修士学位論文

題目

SBFL の精度向上を目的とした実行経路に基づくテストのグループ化

指導教員

楠本 真二 教授

報告者

入山 優

令和 5 年 2 月 1 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻

令和 4 年度 修士学位論文

SBFL の精度向上を目的とした実行経路に基づくテストのグループ化

入山 優

内容梗概

デバッグ作業の負担軽減を目的として，欠陥限局に関する研究が盛んに行われている．欠陥限局とはプログラム中の欠陥箇所を特定する技術である．Spectrum-Based Fault Localization（以下 SBFL）は欠陥限局の手法の 1 つで，各テストケースの成否と実行経路をもとに欠陥箇所を特定する．しかし，SBFL はプログラム中に欠陥箇所が 1 つしかないという仮定のもとに設計されているため，複数の欠陥箇所が存在する場合，各欠陥箇所を高い精度で特定できない．そこで，複数の欠陥箇所を同時に特定するのではなく，各欠陥箇所を個別に特定した後にその結果を組み合わせることで欠陥限局の精度を向上できないかと考えた．本研究では，実行経路に基づいてテストケースをグループ化する手法を提案する．実行経路が類似している失敗テストは同じ欠陥箇所を実行している可能性が高く，その欠陥箇所の特定に役立つ．また，失敗テストと実行経路が類似している成功テストはテストの成否を分ける情報を持っているため，欠陥限局の精度向上に役立つ．実行経路が類似した失敗テストをグループ化し，グループごとに失敗テストと実行経路が類似した成功テストを用いて欠陥箇所を特定する．各グループで得られた結果を組み合わせることで，欠陥限局の精度向上を目指す．データセット Defects4J に含まれる 78 個の欠陥を対象に実験を行い，提案手法は SBFL より高い精度で欠陥限局できることを確認した．提案手法により 21.8% の欠陥箇所において精度が向上し，最大で 1.3% 向上した．28.9% の欠陥箇所において精度が低下したが，精度の低下は最大でも 0.12% に抑えられた．

主な用語

欠陥限局，グループ化，SBFL

目次

図目次

表目次

1　はじめに

ソフトウェア開発において，デバッグは負担の大きい作業である．ソフトウェア開発に必要なコストのうち，半分以上をデバッグ作業が占めているという報告もある [1, 2]．そのため，デバッグの負担を軽減するための研究が盛んに行われている．

デバッグ作業の負担を軽減する技術の 1 つとして欠陥限局がある．欠陥限局とはプログラム中の欠陥箇所を特定する技術である．欠陥限局の手法は様々で，これまでに多くの手法が提案されている [3–6]．その 1 つに Spectrum-Based Fault Localization（SBFL）がある．SBFL は各テストケースの成否と実行経路をもとに欠陥箇所を特定する．失敗したテストケースで実行された文は欠陥箇所の可能性が高く，成功したテストケースで実行された文は欠陥箇所の可能性が低いというアイデアに基づいている．SBFL は入力として欠陥を含むプログラムとテストスイートを受け取ると，各文について疑惑値を出力する．疑惑値とは欠陥箇所を含んでいる可能性の高さを表す指標であり，値が大きいほど欠陥を含んでいる可能性が高いことを表す．疑惑値が大きい文から順に調べることで，欠陥箇所を特定できる．

SBFL はプログラム中に欠陥箇所が 1 つしかないという仮定のもとに設計されている [7]．しかし，実際の開発では，1 つのプログラムに複数の欠陥箇所が存在する場合が多い [8]．そのような場合，SBFL は高い精度で各欠陥箇所を特定できない．

そこで，複数の欠陥箇所を同時に特定するのではなく，各欠陥箇所を個別に特定した後にその結果を組み合わせることで欠陥限局の精度を向上できないかと考えた．本研究では，実行経路に基づいてテストケースをグループ化する手法を提案する．実行経路が類似している失敗テストは同じ欠陥箇所を実行している可能性が高く，その欠陥箇所の特定に役立つ．また，失敗テストと実行経路が類似している成功テストはテストの成否を分ける情報を持っているため，欠陥限局の精度向上に役立つ．実行経路が類似した失敗テストをグループ化し，グループごとに失敗テストと実行経路が類似した成功テストを用いて欠陥箇所を特定する．各グループの結果を組み合わせることで，欠陥限局の精度向上を目指す．

データセット Defects4J に含まれる 78 個の欠陥に対して SBFL と提案手法を適用し，評価実験を行った．提案手法は SBFL より高い精度で欠陥限局できた．提案手法により 21.8% の欠陥箇所において精度が向上し，最大で 1.3% 向上した．28.9% の欠陥箇所において精度が低下したが，欠陥限局の精度の低下は最大でも 0.12% に抑えられた．

以降，2 節で SBFL について述べ，3 節で提案手法のキーアイデアについて述べる．4 節で提案手法について述べ，5 節では本研究で設定した Research Question について述べる．6 節でその Research Question に答えるために行った実験とその結果について述べる．7 節では実験結果に対する考察について述べ，8 節で本研究の妥当性の脅威について述べる．9 節で本研究の関連研究について述べ，最後に 10 節で本研究のまとめと今後の課題について述べる．

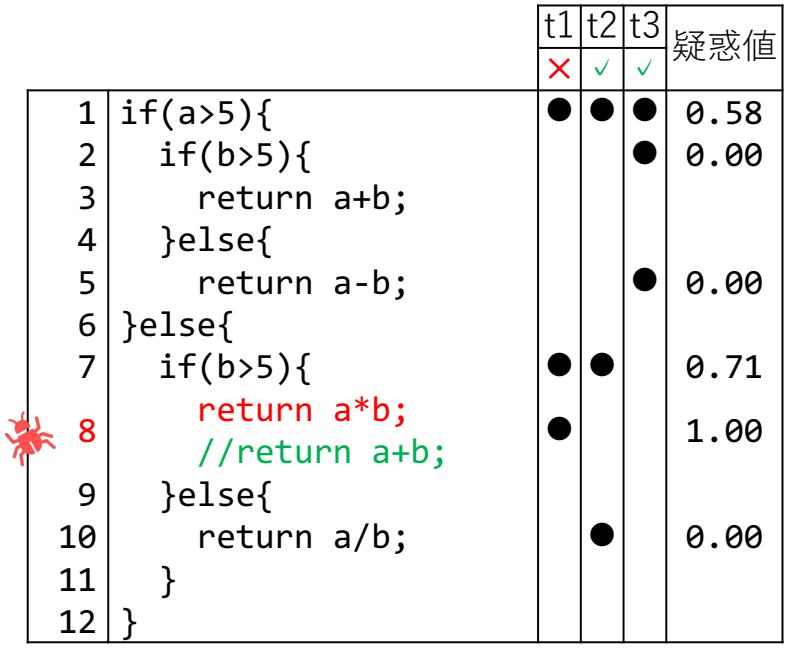


図 1　SBFL により欠陥箇所を特定できるプログラム

2　準備

欠陥限局とはプログラム中の欠陥箇所を特定する技術である．欠陥限局の手法の 1 つに，Spectrum-Based Fault Localization（SBFL）がある．この手法では，欠陥を含むプログラムとテストスイートを受け取ると各テストケースの成否と実行経路を収集し，その情報をもとに疑惑値を算出する．疑惑値とは欠陥箇所を含んでいる可能性の高さを表す数値である．疑惑値の計算手法には様々な種類がある．Abreu らは，7 つの疑惑値の計算手法を比較し，Ochiai [9] が優れた手法であると結論付けている [10]．そのため，本研究では疑惑値の算出に Ochiai を利用する．Ochiai の計算式は以下の式で表される．

$$susp(s) = \frac{fail(s)}{\sqrt{totalfails \times (fails(s) + pass(s))}} \tag{1}$$

式中の各変数は以下の意味を表す．

s: 疑惑値計算対象のプログラム文

$susp(s)$: s の疑惑値

$fail(s)$: s を実行した失敗テストの数

$pass(s)$: s を実行した成功テストの数

図 1 は SBFL により欠陥箇所を特定できるプログラムである．このプログラムは 8 行目が欠陥箇所

であり，t1，t2，t3 はテストケースを表している．8 行目を実行する失敗テストは 1 つであり，この行を実行する成功テストはない．1 行目や 7 行目を実行する失敗テストは 1 つだが，これらの行を実行する成功テストが存在する．そのため，8 行目の疑惑値は 1 行目や 7 行目の疑惑値より大きくなり，高い精度で欠陥限局できている．

