Vol.2021-SE-207 No.24
2021/3/2

共変更に基づく変更推薦に対するブランチ戦略の影響分析

伊勢本 圭亮[1,a)]　小林 隆志[1,b)]　佐伯 元司[1,c)]　林 晋平[1,d)]

概要：共変更情報を用いた変更推薦にブランチが与える影響が指摘されているが、影響につながるブランチの特徴が明らかになっていない。本論文では、ブランチの上の変更をコミットごとに扱う場合と統合して単一のマージコミットとして扱う場合で共変更情報やそれに基づく変更推薦結果を比較し、変更推薦に影響を与えるブランチの特徴を明らかにする。既存の OSS リポジトリに対して、ブランチの扱い方ごとに変更推薦を行い、複数の評価指標で比較した。また、推薦ルールの生成に使用する部分変更履歴の特徴に対する変更推薦結果の傾向とブランチ上のコミットとマージコミットの共変更情報に対する将来の変更の関係から、変更推薦ではブランチ上のコミットで個別に扱うほうが適切だったことが多かった。

1. はじめに

ソフトウェアシステムが進化するにつれ、コードの量とともにコード同士の依存関係が増えて複雑になる。複雑になると、開発者が変更を加えた結果、どこまで、どのような影響があるのか理解することは難しい。そのため、必要な変更が漏れることでバグの原因になる。

そこで、ソフトウェアシステムの進化の過程の情報に基づいて、依存関係を特定することが行われている。ソフトウェア版管理システムなどに記録されたコミットといった情報を用いることで、静的解析や動的解析では得られない依存関係をプログラミング言語に依存せずに得られる。

そのアプローチの一つとして、共変更情報に基づく変更推薦 [1, 2] がある。共変更情報は、開発履歴から同時に変更されたファイルを取得したものである。共変更情報にアソシエーション分析を適用することで、あるファイルが変更されるとある別のファイルが変更されるというルールに基づいた依存関係が抽出できる。

共変更情報に基づく変更推薦結果には様々な要素が影響することが指摘されている [3, 4]。影響の一つとして、Kovalenko らによってブランチの影響が指摘されており、ブランチの扱いを変えた 2 つの履歴戦略を比較し、共変更情報をブランチ上のコミットから個別に得るほうがマージコミットからまとめて得るよりも性能がわずかに向上すると報告している [5]。

[1] 東京工業大学情報理工学院
Tokyo Institute of Technology, Tokyo 152–8550, Japan
[a)] k_isemoto@se.cs.titech.ac.jp
[b)] tkobaya@c.titech.ac.jp
[c)] saeki@se.cs.titech.ac.jp
[d)] hayashi@c.titech.ac.jp

しかし、Kovalenko ら [5] とは違い、共変更分析では作業項目で変更をまとめるべきとした調査 [6] があり、ブランチの適切な扱いが明らかになっていない。Miura らは共変更情報の比較を行い、イシューとの関連から同一作業と特定したコミットの変更をまとめないと共変更情報に欠落が起きるため、まとめて扱うべきとしている [6]。ブランチは、主にバグ修正や機能追加などのタスクのために使われており [7]、ブランチ上の変更は同一作業に近いと考えられる。このことを鑑みると両調査のと結果は異なっており、ブランチの適切な扱いが明確はでない。

また、ブランチはソフトウェア開発の重要な要素の一つであり、マイニング手法を実際のリポジトリに適用するために、ブランチの影響を調べることは重要である。現在、版管理システムとして Git が広く利用されている。Git を用いるプロジェクトでは、ブランチを利用した変更管理が広く行われている [8]。Git のブランチは簡単に作ることができ、バグの修正や機能の追加などのタスクのために使われる [7] こともあれば、リリースごとの管理のために使われることもある。ブランチの使い方によってコミットの粒度も変わり、変更推薦にも影響するため、ブランチは状況に応じて適切に扱う必要がある。また、どのようなブランチがどのような影響を与えるか調べることは、提案されたマイニング手法を実際の開発に導入する際に役に立つ。

本論文では、変更推薦での適切なブランチの扱いを特定するため、次の Research Question（RQ）を設定する。

- RQ_1: 変更推薦ではブランチ上のコミットを個別に扱うほうがわずかに性能が高くなるという Kovalenko らの結果に実験設定の影響はないのか
- RQ_2: どのようなブランチが変更推薦に影響を与えるか

- $RQ_{2.1}$: ブランチの特徴に対して変更推薦結果にどのような関係があるか
- $RQ_{2.2}$: ブランチの特徴に対してブランチ上のコミットとマージコミットの共変更情報にどのような関係があるか

