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

信学技報
TECHNICAL REPORT OF IEICE.

開発者によるバグ限局を考慮した 自動バグ修正への影響に関する研究

山手　響介† 首藤　巧† 浅田　翔† 佐藤　亮介† 亀井　靖高†
鵜林　尚靖†

† 九州大学
E-mail: †{yamate,shuto,asada}@posl.ait.kyushu-u.ac.jp, ††{sato,kamei,ubayashi}@ait.kyushu-u.ac.jp

あらまし　システム開発におけるデバッグコストの削減を目的として，自動バグ修正の研究が盛んに行われている．自動バグ修正ではバグの箇所を自動で推測しそのバグを修正するバグ限局と呼ばれるステップがあるが，バグの箇所を正確に特定できるとは限らない．バグ限局に失敗すれば自動バグ修正の精度が低下し，修正時間が増加する．そこで開発者の直感を考慮してバグ限局を行えば，修正精度の向上および修正時間の短縮が期待できる．本研究ではバグ限局を手作業でも行うことができる自動バグ修正ツールを開発し，簡単なプログラムで初期実験を行い結果を分析した．簡単なプログラムにおいて，バグの箇所が分かっている場合は自動でバグ限局を行うよりも開発者によるバグ限局を行った方が，修正の精度が向上し理論的には修正時間が短くなることが明らかになった．

The Impact of Fault Localization Considering Developers' Intuition on Automatic Bug Repair

Kyosuke YAMATE†, Takumi SHUTO†, Sho ASADA†, Ryosuke SATO†, Yasutaka KAMEI†, and Naoyasu UBAYASHI†

† Kyushu University
E-mail: †{yamate,shuto,asada}@posl.ait.kyushu-u.ac.jp, ††{sato,kamei,ubayashi}@ait.kyushu-u.ac.jp

1. はじめに

ソフトウェア開発を行う際の大切な行程の一つとしてデバッグ作業があげられる．デバッグ作業は基本的に開発者の手作業で行われる為非常にコストがかかる．これを解決するために自動でバグの修正を行うツールの開発，改良が盛んに行われている．

現在提案されている自動バグ修正の手法では，バグを含むソースコードとそのテストスイートを元にバグの箇所を推測し，そのバグを修正するようなパッチを生成することでバグの修正を行う．この手法ではテストが不十分である場合や，対象のプロジェクトが大きく複雑になった場合に，バグの箇所を推測できない可能性が高くなり，自動バグ修正の精度や修正時間に悪影響を及ぼす．

そこでソフトウェアの開発者によるバグの箇所の推測結果を利用することで，自動バグ修正の精度の向上および修正時間の短縮が期待できると考え，本研究ではバグの箇所の推測を手作業でも行うことができる自動バグ修正ツールを開発し，簡単なプログラムで初期実験を行い結果を分析した．

本稿では 2 章で自動バグ修正の背景と関連研究を紹介する．3 章では自動バグ修正の課題と本研究の目的について述べる．4 章で開発中の自動バグ修正ツール jProphet を紹介する．5 章で本研究で行った初期実験について述べ，6 章で初期実験の結果および考察を述べる．7 章で妥当性の脅威について述べ，最後に 8 章でまとめと今後の課題について述べる．

表 1　SBFL の手法の例

名前	疑惑値の計算式
Jaccard	$\frac{N_{CF}}{N_{CF}+N_{UF}+N_{CS}}$
Ochiai	$\frac{N_{CF}}{\sqrt{(N_{CF}+N_{UF})\times(N_{CF}+N_{CS})}}$
Tarantula	$\frac{N_{CF}/(N_{CF}+N_{UF})}{N_{CF}/(N_{CF}+N_{UF})+N_{CF}/(N_{CS}+N_{US})}$

N_{CF}：その行を実行した失敗したテストの数
N_{CS}：その行を実行して成功したテストの数
N_{UF}：その行を実行せず失敗したテストの数
N_{US}：その行を実行せず成功したテストの数

2. 背　　景

2.1 自動バグ修正

自動バグ修正における代表的な手法として，入力としてバグを含むプログラムと失敗するテストを含むテストスイートを受け取り，全テストに通るように書き換えられたプログラムを出力する技術があり，本稿ではこの手法を対象に研究を行う．この自動バグ修正手法の修正過程は，大きく分けるとバグ限局，コード改変の 2 つに分けることができる．バグ限局ではプログラムの中からバグの原因となっている箇所を推測し，コード改変ではバグ限局によって推測された箇所を優先的に改変することでプログラムを修正する．

