識の投稿から抽出したコード片全体に対して，コードクローンが検出される行数の割合を指す．一般的にコードクローン検出では print 文 1 行のみのコードクローンが多く検出される．そのような場合はコードの構造的な類似が低いと考えられる．そのため，網羅率を用いて構造的な類似が低いコード片を除外する．網羅率は次式で算出される．

$$\text{網羅率} = \frac{\text{コードクローンとして検出された行数}}{\text{コード片全体の行数}} \quad (1)$$

図 2 はコードクローンが検出されたコード片の例である．青く強調表示されている部分がコードクローンとして検出された部分を指す．この例では，コード片全体の行数が 15 行に対して，コードクローンとして検出された行数は 8 行であり，網羅率は約 53%となる．

3.3 トピックモデル

本研究ではトピックの抽出を行うために，2.2 節で述べた Allamanis らの手法を参考にトピックモデルを構築する．一般にモデルの学習に使用するテキストデータの作成には，形態素解析を用いる．**表 1** は SO の投稿を用いてトピックモデルを構築した場合の代表的な単語の一部である．このように，形態素解析を用いてテキストデータの作成を行うと，トピックを代表する語句は単語として抽出される．そのため，そのトピックが何を表しているのかを推測するのが困難である．2.2 節で述べたように，Allamanis らは動詞句のみを対象にモデルの学習を行うことで問題の原因ごとに SO の投稿を分類可能であることを示している．しかし，投稿で用いられている API 名や関数名まではわからない．そこで本研究では，「投稿者が何をしたいか」＋「メソッド名」のようなトピックの抽出を目標とする．名詞句，動詞句など複数の単語からなるまとまりを特定することができる構文解析手法に制約付き解析がある．制約付き解析を用いると，形態素解析よりも大きいまとまりで分かち書きをすることが可能なため，本研究で目標とするトピックの抽出が行えると考えられる．そこで本研究では，制約付き解析を用いて本研究ではデータの整形を行う．

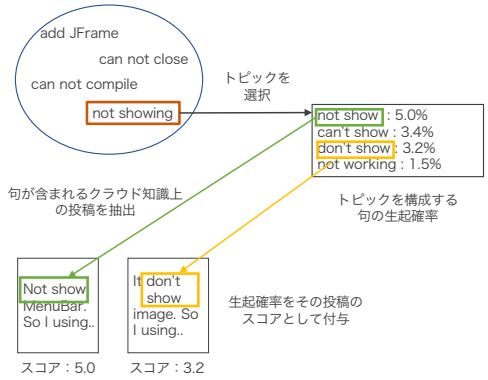

図 3 投稿のスコア付けのイメージ
Fig. 3 Example of scoring a post

LDA を用いたトピックモデルの構築を行う際，事前にトピック数を定義する必要がある．Allamanis らは詳細なトピックの抽出を行うために，トピック数を 150 に定義して学習を行っている．本研究ではトピックをユーザに提示するため，トピック数が 150 の場合，ユーザが探す際の負担が大きい．しかし，トピックス数が少なすぎると，そのトピックが何を表すかを推測するのが困難である．そこで LDA の評価指標である coherence を用いてトピック数の定義を行う．

本研究ではユーザにトピックの提示を行うために，各トピックのラベル付けを行う．適切なトピック数を定義してもトピックを構成する句からそのトピックが何を表しているのかを推測するのが困難な場合もある．そのため，トピックを構成する句からラベルを作成する．Mei らはチャンキングと Ngram Testing によってテキストデータからトピックのラベルを生成している [10]．チャンキングでは，チャンクの出現頻度をもとにラベルの作成することができる．チャンキングを用いることで文法的に意味のあるラベルを生成できる．Ngram Testing では統計的検定に基づいて意味のあるラベルを生成できる．そこで，本研究でも Mei らの手法を用いてラベルの生成を行えるか検証する．

推測されたラベルをユーザに提示し，ユーザが選択したトピックを解決策のスコア付けに用いる．現在は生起確率の高い句が含まれる投稿の提示を検討している．生起確率とは，トピックに関連する単語がそのトピックにおいて生起する確率のことを指す．スコア付けは以下の手順で行う．最初にクラウド知識から句が含まれる投稿を抽出する．次に句の生起確率をその投稿のスコアとして付与する．最後にスコアの高い順にユーザに解決策となりうる投稿として提示する．**図 3** は投稿のスコア付けの例である．図 3 では，「not showing」のラベルが付与されたトピックをユーザが選択している．「not showing」のラベルが付与されたトピックを構成する句の生起確率は，「not show」が 5%，「can't show」が 3.4%，「don't show」が 3.2%，「not working」が 1.5%となっている．クラウド知識上に「not show」，「don't show」のそれぞれが含まれる投稿があった場合，「not show」が含まれている投稿のスコアは 5.0，「don't show」が含まれている投稿のスコアは 3.2 となる．