RQ_1 では、Kovalenko らと同様に 2 つの履歴戦略での変更推薦の比較で、Kovalenko らの結果につながった実験設定の影響を調べる。Kovalenko ら [5] が使用したリポジトリより多くのリポジトリに対して、Kovalenko らの実装 [9] を再現した実装とマージコミットの扱いを変更した実装でブランチの扱いを変えた履歴戦略間の結果の比較し、Kovalenko ら [5] らよりも多くの評価指標に基づいた検証を通じて分析した。

RQ_2 では、ブランチがどのような特徴を持つときに変更推薦に影響を与えるかを調べる。マージコミットの扱いを変更した実装でも変更の影響は限定的で、Kovalenko ら [5] の結果からもブランチの扱いを変えたときの差が小さいことが想定される。しかし、ブランチにも違いがあるため、ブランチの扱いを一律に定められない可能性がある。そこで、ブランチの長さやマージコミットの大きさといったブランチの特徴ごとに変更推薦にどういった影響を与えるかを調べる。$RQ_{2.1}$ では、ブランチの特徴に対して履歴戦略を変えた推薦結果の差の分析を行った。$RQ_{2.1}$ の分析は変更推薦手法の設定の影響を受ける可能性があるため、ブランチの特徴に対してブランチ上のコミットとマージコミットの共変更情報の差の分析を $RQ_{2.2}$ で行った。

2. Kovalenko らの調査とその問題

Kovalenko ら [5] は、ファイル履歴を用いるマイニング手法を利用する際のブランチの適切な扱いの重要性を評価するために、変更推薦を含む 3 つのマイニング手法に対し、ブランチの扱いを変えた First-Parent と Full という履歴戦略を適用した結果の比較した。First-Parent では一番目の親コミットを辿ることで、Full では全ての親コミットを辿ることで到達可能なコミットを対象にする。Full ではマージコミットはブランチ上の変更をまとめたものとして、除外する。一方、First-Parent では、ブランチ上のコミットで行われた変更はマージコミットとしてまとめて扱われる。Kovalenko ら [5] はこの 2 つの履歴戦略を比較し、Full のほうがわずかに推薦性能が高くなることを示した。

Kovalenko らの実装は公開されている [9] が、その実装には議論すべき箇所がある。その実装では First-Parent でコンフリクトなどの例外を除いてマージコミットの変更が無視されており、ブランチ上の変更の多くが欠落している。図 1 に各履歴戦略を示す。色が濃くなっているコミットが各履歴戦略の対象になる。Full では、ブランチ上のコミットが個別に扱われる。マージあり First-Parent では、ブランチ上の変更をマージコミットでまとめる。Kovalenko らの実装では、マージコミットを一部の例外を除き対象から外すマージなし First-Parent になる。Full の性能がわずかに高くなる傾向にあったのは、マージなし First-Parent によりブランチ上の変更が省略されたことが影響している可能性がある。

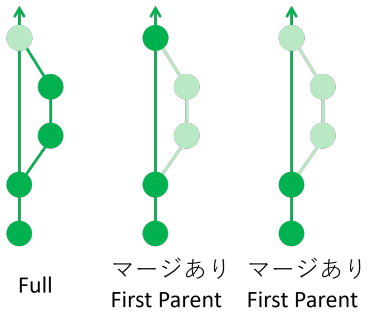

図 1 各履歴戦略で対象になるコミット

そのため、RQ_1 では Full とマージなし First-Parent、Full とマージあり First-Parent 間で変更推薦の性能を比較する。2 つの組合せでの実験を通じて、Kovalenko らの実装 [9] でブランチ上の変更が欠落した影響を確認する。Full とマージなし First-Parent でも実験を行うことで、Kovalenko ら [5] とほぼ同じ条件の実験ができ、平等な検証につながる。また、RQ_2 では、First-Parent でブランチ上の変更が欠落することは、ブランチの扱いの比較として不適切とみなし、Full とマージあり First-Parent 間の推薦結果を対象に行う。

3. 変更推薦手法

変更推薦は、推薦の基準とするファイルの集合（クエリ）を与えることで、順位付きの推薦リストを返す。本論文では、Kovalenko らが公開している実装 [9] の変更推薦手法の 2 つのアルゴリズムのうち、OTHERS に基づいて作成した。

変更推薦は次の 3 つのステップで行われる。

(1) 推薦ルール作成に使用するコミットの抽出
(2) 抽出されたコミットの共変更にアプリオリアルゴリズムを適用し、推薦ルールを作成
(3) 推薦ルールから推薦リストを作成

まず、推薦ルール作成に使用するコミットの抽出では、各履歴戦略の対象になるコミットから、クエリに含まれるファイルが変更されているコミットを抽出する。推薦を行うコミットを起点に、各履歴戦略の対象になるコミットから、少なくとも 1 つクエリのファイルが含まれるコミットを抽出する。ただし、大きな変更を持つコミットでは意味の関係性を持たないとして、コミットで行われたファイルへの変更数が 10 を超えるものを除外する。こうして得たコミットから、直近のものから最大 100 を抽出する。