3　キーアイデア

SBFL はプログラム中に欠陥箇所が 1 つしかないという仮定のもとに設計されているため，プログラム中に複数の欠陥箇所が存在する場合，各欠陥箇所を高い精度で特定できない．そこで，複数の欠陥箇所を同時に特定するのではなく，各欠陥箇所を個別に特定した後にその結果を組み合わせることで欠陥限局の精度を向上できないかと考えた．欠陥箇所を高い精度で特定するためには，全失敗テストのうちその欠陥箇所を実行する失敗テストだけを用いることが重要である．しかしながら，欠陥箇所は未知の情報であるため，欠陥箇所を実行する失敗テストだけを用いて欠陥限局することは難しい．そこで，実行経路が類似した失敗テストは同じ欠陥箇所を実行する可能性が高い [11] ことに注目し，実行経路が類似した失敗テストだけを用いて欠陥限局する．また，失敗テストと実行経路が類似した成功テストは欠陥限局の精度向上に役立つため [12]，実行経路が類似した成功テストも用いて欠陥限局する．実行経路が類似した失敗テストとその失敗テストと実行経路が類似した成功テストを用いることで，各欠陥箇所を高い精度で特定できる．

図 2 は実行経路が類似した失敗テストとその失敗テストと実行経路が類似した成功テストだけを用いることで欠陥限局の精度が向上するプログラムの例である．図 2 のプログラムは 1 行目と 8 行目が欠陥箇所である．全てのテストケースを用いた場合，1 行目の疑惑値が最も大きいので高い精度で欠陥限局できている．しかし，8 行目の疑惑値は欠陥箇所ではない 2 行目の疑惑値よりも小さく，高い精度で欠陥限局できていない．これは 8 行目を実行する失敗テストの数が 2 行目を実行する失敗テストの数より少ないためである．欠陥箇所である 1 行目の直後の行を実行し，実行経路が類似したテストケース `t1`，`t2`，`t3` だけを用いて欠陥限局した場合，1 行目と 2 行目の疑惑値が最大となる．欠陥箇所である 8 行目の直前の行を実行し，実行経路が類似したテストケース `t4`，`t5` だけを用いて欠陥限局した場合，8 行目の疑惑値が最大となる．そのため，実行経路が類似したテストケースだけを用いることで各欠陥箇所を高い精度で特定できている．

本研究では，実行経路に基づいてテストケースをグループ化し，グループごとに欠陥限局して得られる疑惑値を組み合わせる手法を提案する．

		t1	t2	t3	t4	t5	疑惑値		
		×	×	✓	×	✓	全て	{t1,t2,t3}	{t4,t5}
1	if(a>=5){ //if(a>5){	●	●	●	●	●	0.78	0.82	0.71
2	if(b>5){	●	●	●			0.67	0.82	
3	return a+b;	●					0.58	0.71	
4	}else{								
5	return a-b;		●	●			0.41	0.50	
6	}else{								
7	if(b>5){				●	●	0.41		0.71
8	return a*b; //return a+b;				●		0.58		1.00
9	}else{								
10	return a/b;					●	0.00		0.00
11	}								
12	}								

図 2　実行経路が類似したテストケースだけを用いることで精度が向上するプログラムの例

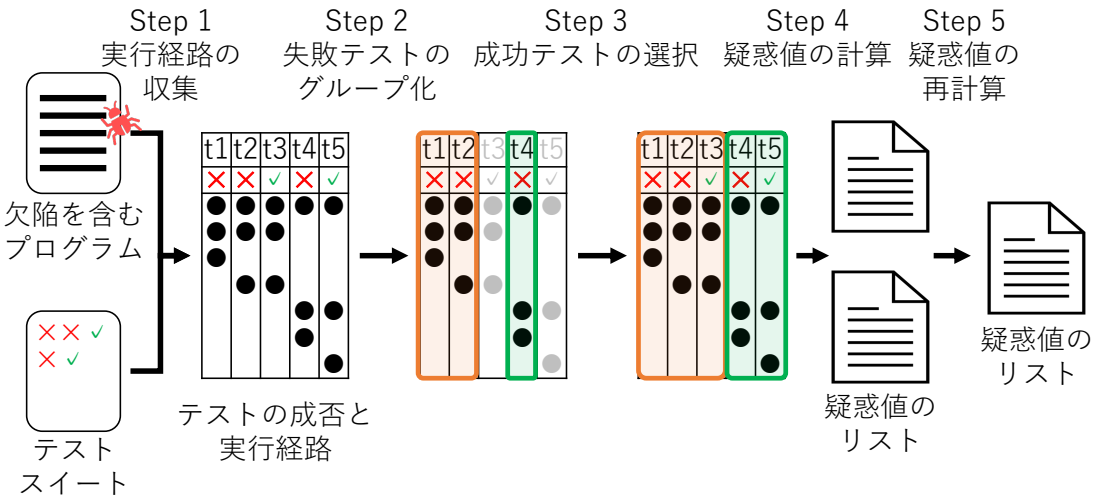


図 3　提案手法の概要

4　提案手法

本研究では，欠陥限局の精度を向上させることを目的として，実行経路に基づいてテストケースをグループ化する手法を提案する．複数の欠陥箇所を同時に特定するのではなく，各欠陥箇所を個別に特定した後にその結果を組み合わせることで欠陥限局の精度向上を目指す．実行経路が類似している失敗テストは同じ欠陥箇所を実行している可能性が高く，その欠陥箇所の特定に役立つ．また，失敗テストと実行経路が類似している成功テストはテストの成否を分ける情報を持っているため，欠陥限局の精度向上に役立つ．実行経路が類似した失敗テストをグループ化し，グループごとに失敗テストと実行経路が類似した成功テストを用いて欠陥箇所を特定した後に，各グループで得られた結果を組み合わせる．

提案手法の概要を図 3 に示す．提案手法は以下の 5 つのステップで構成されている．

Step 1 実行経路の収集

Step 2 失敗テストのグループ化

Step 3 成功テストの選択

Step 4 疑惑値の計算

Step 5 疑惑値の再計算

以降，各ステップについて説明する．

4.1　Step 1: 実行経路の収集

欠陥を含むプログラムに対してテストケースを実行し，全テストケースの成否と実行経路を収集する．

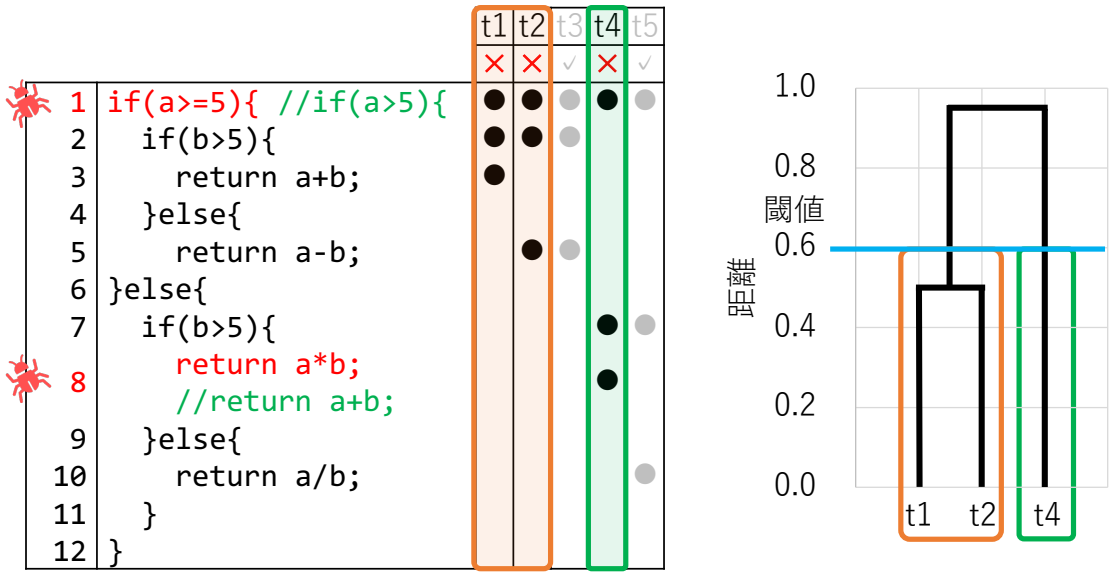


図 4　失敗テストのグループ化の概要

4.2　Step 2: 失敗テストのグループ化

Step 1 で収集した実行経路をもとに失敗テストをグループ化する．同じグループに含まれる失敗テストの実行経路は類似し，別のグループに含まれる失敗テストの実行経路は類似しないように失敗テストをグループ化する．失敗テストをグループ化するためにウォード法 [13] による階層クラスタリングを行う．失敗テストの実行経路がどの程度類似しているかを距離として測定し，距離が近い失敗テストから順にペアにしていく．距離が閾値以下の失敗テストを同じグループにまとめる．本研究では失敗テストの実行経路がどの程度類似しているかを表す距離として Jaccard 距離を用いる．Jaccard 距離は以下の式で表される．

$$jaccard(t_i, t_j) = 1 - \frac{intersection(t_i, t_j)}{union(t_i, t_j)} \tag{2}$$

