修　士　学　位　論　文

題　目

Ablationとテストケース学習に基づく
実行情報の欠落を利用した故障箇所特定

指　導　教　員

岡野 浩三

報　告　者

池田 拓真

2024年2月17日

信州大学大学院総合理工学研究科

令和５年度 (2023 年度)

信州大学大学院総合理工学研究科

修士学位論文

Ablation とテストケース学習に基づく実行情報の欠落を利用した故障箇所特定

専攻名　　工学専攻

分野名　　電子情報システム工学分野

学籍番号 22W2008A

氏名 池田 拓真

論　文　内　容　の　要　旨

氏名	池田 拓真	専攻名	工学専攻	学籍番号	22W2008A
論文題目	Ablation とテストケース学習に基づく実行情報の欠落を利用した故障箇所特定				

自動故障箇所特定は, プログラムのデバッグ作業のコストを削減する手法の 1 つである. 既存手法の中で Spectrum based Fault Localization (SFL) は, スケーラビリティなどの観点から有望な結果を示している. SFL は, テスト実行で得られたコードカバレッジ情報から各スペクトルである, ソースコードの文または関数などの疑わしさスコアを算出する. 従来 SFL 手法は, コードカバレッジ情報から抽出した各文の実行回数などから疑わしさスコアを算出するため, 代入文などの Pass や Fail で実行される文に対して高い疑わしさスコアを与えることができない. したがって, そのような文が故障している場合, 従来 SFL 手法は開発者に対してデバッグに有用な情報を提供することが難しい. 本論文では, 従来 SFL 手法で扱うことが難しい故障を高い精度で特定することができる故障箇所特定手法を提案する. 本論文では, 新しい観点から故障箇所特定手法について議論する. 従来 SFL が基本的には実行されたスペクトルから抽出された情報に依存しているのに対し, 提案手法の新しいアイディアは, 欠落したスペクトルの影響を調査することで故障箇所を特定することである. 特定のスペクトルを削除したプログラムの実行結果を得るには, 不完全なプログラムの実行結果を予測する新しい方法が必要である. 本論文では, 不完全なプログラム実行の結果を予測するために, 機械学習のテストケース分類器を採用する. この分類器は, テストの実行結果を「合格」または「不合格」に分類し, 完全なプログラムまたは不完全なプログラムの実行の両方で動作することができる. 評価実験は, Defects4j と Software Infrastructure Repository (SIR) で利用可能な 6 つのプロジェクトを用いて実施した. 評価実験の結果, 動機例題や多重故障などの, 従来 SFL 手法で扱うことが難しいとされている故障において, 提案手法が従来 SFL 手法よりも高い精度で故障箇所を特定できることを確認した. また, 実験対象とした全ての故障を対象にした比較では, 従来 SFL 手法と有意な差は認められないことを確認した. 本論文では, 異なる故障特性を持つプログラムの実験結果を分析することで, 従来 SFL 手法と比較した際の, 提案手法の利点と欠点について議論した.

主な用語

Spectrum based Fault Localization, 深層学習, テストケース学習, Word2Vec, LSTM, Multi Layer Perceptron, Attention

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, 故障箇所の特定は高いコストを伴う作業である. テストとデバッグは, 開発コストの最大 75 %を占めると報告されている [1]. 自動的な故障箇所特定は, プログラムのデバッグコストを削減する有効な手法であり, 既存の手法の中で, Spectrum-based Fault Localization (SFL) はスケーラビリティ, 処理の軽量さ, および言語に依存しないという点で, 有望な結果を示している [2–4].

代表的な SFL 手法としては, Ochiai [5] と Tarantula [6] が挙げられる. これらの手法は, テストの実行時におけるコードカバレッジに基づいて各文の故障の疑わしさスコアを計算する. 高いスコアを持つ文は, 故障の可能性が高いとみなされ, 開発者は高いスコアを持つ文から始めてプログラムを調査し, 故障箇所を特定する.

Ochiai と Tarantula は, Fail または Pass のテストケースでの各文の実行頻度に基づいて故障の疑わしさスコアを計算する. Fail のテストで頻繁に実行されるが, Pass のテストでは頻繁に実行されない文は, 他の文よりも高い疑わしさスコアを持つ. 従って, もし故障箇所が Pass と Fail のテスト実行回数に差がない文であれば, これらの手法はその故障箇所に高い疑わしさスコアを与えることができない. 実際, 代入文などの Pass または Fail に関わらず実行される文には, これらの手法は高い疑わしさスコアを与えない.

近年, 故障特定のためにいくつかの深層学習ベースのアプローチが提案され, DNN（Deep Neural Networks）の学習能力が故障箇所特定において, 従来 SFL 手法（Ochiai, Tarantula）よりも優れた推論性能を示していることが報告されている [7]. 深層学習ベースの SFL 手法の一つとして, 仮想カバレッジを用いた手法が提案されている [7–11]. これらの手法は, テストカバレッジを入力とし, そのテスト実行カバレッジが Pass または Fail かを分類する DNN モデルを学習する. そして, 特定の文のみが実行されたことを表現する仮想カバレッジが学習済みの DNN モデルに入力され, 各コードブロックの故障の疑わしさを示すスコアを出力する. これらの手法は, コードカバレッジ情報を用いて故障箇所の推定を行うため, 従来 SFL 手法と同様に Pass または Fail に関わらず実行される文には高い疑わしさスコアを与えることは期待できない.

本論文では, 従来 SFL 手法 (Ochiai, Tarantula) やコードカバレッジ情報を利用した深層学習に基づく SFL 手法が, 特定のタイプの故障に対して高い疑わしさスコアを与えることができない場合における効果的なアプローチを提案する.

提案手法の第一段階は, テストケースの学習であり, DNN モデルを使ってテストの実行を Pass または Fail に分類する. 実行トレースの処理時間を改善するため, Tsimpourlas らが報告したテストケース学習手法 [12] を拡張した. トレーニングされた DNN モデルは, 全ての実行トレースを Pass または Fail に分類する. 実行トレースは, コードカバレッジ情報から生成された実行された文の集合として表され, DNN モデルは各テストケースにおける文の実行パターンを学習する. DNN モデルは実行された文の集合を入力とし, Pass と分類される信頼度を表す float 型の単一の値を出力する. 次

に, トレーニングされた DNN モデルに Ablated トレース（実行トレースから文を系統的に削除したもの）を入力する. 削除された文が実際に故障文である場合, 実行トレースには故障の実行に関する情報が含まれないため, DNN モデルの推論結果が変わると期待される. 削除された文が故障文でない場合, 故障文に関する情報は実行トレースに残り, 推論結果に大きな影響を与えない. DNN モデルは, 入力が Pass に分類される信頼度を出力するため, 故障文が実行トレースから削除されると, DNN の出力値は Ablation 前のトレースと比べて大幅に高くなる.

提案手法は Defects4j [13] と Software Infrastructure Repository [14] で利用可能な 6 つのプロジェクト（Math, Lang, Chart, Totinfo, Printtokens, Printtokens2）で評価された. その結果, Wilcoxon Signed-Rank Test [15] により, 提案手法は複数の故障があるプログラムに対して, 従来 SFL 手法よりも高い精度で故障を特定できることが確認された. さらに, 提案手法は, 故障文における Pass と Fail のテストケース間の実行頻度にほとんど差がない場合にも, 従来 SFL 手法よりも高い精度を達成した. 特に, 代入文が故障しているプログラムにおいて, 提案手法は故障を特定するために必要なコード調査量を最大 23 ポイント削減した. また, 全ての故障を対象とした総合的な比較において, 提案手法は従来の SFL 手法と有意な差はみられないことを確認した. 本論文の第 6 節において, 提案手法が従来手法よりも高い精度を達成した特定の故障タイプについて, 詳細な分析と議論を行う.

本論文の残りの部分は以下の通りである. 2 節では背景を説明する. 3 節では提案手法を説明し, 4 節では実験のセットアップについて述べる. 5 節では実験結果について説明し, 6 節では考察について議論する. 最後に 7 節で結論を述べる.

2 背景

本節では, 最初に統計的な SFL 手法と深層学習に基づく SFL 手法について説明し, 最後にそれらの手法の課題について動機例題を用いて説明する.

2.1 統計的 SFL 手法

Tarantula [6] と Ochiai [5] は代表的な統計的な SFL (Spectrum-based Fault Localization) 手法である. これらの手法は, テスト実行中に収集されたテストカバレッジを使用して, 各文の故障の疑わしさを示す疑わしさスコアを計算する. 高い疑わしさスコアを持つ文は, 故障の疑いが高いとみなされ, 開発者はこれらの高得点文を優先的に調査する. Ochiai と Tarantula における各文の疑わしさスコアを計算するための数式を以下に示す.

$$\frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}} \quad (Ochiai) \tag{1}$$

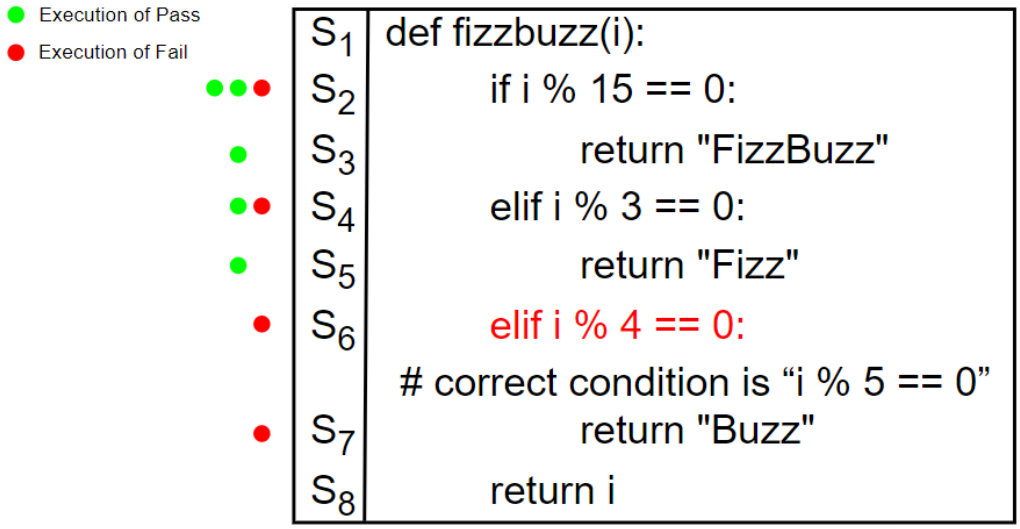
$$\frac{\frac{e_f}{e_f+n_f}}{\frac{e_f}{e_f+n_f} + \frac{e_p}{e_p+n_p}} \quad (Tarantula) \tag{2}$$

これらの数式は以下の値を用いて計算を行う [16].

- e_f は, その文が Fail テストケースで実行された回数を示す.
- e_p は, その文が, Pass テストケースで実行された回数を示す.
- n_f は, その文が Fail テストケースで実行されなかった回数を示す.
- n_p は, その文が Pass テストケースで実行されなかった回数を示す.

Ochiai のプログラムへの適用例を図 1 に示す. 図 1 では, 8 行のソースコードに Ochiai を適用している. 赤文字で示す S_6 がこのソースコードにおける故障箇所である. 緑の丸はその文が Pass のテストで実行された回数を示し, 赤の丸はその文が Fail のテストで実行された回数を示す. Ochiai や Tarantula による疑わしさスコアの算出における直感的な考え方は, Fail のテストで頻繁に実行され, Pass のテストでの実行回数が少ない文が疑わしいとするものである. したがって, Ochiai において, 緑の丸が少なく, 赤の丸が多いほどその文は疑わしい. 図 1 の下部に Ochiai の疑わしさスコアと, スコアによって得ることができる順位を示している. Fail のテストでしか実行されていない S_6, S_7 に最も高い疑わしさスコアが与えられ, 次に疑わしいのは Pass と Fail で 1 回ずつ実行された S_4 である. 図 1 の適用例では, Ochiai は故障箇所である S_6 に最も高い疑わしさスコアを与えており, これを疑わしさ Rank として開発者に提示することで, デバッグに有効な情報を開発者に提供することができる.

Tarantula や Ochiai に類似した疑わしさスコアを計算するためのいくつかの手法 [17–19] が提案されている. さまざまなメトリクスが提案されたため, これらの SFL 手法を統合し, 複数の統計的な SFL 手法から疑わしさスコアを計算する手法 [20] が提案された.



	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
Ochiai	0	0.577	0.0	0.707	0.0	1.0	1.0	0.0
Rank	5	4	5	3	5	1	1	5

図 1: Suspiciousness Score and Rank of Ochiai

これらの手法は, SFL のためにコードカバレッジから収集された各文の実行回数のみを使用する. 一方, 提案手法は各テストケースで実行された文の組み合わせを使用するため, SFL 手法で使用される情報とは異なる.

2.2 深層学習に基づく SFL 手法

既存の関数レベルの故障特定技術 [21,22] は, 疑わしさの値を計算するために関数カバレッジまたはコードカバレッジ情報を使用している. Murtaza ら [21] の手法では, 失敗に関連する関数呼び出しのパターンを識別するために決定木を利用する. 一方, Sohn ら [22] は, 遺伝的プログラミングと線形ランクサポートベクトルマシン (SVM) を使用して, 故障したメソッドを優先的にランク付けする手法を提案した. これらの手法は, 関数単位の SFL アプローチであり, 本論文で議論されている文単位の SFL アプローチとは粒度が異なる. また, 動的解析における機械学習の最新手法の 1 つとして, Li らは DeepRL4FL を提案した [23]. この手法では, 故障特定を画像パターン認識の問題として捉え, 不具合のあるコードを特定する. Li らの手法では, モデルがトレーニング時に故障のある文と非故障の文を区別できるようにするため, 故障のある文を訓練データとしてマークする必要がある. 提案手法では, 各テストケースが Pass または Fail であるかどうかのラベルをトレーニングラベルとして扱う. したがって, Li らの手法で使用されるトレーニングデータは, 提案手法のトレーニングデータよ

りも情報量が多い.