次に抽出されたコミットそれぞれで変更されたファイルをトランザクションとして、Apriori アルゴリズムを適用して推薦ルールを作成する。Apriori アルゴリズムでは、ト

表 1　対象リポジトリ

リポジトリ集合	リポジトリ数	平均コミット数
$Apache_K$	10	3,955.6
$Eclipse_K$	8	15,582.9
$Apache_M$	12	2,829.6

ランザクションとともに最低の *Confidence*（*minconf*）と *Support*（*minsup*）を与えることで、その条件を満たした相関ルールを作成できる。Kovalenko らと同様に、*minconf* と *minsup* 共に 0.1 とする。作成した相関ルールから結論部のファイルが 1 つで、*Support* が高い順に 10 を推薦ルールとして採用する。

最後に推薦ルールにクエリを適用し、推薦を行う。推薦ルールの条件部のすべてのファイルがクエリに含まれる場合、そのルールの結論部が推薦結果として採用される。推薦の順位は、採用されたルールの *Support* が高い順につく。

この変更推薦手法では、推薦ルールを絞り込んでから推薦を行うため、推薦される数が少なくなる。条件部が全てクエリの変更に含まれる推薦ルールに対して、絞り込みを行うのではなく、生成された推薦ルールに絞り込みを行う。条件部が全てクエリの変更に含まれないルールを含む 10 から推薦を行うため、高々 10 と少数の推薦を行う。

4. 実証的調査

4.1 対象のリポジトリ

本論文では、Kovalenko ら [5] が使用した Apache（$Apache_K$）と Eclipse（$Eclipse_K$）、Miura ら [6] が使用した Apache（$Apache_M$）の計 30 リポジトリを対象に調査を行う。表 1 に対象リポジトリの情報を示す。クローンに失敗した Eclipse の 2 リポジトリとブランチの影響がなかった Apache の 2 リポジトリを対象から外した。

4.2 RQ_1: 変更推薦ではブランチ上のコミットを個別に扱うほうがわずかに性能が高くなるという Kovalenko らの結果に実験設定の影響はないのか

4.2.1 動機

Kovalenko ら [5] は 2 つの履歴戦略で変更推薦の比較を行い、Full 戦略がわずかに高い性能となることを示した。一方で Miura ら [6] によると、タスク単位で変更をまとめると共変更情報の欠落を防げるため、共変更分析では共変更情報をタスク単位で扱うべきとしている。

1 つのブランチで行っている変更が一つのタスクであるとみなすと、Kovalenko らの Full 戦略でブランチ上の変更を個別に扱うと性能が高くなることは、Miura らの結果と異なる。ブランチは主に Feature ブランチとして使われている [7] ことから、ブランチ上の変更をタスクとみなすと、タスク単位でまとめることとブランチ上の変更をマージでまとめることは近い。Miura ら [6] が共変更の欠落が防げるとする一方、Kovalenko らの結果はブランチ上のコミットを個別に扱う Full 戦略の推薦成功率と推薦数が共に高い。この違いは実験に使用しているリポジトリが異なることが原因の可能性がある。そこで、RQ_1 で Kovalenko ら [5] と Miura ら [6] の研究で使用されたリポジトリのグループで比較し、実験で選択したリポジトリの影響を調べる。

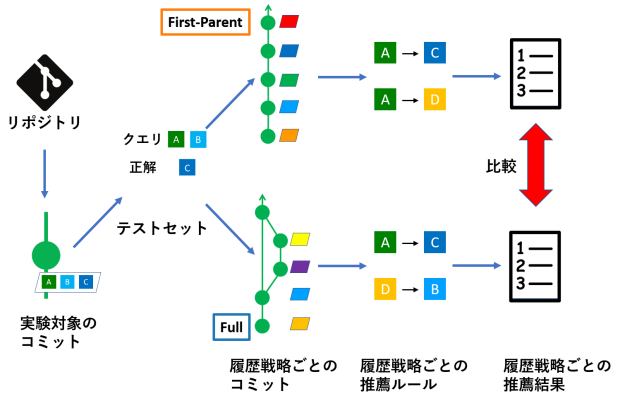

図 2　変更推薦を履歴戦略ごとに比較する全体の流れ

また、Kovalenko ら [5] は成功率と推薦数のみで比較をしており、検証として不十分である。成功率と推薦数だけでは、推薦数を増加させて誤推薦が増えたとしても、推薦の成功率と推薦数は高くなってしまう。そこで、他の変更推薦の研究で使用されている Mean Average Precision（MAP）[1,3,4,10] と勝敗数 [10] での性能の比較も行う。

