7N-03

ソースコード修正履歴を用いた自動バグ修正手法の性能理解

首藤 巧† 亀井 靖高‡ 佐藤 亮介‡ 鵜林 尚靖‡
†‡ 九州大学

1 はじめに

ソフトウェア開発においてバグ修正は重要であり，多くのコストが費やされる．Gazzola らによると，デバッグ作業がソフトウェア製品の開発コスト全体の約 50 %を占めることが多いと報告されている [1]．通常バグの修正は一部ツールを利用しながらも開発者自身が手動で行っており，このバグ修正の作業を自動化することによって開発者の負担を大きく軽減することができる．

現在提案されている自動バグ修正手法は，バグを含むソースコードとテストスイートを元にバグの修正パッチ（修正パッチはプログラムのソースコードとそれに対する編集操作）を生成する．これらの手法ではバグの修正結果はテストケースに依存する可能性が高く，テストケースには全て通るものの，実際に開発者が行うような修正とはかけ離れた無意味な修正パッチを生成する可能性がある．

自動バグ修正のテストケースへの依存による無意味な修正パッチを生成する問題を解決するため，Long らは版管理システムにある過去のバグ修正履歴から機械学習によって確率モデルを生成し，パッチ生成に利用する自動バグ修正手法である Prophet[2] を提案した．この手法では開発者が行うようなバグの修正パッチを生成することによって正しいパッチが生成できるという考えのもと，過去の開発者のパッチと類似度の高いパッチを優先的に生成する設計になっている．

Prophet では学習に利用したパッチやアプリケーションの特徴によって修正パッチの生成結果，及び修正精度が大きく変わってくるものと考えられる．そこで我々は版管理システムのバグ修正履歴を利用した自動バグ修正手法の修正結果への学習データセットの影響を明らかにしたいと考えた．本研究では複数の学習データセットによって確率モデルの学習を行い，そのモデルを用いた修正結果の比較調査を実施する．

Understanding the Performance of Automatic Program Repair Techniques Using Source Code Change Histories
Takumi Shuto†, Yasutaka Kamei‡, Ryosuke Sato‡, Naoyasu Ubayashi‡
†‡Kyushu University, Japan

†shuto@posl.ait.kyushu-u.ac.jp
‡{kamei, sato, ubayashi}@ait.kyushu-u.ac.jp

2 Prophet

2.1 概要

Prophet は過去の開発者のバグ修正履歴から機械学習により確率モデルを生成してプログラムの修復に利用する自動バグ修正手法である．Prophet は欠陥を含む実行可能なプログラム及びテストスイートを入力として受け取り，プログラムの自動修正を行う．出力はテストスイートに含まれるテストスイートを全て通過したプログラムである．

2.2 修正

Prophet は入力されたテストケースを利用して欠陥位置の特定を行う．特定された欠陥の位置に対して適用可能なプログラムの変更を修正パッチ候補として全て生成する．次にそれらの各修正パッチ候補に対して確率モデルによって優先順位を決定する．その後，優先度の高い修正パッチ候補からテストケースを実行し，全てのテストケースを通過した修正パッチ候補を修正結果として出力する．

確率モデル：Prophet は事前に版管理システムの履歴から得られた開発者の多数の修正パッチから確率モデルを学習する．開発者の修正パッチを入力された場合に最も高い確率を割り当てるように確率モデルを学習させることによって，修正パッチ候補が過去の開発者の修正パッチと類似度が高いほど確率モデルが高い確率を割り当てるようになる．

3 アプローチ

Prophet は版管理システムから学習データセットを作成し，確率モデルの学習を行う．この確率モデルによって Prophet は従来手法よりもテストケースに依存しない正しい修正が可能であると報告されているが，利用した学習データセットによって修正できるバグの種類や修正内容に大きく違いが発生するのではないかと考えた．

そこで本研究では，Prophet の確率モデルの学習に利用するデータセットを下記の指標の下で複数作成し修正結果の比較調査を実施する．

- 調査 1 ：ベンチマークアプリケーションとの関連性
- 調査 2 ：パッチの分布と構成

なおベンチマークとして使用するデータセットは Prophet の評価実験で利用されたベンチマークと同様のものを利用する．それらのベンチマークは 8 つの OSS プロジェクト，libtiff，lighttpd，php，python，wireshark，