Wang らは, 失敗した実行から合格した実行を生成する手法を提案した [24]. Wang らの手法では, 失敗した実行において条件分岐の結果を切り替えることで合格した実行が生成される. これは, 実行情報の欠落による不完全な実行トレースがテスト結果に影響を与えるという直感的なアイディアに基づいている提案手法とは異なる.

提案手法に関連する研究として, 欠陥を特定するために仮想カバレッジを使用する手法 [7,8,10,11] が挙げられる. これらの手法では, まずテスト実行のカバレッジを入力として受け取り, テスト結果が合格 (Pass) か不合格 (Fail) かを分類するための DNN モデルを最初に訓練する. 次に, 特定の文またはコードブロックのみが実行されたことを示す仮想カバレッジを構築する. 仮想カバレッジを構築するためのいくつかの手法が提案されており, 文単位でのアプローチ [7] やコードブロック単位でのアプローチ [10,11] が挙げられる. これらの手法は, 欠陥を特定するためにテストケースを分類する DNN モデルの出力を使用するというアイデアにおいて, 提案手法と類似している. 提案手法では実行情報の欠落を利用する一方, これらの手法は特定の 1 行だけが実行されたことを仮想的に表現する仮想カバレッジを利用しており, 故障箇所を推定する方法が大きく異なる.

2.3 動機例題

統計的 SFL 手法 (2.1 節) と従来の深層学習に基づく SFL 手法 (2.2 節) におけるコードカバレッジ情報を用いる手法では, Pass と Fail の両方のテストケースにおいて頻繁に実行される文には高い疑わしさスコアを与えることが難しい. これは, コードカバレッジ情報を用いて疑わしさスコアを計算すると, Fail のテストケースで頻繁に実行され, Pass のテストケースでは稀にしか実行されない文に高いスコアが与えられる傾向にあるためである. このようなケースについて, 実際のプロジェクトに統計的 SFL 手法を適用した例題を用いて説明する. また, 本稿における文は, 実行情報上の一行を指しており, ソースコード上の一文とは異なる点に注意する.

図 2 に Defects4j [13] で利用可能な Chart Version 7 に対して, 統計的 SFL 手法 (Ochiai, Tarantula) を適用した結果を示す. Chart Version 7 は代入文が故障しており, 提案手法の動機例題である. 縦軸は疑わしさスコアを示し, 横軸はプログラムの各文を示す. 赤いプロットは故障箇所を意味しており, 赤いプロットの疑わしさスコアが他の文のスコアよりも大きいことが望ましい.

図 2 において, Ochiai, Tarantula のどちらの手法も故障箇所 (赤プロット) に高いスコアを与えられていない. したがって, 故障箇所には下位 (下から二番目) の疑わしさの順位が与えられ, 疑わしさの順位が上位のものから調査することを想定すると, 故障箇所の特定には多くの文を調査しなければならない.

また, 図 2 は統計的 SFL 手法のもう一つの課題点を示している. Ochiai と Tarantula のどちらも, 故障箇所と同じスコアを他の多くの文に与えている. これは, 故障箇所と同じ疑わしさの順位が与えられる文が多く存在することを意味している. 最悪のケースを考えると, 開発者は同じスコアを与え

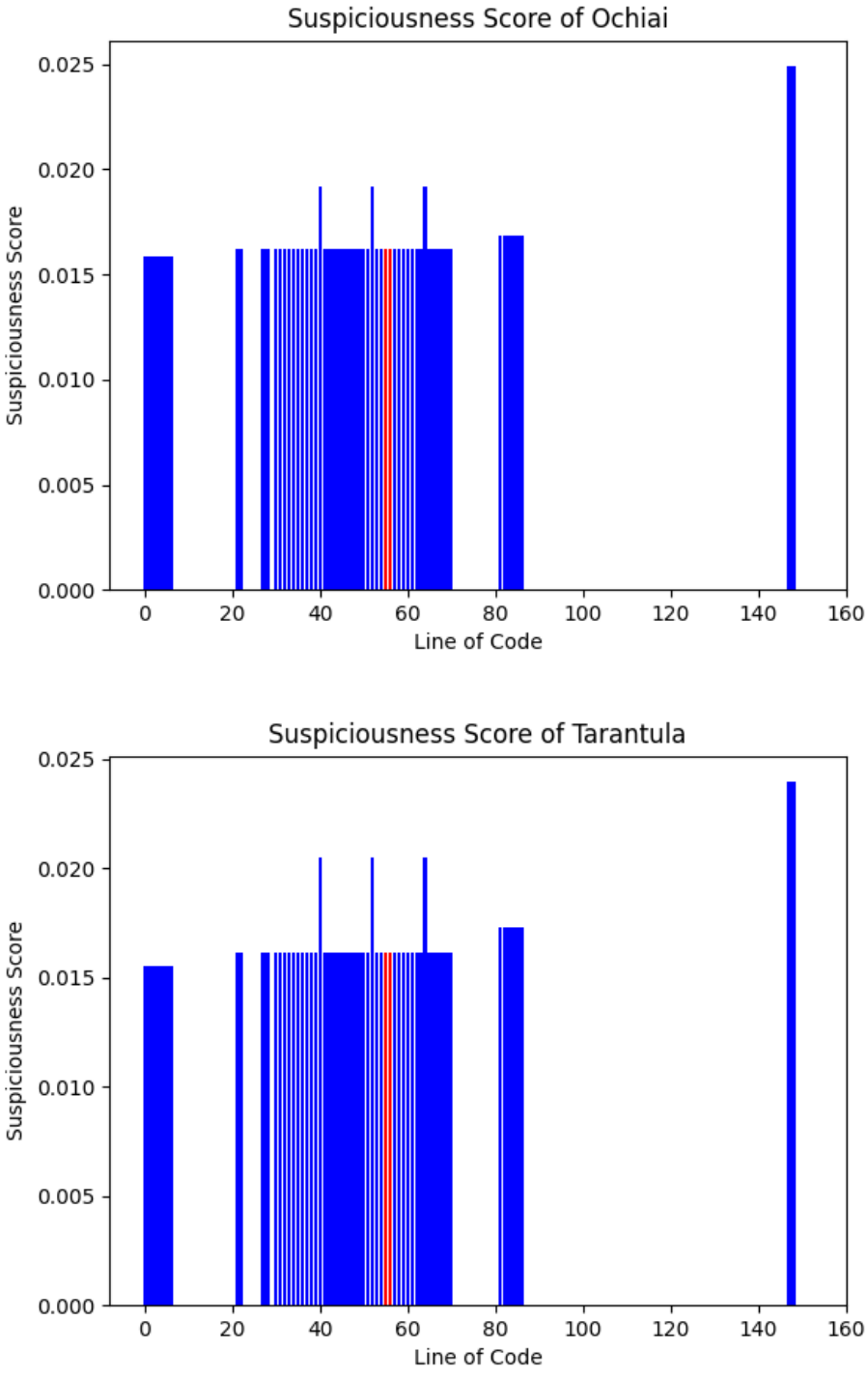


図 2: Suspiciousness Score of Ochiai and Tarantula in Each Statement

られた文を全て調査しなければ故障箇所に到達することができない. したがって, 一つ目の課題点と同様に, 故障箇所の特定に多くの文の調査が必要になると考えられる.

図 2 が示す統計的 SFL 手法の問題点の原因は, 疑わしさスコアを計算するために使用する実行情報である. 統計的 SFL 手法は疑わしさスコアの算出にコードカバレッジ情報から得られる各文の各テストケースにおける実行回数を利用している. 様々な疑わしさスコア算出方法が提案されているが, 基本的な考え方は, Fail で頻繁に実行され, Pass ではあまり実行されない文は疑わしいというものである. そのため, Pass と Fail で頻繁に実行される文 (代入文など) には高い疑わしさスコアが与えられない. また, 実行回数から疑わしさスコアを算出するため, 同一の実行パターンとなる文同士には同じ疑わしさスコアが与えられる. その結果, 同順位の文の数が多くなり, 故障箇所特定の精度に悪影響を与える要因となる. 図 2 が示す問題点の原因はコードカバレッジであり, コードカバレッジ情報を用いる統計的 SFL 手法や深層学習に基づく SFL 手法の限界である.

コードカバレッジ情報以外の情報 (履歴情報など) を用いた深層学習に基づく手法 [23] であれば図 2 が示す問題点を解決できることが期待されるが, これらの手法は教師データの構築に必要なリソースが多く, コードカバレッジ情報を用いる深層学習ベースの SFL 手法 [7, 8, 10, 11] と比較して, DNN モデルの訓練で発生するオーバーヘッドが大きい. また, 統計的 SFL 手法やコードカバレッジ情報を用いる深層学習ベースの SFL 手法と同じリソースを用いて図 2 が示す問題点を解決する手法は開発されていない.

本論文では, 図 2 が示す問題点に対応することが期待される, 深層学習に基づいた新たな SFL 手法を提案する. 提案手法は, 統計的 SFL 手法と同じリソースから生成される実行トレース, コードカバレッジ情報を用いる深層学習ベースの SFL 手法と訓練コストが同じ DNN モデル, 実行トレースにおける実行情報の欠落を利用した新たな故障箇所特定のアイディアを用いる.

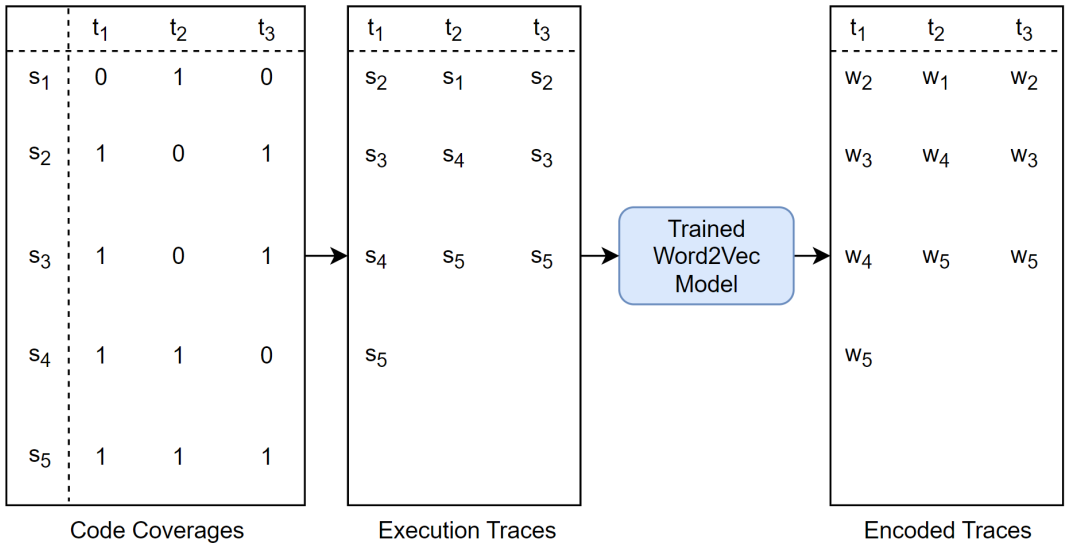


図 3: Create Execution Traces and Encoded Traces

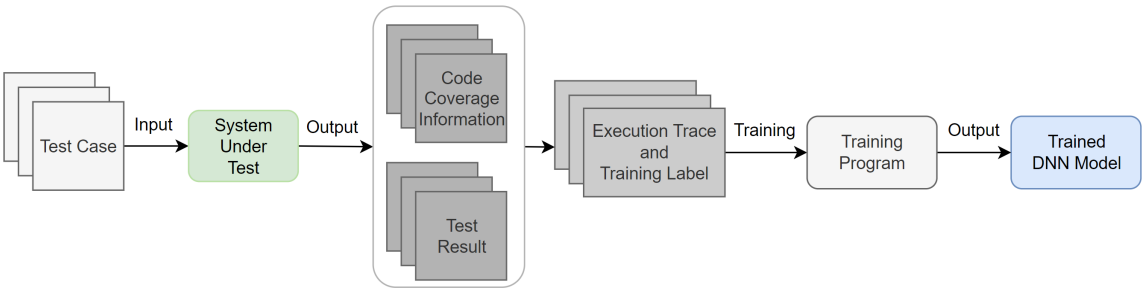


図 4: Overview of DNN Model Learning

3 提案手法

提案手法は以下の 4 のステップで構成される.