Kovalenko らの実装 [9] はマージなし First-Parent で多くのブランチ上の変更を省略しており、その影響で Full の性能がわずかに高くなっていた可能性がある。共変更情報の省略が原因で Full 戦略の性能が高くなっていたとすると、マージあり First-Parent と比較すると Full のほうが性能が良くなるとは限らない。ただ、マージコミットの変更をコンフリクトなどのみとしたことは、ブランチのコミットを取り込んだ変更はマージコミットで行った変更ではないとする意図的な仕様の可能性もある。そこで、Kovalenko らの実装 [9] に近づけた実装と履歴戦略をマージなし First-Parent からマージあり First-Parent に変更とその他の変更も加えた実装での実験を行う。

4.2.2 調査方法

Full とマージなし First-Parent（Imp_{no_merge}）、Full とマージあり First-Parent（Imp_{merge}）、それぞれの履歴戦略間での推薦結果を比較する。実験の流れを図 2 に示す。リポジトリごとに実験対象のコミットごとに、変更のうち 1 ファイルを正解に、その他をクエリに分割してテストセットを作成する。作成したテストセットで、履歴戦略ごとに変更推薦を行う。最後にその結果を比較し、集計する。

実験対象は、次の Kovalenko ら [5] と同様の以下の条件を満たすコミットと正解の変更になる。

- 10 以下のファイルを変更している
- 比較を意味のあるものにするため、履歴戦略ごとに取得する履歴が異なる [5]

- 意味のある規則を学習できるよう、両方の履歴戦略で 5 以上のコミットから共変更情報を取得している [5]
- 両方の履歴戦略で 1 以上の推薦ルールが生成されている [5]

Imp_{no_merge} と Imp_{merge} では、履歴戦略以外にも 3 つの処理が異なる。

- 推薦ルール生成の対象となるコミットの制限が異なる。Imp_{no_merge} では、クエリの変更を含み、10 以下のファイルへの変更を持つコミットから、authoredDate が新しい順に最大 100 コミットを対象にする。Imp_{merge} では、クエリの変更ごとに Git log コマンドでコミットグラフに従って取得した各最大 100 コミットから、10 以下のファイルへの変更を持つコミットに絞り込んで対象にしている。Imp_{no_merge} では 100 になるか、クエリの変更を持つコミットがなくなるまでコミットを取得するが、Imp_{merge} ではクエリのファイルごとに最大 100 コミット取得し、その中から 10 以下の変更を持つ対象コミットを抽出する。大量のテストセットでの実験を行うため、Imp_{merge} での処理が早くなるようにこの変更を加えた。
- 対象リポジトリが異なる。Kovalenko らの実装 [9] ではコミット数が 10000 より大きいリポジトリを対象外としている。Imp_{no_merge} では、その実装 [9] で計算可能なリポジトリのみを対象にしている。Imp_{no_merge} と Kovalenko らの実装 [9] との差異を確認するために対象を揃えた。
- 実際の変更推薦での設定に合わせるため、Kovalenko ら [5] が行っていた同一テストセットの履歴戦略間で推薦ルール生成に使用するコミットの数と最大 10 抽出した推薦ルールの数を少ない履歴戦略に揃える処理を Imp_{merge} では行わない。変更推薦をする際に行われない処理であるが、Imp_{no_merge} では Kovalenko らの実験の再現を重視した。

こうして得られた推薦結果を成功率、MAP、勝敗数 [10] で比較する。

- 成功率 [5] は、推薦結果を成功、失敗、無推薦に分け、履歴戦略ごとに各割合を集計する。成功率に加え、失敗率、無推薦率も算出される。なお、Kovalenko ら [5] は、クエリに含まれるファイルのみ推薦した場合、失敗と判定していたが、本論文では無推薦と判定する。
- MAP [10] は、常に推薦すべきとして無推薦結果の Average Precision（AP）を 0 と厳しく扱った MAP_{all} と無推薦結果を除外して評価する MAP_{app} の 2 種類を用いる。
- 勝敗数 [10] は、AP が高いか、AP が 0 同士のときに誤推薦をしなかった履歴戦略側を勝利とし、テストセットごとに勝敗判定を行って集計する。

表 2　リポジトリグループごとの成功率

	リポジトリ (テスト数)	履歴戦略	平均 推薦数	平均 ルール数	成功	割合 失敗	無推薦
Imp_{no_merge}	$Apache_K$ (25,588)	Full	**1.005**	7.763	**0.179**	0.438	0.383
		First-Parent	0.954	7.763	0.167	0.428	0.405
	$Eclipse_K$ (4,405)	Full	0.775	6.862	**0.156**	0.372	0.472
		First-Parent	**0.788**	6.862	0.148	0.383	0.470
	$Apache_M$ (7,063)	Full	**0.726**	6.491	**0.104**	0.395	0.501
		First-Parent	0.661	6.491	0.099	0.359	0.542
Imp_{merge}	$Apache_K$ (33,728)	Full	0.787	6.808	0.143	0.350	0.507
		First-Parent	**0.823**	7.172	**0.147**	0.365	0.488
	$Eclipse_K$ (5,994)	Full	**0.740**	6.517	**0.150**	0.350	0.499
		First-Parent	0.739	6.792	0.148	0.349	0.503
	$Apache_M$ (105,807)	Full	0.739	5.391	**0.165**	0.319	0.515
		First-Parent	**0.745**	5.516	0.163	0.325	0.512

