

変更ルールマイニングのための解析範囲動的決定手法

内藤 祐樹^{1,a)} 小林 隆志^{1,b)}

概要：改版履歴から変更ルールを抽出することにより現在の変更で影響が生じる可能性のあるファイルを検出できる。先行研究によって、過去のすべてではなく直近の一定数のコミットに制限して解析する方が平均して検出精度が高いことが示されている。しかしながら、実際にはプロジェクトごとに最適なコミット数は可変である。本研究では、コミットごとに動的に解析範囲を決定する手法を提案する。本稿では、12 の OSS を対象に分析することで、解析範囲の最適長はプロジェクトごととコミットごとに異なることを示す。さらに、共変更の出現傾向を考慮することでコミットごとに動的に解析範囲を推定する提案手法は、最適なプロジェクト単位での固定長と同程度の精度であることを明かす。

キーワード：解析範囲の動的決定, 変更影響解析, 改版履歴分析, ソフトウェア保守

1. はじめに

ソフトウェアに対して変更を加える際には、変更すべきソースコードを漏れなく編集することが必要である。しかしながら、ソフトウェアの進化を追従しつつソースコード間の依存関係を常に把握することは困難であり、変更波及箇所の見落としから変更漏れが生じてしまう。

この問題に対処する方法として、静的解析を行って依存関係を追跡し変更波及箇所を特定するアプローチが提案されてきた [1]。しかし、依存関係を網羅的に抽出できる一方で、実際には影響を受けない依存関係も多数検出してしまいう傾向にある [2]。さらに、静的解析は一般的に固有のプログラミング言語を対象として実行するため、複数の言語で開発されるソフトウェアの分析には適していない [3]。

静的解析による変更波及解析の問題に対処するため、Git や Subversion などの版管理システムが保持する改版履歴を分析する研究が行われている。改版履歴から過去に頻繁に共変更されたファイルの集合を探索することでそれらの共変更性を求め、変更ルールとして抽出する [4]。それにより、静的解析では捉えられない重要な依存関係を検出できる [5]。変更ルールは $A \Rightarrow B$ の形式であらわれ、もし A が変更されたならば B も共変更される可能性が高いことを意味する。そして、このようなルールの集合と変更中のファイル集合に基づいて、変更漏れの可能性があるファイルを開発者に推薦することで変更波及箇所の特定作業を支援できる。なお、 A , B はそれぞれルールの左辺、右辺と呼ばれる。

改版履歴分析では解析対象コミット数の制限が有効であるとされる [6, 7]。ファイル間の依存関係は経年変化するため、改版履歴全体の解析では開発初期に形成される変更

ルールがノイズとなり、有益な依存関係の特定は困難になる。一方で、過少なコミットの解析では重要な依存関係を検出できない。

そこで、Moonen ら [8, 9] や森ら [10] は解析対象コミット数が変更波及解析の結果に与える影響を調査している。これら先行研究では、選択した実験対象プロジェクトで変更ルールを満足に抽出するためのコミット数を見積もっている。そのため、示された解析範囲に一般性があるとはいえない。また、あらゆるプロジェクトに対して同じコミット数の解析を前提としており、プロジェクト単位やコミット単位で解析範囲の変化による推薦精度への影響については未調査である。さらに、先行研究で示された解析範囲は改版が少ないプロジェクトを考慮していない。これらの問題に対し我々は、プロジェクト単位やコミット単位に解析範囲を切り替えることで変更推薦の精度が向上するのではないかと考える。

本研究では、12 の OSS に対する事前調査によって、最適な解析範囲はプロジェクトごとやコミットごとで異なることを明らかにする。そして、改版履歴分析による変更推薦の精度向上を目的として、コミットごとに解析範囲を動的に決定する手法を提案する。本手法では、最新コミットから遡って解析する過程で、すでに抽出済みである変更ルールの平均出現回数に着目して解析の継続判定を行う。継続判定で用いる閾値を自動的に算出することで、解析範囲に関するマイニングパラメータの設定を開発者に求めずに、コミットごとにその範囲を動的に決定する。

本稿の貢献は以下のとおりである。

- プロジェクト単位やコミット単位に最適な解析長は変化することを実証した。
- 先行研究で示された固定解析長に従うより、提案手法を適用する方が変更推薦の精度が優れることを示した。
- 提案手法による推薦精度はプロジェクト単位の最適な解析長の適用時と同程度であることを明かした。

¹ 東京工業大学 情報理工学院
School of Computing, Tokyo Institute of Technology

a) y.naito@sa.cs.titech.ac.jp

b) tkobaya@c.titech.ac.jp

2. 関連研究

Zimmermann ら [4] は、クラスやメソッド単位の共変更性を検出し、現在の変更から次に変更すべきソフトウェア・エンティティを推薦するツールとして ROSE を提案した。彼らは、開発者が現在の変更を版管理システムにコミットする状況で ROSE は変更漏れを推薦できるかという実験を行い、変更推薦が正確であることを示した。

ROSE による変更推薦は非常に高速に行われるが、限られた変更ルールしか抽出できず、多くの場合で変更漏れを推薦できないという問題がある。Rolfesnes ら [7] は、Targeted Association Rule Mining [11] に倣ってルールの抽出に用いるコミットを制限することで、効率よく有益な変更ルールを抽出する TARMAQ アルゴリズムを提案した。彼らは、TARMAQ アルゴリズムの方が ROSE より多くの変更ルールを抽出できるとしている。また、両者の計算時間には実用上有意な差はなく、推薦順位も加味した推薦精度をあらわす *Average Precision* の観点で TARMAQ アルゴリズムは ROSE より優れていることも明かした。