表 1: 学習データセットによる正解パッチの順位比較

バグ	全パッチ候補数	正解パッチ順位			
		オリジナル	PHP のみ	関連性高	関連性低
php-307562-307561	22881	600	52	315	1446
php-307846-307853	17610	6936	13	5442	120
php-307914-307915	30838	1	2	5	44
php-308262-308315	70879	55	135	1146	49
php-308734-308761	10645	1415	844	1314	878
php-309111-309159	41554	338	271	209	18
php-309516-309535	21016	6050	750	5727	1866
php-309579-309580	40152	11	89	2	11
php-309688-309716	45647	44	111	505	21
php-309892-309910	27437	14	485	96	49
php-310011-310050	52498	18	815	362	543
php-310991-310999	69387	26	3	4	8
php-311346-311348	5099	14	13	2	12
libtiff-ee2ce5b7-b5691a5a	67241	13	58	393	780
libtiff-d13be72c-ccadf48a	71876	54	9	71	73
libtiff-5b02179-3dfb33b	157153	40	33	99	664
gmp-13420-13421	42424	9253	9617	6676	26234
gzip-a1d3d40-f17cbd1	33615	580	11052	7943	11020

fbc，gzip，gmp の版管理システムから抽出された 105 の欠陥である．

3.1 調査 1 : ベンチマークのアプリケーションとの関連度

動機：ベンチマークのデータセットにはプログラミング言語や画像操作ライブラリなど様々な版管理システムが含まれている．直感的にはこれらのベンチマークのアプリケーションと学習データセットに利用するアプリケーションの種類の関連度がバグの修正結果に影響を与えると考えられる．そこでベンチマークのアプリケーションと関連性の大きい学習データセットと関連性の小さいデータセットの二つを用意，学習し修正結果の比較を行う．

データセット：GitHub にて公開されている OSS プロジェクトを対象とする．ベンチマークに含まれる 8 つの OSS リポジトリの GitHub における Description の単語を用いて GitHub にて検索を行いコミット数 1,000 以上のものからそれぞれ 8 つ OSS プロジェクトを無作為に選びバグ修正履歴を抽出して学習データセットを作成する．また，比較対象として GitHub から無作為に抽出した 8 つの OSS プロジェクトから学習データセットを作成する．

3.2 調査 2 : パッチの分布と構成

動機：Sobreira らは，自動バグ修正で用いられるデータセットは修正技術を進歩させるための重要な要素であり，その特性を理解する必要があるものの，データセットに詳しいバグとそのパッチの特性について情報が含まれているケースは少ないと報告している [3]．そこで学習に利用するデータセットのパッチのサイズや分布，構成などを調べ，特徴に応じてデータセット群を作成することで各パッチの持つ特性がバグ修正結果にどう影響を及ぼすかを調査する．

データセット：データセットの各修正パッチについてパッチの行数，広がり，修正内容など各特性ごとに分類し複数データセットを用意する．対象プロジェクトについては現在検討中である．

4 現状の結果と今後の課題

調査 1 に関して，関連性の高いデータセットとして espruino，goaccess，zstd の三つの OSS から合計 41 のバグを，また関連性の低いデータセットとしては hashcatm，tig の二つの OSS から合計 33 のバグを収集し学習を行った．今回は学習による Prophet の正しい修正を行う能力の向上の度合いを検証するために，用意した学習データセットごとに確率モデルが割り当てた優先順位を割り出した．Prophet の提案論文で行われた評価実験で開発者が行う修正のような正しいパッチを生成できたベンチマークのバグに対して実験を行い，正しいパッチに割り当てられた順位を表 1 にまとめた．

表 1 について，一列目は評価実験で Prophet が正しいパッチを生成できたバグの名前を示し，二列目は Prophet の探索空間が生成し得るパッチ候補の総数を示す．三列目から六列目は各データセットから生成された確率モデルによって割り当てられた全パッチ候補の修正の優先順位における正しいパッチの順位である．この順位が高いものほどより正しいパッチであると Prophet に判断されたものである．三列目は Prophet の評価実験で収集された学習データセットによって生成された確率モデルによる順位．四列目は評価実験の学習データセットの大部分をしめた PHP のデータセットからのみ学習した確率モデルによるランクである．五列目は関連性の高い学習データセット，六列目は関連性の低い学習データセットのものである．

現状では関連性の高いデータセットと低いデータセットの実験結果の間に有意な差は見られない．また，PHP のみを学習データに利用した結果が元の PHP を含まない学習データによる結果よりも PHP のバグ修正の精度が向上していた訳ではなかったため OSS アプリケーションのドメインが今回対象としたバグの修正結果に与える影響はあまり無いと考えられる．今回修正対象としたバグは Prophet の提案論文での評価実験にも利用されたベンチマークのものであるが，それぞれのバグがどの程度アプリケーションのドメインに依存したものなのかは調査出来ていないため今後の課題としたい．他にも今後の課題として学習データセットのパッチの数を増やすことや，調査 2 における学習データセットのパッチの分析なども行なっていきたい．

謝辞 本研究は，JP15H05306, 18H04097 による助成を受けた．

参考文献

[1] Gazzola, L., Micucci, D. and Mariani, L.: Automatic software repair: A survey, *IEEE Transactions on Software Engineering* (2017).

[2] Long, F. and Rinard, M.: Automatic patch generation by learning correct code, *ACM SIGPLAN Notices*, Vol. 51, No. 1, pp. 298–312 (2016).

[3] Sobreira, V., Durieux, T., Madeiral, F., Monperrus, M. and de Almeida Maia, M.: Dissection of a bug dataset: Anatomy of 395 patches from Defects4J, *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, pp. 130–140 (2018).