表 3　全体での集計結果

実装	履歴戦略	成功率	MAP_{all}	MAP_{app}	勝数	引分け
Imp_{no_merge}	Full	0.162	0.144	0.247	3,214	36,708
	First-Parent	0.152	0.135	0.241	3,407	
Imp_{merge}	Full	0.160	0.143	0.294	7,848	130,809
	First-Parent	0.159	0.143	0.289	6,872	

表 4　高い性能になったリポジトリの数

実装	指標	Full	EVEN	First Parent
Imp_{no_merge}	成功率	15	7	2
	MAP_{all}	8	16	0
	勝敗数	13	2	9
Imp_{merge}	成功率	15	3	12
	MAP_{all}	5	20	5
	勝敗数	16	1	13

4.2.3　結果

リポジトリの影響．Kovalenko ら [5] と Miura ら [6] らの結果の違いに対象リポジトリの影響の有無を確認するため、Kovalenko ら [5] と同様のフォーマットでリポジトリグループごとの推薦結果を表 2 に示す。Imp_{no_merge} では、$Eclipse_K$ での推薦数を除いて全てで、推薦数、成功率共に Full が高かった。Imp_{no_merge} の結果は $Apache_M$ でも傾向が変わらず、2 つの結果の違いはリポジトリの影響によるものではなかったと考える。

検証指標の影響．検証指標の影響を確認するため、全リポジトリでの成功率、MAP_{all}、MAP_{app}、勝敗数の結果を表 3 に示す。一番右の 2 列は、各履歴戦略の勝数と引き分けの数を示している。Imp_{no_merge} では、勝敗で First-Parent が勝つことが多いことを除いて、Full のほうが高い性能になった。Imp_{no_merge} の First-Parent が勝数が多いのは、Full で推薦数が増加と共に誤推薦数も増加したことが原因である。これは、First-Parent でブランチ上の変更の欠落で推薦のバリエーションが減り、推薦数が減少した影響と考える。

成功率、MAP、勝敗数の観点で高くなったリポジトリの数の集計結果を表 4 を示す。成功率は高い値になった履歴戦略のリポジトリの数と同値になった場合は EVEN に集計する。MAP_{all} は、ウィルコクソンの符号順位検定で有意水準 5%で有意に差がついたリポジトリで高くなった場合、

高くなった履歴戦略ごとに集計し、有意でなかった場合、EVEN に集計する。MAP_{app} は有意な差があったリポジトリは、Imp_{merge} で Full が高かったもので 1 つだけだったため、表 4 では省略した。勝敗数は勝った数の多かった履歴戦略に集計し、勝ちが同数だった場合、EVEN に集計する。Imp_{no_merge} では、成功率、MAP_{all} では、Full の性能が高くなる傾向があった。勝敗数も、Full のほうが勝ったリポジトリが多かった。

Imp_{no_merge} の結果から、Full で誤推薦が増えていたことが分かったものの Full のほうが性能が高くなる傾向があり、検証指標の影響はなかった。

ブランチの情報が省略された影響. 各表の Imp_{no_merge} と Imp_{merge} の結果を比較し、ブランチの情報が省略された影響を確認する。どの結果でも、高い性能になる履歴戦略が均衡する傾向になった。表 2 の結果では、リポジトリグループによって、高い性能になる履歴戦略が異なった。$Apache_K$ では First-Parent のほうが推薦数、成功率共に高いが、$Eclipse_K$ では Full のほうが推薦数、成功率共に高かった。$Apache_M$ では推薦数は First-Parent が多く、成功率は Full が高かった。また、その差の大きさは Imp_{no_merge} に比べ、小さくなった。表 3 では、MAP_{all} で履歴戦略間の差が小さくなり、勝敗数でも引き分けの割合が 84.7%（36,708/43,329）から 89.9%（130,809/145,529）と増加した。表 4 でも、Full が高い性能になったリポジトリ数と First-Parent が高い性能になったリポジトリ数が Imp_{no_merge} よりも均衡した。

変更推薦ではブランチ上のコミットを個別に扱うほうがわずかに性能が高くなるという Kovalenko ら [5] の結果は、ブランチ上の変更情報の欠落がもたらした可能性があると考えられる。