変更ルールを活用した変更支援手法に関する研究は多数存在する。Alali ら [12] は共変更性の検出で、変更ルールの時間的局所性と空間的局所性について調査した。Kagdi ら [13] は sqminer を提案し、ファイルへの変更の順序関係を分析することで変更予測に役立てようとした。そして、Islam ら [14] は、抽出される変更ルールに対して推移性を適用することで、過去に共変更されなかったソフトウェア・エンティティ間における依存関係の予測が可能であることを示した。

改版履歴分析を効果的に行うため、マイニングパラメータの設定に関する研究が行われている。Moonen ら [6] は大規模コミットを解析対象から除外して変更ルールを抽出した場合における推薦精度への影響を調査した。彼らは 39 の Interestingness Measure を用いて順位付けを行った場合の推薦精度を比較し、従来の支持度と確信度が優れることも示した。Mondal ら [15] は共変更の候補となるメソッドを高精度で推薦するため、5 種類の変更ルールの順位付け基準から最適な基準を動的に決定する HistoRank を提案した。彼らは単一の順位付け基準に従うよりも HistoRank によって順位付け基準を切り替える方が高い精度で変更推薦できることを示した。

マイニングパラメータの設定の一つに、変更ルールの抽出で用いる解析対象コミット数がある。Monnen ら [8,9] はすべての実験対象プロジェクトに対して解析対象コミット数を統一し、それを変化させたときの推薦精度の変動を分析している。結果として、どのプロジェクトも最新 15,000 から 25,000 コミットの解析で十分であるとした。また、森ら [10] はファイルの共変更性には時間的近接性が

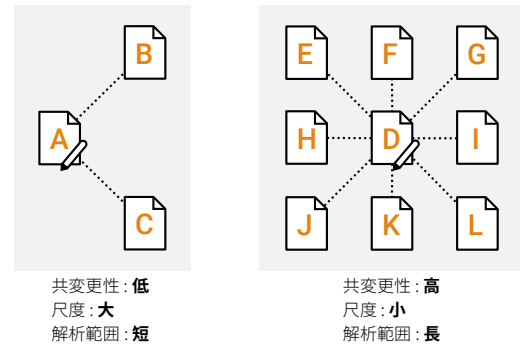


図 1 変更ファイルの共変更性による尺度の傾向と解析範囲の目安

あり、改版履歴全体の解析と比較することで直近 5,000 コミットの分析による推薦精度の向上を示した。しかし、これら先行研究では解析対象コミットをあらゆるプロジェクトで同数にすることを前提に議論している。本研究では、プロジェクト単位やコミット単位で解析範囲を切り替えるべきか議論を行い、コミットごとに解析範囲を動的に決定する手法を提案する。

3. 提案手法

本手法は変更ルールの多寡を把握することで解析範囲を動的に決定する。そのために、ルールの平均出現回数を導入する。以下では、コミット単位の解析範囲が満たすべき性質を議論し (3.1)、平均出現回数概念 (3.2) とそれに基づく解析範囲の決定方針 (3.3)、動的決定で用いる閾値の決定方法 (3.4) について述べる。

3.1 適切な解析範囲の満たすべき性質

本手法では変更中のファイル集合に基づく改版履歴分析 [11] において、抽出される変更ルールの多寡に着目して解析対象コミット数に当たりをつける。最新コミットから遡って変更ルールを順次抽出する過程で、そのときの変更中のファイル集合における共変更性に着目する。図 1 に共変更性の概要を示す。

例えば、改版履歴全体から少数のルールのみ抽出できる状況では変更中のファイル集合の共変更性は低く、限られたファイルと共変更される傾向にある (図 1 の左側)。ある程度の解析範囲で変更ルールを抽出した後、それ以上遡ってもあらたなルールを検出できない場合、すでに重要なルールを発見している可能性が高い。ファイル間の依存関係の経年変化を考慮すると、そのような状況では解析を継続しない方がよい。

一方で、偶発的な共変更を多く検出することで多数の変更ルールを抽出できる場合、変更中のファイル集合は他のファイルと共変更されやすい傾向にある (図 1 の右側)。変更中のファイル集合における高い共変更性を踏まえると、過去に偶然共変更されたファイルが現在の変更漏れファイルとなる可能性がある。そのため、それを検出するには多

くのコミットの解析が必要である。

3.2 変更ルールの平均出現回数の概念

変更中のファイル集合において図 1 に示す共変更性を区別するため、検出済みの変更ルールにおける平均出現回数を尺度として導入する。

解析時の変更ファイル集合を $Query$ とし、 $Query$ 内のいずれかのファイルを含むコミットで構成される集合を $Commit_t$ とする。 $Commit_t$ 上の最新コミットから解析を始め、 i 番目のコミットへ遡るまでに検出したルール集合を $Rule_i$ とする。単純化のため、 $Rule_i$ 内の変更ルールは左辺も右辺も共に 1 つのファイルで構成されるものとする。また、 $Rule_i$ の各ルール $rule_j$ に関して、 i 番目の時点における変更ルールの出現回数を $app_i(rule_j)$ とする。このとき、最新コミットから i 番目の時点で算出されるルールの平均出現回数 $avgApp$ を次式として定義する。

$$avgApp = \frac{\sum_j^{Rule_i} app_i(rule_j)}{|Rule_i|}$$

$Commit_t$ 内のコミットを遡る度に本尺度を求めることで、変更中のファイル集合に対する共変更性を追跡する。それにより、共変更性が低い場合は少数の変更ルールを検出できることから平均出現回数は多くなる (図 1 の左側)。他方で、変更中のファイル集合における共変更性が高い状況では多数のルールを検出することになり平均出現回数は少なくなる (図 1 の右側)。つまり、変更ルールの平均出現回数により変更中のファイル集合の共変更性を区別でき、本尺度が大きいときは長く解析しなくともルールを十分に検出可能であると判断する。