1. 実行トレースの収集
2. DNN モデルの訓練
3. 実行トレースの Ablation
4. DNN モデルによる故障箇所の推定

3.1 実行トレースの収集

提案手法で使用される実行トレースは, コードカバレッジ情報から生成される. 我々は, SUT のコードカバレッジを OpenClover [25], Gcov [26] を使用して収集した. 図 3 に, 文の集まりである実行トレースを生成する手順と, 実行トレースをエンコードする手順を示す. 図 3 において, t_1, t_2, t_3 はテストケースを示し, $s_1, s_2, ..., s_5$ はソースコードの文を示す. また, $w_1, w_2, ..., w_5$ は各文が Word2Vec で

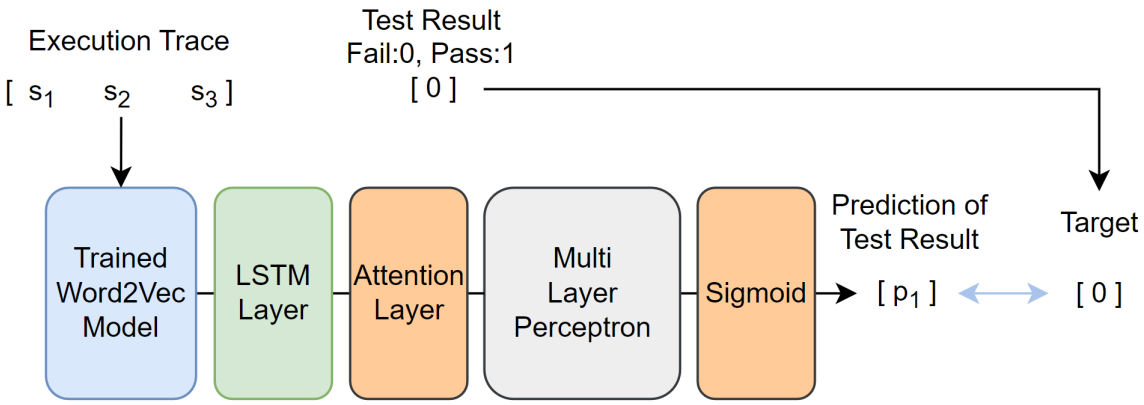


図 5: DNN Model Architecture

エンコードされた分散表現である.

図 3 では, 収集したコードカバレッジ情報から各テストケースで実行された文を特定し, それぞれの文をソースコード内の行番号に従って配置する. その後, Word2Vec [27] を使用して $s_1, s_2, ..., s_5$ の実行トレースをエンコードする. 各テストケース内の文（$s_1, s_2, ...s_5$）を自然言語の"単語"と見なし, その並びを"文"としてエンコードする.

3.2 DNN モデルの訓練

図 4 に, DNN モデルの訓練の概要を示す. DNN モデルは収集した実行トレースを入力とし, テスト結果を教師ラベルとして訓練を行う. テスト結果は, 開発者が要件文書から構築したテストオラクルを使用して収集したものと仮定している. 訓練が完了すると, DNN モデルは入力した実行トレースのテスト結果を分類することが可能である. 提案手法の DNN モデルのアーキテクチャとその入力を, 図 5 に示す. DNN モデルに与えられるテスト結果は, Fail の場合には 0, Pass の場合には 1 である. DNN モデルの出力値が高いほど, 分類結果が Pass であるという確信度が高くなる.

既存手法の DNN モデル [12] は, 文情報と実行トレースの両方をエンコードするために LSTM を使用しており, 大量の実行トレースを処理するためには膨大な計算時間を必要とする. 提案手法で使用する DNN モデルは, 分類性能を維持したまま実行トレースの処理時間を改善したモデルである. 改良点として, 実行トレースのエンコードに Word2Vec [27] と LSTM [28] を併用している. Word2Vec は, 図 5 の s_1, s_2, s_3 が示す, 実行された各文の情報をエンコードするために使用する [29]. Word2Vec の分散表現が LSTM に入力され, テストケースの実行トレースをエンコードする. また, 分類性能を向上させるために Attention [30] を使用した.

図 6 に, Word2Vec が出力した分散表現である W_1, W_2, W_3, W_4, W_5 を LSTM Layer と Attention Layer を用いて 1 つの分散表現 (Output of Attention Layer) を獲得するアーキテクチャを示す. 図 6 において, Word2Vec が出力した分散表現は順次 LSTM Layer に入力され, 各入力に対し LSTM

Layer の出力である O_1, O_2, O_3, O_4, O_5 を獲得する．次に, Attention Layer における最初の処理として, LSTM Layer の最終出力である O_5 と各出力値 (分散表現) との内積を計算する．この計算により, DNN モデルがテスト実行結果の予測分類をする上で, どの文の分散表現に注目しているかを数値として獲得することができる．従来の DNN モデル [12] では予測分類を行うために, LSTM Layer の最終出力 (O_5) を Multi Layer Perceptron の入力に用いており, 高い精度で分類を行うことができている．したがって, LSTM の最終出力である O_5 との類似度を示す内積が大きい値を示す文は, DNN モデルが予測分類を行う上で注目している文と考える．内積計算の結果である A_1, A_2, A_3, A_4 の Softmax を計算した AS_1, AS_2, AS_3, AS_4 は, 入力した各文から最終文を除いた文 W_1, W_2, W_3, W_4 に対する DNN モデルの注目度スコア (Attention Score) を示しており, DNN モデルが予測分類をする上で, どの文に注目したかを数値化している．LSTM Layer の最終出力を除いた各出力値 O_1, O_2, O_3, O_4 と, Attention Score とのスカラー積を計算し, 各計算結果 (AO_1, AO_2, AO_3, AO_4) を足し合わせたものが Attention Layer の出力 (Output of Attention Layer) となり, これは Multi Layer Perceptron への入力として用いられる．

3.3 実行トレースの Ablation

提案手法における Ablation を, 特定の文に関する情報を実行トレースから削除することと定義する．アブレーションの操作を図 7 に示しており, 文 s_4 が対象となり, この文に関する情報を削除している．提案手法において, Ablation した実行トレースは, 特定の文の実行情報が欠落した不完全なプログラム実行トレースである．

3.4 DNN モデルによる故障箇所の推定

提案手法のキーアイディアを図 7 に示す．実行結果が Fail である実行トレースが, Ablation の対象として選択される．Fail となるテストで実行された全ての文が, 選択したトレースに含まれていない場合, 他の Fail トレースを使用して Fail で実行された全ての文を Ablation の対象にする．

図 7 の場合, 元のトレースから s_4 の情報を削除することで, DNN モデルの分類結果が変化する．したがって, s_4 は元のトレースが Fail として分類されるための重要な情報であると考えられ, s_4 は故障の疑わしさが高い文であると判断される．

実際に評価実験で使用した DNN モデルは, 入力された実行トレースのテスト結果を 0 から 1 の Float 値で出力する．DNN モデルの訓練学習時に, 0 を Fail, 1 を Pass の教師ラベルとして与えているため, DNN モデルの出力が 1 に近いほど (大きいほど) 予測結果が Pass である確信度が高い．提案手法における故障箇所推定の考え方は, 元々Fail であった実行トレースから特定の文の実行に関する情報を Ablation することでテスト結果が Pass に変わるというものである．したがって, Ablation した実行トレースを入力した DNN モデルの出力値を疑わしさスコアとして扱う．DNN モデルの出力値が大きいほど予測結果が Pass である確信度が高いため, Ablation した文の疑わしさの度合いは

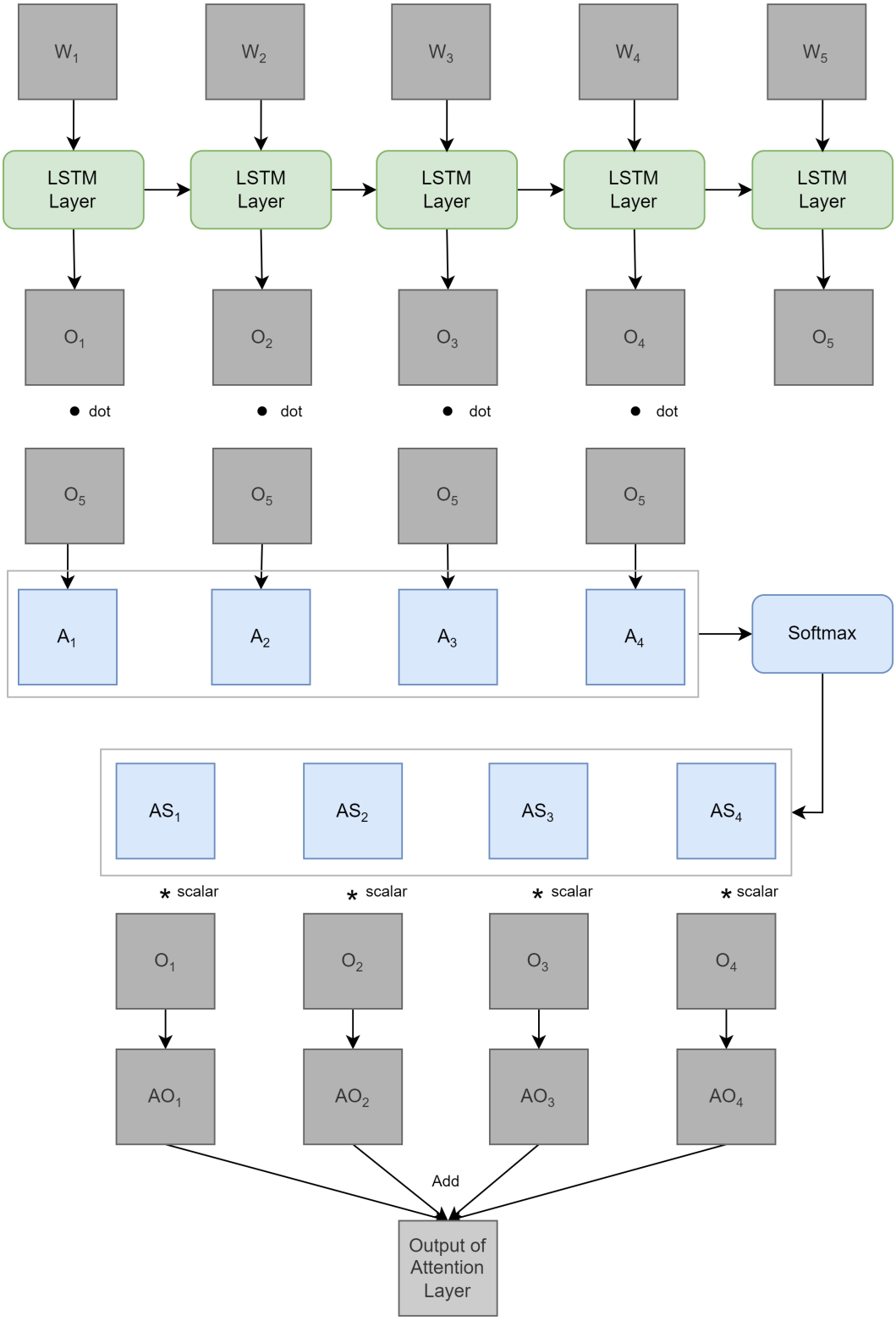


図 6: Attention Architecture

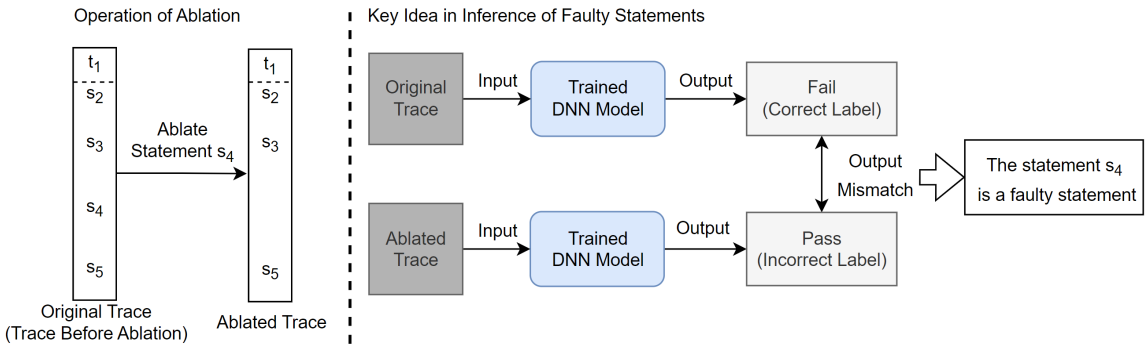


図 7: Operation of Ablation and Key Idea in Our Approach

大きくなる.

本研究での評価実験における実装を GitHub 上で公開している[1].

3.5 提案手法におけるアイディアの整理 (Rationale)

ここで, 提案する故障箇所特定手法の根拠を要約する. 提案手法の第 1 ステップは, 実行トレースを入力とし, テスト結果を分類する DNN モデルを訓練することである. DNN モデルが Pass または Fail を分類する際に, 実行パターンや実行された文の順序を学習することが期待される. 第 2 ステップとして, 特定の文を選択し, その文に関する情報を削除し, DNN モデルが推論する分類結果の確信度を観察する. もし削除された文が故障文であれば, DNN モデルの出力は削除前の結果よりも Pass に近づくことが期待される. 提案手法では, Fail で実行された全ての文を Ablation の対象として選択する. Ablation したトレースは 1 つずつ DNN モデルに入力され, DNN モデルの出力が最も合格に近づく (DNN モデルの出力が大きくなる) 文を最も故障が疑わしいとする. したがって, Ablation したトレースの出力値の降順を, 故障の疑しさを示す順位として利用する.

[1] https://github.com/jdpask21/ablationenv

4　評価実験のセットアップ

4.1　リサーチクエスチョン

評価実験では, 以下のリサーチ・クエスチョンを調査する.

RQ 1.　提案手法と従来手法 [5–7, 11] との故障箇所特定精度の比較.

RQ 2.　提案手法が検出可能な故障タイプの検討.