> 実験設定の影響の可能性がある。Miura らの結果の違いから、対象リポジトリと検証指標の影響を確認したが、Kovalenko らと同様に Full の性能が高くなる傾向だった。しかし、マージの変更が First-Parent で欠落していたことを修正した Imp_{merge} では傾向が均衡し、Kovalenko らの結果はブランチ上の変更情報の欠落がもたらした可能性があると考えられる。

4.3 RQ_2: どのようなブランチが変更推薦に影響を与えるか

4.3.1 動機

ブランチには様々な使い方があり、リポジトリによって使い方が異なる。その多くの使い方は機能の作成やバグの修正 [7] だが、リリースやバージョンごとの管理にも使われる。例えば、本論文の対象リポジトリの apache/accumulo や apache/cassandra ではバージョンごとにブランチを作り、開発を並列して行っている。また、機能の作成にブランチを使ったとしても、作成頻度や粒度の違いがある。

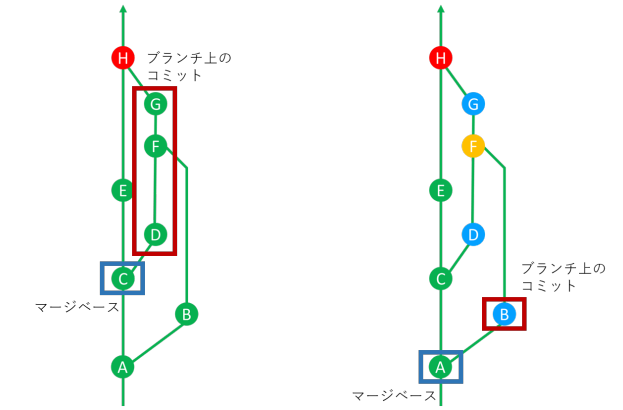

図 3 ブランチの長さ算出の流れ

ブランチごとの特徴によって、変更推薦に適切なブランチの扱い方が異なることが考えられる。例えば、ブランチ上で変更を細かく分けている場合はそれらをマージコミットとして扱うほうが良いかもしれないし、大きな変更を複数回ブランチ上で行ってマージしている場合はブランチ上のコミットごとに扱うほうが良いかもしれない。

そこで RQ_2 では、ブランチの長さとマージコミットの大きさという 2 つの特徴が変更推薦にどのような影響を与えるのかを調べる。

- ブランチの長さとは、マージコミットでマージした各ブランチの共通の親（マージベース）に至るまでのマージしたブランチ上のパス中に存在するコミットの数である。ただし、パス中にマージコミットがあった場合、再帰的にコミットを辿って集計する。図 3 を用いて、マージコミット H を例にブランチの長さの算出方法を説明する。マージコミット H の親コミット E と G のマージベース C までのブランチ上のコミットを取得する。取得したブランチ上のコミットにマージコミット F が含まれていたので、F でマージしたブランチの先頭コミット B と E のマージベース A までのブランチ上のコミットを取得する。そして、取得したブランチ上のコミットからマージコミット F を、ブランチ上の変更を取り込んだだけで実質的な変更がないとして除いた 3 と算出する。
- マージコミットの大きさは、マージコミットで変更されたファイル数である。

$RQ_{2.1}$ では、ブランチの長さとマージコミットの大きさに対して、ブランチが原因で発生した推薦ルールの生成に使用する部分変更履歴の特徴ごとに変更推薦結果にどのような傾向があるのかを調べる。一方、$RQ_{2.2}$ ではブランチの特徴に対して、ブランチ上のコミットとマージコミッ

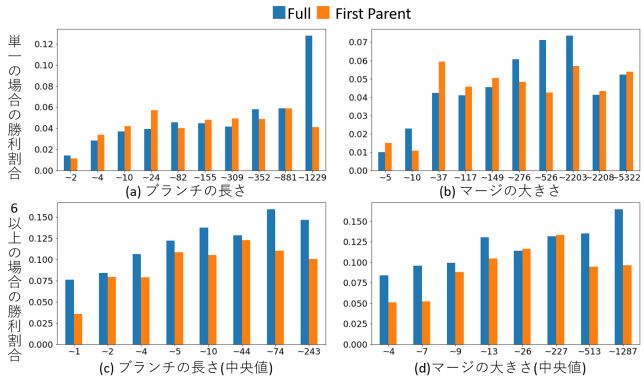

図 4 ブランチの特徴に対する履歴戦略の勝利割合

トの共変更情報がそれぞれ、将来の変更をどの程度正確に示しているのかを調べる。$RQ_{2.1}$ で変更推薦に使用するコミットの違いの原因になったブランチに注目して分析する一方、$RQ_{2.2}$ では、ブランチ上のコミットとマージコミットの共変更を直接比較する。よって、$RQ_{2.2}$ では変更推薦手法の実装の影響を受けない。