式中の各変数は以下の意味を表す．

t_i，t_j: テストケース

$jaccard(t_i, t_j)$: t_i と t_j 間の Jaccard 距離

$intersection(t_i, t_j)$: t_i と t_j の両方で実行されるプログラム文の数

$union(t_i, t_j)$: t_i または t_j で実行されるプログラム文の数

失敗テストのグループ化の概要を図 4 に示す．各テストケース間の Jaccard 距離を求めると，$jaccard(t1, t2) = 0.50$，$jaccard(t2, t4) = 0.80$，$jaccard(t4, t1) = 0.80$ である．テストケース間の距

離が小さく実行経路が最も類似している t1 と t2 が最初にペアとなる．ウォード法の場合，{t1,t2} と t4 間の距離は 0.95 になり，{t1,t2} と t4 が次にペアとなる．距離が閾値以下の t1 と t2 は同じグループになり，距離が閾値より大きい t4 は別のグループになる．

4.3 Step 3: 成功テストの選択

グループごとに失敗テストと実行経路が類似した成功テストを選択する．失敗テストと成功テストの実行経路の類似度を計算し，その類似度が閾値以上なら成功テストを選択する．実行経路の類似度を次のように定義する．

$$similarity(t_f, t_p) = \frac{intersection(t_f, t_p)}{total(t_f)} \tag{3}$$

式中の各変数は以下の意味を表す．

t_f: 失敗テスト
t_p: 成功テスト
$similarity(t_f, t_p)$: t_f と t_p の実行経路の類似度
$intersection(t_f, t_p)$: t_f と t_p の両方で実行されるプログラム文の数
$total(t_f)$: t_f で実行されるプログラム文の数

グループ内に失敗テストが複数ある場合，いずれかの失敗テストについて類似度が閾値以上であれば成功テストを選択する．

4.4 Step 4: 疑惑値の計算

グループ化されたテストケースを用いて欠陥限局を行い，疑惑値を計算する．疑惑値の計算式として従来手法と同様の計算式を用いる．

4.5 Step 5: 疑惑値の再計算

グループごとに算出された疑惑値を組み合わせて提案手法としての疑惑値を計算する．提案手法の疑惑値の計算方法として様々な方法が考えられる．今回は各グループの疑惑値の平均値を提案手法の疑惑値とする方法と，各グループの疑惑値の最大値を提案手法の疑惑値とする方法をとる．値がない箇所は疑惑値を 0 として平均値と最大値を求める．以降は，平均値を取る手法を Grouping-Average Spectrum-Based Fault Localization（GASBFL）とし，最大値を取る手法を Grouping-Max Spectrum-Based Fault Localization（GMSBFL）とする．

疑惑値の再計算の概要を図 5 に示す．Step 2，Step 3 の処理によってテストケースは {t1,t2,t3} と

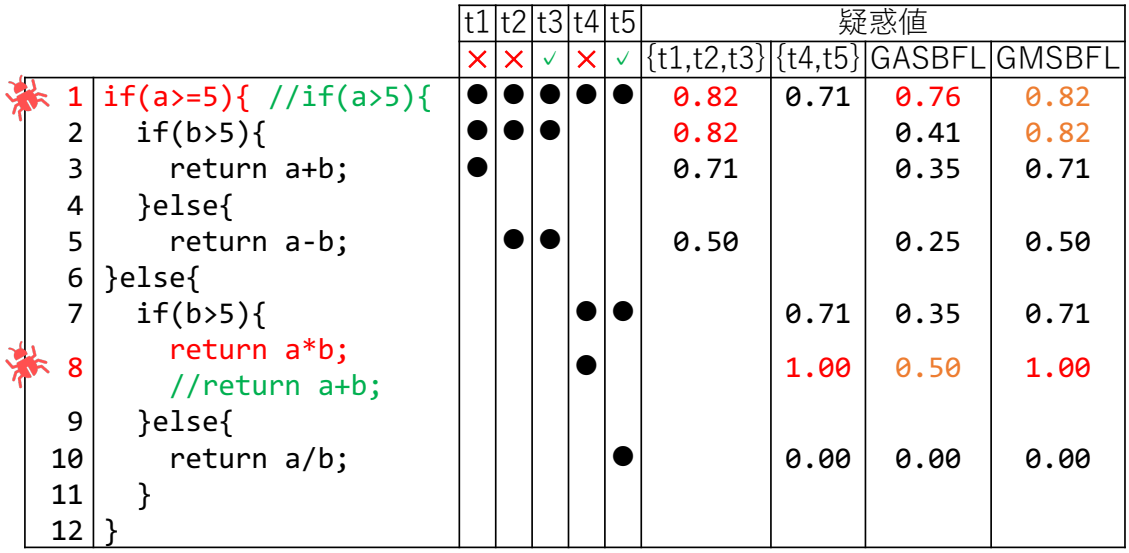


図 5　疑惑値の再計算の概要

{t4,t5} にグループ化される．Step 4 の処理によってグループごとに疑惑値が算出される．GASBFL では各グループの疑惑値の平均値をとるため，1 行目の疑惑値は 0.76 になり，8 行目の疑惑値は 0.50 になる．1 行目の疑惑値は最も大きな値であり，8 行目の疑惑値は 2 番目に大きな値である．そのため，2 つの欠陥箇所を高い精度で特定できている．GMSBFL では各グループの最大値をとるため，1 行目の疑惑値は 0.82 になり，8 行目の疑惑値は 1.00 になる．8 行目の疑惑値が最も大きく，その次に 1 行目の疑惑値が大きい．そのため，2 つの欠陥箇所を高い精度で特定できている．

5 Research Questions

提案手法を評価するために，以下の 2 つの Research Question（以下 RQ）を設定した．

RQ1 失敗テストのグループ化および成功テストの選択においてどの値が閾値として適切か
RQ2 提案手法により欠陥限局の精度が向上するか

RQ1 では，提案手法において失敗テストのグループ化で利用する距離の閾値や成功テストの選択で利用する類似度の閾値をどの値に設定すればよいかを調査する．RQ2 では，RQ1 で調査した閾値を用いて SBFL と比べて提案手法は高い精度で欠陥限局できるかを調査する．

6 実験

提案手法を評価するために 2 つの実験を行った．以降では各実験について説明する．

6.1 実験対象

本実験では，データセット Defects4J [14] に含まれる 78 個の欠陥を実験対象とした．Defects4J は欠陥限局の性能を評価するためのベンチマークとして多くの研究で利用されている [15–17]．Defects4J に含まれるプロジェクトのうち表 1 に示すプロジェクトを実験対象とした．失敗テストが 1 つの場合，提案手法は失敗テストをグループ化できないので，そのような欠陥は除外した．著者の環境において Java のバージョンの違いなどによって正しく動作しなかったプロジェクトや欠陥の個数が少ないプロジェクトは除外した．失敗テストで実行されない欠陥箇所は疑惑値が算出されない．そのような欠陥箇所は除外した．

6.2 評価指標

欠陥限局の精度を評価するための指標として EXAM を用いる．EXAM は以下の式で定義される．

$$EXAM = \frac{\text{欠陥箇所の疑惑値の順位}}{\text{プログラム文の総数}} \times 100\% \tag{4}$$

欠陥箇所の疑惑値の順位は，SBFL により算出された疑惑値の値が大きい順に文を順位づけしたときの欠陥箇所の順位である．欠陥箇所と疑惑値の値が同じ文が複数存在する場合，欠陥箇所の疑惑値の順位はそれらの中で最も低い順位とした．この評価指標は値が小さいほど欠陥限局の精度が高いことを表す．

表 1　実験対象のプロジェクト

プロジェクト名	欠陥の数	欠陥箇所の行数
jfreechart（Chart）	6	17
commons-codec（Codec）	7	16
jackson-core（JacksonCore）	9	19
jsoup（Jsoup）	16	34
commons-lang（Lang）	13	32
commons-math（Math）	27	107

6.3 実験 1

実験 1 では，RQ1 の失敗テストのグループ化および成功テストの選択における適切な閾値について調査する．

6.3.1 実験設定

失敗テストのグループ化における距離の閾値と成功テストの選択における類似度の閾値について，それぞれ 0.1–0.9 の区間を 0.1 区切りで取る．表 1 に示すプロジェクトのうち，欠陥の数および欠陥箇所の行数が最も多い Math を実験対象とした．各閾値を用いたときの EXAM を比較し，GASBFL と GMSBFL における適切な閾値を調査する．

6.3.2 GASBFL における適切な閾値

各閾値を用いて GASBFL を適用したときの EXAM の中央値について示した結果が図 6 である．距離の閾値が 0.1–0.7 区間内で，類似度の閾値が 0.3–0.4 区間内のとき，EXAM の中央値が小さくなる傾向がある．EXAM の中央値が最も低い閾値の組み合わせについて，EXAM を比較した箱ひげ図が図 7 である．類似度の閾値として 0.3 を用いたときの EXAM より 0.4 を用いたときの EXAM は小さくなる傾向がある．距離の閾値について 0.1–0.7 区間で EXAM の値はほとんど変化しない．そのため，距離の閾値として適切な値は中央値の 0.4 とする．