3.3 平均出現回数による解析範囲の決定

平均出現回数の定義により、最新コミットから遡りながら変更ルールを検出する過程で本尺度は変動する。次の解析対象コミットから形成される変更ルールがすでに検出済みである場合では本尺度は増加する。一方で、遡ったときに未検出ルールを抽出する場合には、検出済みルールの種類数が増えるため本尺度は減少する。このような変更ルールの初検出時では、ノイズとなるルールを抽出してしまう可能性がある。ノイズの混入を回避するため、平均出現回数が減少するときに解析を停止させることが望ましい。したがって、そこまで解析した範囲に応じた十分な平均出現回数があり、減少傾向にある場合に変更ルールの抽出を停止させる。

最新コミットから遡る過程で変更ルールの平均出現回数 $avgApp$ により解析の継続判定を行う手順をアルゴリズム 1 に示す。本尺度の減少を検知するために閾値 T を設け、尺度がそれを下回るときに解析を停止させる。なお、共変更性の追跡と同時に、変更漏れファイルの推薦に用いる

Algorithm 1 平均出現回数に基づいた変更ルールの抽出

Require: 制約付きコミット集合: $Commit_t$,
変更ファイル集合: $Query$

Ensure: 変更ルール: $RULE$

```

1:  $RULE \leftarrow \emptyset$ 
2:  $Rule_i \leftarrow \emptyset$ 
3:  $i \leftarrow 0$ 
4: for all  $Com \in Commit_t$  do
5:    $i \leftarrow i + 1$ 
6:   for all  $lhs \subseteq Com \cap Query$  do
7:     for all  $rhs \in Com \setminus Query$  do
8:        $RULE \leftarrow RULE \cup \{lhs \Rightarrow rhs\}$ 
9:       if  $|lhs| = 1$  then
10:         $Rule_i \leftarrow Rule_i \cup \{lhs \Rightarrow rhs\}$ 
11:         $increment(app(lhs \Rightarrow rhs))$ 
12:       end if
13:     end for
14:   end for
15:    $sum \leftarrow 0$ 
16:   for all  $rule \in Rule_i$  do
17:      $sum \leftarrow sum + app(rule)$ 
18:   end for
19:    $avgApp \leftarrow sum / |Rule_i|$ 
20:   if  $avgApp < T$  then
21:     break
22:   end if
23: end for
24: return  $RULE$ 

```

ルールの収集も行う。それら変更ルールはアルゴリズム 1 では $RULE$ と記している。ただし、確信度と支持度の算出に用いるパラメータの更新手順は省略する。

3.4 解析範囲の動的決定で用いる閾値の算出

変更中のファイル集合に対する共変更性を評価して解析範囲を動的に決定するため、本尺度に対する閾値 T を定める。ただし、単純に固定値を閾値として設定するだけでは、解析の過程で尺度が増減することから解析範囲の動的決定における柔軟性を失うという問題がある。そこで、解析の継続判定で用いる閾値は解析対象コミットを遡る度に変化させる。

閾値を設定するために $Commit_t$ 全体の走査によって求まる変更ルールの平均出現回数に着目し、それを基準に解析対象コミットを遡る過程で算出する本尺度を評価する。全体の傾向と比較することで、現在の解析状況が限られた変更ルールしか抽出しない傾向にあるのか、膨大なルールを検出する傾向にあるのか判定する。

$Commit_t$ 全体を走査することで得られる変更ルールの集合を $Rule$ とする。先程の $avgApp$ の定義と同様に、 $Rule$ を構成する変更ルールは左辺も右辺も 1 つのファイルであるとする。また、 $Rule$ の各ルール $rule_k$ における出現回数を $app(rule_k)$ とする。さらに、 $|Commit_t|$ が大きい場合は多くのコミットから変更ルールを抽出することになるため、それに伴い $app(rule_k)$ の総和は大きくなる。こ

れは $|Rule|$ が大きくても膨大な解析対象コミットにより、全体を通した変更ルールの平均出現回数も多くなる可能性を示唆している。そこで、閾値の定義には解析対象コミット数も加味する。したがって、アルゴリズム 1 の 20 行目における閾値 T を次のように定義する。なお、 i は解析の過程で $Commit_t$ 上の最新コミットから遡った回数をあらわす。

$$T = \frac{\sum_k |Rule| \text{app}(rule_k)}{|Rule|} \times \frac{1}{|Commit_t|} \times i$$

これにより、解析対象コミットに関連するパラメータの設定を開発者に求めず、動的に解析範囲を決定する。

4. 評価

4.1 Research Questions

本研究では、実験によって以下に設定する 4 つの Research Question に対する答えを明らかにする。

- **RQ1:** プロジェクト単位で解析範囲を切り替えた方が優れた変更推薦が可能か。

推薦精度が最大となる最適な解析範囲はプロジェクトごとに大きく異なるかどうかを明らかにする。実験対象プロジェクトごとに、解析対象コミット数を 0 から総コミット数の 1/1000 単位ずつ増やし、その過程で評価尺度が最大となる解析範囲を特定する。そして、プロジェクトごとに定まる最適な解析範囲を比較する。