2.2 バグ限局

自動バグ修正におけるバグ限局は，バグを含むプログラムと，失敗するテスト，成功するテストの両方を含むテストスイートを入力として受け取り，バグの箇所を推測する技術が主流である．その中でも実行経路情報に基づいてバグ限局を行う Spectrum-Based Fault Localization [2](以降，SBFL) は近年盛んに研究されている手法の 1 つである．SBFL は失敗するテストで実行される文はバグの原因である可能性が高く，成功するテストで実行される文はバグの原因である可能性が低いという考えに基づいてバグの箇所を推定する．SBFL はこれまでに多くの手法が提案されている．その一例を表 1 に示す．

2.3 関連研究

笠井ら [3] によると，不具合を含む可能性の高いモジュールから順に，そのモジュールが正しく動作をするかどうかを確認することを目的とし，判別モデルにより得た判別得点と目視により得た評価点の 2 つを組み合わせて，モジュールを欠陥を含む可能性の大きい順にランク付けする手法が提案されており，その手法の有効性が示されている．自動バグ修正においても同様に，目視によるバグ限局を考慮することで，修正精度の向上が期待できる．

3. 動　　機

3.1 自動バグ修正の課題

自動バグ修正におけるバグ限局で全てのバグを特定できるとは限らない．バグの箇所が特定できなかった場合正しい修正が後回しにされるため，テストに通るだけの正しくないパッチが出力されやすくなり，修正に失敗する可能性が高くなる．正しいパッチが生成されたとしても，修正時間が増加すると考えられる．

3.2 本研究の目的

開発者がバグの箇所をある程度把握している場合，その情報をバグ限局の結果に反映することで，短時間かつ正確にバグを修正できると考えられる．

本研究では自動バグ修正ツールを開発し，手作業でバグ限局を行う機能を組み込み，バグの箇所が判明している前提で仮想的な実験を行うことで，開発者の直感が当たっている場合の自動バグ修正に与える影響を調査する．

4. jProphet

jProphet は現在著者を含む複数人で開発中の Java 言語を対象とした自動バグ修正ツールである．jProphet は Long らが提案した自動バグ修正技術である Prophet [1] を参考に開発している．入力として，バグを含むプログラムと失敗するテストを含むテストスイートを受け取り，プログラムの修正を行う．jProphet の修正過程を図 4. に示す．

4.1 バグ限局

jProphet ではバグ限局を SBFL と手作業の 2 つの手法に切り替えて実行することができる．

SBFL. SBFL でのバグ限局は入力としてバグを含むプログラムとそのテストスイートを受け取り，それぞれのテストの成否と実行経路情報を用いて，各行の疑惑値を算出する．疑惑値算出の計算式は 2.2 章の表 1 で示したものから選択することができる．

手作業. 手作業でのバグ限局では，ファイルパスと行番号，付与したい疑惑値を入力することで任意の箇所を任意の疑惑値にすることができる．

4.2 修正パッチ候補生成

対象ソースコードから抽象構文木 (AST) を取得し，各ノードに対しテンプレートを適用して修正パッチの候補を生成する．jProphet で用いるテンプレートを表 2 に示す．

4.3 修正パッチ候補の並び替え

バグ限局で算出された疑惑値の高い行を修正する修正パッチ候補が上位に来るように並び替える．

現状，同率順位の修正パッチ候補は生成された順に並ぶようになっている．

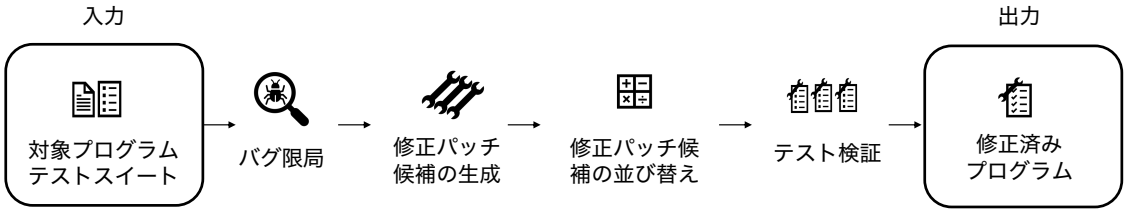

図 1　jProphet の修正過程

表 2　jProphet で用いるテンプレート

名前	操作内容
Condition Refinement	if 文の条件式の書き換え
Condition Introduction	対象の文を if 文で挟み込む
Control Flow Introduction	コントロールフロー制御文を追加
Method Replacement	関数の置換および引数の置換
Value Replacement	対象の文の中の変数などの置換
Copy and Replace	文の複製および変数の置換