4.3.2 $RQ_{2.1}$ の調査方法

変更推薦ルール生成に関わってくるブランチの特徴に対する推薦結果を分析する。推薦ルール生成に使用するコミットは、履歴戦略を切り替えることによって、ブランチに関係するコミットが入れ替わる。入れ替わったコミットから、その原因となったマージコミットとそこで取り込まれたブランチ上のコミットを特定し、その特徴に対する推薦結果の傾向を分析する。入れ替わりの原因となったマージコミットが複数ある場合があるため、単一の場合と 6 以上の場合で比較する。2 以上とすると非常に長いブランチが含まれている場合などにその影響を大きく受け、単一の場合と近い結果になったため、分析対象のテストセットごとのコミットの入れ替わりの原因となったマージコミットの数の第三四分位数から 6 以上と設定した。

推薦結果を比較する尺度として、勝敗 [10] を利用する。関係しているブランチの特徴の値を用いて、サンプル数がなるべく均等になるように推薦結果を分割し、どちらの履歴戦略が勝つ傾向なのかを確認する。6 以上の場合では、各マージコミットのブランチの特徴の値を集約するために最大値、平均値も試したが、突出した値のマージコミットが含まれるとその影響を受けたため、中央値で分割する。

なお、ブランチの戦略の切り替えの使用コミットの変化をより際立たせるため、使用するコミット数が多くならないように限定して分析を行った。具体的にはデータセット全体での First-Parent で使用するコミット数の中央値 53 以下のものに限定して得た 71832 を対象とした。

4.3.3 $RQ_{2.1}$ の結果

ブランチの長さ．ブランチの長さとマージコミットの大きさに対する勝った履歴戦略の割合を図 4 に示す。引き分けになった場合は、省略されており、勝敗のついた割合のみ表示している。単一の場合、ブランチの長さが 10〜24 まではブランチの長さに応じて推薦結果への影響が大きくなっていく。881 より長いブランチが関連していると影響が大きくなり、そこまで 2%以下だった履歴戦略間の勝つ確率の差は、Full のほうが 8%高くなる。6 以上の場合、ブランチの長さが 5 になるまでは勝敗のついた割合が大きくなるが、そこでおおよそ頭打ちする。

ブランチの長さが長くなるとブランチの影響が大きくなり、非常に長いブランチが関係しているとブランチ上のコミットで個別に扱うほうが良い。

マージの大きさ．単一の場合、マージコミットの大きさが 10 を超えるまで、約 97%の場合で勝敗がつかず、ブランチの扱いを変える影響が小さい。しかし、10 を超えると急激に勝敗のついた割合が 3%から 10%に増える。これは、今回の実験で使用した変更推薦で 11 以上の変更を持つコミットを除いて推薦ルールを生成しているためと考える。6 以上の場合、マージコミットの大きさの中央値に対する勝敗のついた割合は増加する傾向がある。

マージコミットの大きさが大きくなるとブランチの影響が大きくなる。また、単一ブランチの場合にマージコミットの大きさが 10 を超えたときの勝敗のついた割合の急な増加は、推薦手法の影響と考えられる。

4.3.4 $RQ_{2.2}$ の調査方法

マージコミット、もしくは、そのマージコミットで取り込んだブランチ上のコミットで変更された 1 ファイルを基準とし、それと共変更されたファイルについて、ブランチ上のコミットとマージコミットで比較する。分析対象のマージコミット、そのマージコミットで取り込んだブランチ上のコミットで、基準としたファイルと共変更されたファイル（$F_{changed}$）を集計する。それに対し、マージコミットから 100 コミットで到達可能な将来のコミットで基準としたファイルと共変更されたファイル（F_{answer}）を正解として $Precision$ ($= |F_{changed} \cap F_{answer}|/|F_{changed}|$) を計算する。基準としたファイルごとに集計すると多くのファイルを含むマージコミットの影響が大きくなるので、コミットごとに平均を取り、ブランチの特徴ごとにそれぞれの分布と傾向を確認する。

対象は、デフォルトブランチ上のマージコミット、すなわち、HEAD から、1 番目の親コミットを辿ることで到達可能なマージコミットとする。ブランチの長さが 1 でマージコミットと同じファイルのみへの変更が行われていた場合は、共変更の差が起きないため除外した。

4.3.5 $RQ_{2.2}$ の結果

ブランチの長さ．ブランチの特徴に対する $Precision$ のコミットごとの平均値の分布を図 5 に示す。ブランチの長さが 1 以下（0 のものも 1 つ存在した）を除いて、ブランチを利用するほうがマージコミットを利用する場合と比べて高い値で分布しており、長くなるにつれその差も大きく