- **RQ2:** プロジェクト単位よりコミット単位で解析範囲を切り替えた方が優れた変更推薦が可能か。

コミットごとに解析範囲を変化させることで、プロジェクト単位の最適な解析範囲を指定するより優れた変更推薦が可能か調査する。まず、コミットごとの最適な解析範囲は一つに定まらないことを確認する。そして、コミットごとに最適な解析範囲を設定したときと RQ1 のプロジェクト単位の最適固定長を指定したときの推薦精度を比較する。なお、コミットごとの最適な解析範囲とは、変更漏れファイルの推薦順位が最上位になるときの解析範囲と定義する。

- **RQ3:** 先行研究の固定解析長を用いるより提案手法によって推薦精度が向上するか。

先行研究で示された固定解析長に従う場合と比較して、提案手法を適用することで変更推薦の精度が向上するか調査する。固定解析長として 5,000 [10] と 15,000, 25,000 [8,9] を選択し、それらの設定時と提案手法による推薦精度を比較する。また、Wilcoxon の符号順位検定 ($\alpha=0.05$) により提案手法の方が有意に優れるか評価する。さらに、Cliff の δ [16] を効果量として算出し、提案手法と固定解析長の精度の差を評価する。なお、固定解析長よりコミット数が少ないプロジェクトでは、評価対象コミットから過去の全コミットを解析対象として使用し実験を行う。

表 1 実験対象プロジェクト

プロジェクト	総コミット数	開発期間
Pig	3,696	2007/10 - 2020/10
Calcite	4,460	2012/04 - 2020/10
Struts	5,978	2006/02 - 2020/10
Accumulo	10,650	2011/10 - 2020/10
Tomcat	22,453	2006/03 - 2020/08
OpenLDAP	22,980	1998/08 - 2020/08
Eclipse JDT	24,642	2001/06 - 2020/08
OpenSSL	26,897	1998/12 - 2020/08
Camel	48,524	2007/03 - 2020/11
GCC	179,103	1988/11 - 2020/08
gecko-dev	724,474	1998/03 - 2020/08
Linux	949,627	2005/04 - 2020/08

- **RQ4:** プロジェクト単位やコミット単位の最適長による推薦精度と比較して提案手法に有効性があるか。

RQ3 と同様の評価方法で、プロジェクト単位やコミット単位の最適な解析長を設定したときの結果と比較し、提案手法の推薦精度を評価する。それぞれの最適な解析長は本来未知数であり、その結果と比較することで提案手法の有効性を裏付ける。

4.2 実験対象

本実験では表 1 に示す 12 の OSS を対象とする。開発規模が大きく異なるプロジェクトを選択している。

4.3 実験手順

本実験は、leave-one-out 交差検証 [10] により仮想的な欠陥に対する変更推薦の精度を評価する。各実験対象プロジェクトで最新 2,000 コミットを評価対象コミットとする。評価対象コミットで変更された 1 ファイルを欠損させ、そのファイルを変更し忘れた状況を仮想的に作り出す。欠損後のファイル集合を現在変更している最中であるとし、それより過去のコミットを解析して変更ルールの検出を行う。第一基準を確信度、第二基準を支持度 [17]、第三基準をファイルパスの辞書順 [18] として抽出したルールを順位付けし、変更漏れファイルの候補をランキング形式で出力する。その候補中における欠損ファイルの有無と推薦順位から推薦精度を評価する。

同時変更ファイル数が 1 のコミットは leave-one-out 交差検証が行えないため評価対象から除外する。同時変更ファイル数の多いコミットはライセンス情報の変更やソースコードの書式変更などにより発生する [6, 18–20]。このようなコミットから形成される変更ルールは、偶発的な依存関係を示しノイズとなる可能性が高い。ノイズとなりうる変更ルールが関与する評価を避けるため、同時変更ファイル数が 30 以上のコミットを評価対象と解析対象から取り除く。

抽出する変更ルールに制限を設け、解析の効率化を図る。

Targeted Association Rule Mining [11] に倣って変更中のいずれかのファイルを左辺に含み、右辺にそれ以外のファイルを置くルールのみを抽出する。なお、それらルールの左辺は要素数が 1 以上 3 以下のファイル集合で、右辺は単一のファイルとする。本手法の変更ルールの平均出現回数は、左辺も右辺も単一ファイルの変更ルールのみを扱うことに注意する。また、最小支持度と最小確信度 [4] は 0.0 とし、変更ルールの支持度および確信度に対して制限を設けない。

なお、本実験では Git の改版履歴から抽出した変更ルールに基づいて推薦を行うツール LCEExtractor [10] を用いる。我々は LCEExtractor に解析範囲を動的に決定するアルゴリズムを実装した。

4.4 評価尺度

変更推薦の評価尺度は森ら [10] と同様に、各評価対象コミットにおける推薦の正確性をあらわす mrr_i と推薦の網羅性を示す $recall_i$ を求め、トレードオフの関係より、それらの調和平均である f_{mrr_i} を用いる。