4.4　テスト実行による修正パッチ候補の検証

並び替えられた修正パッチ候補を上位から順に修正対象のプログラムに適用し，入力として受け取った失敗するテストを含むテストスイートを実行する．全てのテストが成功するようなパッチが見つかった時点で修正パッチ候補の検証を終了し，そのパッチを修正パッチとして出力する．

5.　初 期 実 験

初期実験として，jProphet で小さなプログラムを修正することで手作業によるバグ限局が有効であるかを調査する．本章では初期実験で用いるデータセットおよび実験手順について説明する．

5.1　データセット

修正対象のプログラムとして，二分探索 (BinarySearch) のプログラムを 1 行書き換えることによってバグを埋め込んだものを用意する．本研究で用いるバグを埋め込む前の二分探索のプログラムは，入力としてソート済みの整数のリスト，探索したい整数一つを受け取り，リストの中から探索したい整数を探し出し，見つかればその配列のインデックス番号を，見つからなかったら-1 が出力される．バグを埋め込んだ二分探索のプログラムをプログラム 1 に示す．

テストスイートには，成功するテスト 3 つと失敗するテスト 1 つを用意する．テストの詳細を表 3 に示す．

SBFL では疑惑値の算出に用いる計算式の 1 つである Jaccard [2] を用いて疑惑値を算出する．

バグの修正には jProphet を用いるが，修正パッチ候補生成における 2 つのテンプレート Condition Refinement および Condition Introduction で生成されるパッチを適用してテスト実行による検証を行うと，実装上の問題により正常に動作しなかったため，これら 2 つのテンプレート以外で生成される修正パッチ候補のみを対象として実験を行った．

表 3　初期実験で用いるテストスイート

名前	整数のリスト	探索する整数	想定される出力	テストの成否
テスト 1	[1, 2, 3, 4, 5]	4	3	T
テスト 2	[1]	1	0	T
テスト 3	[1, 2, 3, 4, 5]	6	-1	T
テスト 4	[1, 2, 3, 4]	2	1	F

5.2　実 験 手 順

バグ限局の手法を SBFL，手作業，SBFL+ 手作業の 3 通りに切り替えて修正を行う．SBFL+ 手作業によるバグ限局では，SBFL と手作業の両方でバグ限局を行い，各行の疑惑値を比較して大きい方をその行の疑惑値とする．それぞれの手法で生成された修正パッチのランクと内容を比較する．手作業によるバグ限局は以下の 3 通りの方法で行う．

・バグの箇所一点の疑惑値を最大にする．
・バグの箇所を含む複数行の疑惑値を最大にする．（本実験ではバグの箇所を含む if 文全体を指定）
・ある行 x の疑惑値を最大にし，その前後 y 行の疑惑値を変化させる．ただし前後 y 行の中，もしくは行 x にバグの箇所を含むように y をとる．（本実験では y=2 とし，行 x から 1 行離れるごとに最大値 1 から 0.2 ずつ小さい値を付ける）

6.　実験結果と考察

6.1　結　　果

SBFL でのバグ限局によって算出された各行の疑惑値をプログラム 2 に示す．本実験で用意したバグを埋め込んだ二分探索のプログラムに対して SBFL でバグ限局を行うと，バグの箇所でない行の疑惑値の方が高くなった．

実際に生成された 2 種類のパッチ A，B をプログラム 3，~~プログラム~~ 4 に示す．パッチ A は想定していた正しい修正を行うパッチで，パッチ B は正しく動作するが無

プログラム 1　バグを埋め込んだ BinarySearch ソースコード

```
int search(int[] list, int target) {
    int l = 0;
    int r = list.length;
    int index = -1;
    while (l < r) {
        int m = (l + r) / 2;
        if (list[m] == target) {
            index = l; //index = m;
            break;
        } else if (list[m] > target) {
            r = m;
        } else if (list[m] < target) {
            l = m + 1;
        }
    }
    return index;
}
```

表 4　実験結果

	疑惑値の指定方法	出力された修正パッチ	正しいパッチの順位
SBFL	-	B	96
手作業	1 箇所を指定	A	29
	複数行を指定	B	86
	2 行前の前後を変化	A	88
	1 行前の前後を変化	A	77
	バグの前後を変化	A	29
	1 行後の前後を変化	B	60
	2 行後の前後を変化	B	96
SBFL + 手作業	1 箇所を指定	A	29
	複数行を指定	B	86
	2 行前の前後を変化	A	88
	1 行前の前後を変化	A	77
	バグの前後を変化	A	29
	1 行後の前後を変化	B	60
	2 行後の前後を変化	B	96