本論文では, まず RQ1 として, 提案手法と従来手法の実験対象の全ての故障を対象にした故障箇所特定性能を比較する. 全体の結果を踏まえた上で, RQ2 として, 動機例題で説明したような故障に対する提案手法の有効性について議論する. 不具合特定精度の指標として TopN % を用いる. TopN % は, バグが全体の上位 N %に分類されたことを表し, N の値が小さいほど故障の特定性能が高いことを示す. 複数のバグがある場合, 最大の TopN % がこのような指標として使用される. 本手法は, 各テストケースで実行された文の集合に対して学習させた DNN モデルに基づいているため, 実行頻度情報のみでは識別が困難な欠陥に対して有効であることが期待される. 考察では, 実験結果を故障の種類の観点から議論する.

4.2　比較手法

評価実験で用いた比較手法について, 手法名と選定理由は以下の通りである.

1. Tarantula [6].　Tarantula は SFL のための最初の技術の一つであり [2], 従来研究でベンチマークのための比較手法として利用されてきた [31, 32]. 本論文においても, Tarantula と統計的 SFL の代表的な手法として比較手法に選定した.

2. Ochiai [5].　Ochiai は統計的 SFL 手法として代表的な手法の一つであり, いくつかの研究で疑わしさスコアを計算する手法として利用されている [33, 34]. また, Abreu らによると, Ochiai は Tarantula よりも故障箇所特定に有効である [35]. コードカバレッジ情報を用いる手法として代表的なものであるため, 比較手法として選定した.

3. OneHot [7].　Zhang らはコードカバレッジ情報を用いた深層学習に基づく SFL 手法として仮想カバレッジを利用する手法を提案した [7]. この手法の教師データ構築に必要なリソースは提案手法と同じであり, 比較手法として適切である. いくつかの DNN モデルアーキテクチャが提案されているが, 畳み込みニューラルネットワーク (CNN) を用いた DNN モデルが最も高い精度を達成しているため [7], CNN のアーキテクチャを比較手法の DNN モデルとして採用した.

4. BlockHot [11].　桐生らは Zhang らの SFL 手法 [7] で使用できる新たな仮想カバレッジを提案した [10]. また, 池田らは桐生らの手法を改良し, 一部のプロジェクトにおいて故障箇所特定性能が改善することを示した [11]. この手法は一部のプロジェクトにおいて, 従来の仮想カバレッジを用いる手法よりも高い故障箇所特定性能を達成するため, 比較手法として選定した.

表 1: Details of Target Programs

Project	Number of Versions	LOC	Number of Tests
Lang	30	58389	54987
Math	29	23623	83364
Chart	15	10094	27036
Print_tokens	7	336	4130
Print_tokens2	9	343	2064
Tot_info	23	268	1052

4.3 実験対象プログラム

Defects4J [13] および Software Infrastructure Repository (SIR) [14] から提供されたバグとその修正内容を使用して評価実験を行った. Defects4J は Java プロジェクトにおける実際の故障を公開しているデータベースであり, SIR は様々なプロジェクトにおける実際の故障, もしくは挿入された故障を公開している. 実験の対象として, Defects4J からは Lang, Math, Chart, SIR からは Printtokens, Printtokens2, Totinfo を対象プロジェクトとして選択した. 評価実験において, 以下の条件を満たす故障を実験で使用した.

- 故障の修正内容がコードの追加のみでない. もしバグがコードの追加のみで修正された場合, 元のバグのあったソースコードには指摘すべき欠陥が存在しないためである.
- 故障文はが, 少なくとも Fail テストと Pass テストのそれぞれで少なくとも一度は実行される. SFL 手法では, 故障が Pass と Fail のテストで少なくとも一度は実行されることが必要なためである.

OpenClover [25] と gcov [26] を使用して, 実行カバレッジと実行回数のレポートを収集した. Defects4j のプロジェクトのカバレッジを収集するために OpenClover を使用し, SIR のプロジェクトのカバレッジを収集するために gcov を使用した. OpenClover の仕様により, クラスメンバー変数の定義などの実行が記録されないため, 実行カバレッジとして記録されていないプログラムの一部を故障セットから除外した. 実験プログラムの行数 (LOC) とテスト数を表 1 に示す.

4.4 DNN モデルのセットアップ

DNN モデルの訓練学習を行う際, LSTM と Multi-Layer Perceptrons (MLP) の重みパラメータはランダムな値で初期化した. Word2Vec でエンコードされた単語のベクトル表現のサイズは 128 とした. 分散表現のサイズをさらに大きくしても, 分類の性能にはほとんど影響がないことを実験的に確認した. LSTM ネットワークの出力ベクトルのサイズは 256 とした. MLP は, サイズが $256, 128, 64$ の 3 つの隠れ層を持つ. 各層のパラメータは, 計算時間と分類性能の観点から既存手法 [12] に基づい

て選択した．オプティマイザーとして Adam [36] を使用し, 学習率は実験対象プログラムのコードの行数に応じて調整した．学習率は $1e-06$ から $5e-05$ の間で選択した．$1e-06$ 未満の学習率は収束に必要な計算時間を増やし, $5e-05$ を超える値は学習結果に悪影響を与える．機械学習フレームワークとして PyTorch (ver. 1.10.1) を使用した．評価実験では, 1 つの対象プログラムに対して 10 の DNN モデルの訓練を行う．各 DNN モデルで提案手法による SFL を実行し, その平均値を最終的な実験結果とした．各プログラムに対して提案手法を 10 回適用しているのは, DNN モデルの訓練学習と推論における乱数的な要素を考慮するためである．

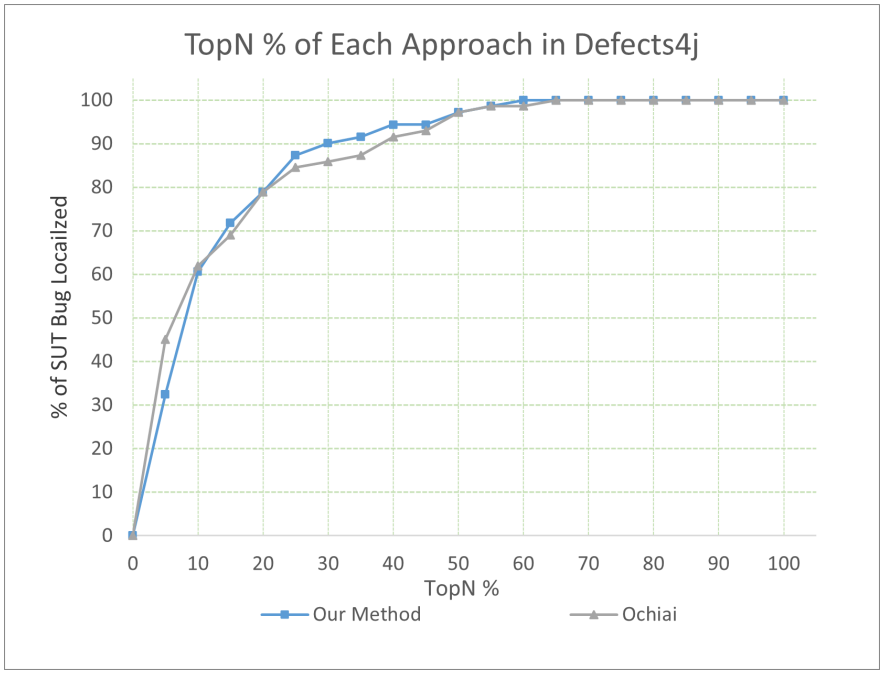


図 8: SFL Performance of The Proposed Method and Ochiai in Defects4j

5　実験結果

5.1　RQ1 における実験結果

図 8, 9, 10, 11 に Defects4J の RQ1 の実験結果を示す. 図 8, 9, 10, 11 の横軸は TopN %を表し, 開発者が調査したソースコードの量を意味する. 縦軸は特定された不具合の割合を示し, 縦軸の 100 %はすべての不具合が特定されたことを意味する. 例えば, Top 50 %の縦軸プロットは 90 %を超えており, ソースコードの半分を調査することで 90 %以上の不具合が特定されることを示している. 青プロットは提案手法を示しており, 灰プロット (図 8) が Ochiai, オレンジプロット (図 9) は Tarantula, 黄色プロット (図 10) は OneHot, 緑プロット (図 11) が BlockHot を示している.

結果として, Ochiai, Tarantula との比較結果に関しては, 提案手法との大きな差はみられない. OneHot, BlockHot との比較結果に関しては, 同じコード調査量 (Top N %) における特定した故障の割合が, 提案手法のほうが僅かに高いことがわかる.

表 2 に Defects4j における実験結果に Wilcoxon Signed-Rank Test を適用した結果を示す. 表 2 における Conclusion の列が検定の結果を示しており, SIMILAR は提案手法と比較手法との間に有意な差が認められないことを意味し, BETTER は提案手法が比較手法よりも故障箇所特定性能が有意に高いことを意味する.

検定の結果として, BlockHot 以外の比較手法は, 提案手法との間に有意な差が認められない (SIMILAR) ため, これらの手法と提案手法の故障箇所特定性能に有意な差はない.

図 12, 13, 14, 15 に SIR における RQ1 の実験結果を示す. また, 表 3 に SIR における実験結果に

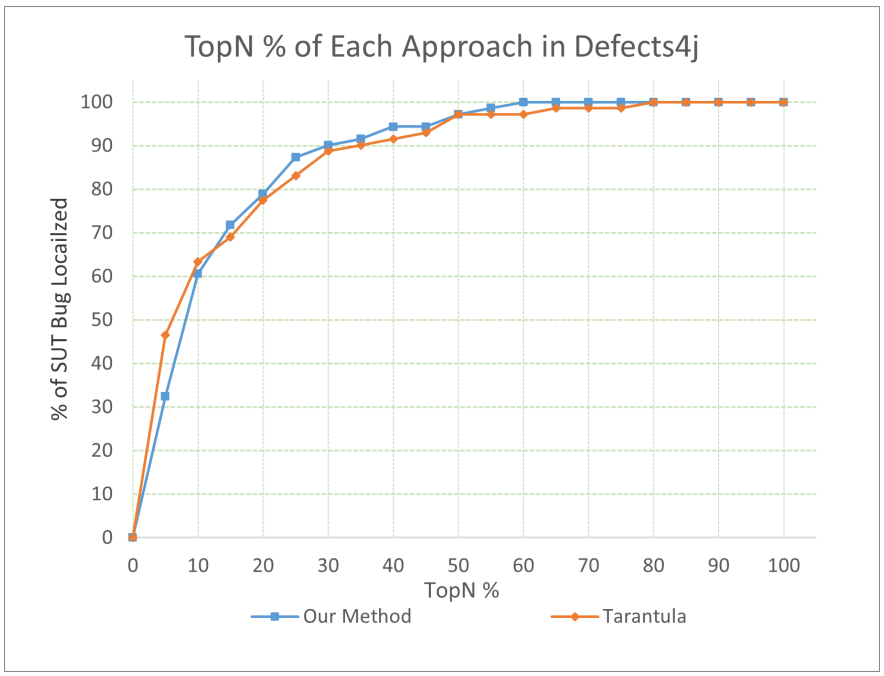


図 9: SFL Performance of The Proposed Method and Tarantula in Defects4j

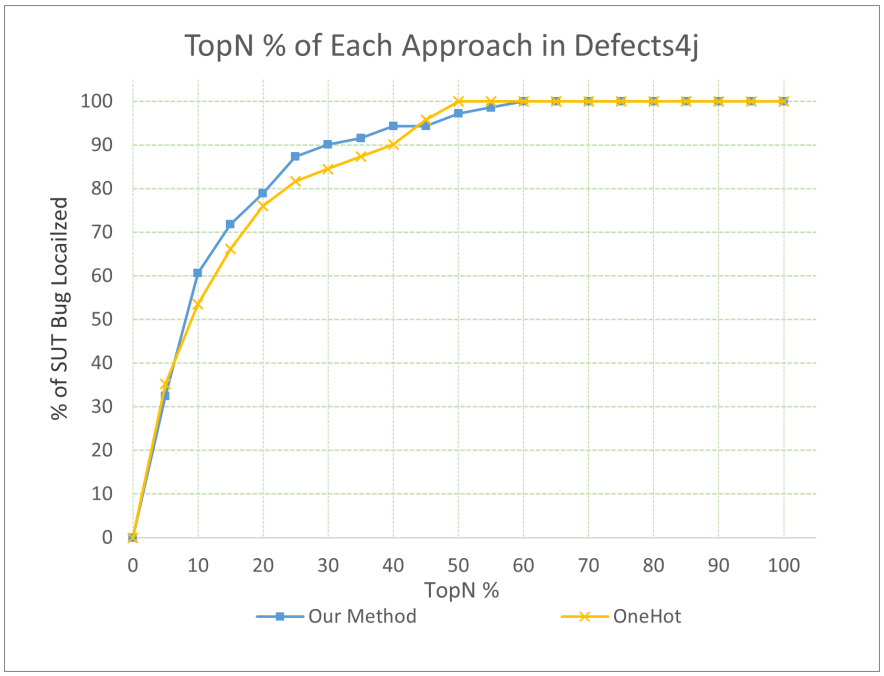


図 10: SFL Performance of The Proposed Method and OneHot in Defects4j

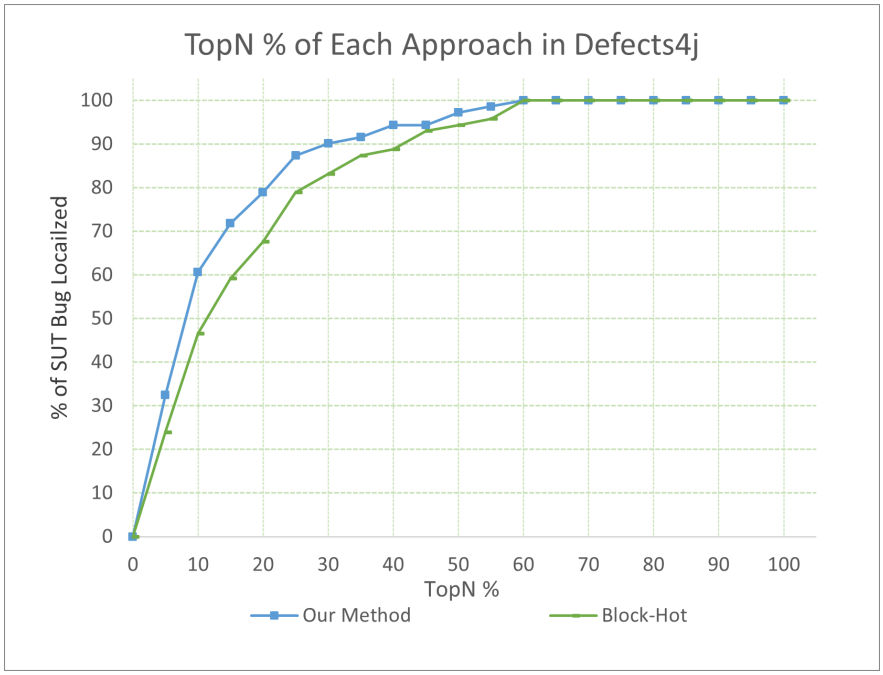


図 11: SFL Performance of The Proposed Method and BlockHot in Defects4j

表 2: Test Results in Defects4j (All Version)