任意の評価対象コミットを com_i とし、そのコミットで変更されたファイル集合をあらわす。 com_i から 1 ファイル $c \in com_i$ を欠損させた集合に基づいて推薦されるファイル集合を $recom_{i,c}$ とする。また、 $recom_{i,c}$ の順位付けによる欠損ファイル c の推薦順位を $rank_{i,c}$ と定義する。このとき、評価対象コミット com_i における推薦の発生率をあらわす $feedback_i$ を算出し、 $recall_i$, mrr_i , f_{mrr_i} を次のように定義する。

$$feedback_i = \frac{1}{|com_i|} \sum_{c \in com_i} \begin{cases} 1 & (recom_{i,c} \neq \emptyset) \\ 0 & (otherwise) \end{cases}$$

$$recall_i = \frac{1}{|com_i|} \sum_{c \in com_i} \begin{cases} 1 & (c \in recom_{i,c}) \\ 0 & (otherwise) \end{cases}$$

$$mrr_i = \frac{1}{feedback_i \cdot |com_i|} \sum_{c \in com_i} \begin{cases} \frac{1}{rank_{i,c}} & (c \in recom_{i,c}) \\ 0 & (otherwise) \end{cases}$$

$$f_{mrr_i} = \begin{cases} 2 \cdot \frac{mrr_i \cdot recall_i}{mrr_i + recall_i} & (mrr_i \neq 0, recall_i \neq 0) \\ 0 & (otherwise) \end{cases}$$

4.5 実験結果 (RQ1) : プロジェクト単位の変換効果

各プロジェクトで、解析範囲の切り替えによって f_{mrr_i} の平均値が最大となる最適解析長を表 2 に示す。また、プロジェクトの総コミット数に対する最適解析長の比率も記載する。それぞれ太字は最大値、下線は最小値をあらわす。最適解析長を比較すると、一番長い Linux の 773,946 と一番短い Eclipse JDT の 2,488 で大きな差が生じた。比率

表 2 プロジェクト単位の最適解析長

プロジェクト	最適解析長	総数に対する比率
Pig	3,152	0.85
Calcite	3,090	0.69
Struts	5,141	0.85
Accumulo	8,882	0.83
Tomcat	19,646	0.87
OpenLDAP	13,512	0.58
Eclipse JDT	<u>2,488</u>	<u>0.10</u>
OpenSSL	24,153	0.89
Camel	44,739	0.92
GCC	126,625	0.70
gecko-dev	277,473	0.38
Linux	773,946	0.81

の観点でも、一番高い Camel の 92% と一番低い Eclipse JDT の 10% で差が確認できる。これらから、推薦精度を向上させるにはプロジェクト単位で解析範囲を切り替えるべきである。

RQ1 に対する回答: 定説の固定解析長に従う場合は優れた変更推薦ができるとはいえ、プロジェクト単位に解析範囲を切り替えると推薦精度が向上する。

4.6 実験結果 (RQ2) : コミット単位の変換効果

図 2 に、コミット単位の最適解析長と表 2 に示すプロジェクト単位の最適解析長の差分の分布を示す。コミット単位の最適長は複数存在する場合があるため、プロジェクト単位の最適長と最近傍の長さを選択して差分を求める。プロジェクト単位の最適長の指定により、図 2 で 0 より高い値はコミット単位より長く、0 未満はコミット単位の最適長に満たないことを意味する。図 2 では差分が同一の値に密集せず全体的にばらつくことが確認できるため、コミットごとに最適長は変化するといえる。

続いて、プロジェクト単位およびコミット単位の最適長を設定した場合における評価対象コミットごとの f_{mrr_i} の分布を図 4 に示す。すべてのプロジェクトで共通して、コミット単位の結果 (右の箱ひげ図) はプロジェクト単位の結果 (中央の箱ひげ図) より全体的に高い値を指す。このことから、高い精度で変更推薦するにはコミット単位で解析範囲を切り替える必要がある。

RQ2 に対する回答: 同一プロジェクトで解析範囲を固定にしてルール抽出を行うよりも、コミット単位で解析範囲を切り替えることで推薦精度が向上する。

4.7 実験結果 (RQ3) : 固定長と提案手法の精度比較

3 種の固定解析長と本手法を適用したとき、評価対象コミットごとに求まる f_{mrr_i} の分布を図 3 に示す。図 3 の各

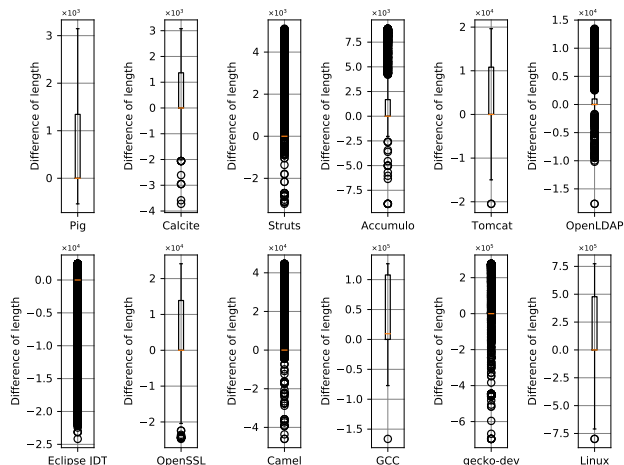


図 2 プロジェクト単位およびコミット単位の最適解析長さの分布

箱ひげ図では f_{mrr_i} の平均値を記している。その値に着目すると OpenSSL を除くプロジェクトで提案手法の推薦精度は固定解析長より優れている。Pig や Calcite などの小規模プロジェクトではそれぞれの結果に大きな差が確認できない。一方で、Linux や gecko-dev などの大規模プロジェクトでは、第一四分位数や中央値の観点でも提案手法の結果の方が高い。

続いて、Wilcoxon の符号順位検定と効果量の解釈 [21] を表 3 に示す。統計検定の結果、どの固定解析長の結果と比較しても 7 のプロジェクトで統計的に有意差が認められており、提案手法を適用した方が推薦精度が優れるといえる。しかし、Cliff の δ の示す効果量の解釈では、多くの場合で差は無視可能な程度にとどまる。大規模プロジェクトである Linux においては、どの固定長の結果とも無視できないほどの差で提案手法が統計的に有意に優れていることを確認した。

