5H-01

自動翻訳された日本語バグレポートを用いたバグ局所化の試行

上杉 正紀† 森崎 修司† 渥美 紀寿‡ 山本 修一郎†
名古屋大学大学院情報科学研究科† 京都大学学術情報メディアセンター‡

1. はじめに

継続的なソフトウェアの開発において，保守作業には多くの労力やコストがかかる．中でも，個々のデバッグにおいて毎回原因箇所を特定するのは非効率である．そのため，ソフトウェア保守を支援するための手法が必要である．

ソフトウェア保守支援のための手法としてバグ局所化[1]がある．これは，情報検索のアルゴリズムを用いて，バグレポートに出現する語彙と，ソースコードファイル内の関数名や変数名としてつけられた識別子に出現する語彙との類似度を算出する．その値によりソースコードファイルを順位付けし，上位のものから修正すべきソースコードファイルを特定しようとする手法である．

従来の多くの手法では，バグレポートが記述された言語とソースコードファイルに現れる識別子の言語の両方が英語であることを前提としている．しかし，英語圏以外で開発されるソフトウェアや英語圏以外にユーザがいるソフトウェアでは，ソースコードファイルに現れる識別子が英語で記述され，バグレポートは英語圏以外で記述されることがある．このような状況ではそれぞれの言語が異なるため，出現する語彙が異なり，従来のバグ局所化の手法はうまく動作しない．

そこで，自動翻訳エンジンを用いて英語以外で書かれたバグレポートを英語に翻訳してからバグ局所化の手法を適用するという方法が考えられる．しかし，実際のバグレポートに対して適用した例は中国語によるもののみである[2].

2. 提案手法

手法は次の手順からなる．

1. 日本語のバグレポートを英語に翻訳する．
2. 翻訳結果の文章を単語ごとに分割する．
3. ソースコードファイルから識別子を取り出す．
4. 手順 2.と手順 3.の結果をそれぞれ重み付けしてベクトルとする．
5. バグレポートごとに全ソースコードファイルとの類似度を求めていき，順位付けを行う．

手順 1.では，自動翻訳エンジンとして excite 翻訳[1]を用いる．

手順 2.では，NLTK[2]の tokenizer を用いて単語の分割を行う．このとき，同時に stemmer を用いて語幹のみを取り出し，活用形や派生形の影響が少なくなるように考慮する．

手順 4.では tf-idf による重み付けを行う．tf とはある語彙の出現頻度であり，idf とはある語彙が出現する文書数の逆数を取ったものである．tf-idf は tf と idf をかけあわせたものである．語彙tが文書dに出現する回数をf，dの全単語数（重複を許す）をT，tが出現する文書数をn，全文書数をNとすると，d内のtに対する tf-idf は以下の式によって求められる．

$$tf(t,d) = \frac{f}{T}$$

$$idf(t) = \log\left(\frac{N}{n}\right)$$

$$tf\text{-}idf(t,d) = tf(t,d) * idf(t)$$

tf-idf によって重み付けした各語彙を成分としてベクトルにする．

手順 5.ではコサイン類似度を用いてバグレポートとソースコードファイル間の類似度を計算し順位付けを行う．コサイン類似度は 2 つのベクトルがどれだけ類似しているかを 0 から 1 の値で表す手法である．ベクトル$\boldsymbol{a}, \boldsymbol{b}$を用いると，コサイン類似度$Similarity(\boldsymbol{a}, \boldsymbol{b})$は次の式によって求められる．

$$Similarity(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \bullet \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$

3. 実験と考察

3.1. 実験方法

実験にはオープンソースソフトウェアである EC-CUBE[3]のリポジトリ内のバグレポート(2014

An Empirical Evaluation of Cross-language Bug Localization by Using Japanese Bug Report
†Graduate School of Information Science, Nagoya University
‡Academic Center for Computing and Media Studies, Kyoto University

[1]http://www.excite.co.jp/world/
[2]http://www.nltk.org
[3]http://svn.ec-cube.net/open_trac/