Subject	1-tailed (left)	Conclusion
Our Approach vs Ochiai	6.258E-01	SIMILAR
Our Approach vs Tarantula	6.149E-01	SIMILAR
Our Approach vs One-Hot	3.275E-01	SIMILAR
Our Approach vs Block-Hot	1.738E-03	BETTER

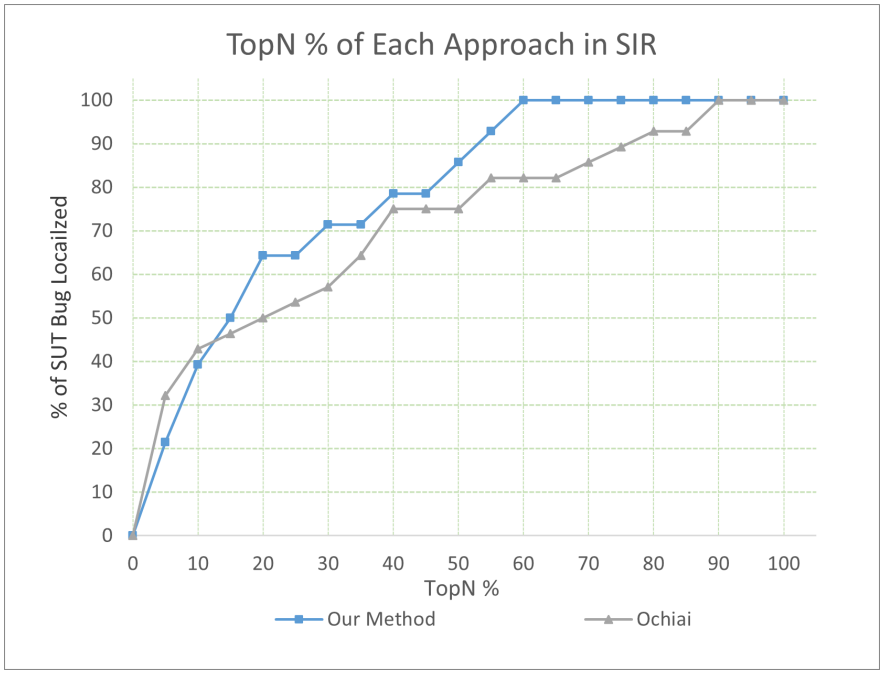


図 12: SFL Performance of The Proposed Method and Ochiai in SIR

表 3: Test Results in SIR

Subject	1-tailed (left)	Conclusion
Our Approach vs Ochiai	4.099E-01	SIMILAR
Our Approach vs Tarantula	1.251E-01	SIMILAR
Our Approach vs One-Hot	3.858E-03	BETTER
Our Approach vs Bock-Hot	2.937E-02	BETTER

Wilcoxon Signed-Rank Test を適用した結果を示す. 図 12, 13, 14, 15 において, 提案手法と各比較手法における, 同じコード調査量での特定できる故障の割合には差がみられる. Ochiai と Tarantula に関しては, Top 5%と Top 10%においては比較手法の特定した故障の数が多いが, それ以降の Top N%に関しては, 提案手法が多くの故障を特定できていることがわかる. OneHot に関しては, 各 Top N%において提案手法が特定した故障の割合が高いことが認められる. BlockHot に関しても, Top 45%, Top 50%の 2 プロットを除いて, 提案手法が特定した故障の割合が高いことが認められる. しかし, 検定の結果として, Ochiai, Tarantula と提案手法との間には有意な差は認められない. OneHot と BlockHot に関しては, 提案手法の故障箇所特定性能が有意に高いことが認められる.

5.2 RQ2 における実験結果

次に, RQ2 の考察で議論する実験結果として, Defects4j の多重故障に限定した実験結果を説明する. 図 16, 17, 18, 19 に Defects4j の多重故障に限定した Top N%における各手法の故障箇所特定性

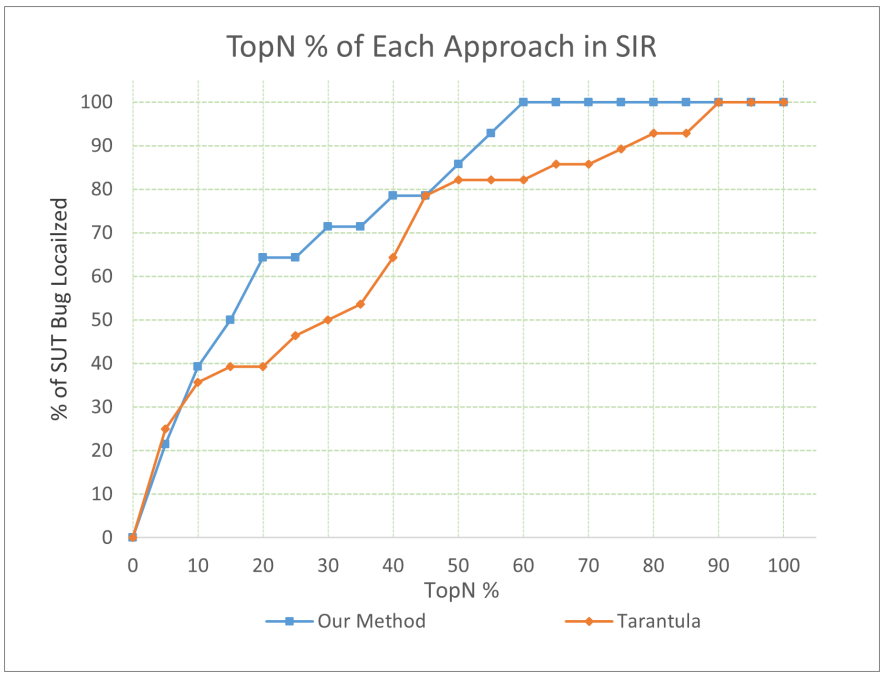


図 13: SFL Performance of The Proposed Method and Tarantula in SIR

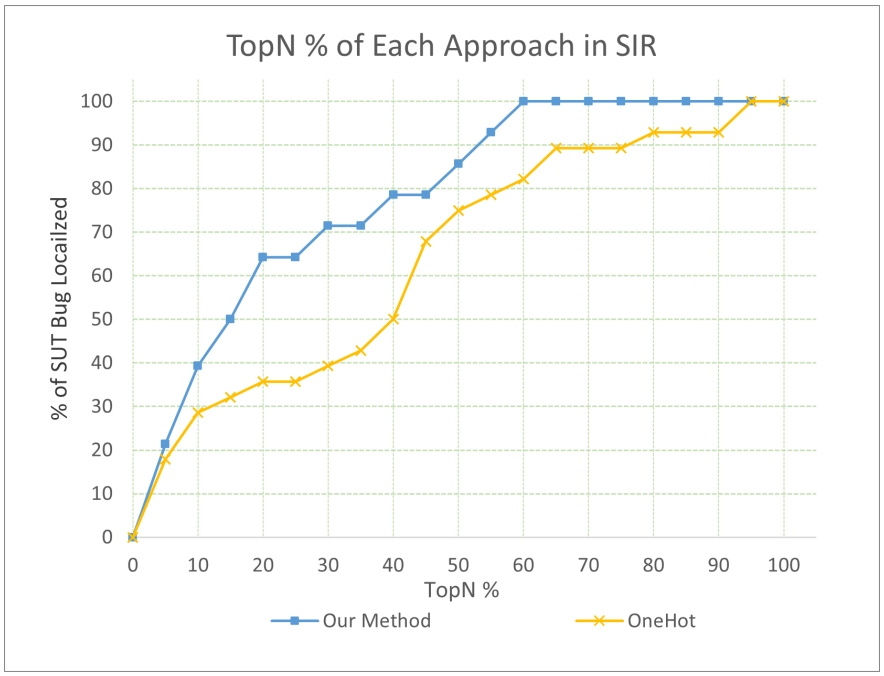


図 14: SFL Performance of The Proposed Method and OneHot in SIR

図 15: SFL Performance of The Proposed Method and BlockHot in SIR

能を示す. また, 表 4 に, Defects4j の多重故障における実験結果に Wilcoxon Signed-Rank Test を適用した結果を示す. SIR のプログラムにおける全ての故障が単一故障であったため, 多重故障の評価対象として SIR のプログラムは含まれていない.

図 16, 17, 18, 19 において, 提案手法はどの比較手法よりも多くの故障を特定することができる傾向にあることが認められる. Tarantula に関して, Top 5%, Top 10%において特定した故障の割合は提案手法と同じであるが, Top 15%以降の故障箇所特定性能は提案手法のほうが高い. OneHot に関して, Top 55%においてのみ提案手法よりも高い性能を達成しているが, それ以外の Top N%においては提案手法の故障箇所特定性能が高い. Top N%は開発者が故障を特定するのに必要なソースコードの調査量を示しているため, より小さい N の値において特定した故障の数 (割合) が重要である. この観点から, Defects4j における多重故障プログラムにおいて, 提案手法はどの比較手法よりも高い故障箇所特定性能を達成する傾向にあることがわかる. また, 表 4 は Defects4j の多重故障プログラムにおいて, 提案手法がどの比較手法に対しても有意差がある (BETTER) ことを示している. したがって, Wilcoxon-Signed Rank Test による評価の結果, Defects4j の多重故障プログラムにおいて, 提案手法は従来 SFL 手法よりも故障箇所特定性能が高いことが認められた.

次に, 考察で議論するための実験結果として, 動機例題 (2.3 節) で説明した Chart Version 7 における提案手法の適用結果を図 20 に示す. 図 20 の横軸はソースコードの行数を示しており, 縦軸は疑わしさスコアの値を示す. 赤いプロットは故障箇所の疑わしさスコアを示しており, 赤いプロットが示す疑わしさスコアが他の青いプロットよりも大きいことが望ましい. 提案手法の適用結果である図