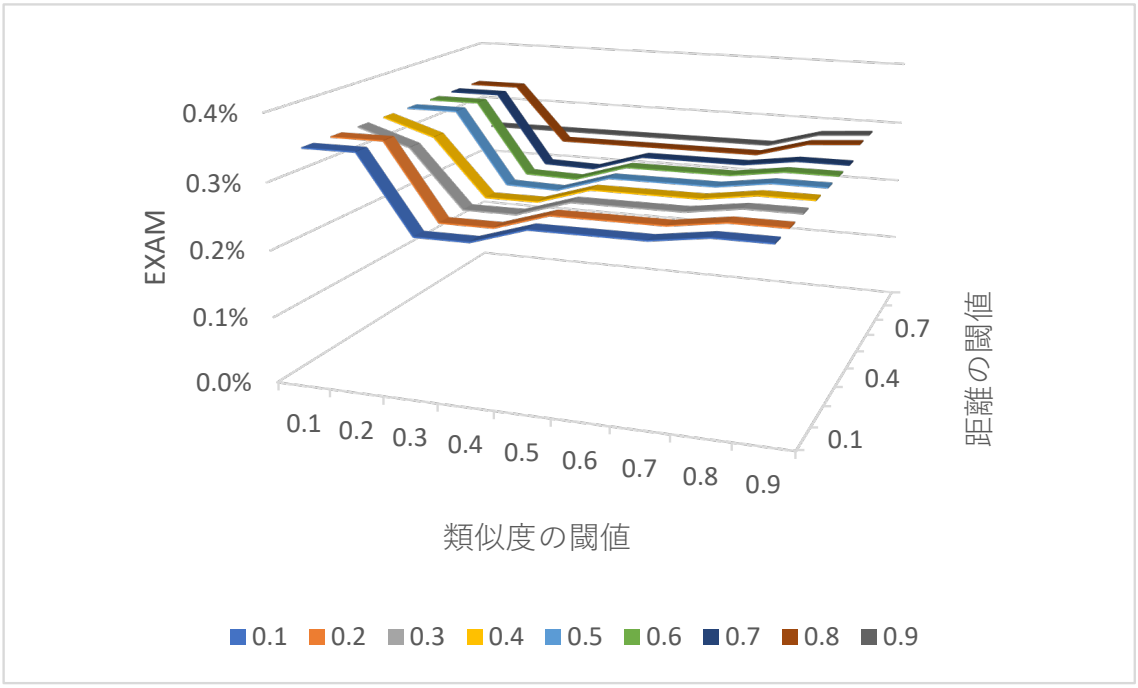


図 6　各閾値を用いたときの GASBFL における EXAM の中央値

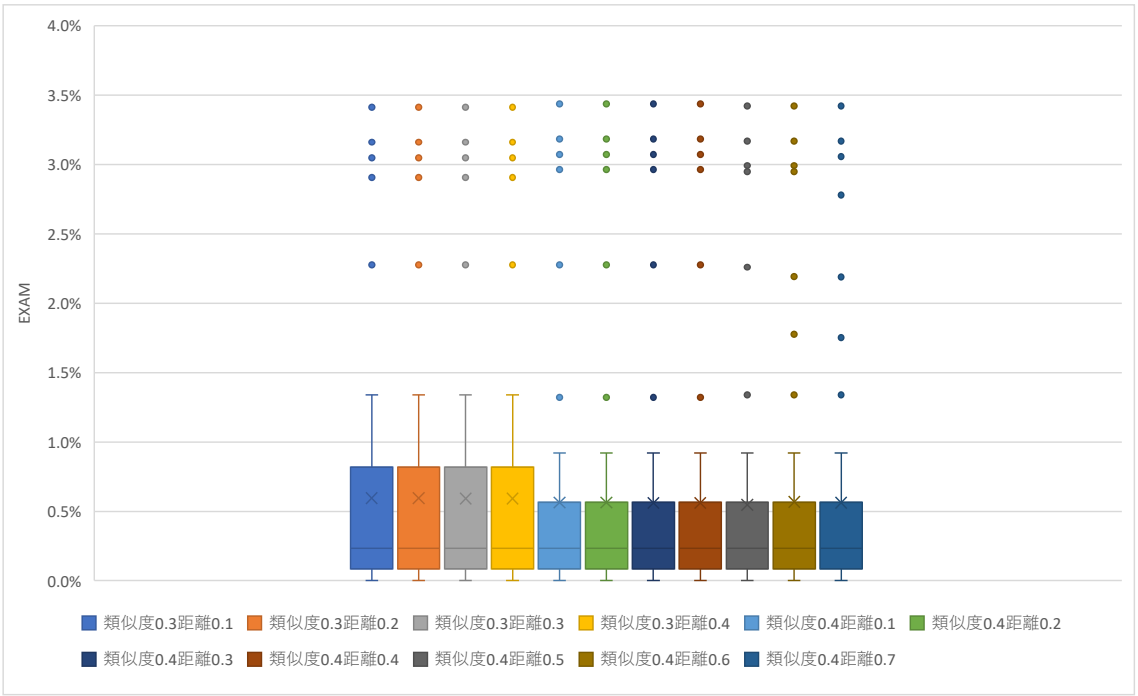


図 7　各閾値を用いたときの GASBFL における EXAM

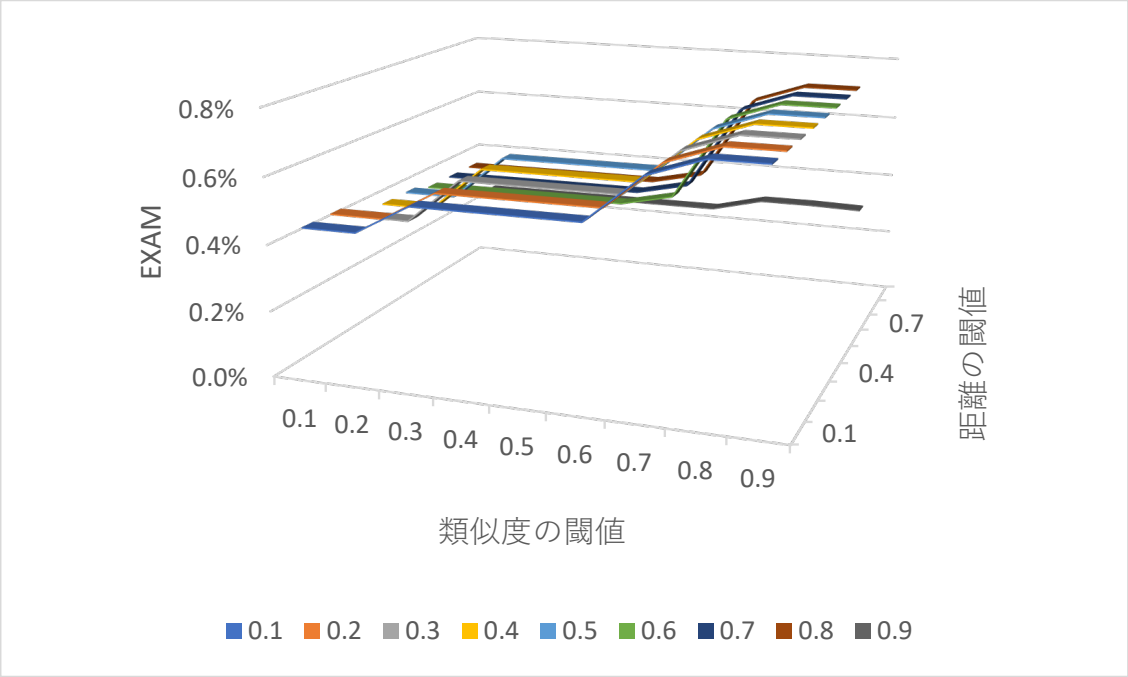


図 8　各閾値を用いたときの GMSBFL における EXAM の中央値

6.3.3　GMSBFL における適切な閾値

各閾値を用いて GMSBFL を適用したときの EXAM の中央値について示した結果が図 8 である．距離の閾値が 0.9 で類似度の閾値が 0.1–0.6 区間内のとき EXAM の中央値が小さくなる傾向がある．EXAM の中央値が最も低い閾値の組み合わせについて，EXAM を比較した箱ひげ図が図 9 である．類似度の閾値として 0.2 を用いたとき，EXAM の第 1 四分位数と第 3 四分位数，最小値，最大値が最も小さかった．

RQ1 への回答：GASBFL について距離の閾値として適切な値は 0.4 であり，類似度の閾値として適切な値は 0.4 である．GMSBFL について距離の閾値として適切な値は 0.9 であり，類似度の閾値として適切な値は 0.2 である．