したがって、平均 f_{mrr_i} の観点で提案手法は先行研究の固定解析長の適用時における精度を上回る傾向にある。また、小規模プロジェクトで大きな差は確認されないが、大規模プロジェクトで劇的に精度が向上している。

RQ3 に対する回答: 定説の固定解析長に従って解析するよりも、提案手法で動的に定まる解析範囲を適用することによって推薦精度が向上する。

4.8 実験結果 (RQ4) : 最適長と提案手法の精度比較

図 4 にプロジェクト単位およびコミット単位の最適解析長と本手法の適用における評価対象コミットごとの f_{mrr_i} の分布を示す。各箱ひげ図に f_{mrr_i} の平均値も記す。平均値や四分位範囲を比較すると、本手法とプロジェクト単位の結果に大きな差が確認できない。また、本手法とコミット単位の結果を比較すると、後者の方が大きく優れる。そもそもコミット単位の最適解析長の適用で求まる評価尺度

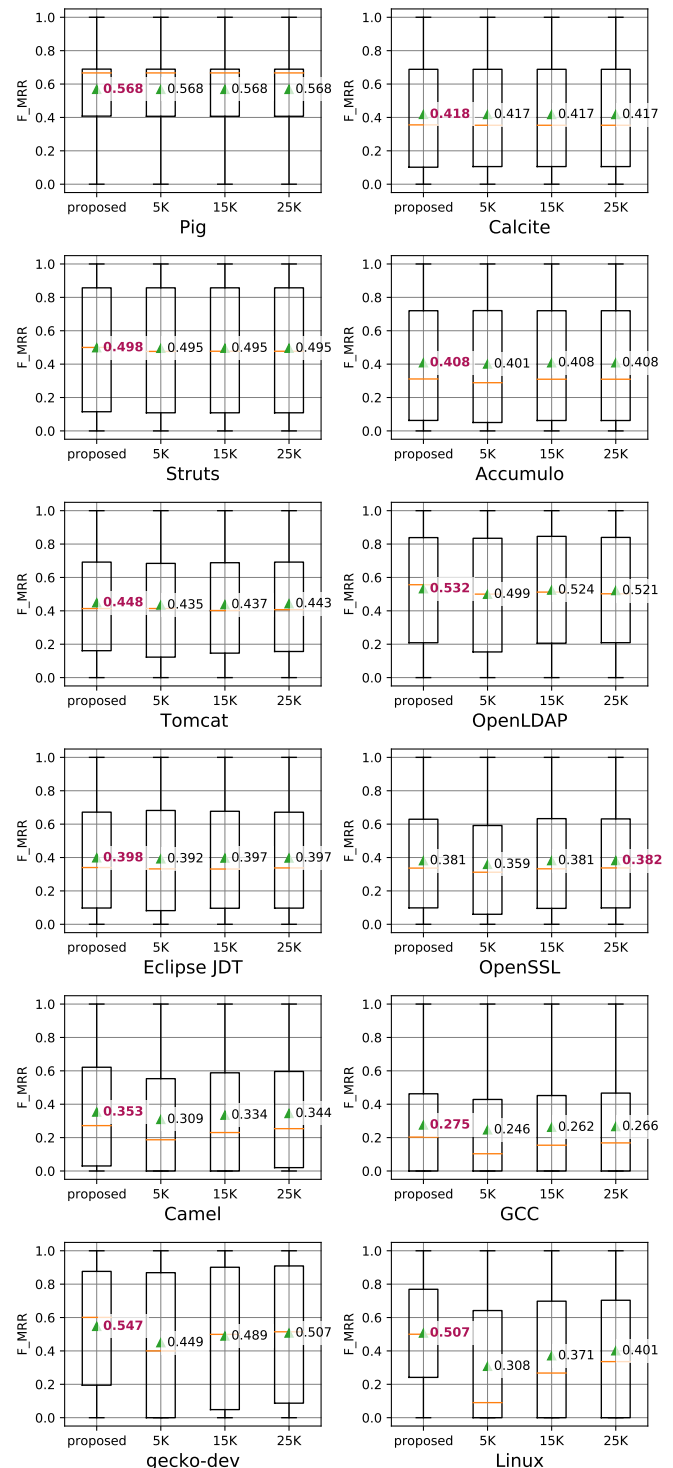


図 3 提案手法と 3 種の固定解析長における f_{mrr_i} の分布

は本実験設定における理論値を指す。そのため、本手法は理論値には到達していないことを意味する。

表 4 に Wilcoxon の符号順位検定と効果量の解釈を示す。提案手法はプロジェクト単位の最適長における結果より統計的に有意に優れているとはいえない。効果量の観点では、どのプロジェクトにおいても無視できるほどの差しかないことを意味する。一方で、コミット単位の最適解析長における結果とは大きな差が生じる。したがって、本手

表 3 提案手法と 3 種の固定解析長の f_{mrr_i} 間における統計検定結果^aと効果量の解釈^b

プロジェクト	5K	15K	25K	プロジェクト	5K	15K	25K	プロジェクト	5K	15K	25K
Pig	○ N	○ N	○ N	Tomcat	○ N	○ N	* S	Camel	* N	* N	○ N
Calcite	* N	* N	* N	OpenLDAP	* N	* N	* S	GCC	* N	* N	○ N
Struts	○ N	○ N	○ N	Eclipse JDT	○ N	○ S	* N	gecko-dev	* S	* N	* N
Accumulo	○ N	* N	* N	OpenSSL	* N	○ N	○ N	Linux	* M	* S	* S