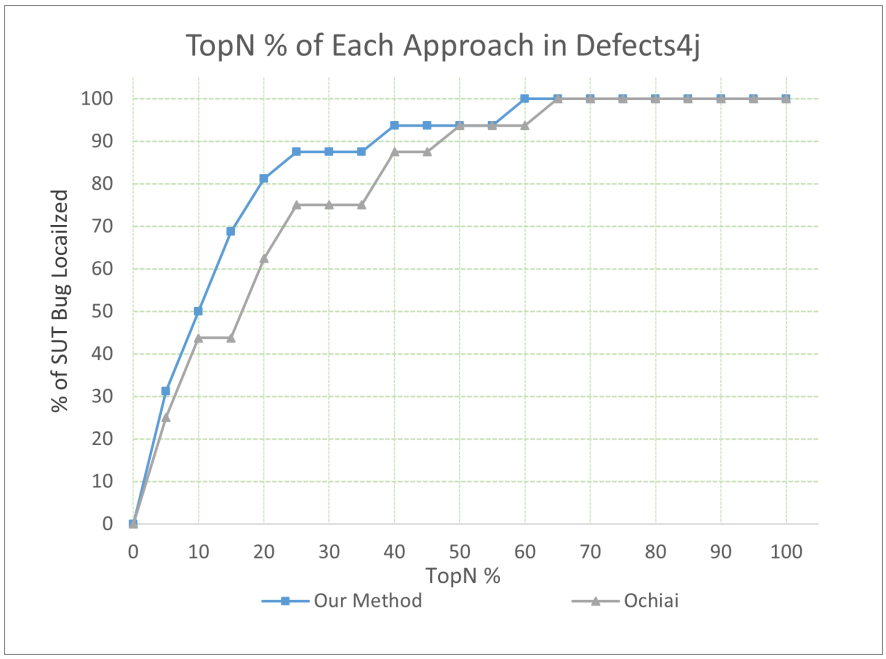


図 16: SFL Performance of The Proposed Method and Ochiai in Multi Fault Programs

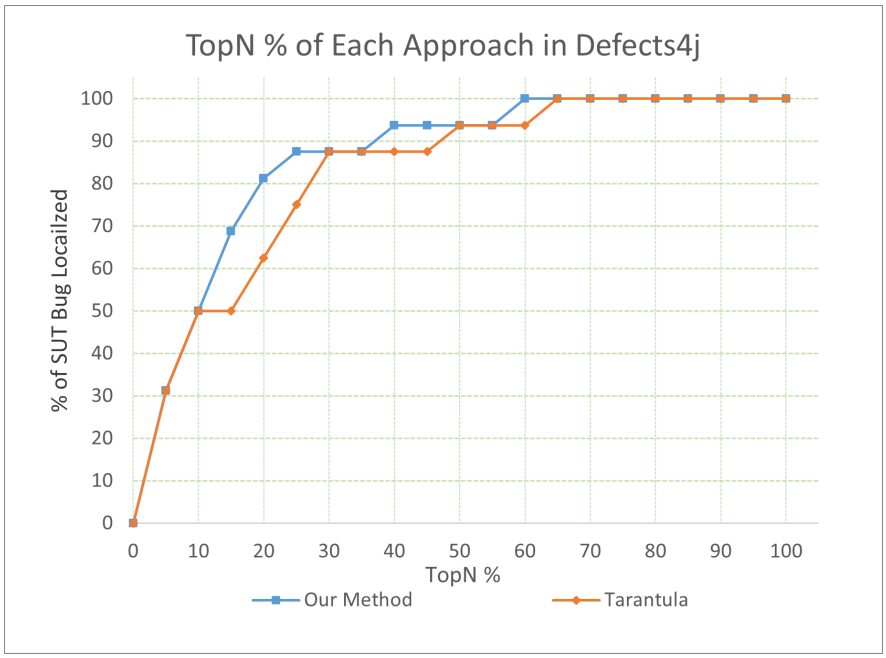


図 17: SFL Performance of The Proposed Method and Tarantula in Multi Fault Programs

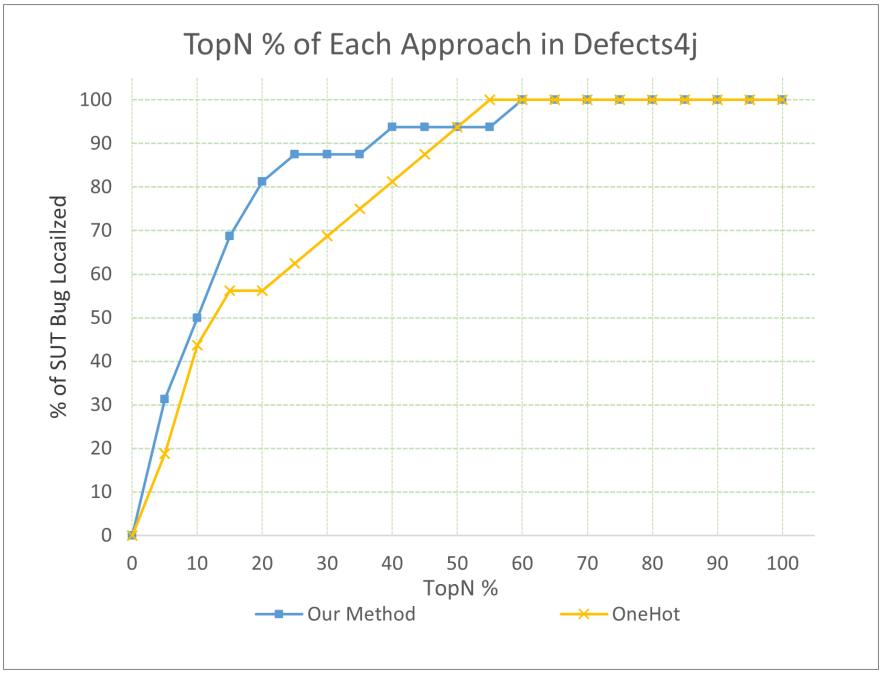


図 18: SFL Performance of The Proposed Method and OneHot in Multi Fault Programs

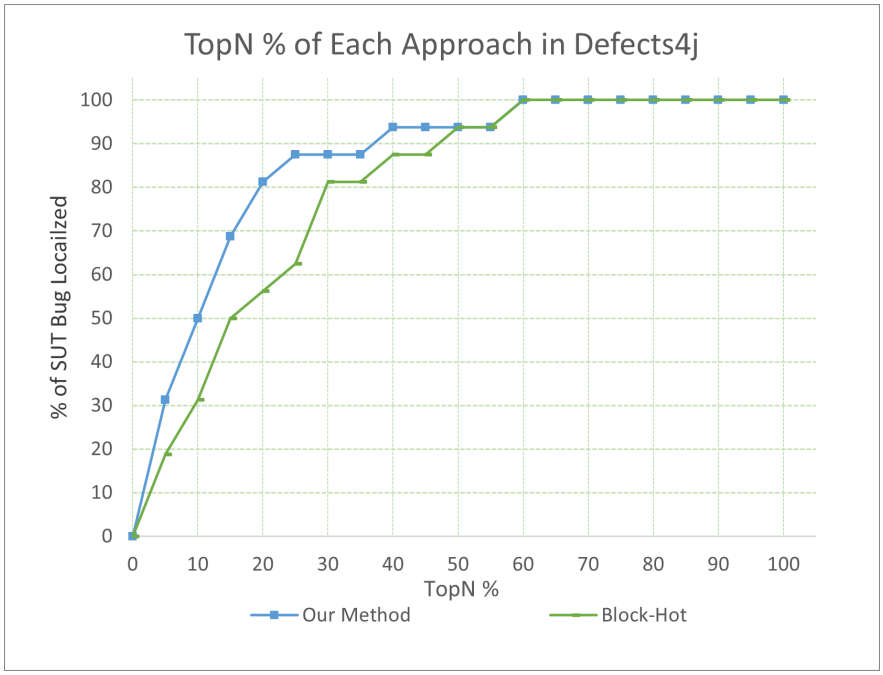


図 19: SFL Performance of The Proposed Method and BlockHot in Multi Fault Programs

表 4: Test Results in Defects4j (Multi Fault Programs)

Subject	1-tailed (left)	Conclusion
Our Approach vs Ochiai	1.494E-02	BETTER
Our Approach vs Tarantula	3.937E-02	BETTER
Our Approach vs One-Hot	2.187E-02	BETTER
Our Approach vs Bock-Hot	9.985E-03	BETTER

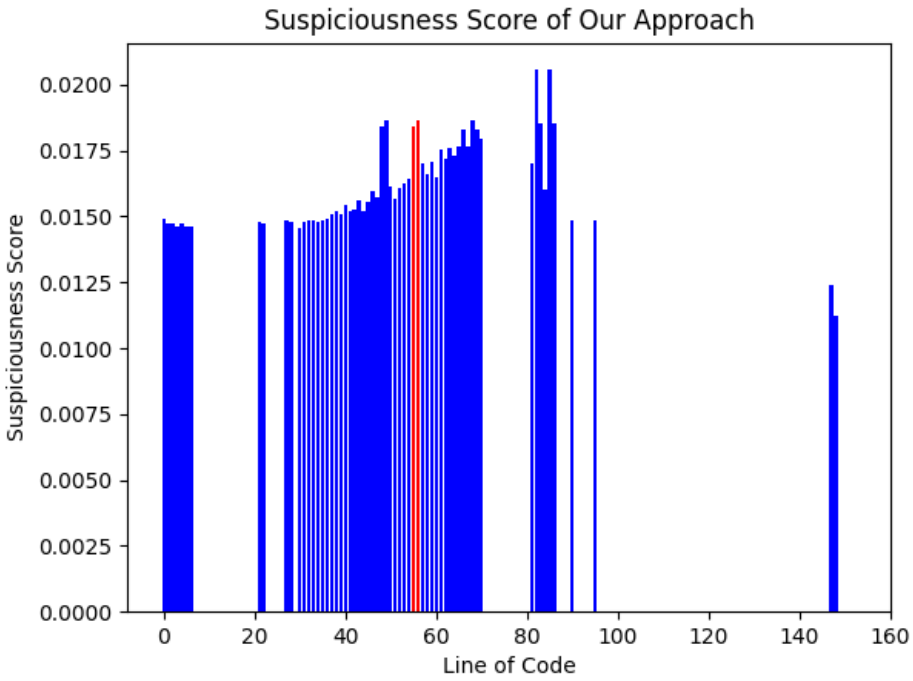


図 20: Suspiciousness Score of Our Approach in Each Statement

20 を, 図 2 が示す Ochiai, Tarantula の疑わしさスコアを比較すると, 提案手法は故障箇所に他の文よりも高い疑わしさスコアを与えていることがわかる. また, Ochiai, Tarantula が故障箇所に, 他の多くの文と同じ疑わしさスコアを与えているが, 提案手法はこの観点においても良好な結果を示している. 実際に提案手法と従来 SFL 手法の Top N%における精度を比較すると, Chart Version 7 において提案手法は Ochiai, Tarantula, OneHot と比較して, 故障箇所を特定するための必要なコード調査量を 20 ポイント以上削減した. BlockHot と比較すると, 提案手法は必要なコード調査量を 12 ポイント以上削減した.

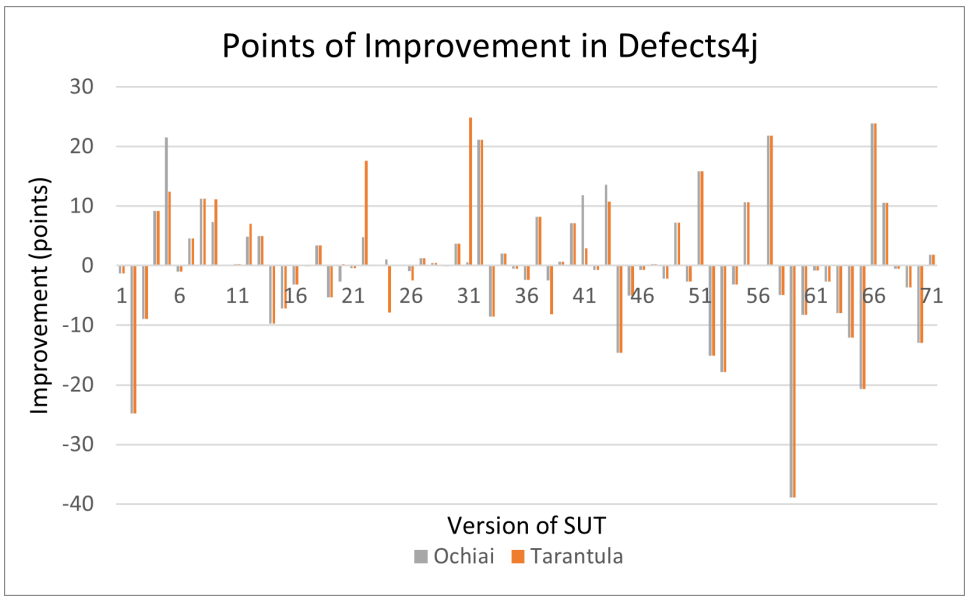


図 21: The Gap Between Statistical and Proposed Methods in The Amount of Code Investigated in Defects4j

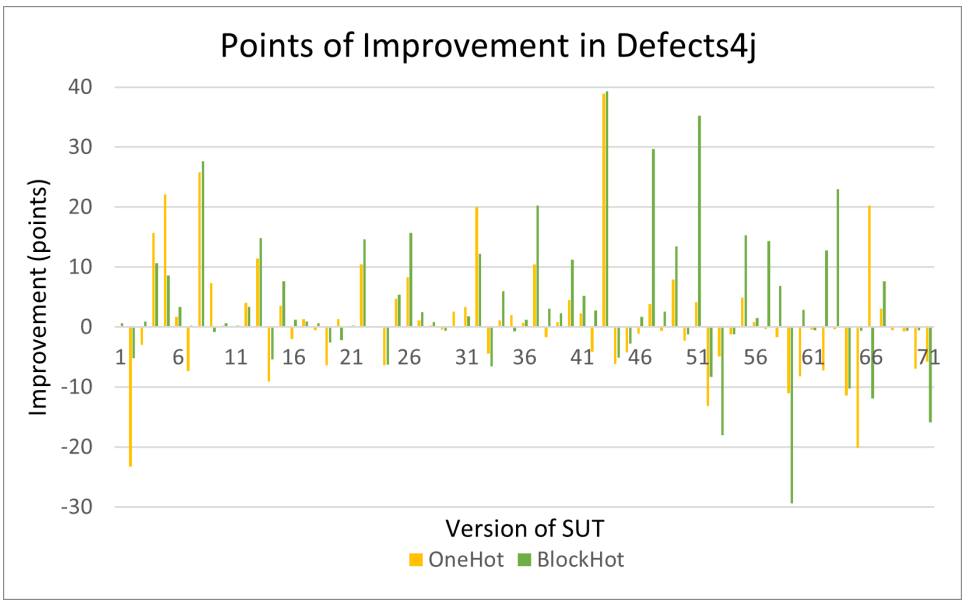


図 22: The Gap Between DNN-based and Proposed Methods in The Amount of Code Investigated in Defects4j

6 考察

6.1 RQ1

図 8, 9, 10, 11 と表 2 が示す Defects4j の全評価対象プログラムを総合した実験結果から, 提案手法と従来手法との間に有意な差はみられない. また, 表 3 が示す検定結果から, SIR を対象にした実験結果においても統計的 SFL 手法 (Ochiai, Tarantula) と提案手法の間に有意な差はみられない. 図 21, 22, 23, 24 に Defects4j と SIR それぞれにおけるプログラムごとの提案手法と従来 SFL 手法の精度差を示す. 図 21, 22, 23, 24 において, 縦軸は提案手法と従来手法の, 故障箇所を特定するのに必要なコード調査量の差を示しており, 横軸は Defects4j において対象となった各プログラムを示す. 縦軸が負の値を示す場合, 従来手法よりも故障箇所特定に必要なコード調査量が多いことを示し, 正の値を示す場合, 必要なコード調査量が少ないことを意味する. したがって, 縦軸が正の値の場合は, 精度が向上したことを意味する.