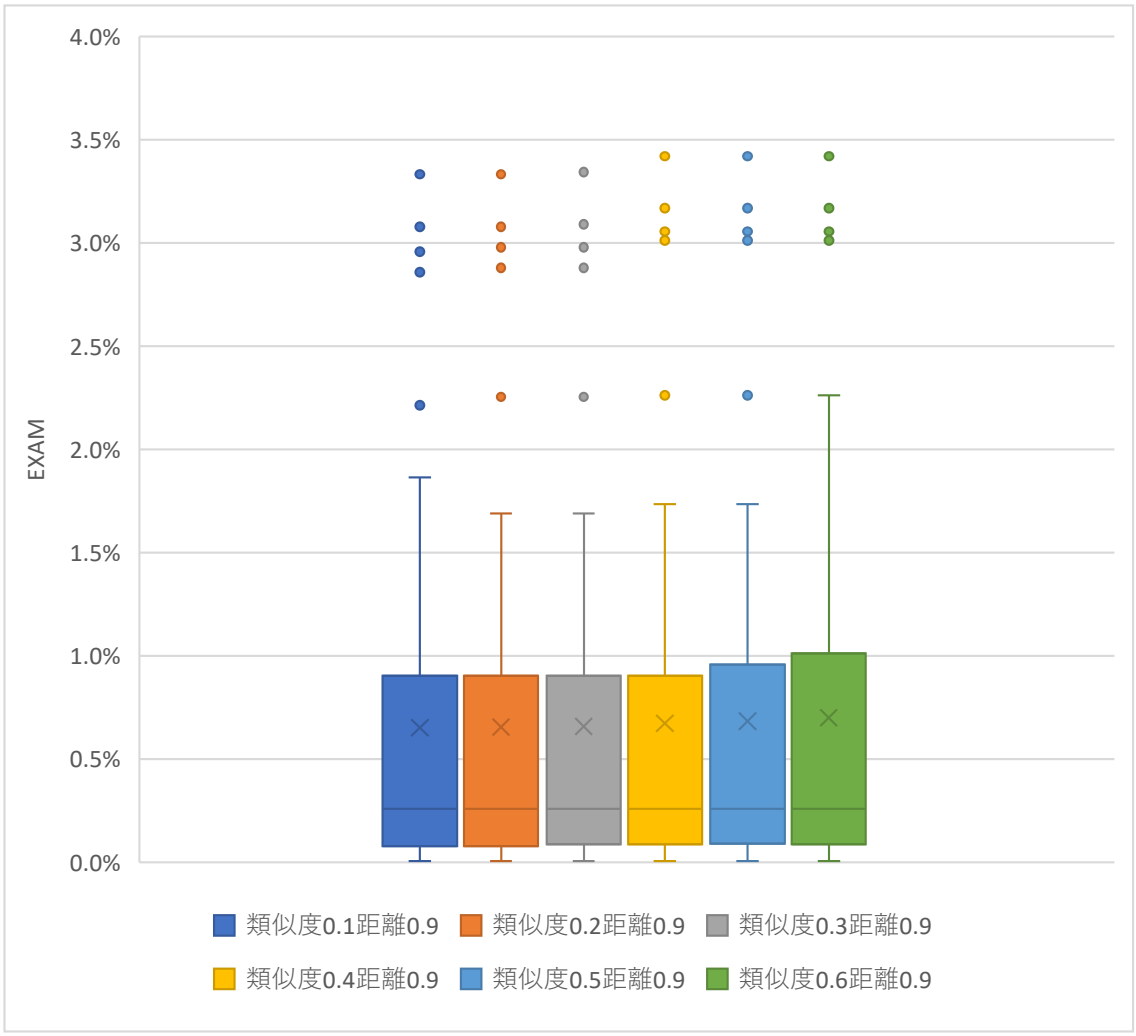


図 9　各閾値を用いたときの GMSBFL における EXAM

6.4 実験 2

実験 2 では，RQ2 の提案手法により欠陥限局の精度が向上するのかについて調査する．

6.4.1 実験設定

実験 1 の結果から GASBFL について距離の閾値として 0.4 を用い，類似度の閾値として 0.4 を用いる．GMSBFL について距離の閾値として 0.9 を用い，類似度の閾値として 0.2 を用いる．表 1 に示すプロジェクトに対して SBFL，GASBFL，GMSBFL を適用し，EXAM を比較する．

6.5 EXAM の比較

SBFL，GASBFL，GMSBFL を用いたときの EXAM に対する箱ひげ図を図 10 に示す．SBFL と GASBFL を比較すると，Chart について GASBFL の EXAM は SBFL の EXAM より大きくなったが，それ以外のプロジェクトについては GASBFL の EXAM は SBFL の EXAM と同程度もしくは小さくなった．GASBFL は SBFL より高い精度で欠陥限局できた．SBFL と GMSBFL を比較すると，GMSBFL の EXAM は SBFL の EXAM と同程度もしくは大きくなった．GMSBFL は SBFL より高い精度で欠陥限局できなかった．

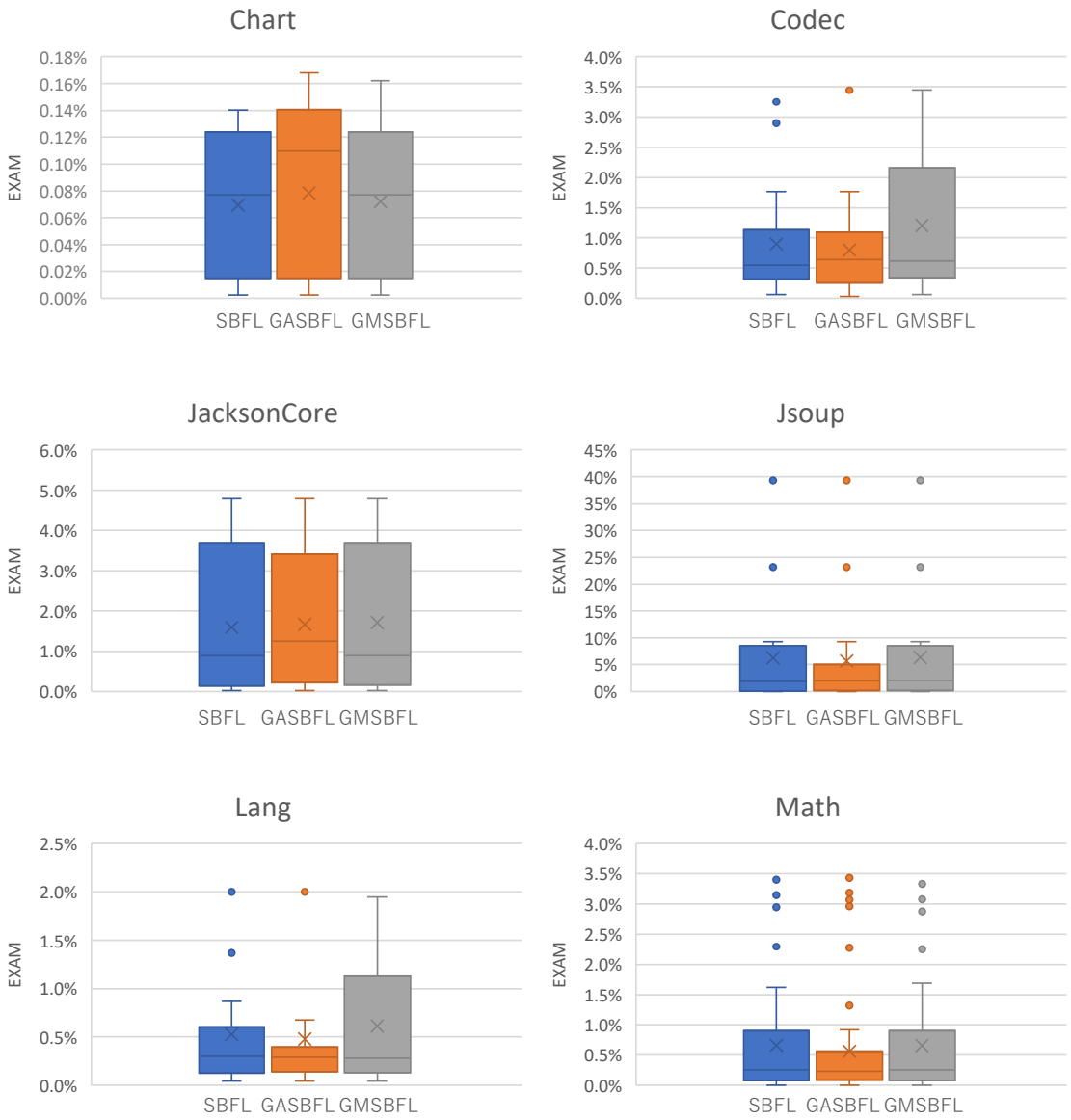


Chart
0.18%
0.16%
0.14%
0.12%
0.10%
0.08%
0.06%
0.04%
0.02%
0.00%
EXAM
SBFL GASBFL GMSBFL
Codec
4.0%
3.5%
3.0%
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
EXAM
SBFL GASBFL GMSBFL
JacksonCore
6.0%
5.0%
4.0%
3.0%
2.0%
1.0%
0.0%
EXAM
SBFL GASBFL GMSBFL
Jsoup
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
EXAM
SBFL GASBFL GMSBFL
Lang
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
EXAM
SBFL GASBFL GMSBFL
Math
4.0%
3.5%
3.0%
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
EXAM
SBFL GASBFL GMSBFL