駄のある修正を行うパッチである．

3 通りの手法でバグ限局を行って修正を行った結果を表 4 に示す．手作業によるバグ限局と，SBFL+ 手作業によるバグ限局での修正結果は全ての疑惑値の指定方法において同じであった．SBFL でバグ限局を行った場合 B のパッチが修正パッチとして出力されたが，手作業および SBFL+ 手作業でバグ限局を行った場合は，7 つの指定方法のうち 3 つで B のパッチ，4 つで A のパッチが修正パッチとして出力された．また，手作業でのバグ限局で B のパッチが出力された 3 つの指定方法のうち 2 つは，候補パッチとして生成された A のパッチのランクが SBFL よりも高かった．

プログラム 2　SBFL によるバグ限局で算出された各行の疑惑値

```
int search(int[] list, int target) {
    int l = 0;                                  0.25
    int r = list.length;                        0.25
    int index = -1;                             0.25
    while (l < r) {                             0.25
        int m = (l + r) / 2;                    0.25
        if (list[m] == target) {                0.25
            index = l; //index = m;             0.3
            break;                              0.3
        } else if (list[m] > target) {          0.3
            r = m;                              0.5
        } else if (list[m] < target) {          0.0
            l = m + 1;                          0.0
        }
    }
    return index;                               0.25
}
```

プログラム 3　jProphet が生成したパッチ A

```
        int m = (l + r) / 2;
        if (list[m] == target) {
-           index = l;
+           index = m;
            break;
```

プログラム 4　jProphet が生成したパッチ B

```
        int m = (l + r) / 2;
        if (list[m] == target) {
            index = l;
+           index = m;
            break;
```

6.2 考　　察

SBFL が失敗するのはどのようなときか．本研究で用いたバグを埋め込んだ二分探索のプログラムにおいて，表 3 のテスト 1，テスト 2，テスト 4 のように探索したい要素がリストの中にあるようなテストを実行した場合，バグの箇所が必ず実行される．よって探索したい要素がリストの中にあり，かつ成功するようなテストが増えればバグの箇所の疑惑値が下がる．二分探索のプログラムに限らず，このように一定の条件を満たすプログラムとテストスイートを入力として受け取ると，SBFL によるバグ限局が失敗する場合がある為，バグの箇所の目処がつ

いていれば手作業によるバグ限局が有効であると考えられる．

一般の大きなプロジェクトではどうなるか．本研究で用いた二分探索のような小さなプログラムで SBFL が失敗することから，一般の大きなプロジェクトでも SBFL が失敗することは十分に考えられる．SBFL が失敗した場合，対象プロジェクトが大きいほど正しいパッチの順位が大きく下がるため，一般の大きなプロジェクトでは正しいが無駄のあるパッチやテストに通るだけの正しくないパッチが出力される可能性が高くなる．よって一般の大きなプロジェクトでも，バグの箇所の目処がついていれば手作業によるバグ限局が有効であると考えられる．

7. 妥当性への脅威

本研究では，開発者がプログラム内のバグの位置を把握している前提で実験を行ったが，実際にそのような状況がどれくらいあるのかが明らかではないので，今後調査する必要がある．

8. まとめと今後の課題

本研究では，手作業でのバグ限局を行う機能を組み込みんだ自動バグ修正ツールを開発し，小さなプログラムで初期実験を行うことで，手作業でのバグ限局が自動バグ修正の結果へ及ぼす影響について調査した．SBFL でのバグ限局で修正を行うと正しいパッチが出力されないが，手作業によるバグ限局で修正を行うと正しいパッチが出力される場合があることが確認できた．

今後の課題としては，OSS から収集した修正履歴を元に修正パッチ候補にスコアを付ける機能を実装し，自動で行うバグ限局の精度が向上した時の，手作業によるバグ限局の有効性を調査することや，一般の大きなプロジェクトに対して実際に調査を行うことが挙げられる．また，今後の展望としてエディタ上で簡単に使用することができるツールに改変することが挙げられる．

文　献

[1] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '16*, pp. 298–312, 2016.

[2] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, Vol. 42, No. 8, pp. 707–740, Aug 2016.

[3] 則充笠井, 修司森崎, 健一松本. 目視評価と判別モデルを組み合わせた fault-prone モジュールのランク付け手法. 情報処理学会論文誌, Vol. 53, No. 9, pp. 2279–2290, sep 2012.