図 21, 22 から, 提案手法とコードカバレッジ情報を用いる各従来手法とで, 精度が大きく向上する, もしくは精度が低下する故障が一致していることが認められる. また, 図 23, 24 においても, 同様の傾向が認められる. 提案手法は実行された文の組み合わせを実行情報とし, 実行情報の欠落を利用して故障箇所を特定する. 一方で, 従来 SFL 手法はコードカバレッジ情報を利用して故障箇所を特定する. SFL で使用する実行情報と, 故障箇所を特定するためのアプローチの違いが, 図 21, 22, 23, 24 が示す結果の要因であり, 提案手法と従来手法で高い精度を達成することができる故障の種類が異なると考えられる. 提案手法は動機例題 (2.3 節) のような, コードカバレッジ情報のみを用いる従来 SFL 手法では扱うことが難しい故障を, 高い精度で特定することが期待される. RQ2 における考察では, 具体的にどのような故障に対して提案手法が有効に動作するのかについて議論する.

6.2 RQ2

本節では故障の種類ごとに提案手法が有効に動作した要因について議論する.

6.2.1 動機例題の故障

図 20 が示す動機例題 (Chart Version 7) に対する提案手法の適用結果と, 図 2 が示す統計的 SFL 手法の適用結果から, 以下の二点が提案手法が動機例題の故障に有効に動作した要因であると考えられる.

1. Pass で頻繁に実行される文に高い疑わしさスコアを与えている.

コードカバレッジ情報を用いて疑わしさスコアを算出する従来 SFL 手法では, 疑わしさスコアの算出式 ((1) 式, (2) 式) が示すように, Fail で頻繁に実行され, Pass では少ない実行回数の文に高い疑わしさスコアを与える. 動機例題は代入文が故障しており, 故障文が Pass と Fail の両方のテストケースで頻繁に実行される. したがって, 統計的 SFL 手法では他の文よりも高い疑わしさスコアを

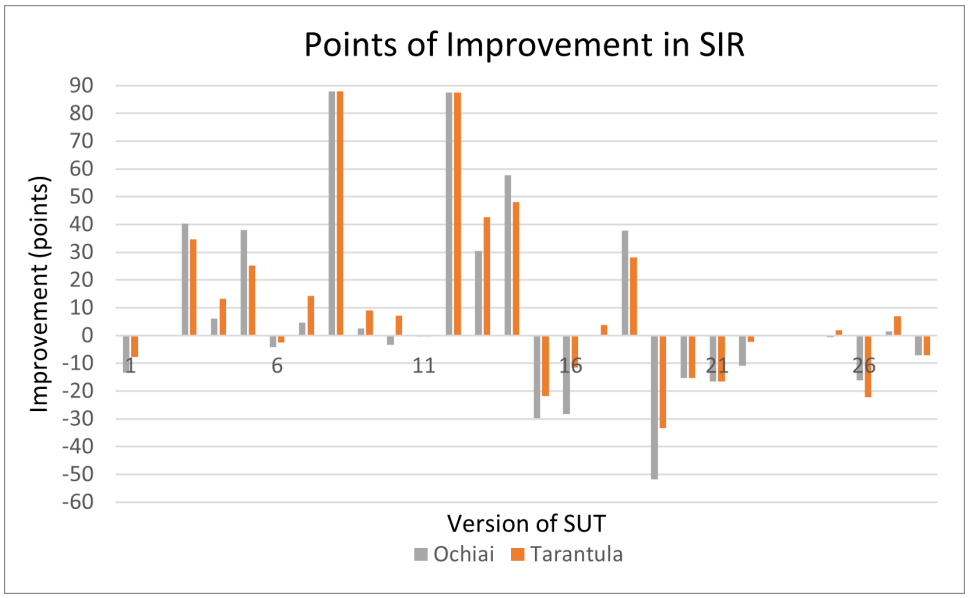


図 23: The Gap Between Statistical and Proposed Methods in The Amount of Code Investigated in SIR

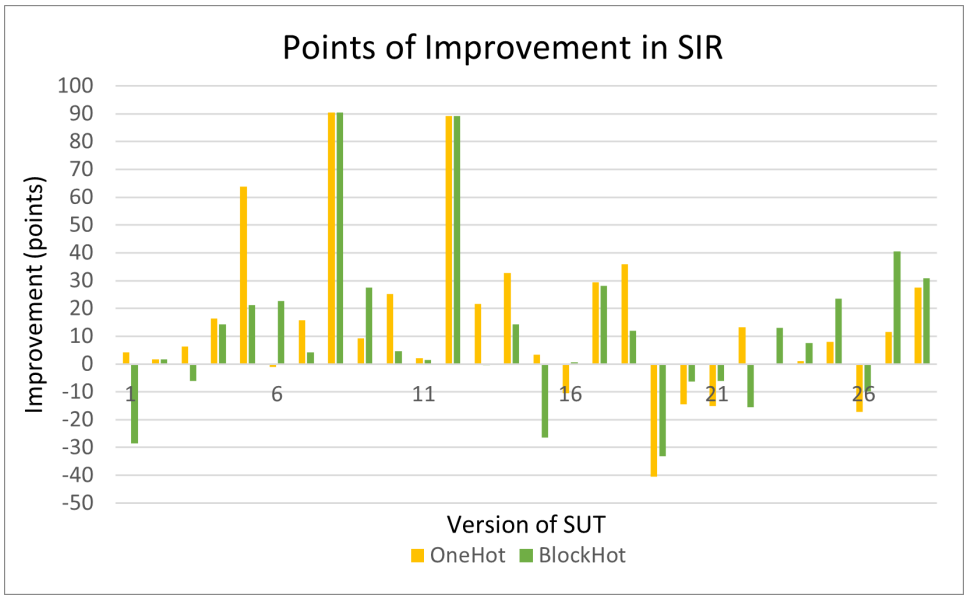


図 24: The Gap Between DNN-based and Proposed Methods in The Amount of Code Investigated in SIR

故障文に与えることができない. 代入の結果が使用される他の文との実行の組み合わせパターンを考慮することで, このような文に高い疑わしさスコアを与えることが期待される. なぜなら, Pass のテストケースで特定の代入文が頻繁に実行されていたとしても, その代入文と他のある文がどちらも実行される組み合わせのパターンは Pass のテストケースには存在せず, Fail のテストケースでしかみられない可能性があるためである. 提案手法における DNN モデルは, 特定の文の組み合わせを学習し, Fail となるテストの実行トレースのテスト結果を Fail として予測分類することが期待される. 特定の文の実行情報の削除 (Ablation) により, 実行情報の組み合わせに欠落を与えることで, DNN モデルの分類結果を観察した際に Pass で頻繁に実行される文にも高い疑わしさスコアを与えることができたと考えられる. 提案手法は故障文と他の文の実行組み合わせを考慮することができ, それが動機例題の故障において従来 SFL 手法よりも高い精度で故障文を特定することができた要因であると考えられる.

2. 同じ疑わしさスコアが与えられる文が少ない.

統計的 SFL 手法は各文の実行回数情報を用いて疑わしさスコアを算出するため, 同一の実行回数となる文同士には同じ疑わしさスコアが与えられる. 実際に図 2 では, 統計的 SFL 手法が故障箇所と同じ疑わしさスコアを他の多くの文に与えていることを示している. 故障文と同じ疑わしさスコアが他の複数の文に与えられた場合, 開発者は自身の経験や知識に基づいて, どの文から優先的に調査するかを決定すると考えられる. 最悪の場合, 開発者は同じ疑わしさスコアが与えられた全ての文を調査しなければ故障箇所を特定することができない. 評価実験で用いた Top N%による評価では, 故障箇所と同じ疑わしさスコアが他の文にも与えられた場合, 最悪の場合を想定し, 開発者は同じスコアが与えられた全ての文を調査したとして評価している. 最悪の場合を想定すると, 図 2 が示す, 統計的 SFL 手法の同じスコアを多くの文に与えるという傾向は, 故障文の特定に悪影響を及ぼす. 一方で, 提案手法の適用結果である図 20 は, 同じスコアが与えられる文の数が統計的 SFL 手法と比較して少ないことを示している. 提案手法は DNN モデルの予測結果である Float 値を疑わしさスコアとして利用するため, 同じ疑わしさスコアを他の文に与えるケースは少ない. したがって, 最悪の場合を想定した際に, 提案手法は統計的 SFL 手法と比較して精度面におけるアドバンテージがあると考えられる.

6.2.2 多重故障

図 16, 17, 18, 19 が示す結果と, 表 4 が示す検定結果から, 多重故障において提案手法が従来手法よりも高い精度で故障箇所を特定している. 図 8, 9, 10, 11 が示す Defects4j における全ての故障を対象にした実験結果と, 図 16, 17, 18, 19 が示す結果を比較すると, 図 16, 17, 18, 19 の従来 SFL 手法における各 Top N%で特定した故障の割合が小さくなっていることが認められる. これは, 従来 SFL 手法が多重故障を対象にした場合に, 特に故障箇所特定性能が低下することを示している. 提案手法においても, 多重故障を対象にした場合に故障箇所特定性能が低下しているが, その低下度合いは従

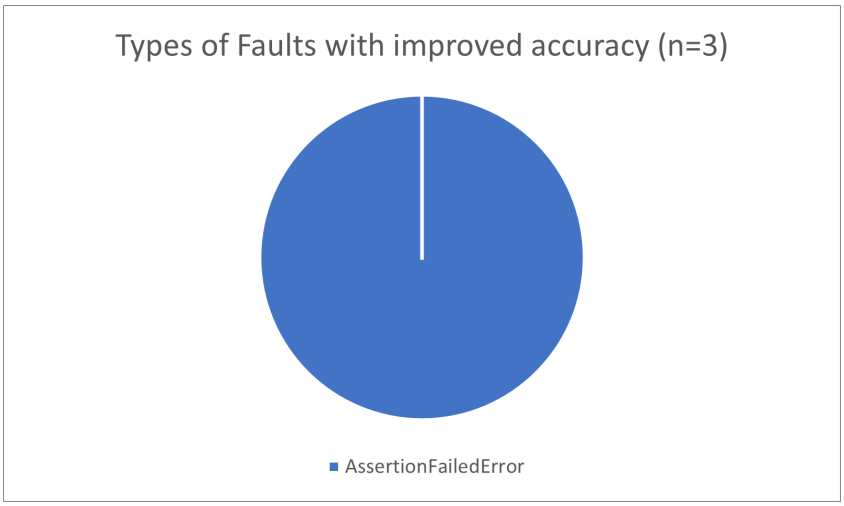


図 25: Percentile of Type of Faults with Improved Accuracy

来 SFL 手法よりも小さい. 特に低下度合いが大きい統計的 SFL 手法に関しては, 6.2.1 節で説明した, 同じ疑わしさスコアを他の多くの文に与えていることが特定性能低下の要因であると考えられる. 提案手法は同じ疑わしさスコアを与える文が少ないため, 特定性能の低下の度合いが小さいと考えられ, 多重故障においては従来 SFL 手法よりも高い精度で故障箇所を特定することができると考えられる.

6.2.3 他の種類の故障

本節では, 単一故障におけるどのような故障に対して提案手法が有効に動作するのかについて議論する. Defects4j では, 各プログラムに Repair Action Tag と Error Tag が与えられている. Repair Action Tag は故障を修正するために行われた作業の種類 (Assignment Expression Modification) を示しており, Error Tag は各故障の Error の種類 (AssertionFailedError など) を示す. これらのタグを, 評価実験で用いた単一故障プログラムの故障の種類を分析するために使用した. また, SIR の故障にはこのようなタグが与えられていないため, 本節では議論しない. 従来手法 (Ochiai) と提案手法との間に 15 ポイント以上の精度差がみられない故障については分析の対象から除外した. Ochiai と Tarantula の精度差は小さいため (5 ポイント未満) , 議論から除外する故障を決定するための比較手法を Tarantula にした場合でも, 本節の議論に影響はない.

分析の結果, Repair Action Tag として, Variable Type Change や Assignment Expression Modification のタグが与えられた故障において, 提案手法が従来手法よりも 15 ポイント以上の精度向上を達成したことを確認した. これらのタグは 6.2.1 で説明した, Pass のテストでも頻繁に実行される文が故障している場合に与えられるタグであり, 動機例題 (2.3 節) に類似した種類の故障である. この結果から, 動機例題の Chart Version 7 以外の, Pass のテストでも頻繁に実行される文の故障に対しても, 提案手法は有効に動作すると考えられる.