図 10　EXAM

6.6 精度が向上した欠陥箇所の個数

各欠陥箇所について SBFL と GASBFL を用いたときの EXAM を比較し，精度が向上した欠陥箇所，精度が変化しなかった欠陥箇所，精度が低下した欠陥箇所の個数を示した結果が表 2 である．Jsoup において精度が低下した欠陥箇所の数より精度が向上した欠陥箇所の数は多かった．しかし，Jsoup および Lang 以外の 4 つのプロジェクトにおいて精度が向上した欠陥箇所の数より精度が低下した欠陥箇所の数は多かった．全体では，21.8% の欠陥箇所において精度が向上したが，28.9% の欠陥箇所において精度が低下した．

各欠陥箇所について SBFL と GMSBFL を用いたときの EXAM を比較し，精度が向上した欠陥箇所，精度が変化しなかった欠陥箇所，精度が低下した欠陥箇所の個数を示した結果が表 3 である．Codec および Math において精度が低下した欠陥箇所の数より精度が向上した欠陥箇所の数は多かった．しかし，それ以外の 4 つのプロジェクトにおいて精度が向上した欠陥箇所の数より精度が低下した欠陥箇所の数は多かった．全体では，13.3% の欠陥箇所において精度が向上したが，19.6% の欠陥箇所において精度が低下した．

表 2　SBFL と GASBFL の比較

プロジェクト名	精度向上	変化なし	精度低下
Chart	2（11.8%）	8（47.1%）	7（41.2%）
Codec	2（12.5%）	5（31.3%）	9（56.3%）
JacksonCore	5（26.3%）	6（31.6%）	8（42.1%）
Jsoup	5（14.7%）	26（76.5%）	3　（8.8%）
Lang	11（34.4%）	10（31.3%）	11（34.4%）
Math	24（22.4%）	56（52.3%）	27（25.2%）
合計	49（21.8%）	111（49.3%）	65（28.9%）

表 3　SBFL と GMSBFL の比較

プロジェクト名	精度向上	変化なし	精度低下
Chart	0　(0.0%)	15 (88.2%)	2 (11.8%)
Codec	7 (43.8%)	3 (18.8%)	6 (37.5%)
JacksonCore	0　(0.0%)	12 (63.2%)	7 (36.8%)
Jsoup	2　(5.9%)	27 (79.4%)	5 (14.7%)
Lang	8 (25.0%)	9 (28.1%)	15 (46.9%)
Math	13 (12.1%)	85 (79.4%)	9　(8.4%)
合計	30 (13.3%)	151 (67.1%)	44 (19.6%)

6.7 精度の変化の度合い

各欠陥箇所について提案手法により精度がどの程度低下・向上したのかを調べるための指標として Diff を次のように定義する．

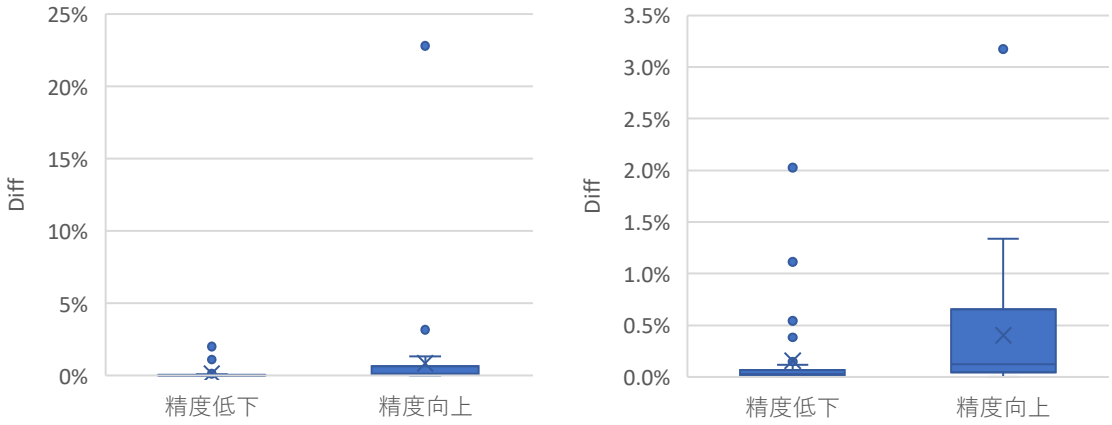
$$Diff = |EXAM_{SBFL} - EXAM_{\text{提案手法}}| \tag{5}$$

この指標は値が小さいほど精度の変化の度合いが小さいことを表す．精度が低下した欠陥箇所についての Diff と精度が向上した欠陥箇所についての Diff を比較することで，提案手法により精度がどの程度向上したのか，精度低下の度合いは小さいのかを調査する．

SBFL と GASBFL を用いたときの Diff に対する箱ひげ図を図 11(a) に示す．データのばらつき具合を比較しやすいように，外れ値 Diff=23% の欠陥箇所を取り除いたときの箱ひげ図を図 11(b) に示す．精度が低下した欠陥箇所について，Diff の中央値は 0.027% で，最大値は 0.12% であった．精度が向上した欠陥箇所について，Diff の中央値は 0.12% で，最大値は 1.3% であった．精度が向上した欠陥箇所についての Diff は精度が低下した欠陥箇所についての Diff より大きかった．GASBFL において精度向上の度合いは精度低下の度合いより大きかった．

SBFL と GMSBFL を用いたときの Diff に対する箱ひげ図を図 12 に示す．精度が低下した欠陥箇所について，Diff の中央値は 0.066% で，最大値は 0.64% であった．精度が向上した欠陥箇所について，Diff の中央値は 0.068% で，最大値は 0.070% であった．精度が向上した欠陥箇所についての Diff は精度が低下した欠陥箇所についての Diff より小さかった．GMSBFL において精度向上の度合いは精度低下の度合いより小さかった．

RQ2 への回答：GASBFL は SBFL より高い精度で欠陥限局できた．21.8% の欠陥箇所において精度が向上し，最大で 1.3% 向上した．28.9% の欠陥箇所において精度が低下したが，欠陥限局の精度の低下は最大でも 0.12% に抑えられた．GMSBFL は SBFL より高い精度で欠陥限局できなかった．



(a) 23% の外れ値あり　　(b) 23% の外れ値なし

図 11　SBFL と GASBFL の Diff

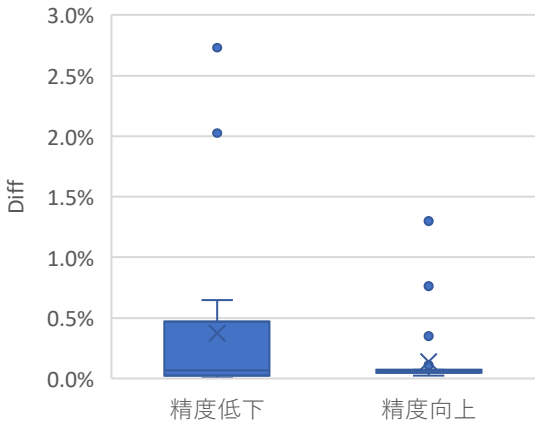


図 12　SBFL と GMSBFL の Diff

		疑惑値					
		{t1}	{t2}	{t3,t4}	{t5}	SBFL	GASBFL
1	final class SoundexUtils {						
2	static String clean(String str) {						
3	…	…	…			…	…
4	if (count == len) {	0.5000	0.2041			0.1690	0.1760
5	return str.toUpperCase();	0.5774	0.2182			0.1826	0.1989
	//return str.toUpperCase(java.util.Locale.ENGLISH);						
6	}						
7	…	…	…			…	…
8	}						
9	}						
10	public class Metaphone …						
11	public String metaphone(…						
12	…				…	…	…
13	if (txt.length() == 1) {				0.2425	0.1534	0.0606
14	return txt.toUpperCase() ;				0.7071	0.4472	0.1767
	//return txt.toUpperCase(java.util.Locale.ENGLISH) ;						
15	}						
16	…				…	…	…
17	}						
18	}						
19	public class Caverphone implements StringEncoder {						
20	public String caverphone(String txt) {						
21	…			…		…	…
22	txt = txt.toLowerCase();			0.5774		0.2582	0.1443
23	txt = txt.replaceAll("[^a-z]", "");			0.5774		0.2582	0.1443
24	…			…		…	…
25	}						
26	}						