表 2 網羅率の検証結果
Table 2 Verification results of coverage ratio

網羅率（%）	コードクローンが検出されたコード片（個）	解決策となるコード片（個）
3.5 未満	1257	0
3.5 - 7.0	468	0
7.0 - 10.5	177	8
10.5 - 14.0	88	0
14.0 - 17.5	34	4
17.5 - 21.0	37	0
21.0 - 24.5	8	1
24.5 - 28.0	30	0
28.0 - 31.5	6	0
31.5 - 35.0	15	0
35.0 - 38.5	1	0
38.5 - 42.0	1	0
42.0 - 45.5	1	0

4. 網羅率の検証

コードクローンが検出されたコード片を用いて網羅率の検証を行なった．コードクローン検出には，3.1 節で述べた CloneGear を用いた．検証ではまず，クラウド知識のコード片として SO で Java のタグが付与されている 5000 件の投稿から 22766 個のコード片を抽出した．次に，バグの発生しているコード片として teratail から 4 件の投稿に含まれる 4 個のコード片の抽出を行なった．クラウド知識のコード片には，それぞれの解決策となる投稿のコード片を 4 個含めている．これは，解決策となりうる投稿のコード片とクラウド知識のコード片，それぞれの網羅率を算出するためである．4 件の投稿から検出されたコードクローンの総数は 2123 個であった．

検証の結果を**表 2** に示す．解決策となりうるコード片の網羅率は 7%〜21%であり，3.5%未満のコード片が 1257 個，3.5%以上 7%未満のコード片は 468 個となり，合計 1725 個のコード片の網羅率は 7%と低い数値になっていた．バグの発生している 4 件の投稿のうち 1 件の投稿では，網羅率 7%未満のコード片を除外すると，485 個のコード片を 89 個まで絞り込むことができる．解決策となりうるコードの割合が網羅率を用いた絞り込みを行う前は約 1%だったものが，約 6%に向上した．今回の検証では網羅率を 7%に設定することで解決策となりうるコードの割合が 5 ポイント向上した．網羅率が 35%以上のコード片が含まれる投稿は，解決策にはなり得ないような投稿であった．そのため，

網羅率の値によって，解決策となりうる投稿が出現しやすくなるとは限らないことが示された．しかし，別のバグが発生しているコード片を用いた場合やクラウド知識のコード片を増やした場合，解決策となりうるコード片の網羅率の分布が変化する．また，今回の検証のように網羅率 7% 未満のコード片を除外した場合，解決策となりうるコード片が除外される可能性があるため，クラウド知識上のコード片を増やし，様々な種類のバグが発生しているコード片を用いて検証を行う必要がある．

5. おわりに

本稿ではクラウド知識を利用する際のユーザの負担軽減を目的としたシステムの検討を行った．SO の投稿を用いて網羅率の検証を行なった結果，解決策とならないコード片の網羅率が非常に低く，網羅率を用いることで有効な絞り込みを行うことができた．また，網羅率の値によって，解決策となりうる投稿が出現しやすくなるとは限らないことがわかった．今後は，様々な種類のバグが発生しているコード片を用いた網羅率の検証，トピックモデルの構築，評価実験を行う予定である．

参考文献

[1] Brandt, J., Guo, P., Lewenstein, J., Dontcheve, M. and Klemmer, S. R.: Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, pp.1589-1598 (2009)

[2] Chen, F. and Kim, S.: Crowd Debugging, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, pp.320-332 (2015).

[3] Allamanis, M. and Sutton, C.: Why, when, and what: analyzing stack overflow questions by topic, type, and code, *2013 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, pp.53-56 (2013).

[4] Blei, D. M., Ng, A. Y. and Jordan, M. I.: Latent dirichlet allocation, Journal of machine Learning research, vol.3, no.Jan, pp.993-1022 (2013).

[5] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Transactions on software engineering*, Vol.33, No.9, pp577-591 (2007).

[6] 村上寛明, 堀田圭佑, 肥後芳樹, 井垣宏, 楠本真二: Smith-Waterman アルゴリズムを利用したギャップを含むコードクローン検出, 情報処理学会論文誌, Vol.55, No.2, pp.981-993 (2014).

[7] Smith, T. F. and Waterman, M. S.: Identification of common molecular subsequences, Journal of molecular biology, vol147, No.1, pp195-197 (1981).

[8] 肥後芳樹, 楠本真二: プログラム依存グラフを用いたコードクローン検出法の改善と評価, 情報処理学会論文誌, Vol.51, No.12, pp2149-2168 (2010).

[9] Schramm, C., Wang, Y., and Bry, F.: Codekōan: a source code pattern search engine extracting crowd knowledge, *Proceedings of the 5th International Workshop on Crowd Sourcing in Software Engineering*, pp.1-8 (2018).

[10] Mei, Q., Shen, X., and Zhai, C.: Automatic labeling of multinomial topic models, *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.490-499 (2007).