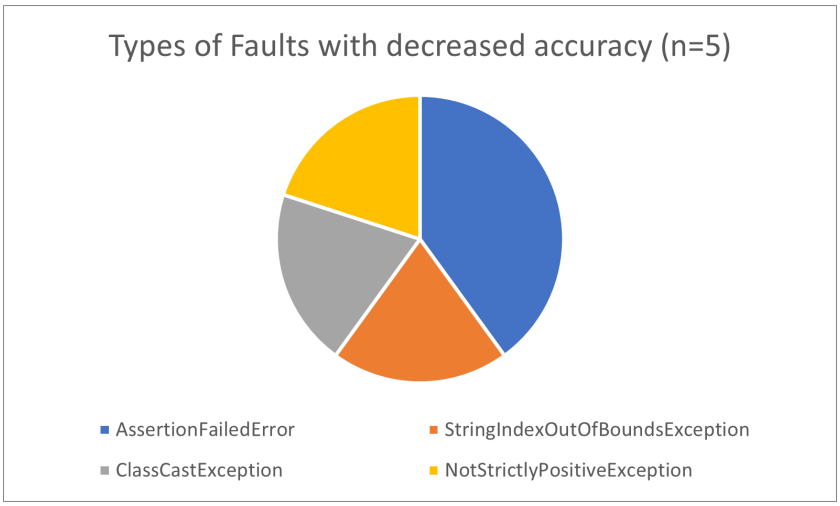


図 26: Percentile of Type of Faults with Decreased Accuracy

図 25, 図 26 に Error Tag の分析結果であるパーセンタイルを示す. 図 25 は提案手法が Ochiai よりも 15 ポイント以上高い精度を達成した故障の種類を示すパーセンタイルであり, 図 26 は Ochiai が提案手法よりも 15 ポイント以上高い精度を示した故障の種類を示すパーセンタイルである. 図 25 から, 提案手法が従来手法よりも高い精度で特定できる故障は, 全て AssetionFailedError であり, Exception 系のエラーは含まれていない. また, 図 26 から, 提案手法が従来手法よりも低い精度を示す故障の種類には, いくつかの例外系 (Exception) のエラーが含まれていることが認められる. Exception 系のエラーの場合, 故障文は異常終了直前に実行された文であることが多い. 異常終了する故障文は, Pass のテストケースでは実行されないケースがほとんどであるため, コードカバレッジ情報 (実行回数情報) を用いて故障箇所を特定する従来 SFL 手法が提案手法よりも高い精度を示したと考えられる. したがって, 提案手法が従来 SFL 手法よりも高い精度で故障を特定することが期待できないエラーの種類としては, Exception 系エラーといった, 異常終了直前に実行された文が故障している可能性が高いエラーが考えられる.

7　まとめ

本論文では, 教師ありテストケース学習を用いた故障箇所特定への新しいアプローチを提案した. 提案する手法は, Ablation された実行トレースを DNN モデルに入力し, その出力値を用いて故障を特定するものである. 本論文では, 提案手法を 6 つの異なるプロジェクトを用いて評価した. RQ1 (6.1 節) では, 提案手法と従来手法における, 全ての故障を対象にした総合的な精度の比較を行い, 結果として従来手法と提案手法との間に有意な差は認められなかった. 本論文では, どのような故障において提案手法が有効に動作するのかについて調査するため, RQ2 (6.2 節) で評価実験で用いた故障の種類を調査し, その種類別に実験結果を分析した. 分析の結果, 複数の故障が存在する場合や, 故障文が Pass のテストでも頻繁に実行されるようなケースにおいて, 提案手法は従来手法よりも高い精度を達成することが認められた. また, Exception 系のエラーに対しては, 提案手法よりも従来手法を用いたほうが高い精度で故障箇所を特定できると考察した. また, 考察 (6 節) では, 一部の故障において提案手法は従来手法よりも故障特定性能が高く, 他の故障に対しては従来手法よりも故障特定性能が低いことを示した.

この結果は, 提案手法を他の SFL 手法と組み合わせて使用することで, 開発者のデバッグ作業の負担をより軽減できる可能性を示している. 提案手法が多重故障や代入文における故障を高い精度で特定したように, 他の SFL 手法においても高い精度で特定できる故障の種類などの特性が異なると考えられる. したがって, 各手法において高い精度で特定できる故障の特性を理解し, それらを組み合わせて使用することで, あらゆる故障やプログラムに対応することができる. 本論文と提案手法は, そのような汎化性能が高い SFL 手法の開発に貢献することができると考えられる. その観点において, 我々が分析した故障の種類は実際に発生する可能性のある故障の一部であり, より多くの故障における提案手法の有効性の分析を行うことが今後の課題である.

謝辞

本研究を行うにあたり，全過程を通して理解あるご指導，ご助言を賜わりました岡野浩三教授に深く感謝を申し上げます.

本研究に対して，細部にわたる熱心かつ丁寧なご指導を頂きました小形真平准教授に感謝の意を表します.

本研究を行うにあたり，公明なアイデアや大変丁寧なご指導，ご協力頂きました国立情報学研究所中島震名誉教授に厚くお礼申し上げます.

本研究を行う過程で様々な形でのご助言やご協力, 励ましをいただきました岡野・小形研究室の皆様に深く感謝いたします.

参考文献

[1] G. Tassey, “The economic impacts of inadequate infrastructure for software testing,” 2002.

[2] H.A. deSouza, M.L. Chaim, and F. Kon, “Spectrum-based software fault localization: A survey of techniques, advances, and challenges,” CoRR, vol.abs/1607.04347, 2016. http://arxiv.org/abs/1607.04347

[3] A. Perez and R. Abreu, “A qualitative reasoning approach to spectrum-based fault localization,” Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pp.372–373, ICSE ’18, Association for Computing Machinery, New York, NY, USA, 2018.

[4] Q.I. Sarhan and A. Beszédes, “A survey of challenges in spectrum-based software fault localization,” IEEE Access, vol.10, pp.10618–10639, 2022.

[5] R. Abreu, P. Zoeteweij, and A.J. Van Gemund, “An evaluation of similarity coefficients for software fault localization,” 2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC’06), pp.39–46, 2006.

[6] J.A. Jones, M.J. Harrold, and J. Stasko, “Visualization of test information to assist fault localization,” Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, pp.467–477, 2002.

[7] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and X. Zhang, “A study of effectiveness of deep learning in locating real faults,” Information and Software Technology, vol.131, p.106486, 2021.

[8] W.E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, “Effective software fault localization using an rbf neural network,” IEEE Transactions on Reliability, vol.61, no.1, pp.149–169, 2012.

[9] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and J. Wen, “Improving deep-learning-based fault localization with resampling,” J. Softw. Evol. Process, vol.33, no.3, mar 2021. https://doi.org/10.1002/smr.2312

[10] H. Kiryu, S. Ogata, and K. Okano, “Improve measuring suspiciousness of bugs in spectrum-based fault localization with deep learning,” Proceedings of International Workshop on Informatics, pp.3–8, IWIN ’22, Informatics Laboratory, Kii-Katsuura, Japan, 2022.

[11] T. Ikeda, H. Kiryu, S. Ogata, and K. Okano, “Dnn-based fault localization with virtual coverage based on number of executions,” Proceedings of International Workshop on Informatics, IWIN ’23, Informatics Laboratory, Mori-machi, Hokkaido, Japan, 2023.

[12] F. Tsimpourlas, A. Rajan, and M. Allamanis, “Supervised learning over test executions as a test oracle,” Proceedings of the 36th Annual ACM Symposium on Applied Computing, p.1521–1531, SAC ’21, Association for Computing Machinery, New York, NY, USA, 2021.

[13] R. Just, D. Jalali, and M.D. Ernst, “Defects4j: A database of existing faults to enable controlled testing studies for java programs,” Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp.437–440, ISSTA 2014, Association for Computing Machinery, New York, NY, USA, 2014.

[14] H. Do, S. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” Empirical Software Engineering, vol.10, pp.405–435, Oct. 2005.

[15] F. Wilcoxon, “Individual comparisons by ranking methods,” Biometrics Bulletin, vol.1, no.6, pp.80–83, 1945.

[16] G. Laghari, K. Dahri, and S. Demeyer, “Comparing spectrum based fault localisation against test-to-code traceability links,” 2018 International Conference on Frontiers of Information Technology (FIT), pp.152–157, 2018.

[17] R. Abreu, P. Zoeteweij, and A.J. vanGemund, “On the accuracy of spectrum-based fault localization,” Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007), pp.89–98, 2007.

[18] W. Masri, “Fault localization based on information flow coverage,” Software Testing, Verification and Reliability, vol.20, no.2, pp.121–147, 2010. https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.409

[19] W.E. Wong, V. Debroy, R. Gao, and Y. Li, “The dstar method for effective software fault localization,” IEEE Transactions on Reliability, vol.63, no.1, pp.290–308, 2014.

[20] Lucia, D. Lo, and X. Xia, “Fusion fault localizers,” Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, pp.127–138, ASE ’14, Association for Computing Machinery, New York, NY, USA, 2014. https://doi.org/10.1145/2642937.2642983

[21] S. Murtaza, N. Madhavji, M. Gittens, and A. Hamou-Lhadj, “Identifying recurring faulty functions in field traces of a large industrial software system,” Reliability, IEEE Transactions on, vol.64, pp.269–283, 03 2015.

[22] J. Sohn and S. Yoo, “Fluccs: Using code and change metrics to improve fault localization,” Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp.273–283, ISSTA 2017, Association for Computing Machinery, New York, NY, USA, 2017. https://doi.org/10.1145/3092703.3092717

[23] Y. Li, S. Wang, and T.N. Nguyen, “Fault localization with code coverage representation learning,” Proceedings of the 43rd International Conference on Software Engineering, pp.661–673, ICSE ’21, IEEE Press, 2021.

[24] T. Wang and A. Roychoudhury, “Automated path generation for software fault localization,” Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, p.347–351, ASE ’05, Association for Computing Machinery, New York, NY, USA, 2005. https://doi.org/10.1145/1101908.1101966

[25] “OpenClover,” https://openclover.org/, (Accessed 18 October 2023).

[26] “gcov - A Test Coverage Program,” https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[27] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” CoRR, vol.abs/1310.4546, 2013.

[28] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, vol.9, no.8, pp.1735–1780, Nov. 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[29] 池田拓真，岡野浩三，小形真平，中島震，“テスト実行結果を自動分類するためのメソッドにおける近接情報を活用した実行トレースの符号化,” 信学技報，第 121 巻，pp.83–88，MSS2021-46，SS2021-33，長崎，1 月 2022．2022 年 1 月 11 日 (火)-1 月 12 日 (水) 長崎県建設総合会館 (SS，MSS)．

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[31] W. Eric Wong, V. Debroy, and B. Choi, “A family of code coverage-based heuristics for effective fault localization,” Journal of Systems and Software, vol.83, no.2, pp.188–208, 2010. Computer Software and Applications. https://www.sciencedirect.com/science/article/pii/S0164121209002465

[32] X. Xie, T.Y. Chen, F.-C. Kuo, and B. Xu, “A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization,” ACM Trans. Softw. Eng. Methodol., vol.22, pp.31:1–31:40, 2013. https://api.semanticscholar.org/CorpusID:14834203

[33] R. Santelices, J.A. Jones, Y. Yu, and M.J. Harrold, “Lightweight fault-localization using multiple coverage types,” Proceedings of the 31st International Conference on Software Engineering, pp.56–66, ICSE ’09, IEEE Computer Society, USA, 2009. https://doi.org/10.1109/ICSE.2009.5070508

[34] N. Digiuseppe and J.A. Jones, “Fault density, fault types, and spectra-based fault localization,” Empirical Softw. Engg., vol.20, no.4, pp.928–967, aug 2015. https://doi.org/10.1007/s10664-014-9304-1

[35] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J. van Gemund, “A practical evaluation of spectrum-based fault localization,” Journal of Systems and Software, vol.82, no.11, pp.1780–1792, 2009. SI: TAIC PART 2007 and MUTATION 2007. https://www.sciencedirect.com/science/article/pii/S0164121209001319

[36] D.P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, eds. by Y. Bengio and Y. LeCun, 2015.

[37] 池田拓真，岡野浩三，小形真平，中島震，“テスト実行結果を自動分類するための機械学習モデルを利用した実行トレースのアブレーションとメソッド単位でのバグ箇所推定,” 信学技報，第 121 巻，pp.13–18，SS2021-44，ONLINE，3 月 2022．2022 年 3 月 7 日 (月)-3 月 8 日 (火) オンライン開催 (SS).

[38] 池田拓真，岡野浩三，小形真平，中島震，“アブレーションによる故障箇所特定における符号化方法とアブレーション方法の改善,” 信学技報，第 122 巻，pp.121–126，SS2022-67，沖縄，3 月 2023．2023 年 3 月 14 日 (火)-3 月 15 日 (水) 名護市産業支援センター (SS).

[39] T. Ikeda, K. Okano, S. Ogata, and S. Nakajima, “Fault localization with dnn-based test case learning and ablated execution traces,” Proceedings of the 2nd International Workshop on Intelligent Software Engineering, ISE ’23, CEUR-WS, Grand Walkerhill Seoul, Gwangjin-gu, Seoul, Korea, to appear, 2023.

[40] T. Ikeda, H. Kiryu, S. Suda, S. Ogata, and K. Okano, “Fault localization with virtual coverage and supervised learning based on execution count information,” International Journal of Informatics Society (IJIS), vol.16, to appear, 2024.