図 13　他の欠陥箇所の順位が向上したことで精度が低下した欠陥箇所の例

7　考察

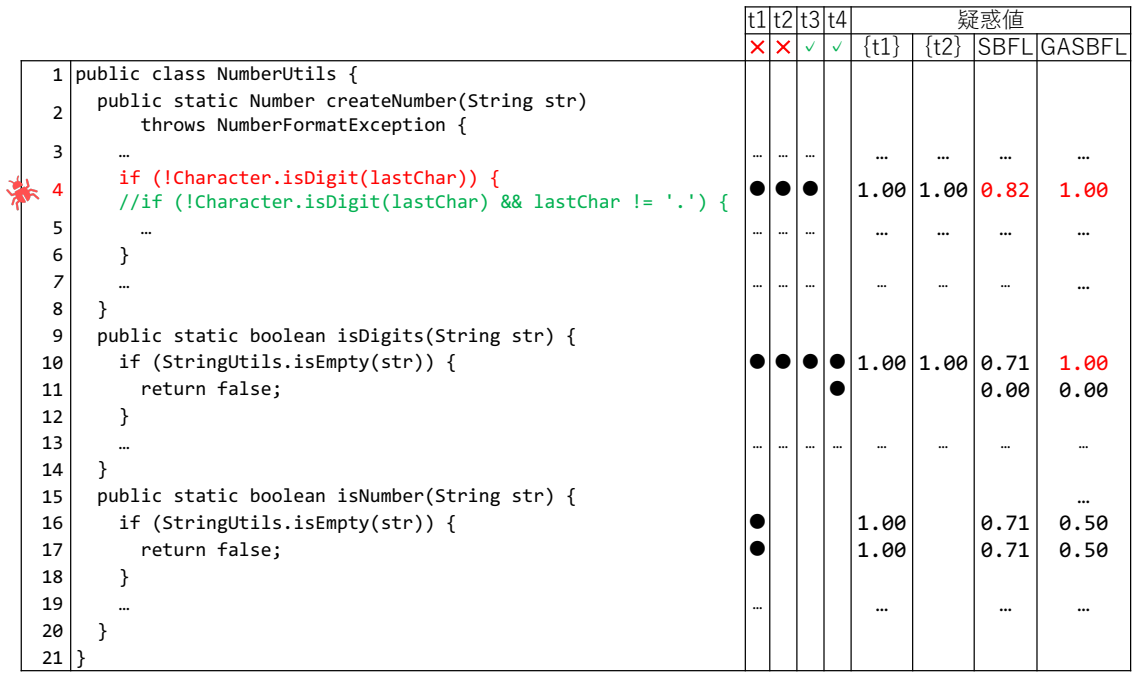
GASBFL を用いることで SBFL より欠陥限局の精度が低下した欠陥箇所について考察する．また GMSBFL を用いることで SBFL より欠陥限局の精度が低下した欠陥箇所について考察する．

7.1　GASBFL により精度が低下した欠陥箇所

実験 2 において GASBFL を用いることで SBFL を用いるよりも欠陥限局の精度が低下した欠陥箇所が存在した．本節ではその原因について考察する．

GASBFL を用いることで SBFL を用いるよりも精度が低下した欠陥箇所の例（Codec の欠陥 ID1）を図 13 に示す．図 13 に示すプログラムは 5 行目と 14 行目が欠陥箇所である．GASBFL を用いると，14 行目の欠陥箇所の順位は 2 位から 5 位に下がった．一方で，5 行目の欠陥箇所の順位は 84 位から 2 位に上がった．欠陥箇所である 5 行目の順位が向上したことで相対的に 14 行目の順位が下がった．図 13 のように欠陥限局の精度が少し低下した欠陥箇所が存在する場合でも，他の欠陥箇所の順位が大幅に向上する場合は問題がない．

しかし，他の欠陥箇所の順位は向上せずに精度が低下する場合がある．図 14 に示す欠陥箇所は成功



		t1	t2	t3	t4	疑惑値			
		✕	✕	✓	✓	{t1}	{t2}	SBFL	GASBFL
1	public class NumberUtils {								
2	public static Number createNumber(String str) throws NumberFormatException {								
3	…	…	…	…		…	…	…	…
4	if (!Character.isDigit(lastChar)) {	●	●	●		1.00	1.00	0.82	1.00
	//if (!Character.isDigit(lastChar) && lastChar != '.') {								
5	…	…	…	…		…	…	…	…
6	}								
7	…	…	…	…		…	…	…	…
8	}								
9	public static boolean isDigits(String str) {								
10	if (StringUtils.isEmpty(str)) {	●	●	●	●	1.00	1.00	0.71	1.00
11	return false;				●			0.00	0.00
12	}								
13	…	…	…	…	…	…	…	…	…
14	}								
15	public static boolean isNumber(String str) {								…
16	if (StringUtils.isEmpty(str)) {	●				1.00		0.71	0.50
17	return false;	●				1.00		0.71	0.50
18	}								
19	…	…				…		…	…
20	}								
21	}								

図 14　成功テストが選択されなかったことで精度が低下した欠陥箇所の例

テストが選択されなかったことで精度が低下した例（Lang の欠陥 ID36）である．GASBFL を用いると，4 行目の欠陥箇所の順位は 55 位から 66 位に下がった．成功テスト `t4` の実行経路はどの失敗テストの実行経路とも類似していないため，GASBFL において `t4` は選択されなかった．この成功テストは 4 行目と 10 行目を区別するのに役立つ成功テストである．このような成功テストが選択されず，欠陥箇所と同じ疑惑値をもつ行が増えため，欠陥限局の精度が低下した．

		t1	…	t4	…	疑惑値				
		✕	…	✕	…	{t1-t3}	{t4-t10}	SBFL	GASBFL	GMSBFL
1	public class FastDateParser implements DateParser, Serializable {									
2	private static StringBuilder escapeRegex(StringBuilder regex, String value, boolean unquote) {									
3	boolean wasWhite= false;	●	…	●	…	0.29	0.42	0.49	0.36	0.42
4	…	…	…	…	…	…	…	…	…	…
5	}									
6	}									
7	public class FastDatePrinter implements DatePrinter, Serializable {									
8	protected String parseToken(String pattern, int[] indexRef) {									
9	StringBuilder buf = new StringBuilder();			●	…		0.58	0.48	0.29	0.58
10	…			…	…		…	…	…	…
11	}									
12	}									

図 15　GMSBFL を用いることで精度が低下した欠陥箇所の例

7.2　GMSBFL により精度が低下した欠陥箇所

実験 2 において GMSBFL を用いることで SBFL を用いるよりも欠陥限局の精度が低下した欠陥箇所が存在した．本節ではその原因について考察する．

GMSBFL を用いることで SBFL を用いるよりも欠陥限局の精度が低下した欠陥箇所の例（Lang の欠陥 ID10）を図 15 に示す．GMSBFL を用いると，3 行目の欠陥箇所の順位は 177 位から 259 位に下がった．一方で GASBFL を用いると，3 行目の欠陥箇所の順位は 177 位から 138 位に上がった．これは GASBFL では SBFL と同様に実行する失敗テストが多い行ほど大きい疑惑値になりやすいが，GMSBFL では必ずしも実行する失敗テストが多い行ほど大きい疑惑値になるとは限らないためである．提案手法によって {t1-t3} と {t4-t10} の 2 つのグループができる．GASBFL では各グループの疑惑値の平均値を取るため，9 行目のように 1 つのグループでのみ疑惑値が算出される行と比べて 3 行目のように 2 つのグループで疑惑値が算出される行は疑惑値が大きくなりやすい．9 行目の疑惑値を算出するために 7 個の失敗テストを利用するのに対して，3 行目の疑惑値を算出するために 10（= 3 + 7）個の失敗テストを利用しており，GASBFL では SBFL と同様に実行する失敗テストが多い行ほど大きい疑惑値になっている．それに対して GMSBFL では各グループの疑惑値から最大値を選ぶため，3 個の失敗テストを用いて得られた疑惑値と 7 個の失敗テストを用いて得られた疑惑値のどちらかを選ぶことになる．3 行目と 9 行目は 7 個の失敗テストを用いて得られた疑惑値であり，同じ失敗テストを利用している．3 行目を実行する成功テストの数が 9 行目を実行する成功テストの数より少なければ，3 行目の疑惑値が 9 行目の疑惑値より大きくなる．GMSBFL は GASBFL と比べて欠陥箇所を実行する成功テストの数が欠陥限局の精度に与える影響が大きい．

8 妥当性の脅威

8.1 疑惑値の計算

文献 [10] において 7 つの疑惑値の計算手法を比較し，Ochiai が優れた手法であると結論付けているため，本研究では疑惑値の算出に Ochiai を利用した．他の計算式を用いた場合でも，GASBFL が SBFL より高い精度で欠陥限局できるのかを調査する必要がある．

8.2 実験対象

本研究では，Dfects4J に含まれる 6 つの Java プロジェクトの欠陥を実験対象とした．他のプロジェクトの欠陥を対象にして実験を行なった場合，実験結果が変化する可能性がある．しかし，Defects4J は欠陥限局の性能を評価するためのベンチマークとして多くの研究で利用されているため，本研究の実験結果は有用である．

8.3 閾値

実験 1 では，欠陥の数および欠陥箇所の行数が最も多い Math を実験対象としている．他のプロジェクトを対象にして実験を行った場合，適切な閾値の値が変化する可能性がある．実験 2 では，実験 1 の結果をもとに距離の閾値と類似度の閾値を設定した．閾値の値を変えて実験を行った場合，実験結果が変化する可能性がある．

9 関連研究

9.1 テストケースの選択

提案手法におけるテストケースのグループ化は有用なテストケースのみを選択して欠陥限局するという処理の繰り返しと考えられる．これまでに，欠陥限局の精度向上を目的としたテストケースの選択に関する研究が行われている [18–20].