^a 統計的有意性 ○: $p \geq 0.05$, *: $p < 0.05$

^b 効果量 N(無視可能): $|\delta| \leq 0.147$, S(小): $|\delta| \leq 0.330$, M(中): $|\delta| \leq 0.474$, L(大): $|\delta| > 0.474$

表 4 提案手法とプロジェクト単位 (RQ1) およびコミット単位 (RQ2) の最適解析長の f_{mrr_i} 間における統計検定結果^aと効果量の解釈^b

プロジェクト	RQ1	RQ2	プロジェクト	RQ1	RQ2	プロジェクト	RQ1	RQ2	プロジェクト	RQ1	RQ2
Pig	○ N	○ L	Accumulo	○ N	○ L	Eclipse JDT	○ N	○ L	GCC	○ N	○ L
Calcite	○ N	○ L	Tomcat	○ N	○ L	OpenSSL	○ N	○ L	gecko-dev	○ N	○ L
Struts	○ N	○ L	OpenLDAP	○ N	○ L	Camel	○ N	○ L	Linux	○ N	○ L

^a 統計的有意性 ○: $p \geq 0.05$, *: $p < 0.05$

^b 効果量 N(無視可能): $|\delta| \leq 0.147$, S(小): $|\delta| \leq 0.330$, M(中): $|\delta| \leq 0.474$, L(大): $|\delta| > 0.474$

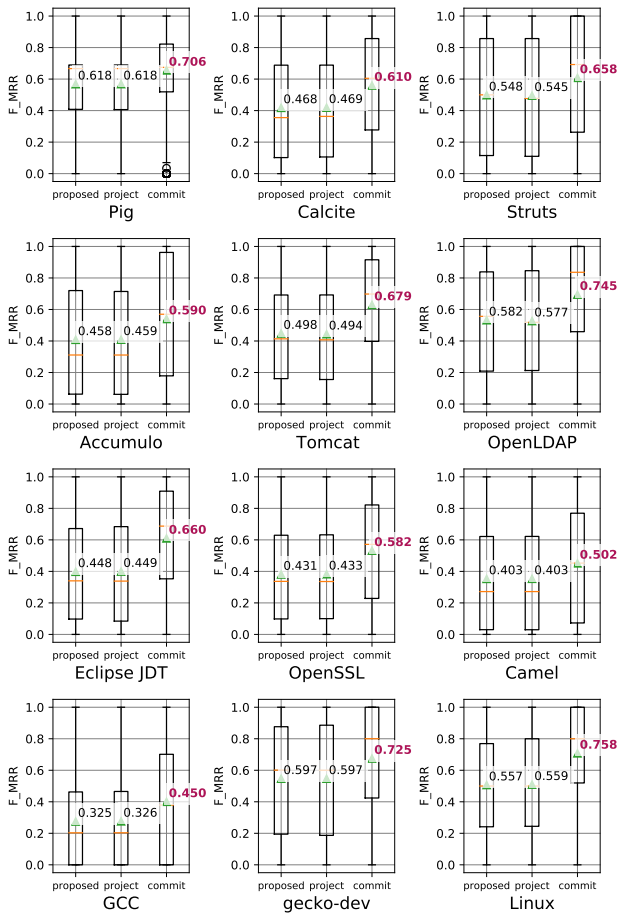


図 4 提案手法とプロジェクト単位およびコミット単位の最適解析長における f_{mrr_i} の分布

法の推薦精度は理論値には到達していないが、本来未知の長さであるプロジェクト単位の最適解析長を適用したときと同程度の推薦精度を発揮する。

RQ4 に対する回答: 提案手法はプロジェクト単位の最適解析長を設定したときと同程度の精度で変更推薦できる。プロジェクト単位の最適長は RQ1 の調査で明らかとなり、同程度の性能を発揮する提案手法には有効性がある。

4.9 考察

先行研究では、プロジェクトの差異を超えてあらゆる解析で共通の範囲を用いる手法がとられてきた。しかし、実際にはプロジェクト単位で最適解析長は大きく異なる。RQ3 で用いた先行研究の固定解析長と比べ、小規模プロジェクトでは短く、大規模プロジェクトでは長い範囲が最適となる。また、プロジェクト単位の最適長で解析しても過学習や学習不足が起これ、推薦精度への影響は無視できない。そのため、コミットごとに解析範囲を切り替える意義がある。本手法はその強みを具えており、プロジェクト固有の最適長を設定したときと同程度の精度で変更推薦が可能である。

さらに、プロジェクト単位の最適解析長は RQ1 の調査より明らかになる。ただし、ファイル間の依存関係の経年変化にともない、最適長も変化することが考えられる。つまり、最適長が変化したとしても、仮想欠陥によって正解が既知である場合に求まる最適長の適用時と同程度の精度を発揮する本手法には有効性がある。今後の研究として、最小確信度と最小支持度により価値の高い変更ルールを重視する解析においても本手法が有効であるか調査する。