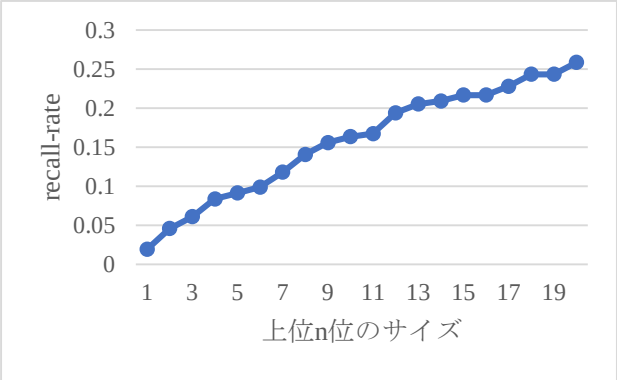

図 1 nの値ごとの recall-rate

年 11 月時点で 1859 件)を用いた．開発言語が PHP であったため，PHP の基本モジュールである token_get_all を用いてソースコードファイルから識別子（クラス名，関数名，変数名）を取り出した．クラス名，関数名は「_」で区切られた複数の語彙からなることが多かったため，「_」ごとに区切り 1 つの語彙として取り出した．また，変数の前につけられる「$」は削除した．

提案手法の評価には recall-rate[1]を用いた．recall-rate の本研究における定義は，あるバグレポートBに対して，順位付けされたソースコードファイルのリストのうち上位n位までに正解のソースコードファイルS_Bを挙げられたバグレポートの割合である．ここで正解のソースコードファイルとは，解決済みのバグレポートBにおいて実際に修正されたソースコードファイルS_Bのことを指す．本研究ではnの値を 1 から 20 まで変動させ，それぞれ recall-rate を求めた．

3.2. 実験結果

提案手法を用いて求めた，許容する上位n位のサイズに対する recall-rate の値を図 1 にまとめた．n=1 のとき 0.019，n=5 のとき 0.091，n=10 のとき 0.163，n=20 のとき 0.258 であった．

3.3. 考察

実験結果より，日本語で記述されたバグレポートを自動翻訳するとバグ局所化を行うことが可能になることがわかった．

日本語のバグレポートに記述されたバグの再現方法が英語に翻訳された際に，ソースコードファイルにおけるクラス名や関数名に出現する語彙と一致するということが多く見受けられた．多くの場合，バグの再現方法はバグレポートに含まれているので，提案手法は他の日本語のバグレポートにも適用できると考えられる．

Ruby-China[4] というオープンソースプロジェクトではバグレポートが中国語で記述されており，ソースコードファイルに出現する識別子は英語である．Xia ら[2]はこのバグレポートを Microsoft Translator で翻訳し，BugLocator[1]というバグ局所化手法を適用している．その recall-rate による評価結果は n=1 のとき 0.06，n=5 のとき 0.14，n=10 のとき 0.26 であると報告されていた．

[4]https://github.com/ruby-china/ruby-china

4. おわりに

英語圏以外で開発されるソフトウェアや英語圏以外にユーザがいるソフトウェアでは，ソースコードファイルに現れる識別子が英語で記述され，バグレポートは英語以外の言語で記述されることがある．この場合従来のバグ局所化の手法を用いて修正すべきソースコードファイルを特定することはできない．本研究では，日本語で記述されたバグレポートを使用し，自動翻訳エンジンを用いて英語に翻訳してからバグ局所化を行うという手法を試行し，recall-rate において n=1 のとき 0.019，n=5 のとき 0.091，n=10 のとき 0.163，n=20 のとき 0.258 という結果を得た．

本研究で用いた tf-idf とコサイン類似度によるアプローチは有名ではあるが新しい手法とは言い難い．近年ではより新しいバグ局所化の手法が提案され，このアプローチよりも優れていることが実証されている[1][3][4]．また，複数の自動翻訳エンジンを用いてバグレポートの翻訳を行う手法も提案されている[2]．今後はこのような，より優れた手法を日本語のバグレポートに対して適用することを検討している．

参考文献

[1] Zhou Jian, Hongyu Zhang, and David Lo. “Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports.” 2012 34th International Conference on Software Engineering (ICSE). pp. 14-24. IEEE, 2012.
[2] Xia Xin, et al. “Cross-language bug localization.” Proceedings of the 22nd International Conference on Program Comprehension. pp. 275-278. ACM, 2014.
[3] Lukins Stacy K., Nicholas A. Kraft, and Letha H. Etzkorn. “Bug localization using latent Dirichlet allocation.” Information and Software Technology, Vol. 52, No. 9, pp. 972-990, 2010.
[4] Rao Shivani, and Avinash Kak. “Retrieval from software libraries for bug localization: a comparative study of generic and composite text models.” Proceedings of the 8th Working Conference on Mining Software Repositories. pp. 43-52. ACM, 2011.