9.2 複数の欠陥箇所に対する欠陥限局

複数の欠陥箇所に対する欠陥限局の手法としてクラスタリングを利用する手法がある [11, 21, 22]．同じクラスターに含まれる失敗テストが同じ欠陥箇所に関連するようにクラスタリングを行い，各クラスター内の失敗テストと成功テストを組み合わせて、各クラスターで 1 つの欠陥箇所を特定する．この手法では欠陥を含むプログラムごとに適切なクラスター数を設定する必要がある．またクラスター数の数だけ疑惑値のリストが生成されるため，疑惑値が大きい行から順に欠陥箇所かどうかを目視で確認するときの負担が大きくなる．提案手法はあらかじめ設定した閾値をもとにテストをグループ化する．また各グループの疑惑値を組み合わせた値を提案手法の疑惑値とするため，疑惑値のリストは 1 つしか生成されない.

10　おわりに

本研究では，欠陥限局の精度を向上させることを目的として，実行経路に基づいてテストケースをグループ化する手法を提案する．複数の欠陥箇所を同時に特定するのではなく，各欠陥箇所を個別に特定した後にその結果を組み合わせることで欠陥限局の精度向上を目指す．実行経路が類似している失敗テストは同じ欠陥箇所を実行している可能性が高く，その欠陥箇所の特定に役立つ．また，失敗テストと実行経路が類似している成功テストはテストの成否を分ける情報を持っているため，欠陥限局の精度向上に役立つ．実行経路が類似した失敗テストをグループ化し，グループごとに失敗テストと実行経路が類似した成功テストを用いて欠陥箇所を特定した後に，各グループで得られた結果を組み合わせる．データセット Defects4J に含まれる 78 個の欠陥を対象に実験を行い，提案手法 GASBFL は SBFL より高い精度で欠陥限局できることを確認した．21.8% の欠陥箇所において精度が向上し，最大で 1.3% 向上した．28.9% の欠陥箇所において精度が低下したが，欠陥限局の精度の低下は最大でも 0.12% に抑えられた．

今後の課題としては次のようなものが考えられる．

成功テストの選択方法の改善　提案手法では失敗テストと実行経路が類似した成功テストのみを用いて欠陥限局を行う．選択されなかった成功テストの中には欠陥限局の精度向上に役立つ成功テストが存在する．より高い精度で欠陥限局するために，実行経路の類似度から成功テストに重みを持たせるなど成功テストの選択方法を改善する必要がある．

疑惑値の計算式の違いによる精度の変化の調査　Ochiai 以外の計算式でも GASBFL は SBFL より高い精度で欠陥限局できるかを調査する．

GASBFL が SBFL より高い精度で欠陥限局できるプロジェクトの特徴の分析　GASBFL が SBFL より高い精度で欠陥限局できるプロジェクトに見られる傾向を，行数や複雑度などの観点から調査し，どのようなプロジェクトに対して GASBFL が有効なのかを調査する．

謝辞

本論文の執筆にあたり，多くの方々にお世話になりました。ここに記して感謝を申し上げます。

本研究を行うにあたって有益なご意見を賜りました楠本真二教授に心より感謝申し上げます．発表練習の場では多くの貴重な意見を頂きました．

本研究において，研究の方針や実現方法，提案手法の評価方法など全過程を通して適切なご指導およびご助言を頂きました肥後芳樹教授に深く感謝申し上げます．また，論文執筆や発表のご指導まで多大なご尽力を賜り，誠にありがとうございます．

本研究に対する有益なご助言および発表に対するご指導等を賜りました柗本真佑助教に深く感謝申し上げます。発表練習の場では，研究内容だけでなく発表資料についても様々なご指摘を頂き大変勉強になりました．

研究活動を円滑に進めるにあたり，様々な支援を頂きました事務補佐員の橋本美砂子氏に深く感謝申し上げます．特に海外出張の旅費の立て替えの際には大変お世話になりました．

研究生活において，様々な場面で支えて頂きました楠本研究室の皆様に心より感謝申し上げます．同期の皆様には研究に関する相談のみならず日常生活の様々な面で支え頂き，有意義な研究生活を送ることができました．

参考文献

[1] Tom Britton, Lisa Jeng, Graham Carver, and Paul Cheak. *Quantify the time and cost saved using reversible debuggers.* Technical report, Cambridge Judge Business School, 2012.

[2] Brent Hailpern and Padmanabhan Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, Vol. 41, No. 1, pp. 4–12, 2002.

[3] Wei Jin and Alessandro Orso. F3: Fault Localization for Field Failures. In *proceedings of the International Symposium on Software Testing and Analysis*, pp. 213–223, 2013.

[4] Bogdan Korel. PELAS-program error-locating assistant system. *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, pp. 1253–1260, 1988.

[5] Chao Liu, Xifeng Yan, Long Fei, Jiawei Han, and Samuel P. Midkiff. SOBER: Statistical Model-Based Bug Localization. In *proceedings of the European Software Engineering Conference Held Jointly with ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 286–295, 2005.

[6] Xiaoguang Mao, Yan Lei, Ziying Dai, Yuhua Qi, and Chengsong Wang. Slice-based statistical fault localization. *Journal of Systems and Software*, Vol. 89, pp. 51–62, 2014.

[7] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering*, Vol. 42, No. 8, pp. 707–740, 2016.

[8] Hao Zhong and Zhendong Su. An Empirical Study on Real Bug Fixes. In *proceedings of the IEEE/ACM International Conference on Software Engineering*, pp. 913–923, 2015.

[9] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. An Evaluation of Similarity Coefficients for Software Fault Localization. In *proceedings of the Pacific Rim International Symposium on Dependable Computing*, pp. 39–46, 2006.

[10] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan JC Van Gemund. A Practical Evaluation of Spectrum-Based Fault Localization. *Journal of Systems and Software*, Vol. 82, No. 11, pp. 1780–1792, 2009.

[11] Chao Liu and Jiawei Han. Failure Proximity: A Fault Localization-Based Approach. In *proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 46–56, 2006.

[12] Manos Renieres and Steven P Reiss. Fault localization with nearest neighbor queries. In *proceedings of the IEEE International Conference on Automated Software Engineering*, pp.

30–39, 2003.

[13] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, Vol. 58, No. 301, pp. 236–244, 1963.

[14] René Just, Darioush Jalali, and Michael D Ernst. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *proceedings of the International Symposium on Software Testing and Analysis*, pp. 437–440, 2014.

[15] Mengshi Zhang, Yaoxian Li, Xia Li, Lingchao Chen, Yuqun Zhang, Lingming Zhang, and Sarfraz Khurshid. An Empirical Study of Boosting Spectrum-Based Fault Localization via PageRank. *IEEE Transactions on Software Engineering*, Vol. 47, No. 6, pp. 1089–1113, 2021.

[16] Ming Wen, Junjie Chen, Yongqiang Tian, Rongxin Wu, Dan Hao, Shi Han, and Shing-Chi Cheung. Historical Spectrum Based Fault Localization. *IEEE Transactions on Software Engineering*, Vol. 47, No. 11, pp. 2348–2368, 2021.

[17] Qusay Idrees Sarhan, BélaVancsics, ÁrpádBeszédes. Method Calls Frequency-Based Tie-Breaking Strategy For Software Fault Localization. In *proceedings the of International Working Conference on Source Code Analysis and Manipulation*, pp. 103–113, 2021.

[18] Tung Dao, Max Wang, and Na Meng. Exploring the Triggering Modes of Spectrum-Based Fault Localization: An Industrial Case. In *proceedings of the IEEE Conference on Software Testing, Verification and Validation*, pp. 406–416, 2021.

[19] Farid Feyzi and Saeed Parsa. Kernel-based detection of coincidentally correct test cases to improve fault localisation effectiveness. *International Journal of Applied Pattern Recognition*, Vol. 5, No. 2, pp. 119–136, 2018.

[20] Long Zhang and Zhenyu Zhang. SeTCHi: Selecting Test Cases to Improve History-Guided Fault Localization. In *proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 200–207, 2018.

[21] Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. Automated support for classifying software failure reports. In *proceedings of the International Conference on Software Engineering*, pp. 465–475, 2003.

[22] A Zheng, MI Jordan, B Liblit, M Naik, and A Aiken. Statistical debugging: Simultaneous isolation of multiple bugs. In *proceedings of the International Conference on Machine Learning*, pp. 26–29, 2006.