4.10 妥当性への脅威

本手法の実装と、プロジェクト単位の最適な解析範囲の

特定方法に関して内的妥当性への脅威が存在する。解析範囲を指定してルール抽出を行う既存手法や提案手法のアルゴリズムの実装において、実験結果に影響を与える欠陥を含んでいる可能性がある。RQ1の実験では、実験対象プロジェクト固有の最適解析長を特定するため、そのプロジェクトにおける総コミット数の1/1000単位を順次加算させ、平均 f_{mrr_i} が最大となる解析範囲を探索した。そのため、さらに細かい粒度で調査を行えば、最適長は異なる長さを指す可能性がある。しかし、本稿では紙面の都合上割愛したが、解析範囲が長くなるにつれて平均 f_{mrr_i} は大きく変動しないことを確認した。したがって、総コミット数の1/1000単位で解析対象コミット数を変化させて特定した最適解析長に大きな誤差は生じておらず、それをういたRQ2とRQ4の結果の妥当性に影響を与えないと考える。

実験結果の一般性に関して外的妥当性への脅威が存在する。本実験では、12のプロジェクトを選択した。これらは、総コミット数や開発期間、主に使用される開発言語(Java, C/C++など)、ドメイン(CLIアプリケーション, Webアプリケーションフレームワークなど)に違いがあり、多種のプロジェクトにより本手法の有効性を主張する。しかし、さらに一般性を高めるにはより多くのプロジェクトを用いた調査が必要と考える。また、プロプライエタリなプロジェクトに対して提案手法を適用した場合、本実験結果とは異なる傾向が観測される可能性がある。

5. おわりに

変更ルールの抽出における解析範囲の定説では、改版履歴全体ではなく直近の一定数のコミットを解析することで有益なルールを検出できるとされる。しかし、先行研究で示される解析範囲はあらゆるプロジェクトで共通する解析長を前提として実証的に決定づけられている問題がある。本研究では、コミットごとに解析範囲を動的に切り替えることで変更推薦の精度は向上すると考え、手法を提案した。

我々は12のOSSに対して実験を行った。まず、プロジェクト単位やコミット単位に解析範囲を可変にすることで、固定の場合よりも推薦精度が向上することを実証した。そして、本手法を適用することで、先行研究で示された固定長を指定したときより推薦精度が優れる傾向にあることを示した。さらに、本手法の適用で、実際は未知数であるプロジェクト単位の最適な固定長を設定した場合と同程度の精度で変更推薦できることを明らかにした。

謝辞 本研究の一部は科研費(#18H03221)の助成を受けた。

参考文献

- [1] L. C. Briand, et al. Using coupling measurement for impact analysis in object-oriented systems. In *Proc. ICSM*, pp. 475–482, 1999.
- [2] M. M. Geipel and F. Schweitzer. Software change dynamics: Evidence from 35 java projects. In *Proc. FSE*, pp. 269–272, 2009.
- [3] A. R. Yazdanshenas and L. Moonen. Crossing the boundaries while analyzing heterogeneous component-based software systems. In *Proc. ICSM*, pp. 193–202, 2011.
- [4] T. Zimmermann, et al. Mining version histories to guide software changes. *IEEE TSE*, Vol. 31, No. 6, pp. 429–445, 2005.
- [5] G. A. Oliva and M. A. Gerosa. On the interplay between structural and logical dependencies in open-source software. In *Proc. SBES*, pp. 144–153, 2011.
- [6] L. Moonen, et al. Practical guidelines for change recommendation using association rule mining. In *Proc. ASE*, pp. 732–743, 2016.
- [7] T. Rolfesnes, et al. Generalizing the analysis of evolutionary coupling for software change impact analysis. In *Proc. SANER*, pp. 201–212, 2016.
- [8] L. Moonen, et al. Exploring the effects of history length and age on mining software change impact. In *Proc. SCAM*, pp. 207–216, 2016.
- [9] L. Moonen, et al. What are the effects of history length and age on mining software change impact? *Empirical Software Engineering*, Vol. 23, No. 4, pp. 2362–2397, 2018.
- [10] 森ほか. 改版履歴の分析に基づく変更支援手法における時間的近接性の考慮と同一作業コミットの統合による影響. 情報処理学会論文誌, Vol. 58, No. 4, pp. 807–817, 2017.
- [11] R. Srikant, et al. Mining association rules with item constraints. In *Proc. KDD*, pp. 67–73, 1997.
- [12] A. Alali, et al. A preliminary investigation of using age and distance measures in the detection of evolutionary couplings. In *Proc. MSR*, pp. 169–172, 2013.
- [13] H. Kagdi, et al. Mining sequences of changed-files from version histories. In *Proc. MSR*, pp. 47–53, 2006.
- [14] M. A. Islam, et al. Detecting evolutionary coupling using transitive association rules. In *Proc. SCAM*, pp. 113–122, 2018.
- [15] M. Mondal, et al. Historank: History-based ranking of co-change candidates. In *Proc. SANER*, pp. 240–250, 2020.
- [16] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, Vol. 114, No. 3, pp. 494–509, 1993.
- [17] 石田, 小林. 改版履歴分析に基づく変更漏れ防止支援における変更ルール集約と順位付けの効果. 信学技報, Vol. 118, No. 471, pp. 79–84, 2019.
- [18] A. T. Ying, et al. Predicting source code changes by mining change history. *IEEE TSE*, Vol. 30, No. 9, pp. 574–586, 2004.
- [19] A. Alali, et al. What’s a typical commit? a characterization of open source software repositories. In *Proc. ICPC*, pp. 182–191, 2008.
- [20] H. Kagdi, et al. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering*, Vol. 18, No. 5, pp. 933–969, 2012.
- [21] J. Romano, et al. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen’s d for evaluating group differences on the nsse and other surveys? *Annual meeting of the Florida Association of Institutional Research*, Vol. 177, pp. 1–33, 2006.