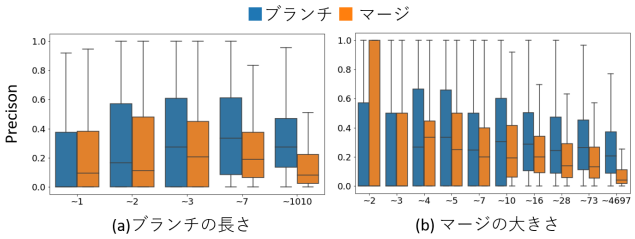

図 5 ブランチの特徴に対する *Precision* の分布

なる。長さが 1 以下のものでは、マージ時に追加の変更や不要な変更の削除が行われていた場合に限定されているため、マージコミットを利用するほうが *Precision* が高くなったと考える。

以上をまとめると、ブランチが 1 以下を除いて、ブランチ上のコミットで個別に扱うほうが高い値になり、ブランチが長くなるほどその差は大きくなる。

マージコミットの大きさ．マージコミットの大きさが小さい間はマージコミットで扱うほうが中央値や第三四分位数が高くなるものの、大きくなるにつれてブランチで扱うほうが高い分布になった。どちらの扱うほうが高い値になるかマージコミットごとに集計したところ、マージコミットが小さい間はマージコミットで扱うほうが高い場合が多いものの、マージが大きくなるにつれ、ブランチで扱うほうが多くなった。マージでまとめられることで追加された共変更がマージが大きくなると、追加された共変更が将来行われない割合が上がることが考えられる。

4.3.6 まとめ

$RQ_{2.1}$、$RQ_{2.2}$、どちらの分析でもブランチが長くたり、マージが大きくなったりすると、ブランチ上のコミットで個別に扱うほうが良い評価になる。長いブランチや大きな変更の場合、複数の目的の変更が行われており、マージでまとめることが悪影響につながっている。短いブランチや小さな変更の場合、共変更ではマージの方が良くなる傾向が見られるが、推薦結果にはほとんど反映されていない。関係しているブランチの長さの最大が 1 以下の推薦結果を確認しても、First-Parent が勝つ割合が高いものの 98%で引き分けになっていた。単一の短いブランチの影響しかないとその影響は無視できるほどになるため、ブランチの使用頻度が低く、そのブランチも短いリポジトリではブランチの扱いを変更する必要性が小さい。

なお、$RQ_{2.1}$ では、推薦に使用コするミット数が多くならないテストに限定しているが、RQ_1 と同じく使用コミット数に対する履歴戦略を変えた勝利割合と使用コミット数に対する履歴戦略の違いによるコミットの差の割合を確認したところ、使用コミット数が増えると履歴戦略の違いによるコミットの差の割合が変わらなくても、勝敗が付く割合が減り、影響が小さくなることが分かった。使用するコミット数はクエリのファイルが過去に変更された回数に依存するため、変更された回数が多いファイルをクエリとして変更推薦する場合は、使用するコミットが増え、ブランチのコミットの扱いを変更する必要性が小さくなる。

推薦結果でマージが 10 を超えたときに影響が急に増加したことは、10 以下の変更を持つコミットに制限していることが影響している可能性が高い。共変更の分析では、マージの大きさが 10 前後で大きな変化はない。マージコミットの共変更が取り扱われなくなることで、良くも悪くもブランチの影響が大きくなっていると考えられる。

> 長いブランチや大きなマージコミットが関係していると影響が大きくなり、ブランチ上のコミットで共変更を扱うほうが性能が良くなる。一方で、短いブランチや小さいマージコミットの場合、マージコミットで共変更を扱うほうが性能は良くなるが、その差は非常に小さく、影響も小さい。また、変更推薦手法における除外コミットサイズのパラメータが推薦結果の違いに影響している。

4.4 議論

RQ_2 の結果、ブランチが長くなったり、マージコミットが大きくなったりすると、ブランチ上のコミットで共変更を扱うほうが性能が良くなることが分かった。しかし、図 4 を確認すると一方的に Full が勝つ割合が上がっているのではなく、First-Parent が勝つ割合も上昇している。そこで、マージで追加された共変更が 7 以上（数が少ないと 1 つの共変更の影響を大きく受け、3 ファイルへの変更が持つ最大の共変更数 6 以下を除外するため）で、その共変更が将来も変更される確率が高いマージコミットについて、どのような共変更が追加されたかを少数のマージコミット（40 個）をサンプリングし、調べた。特徴的なものとして、apache/cassandra で更新内容を記録した CHANGE.txt との共変更で、対象の apache/cassandra のマージコミット 20 個中 19 個がそれに該当した。ほかのリポジトリでも設定ファイルとの共変更が 3 個含まれていた。ただ、apache/cassandra では複数のタスクをマージしている傾向があった。

また、マージコミットで新たに追加された変更によって追加された共変更が 40 個中 6 個で起きていた。追加された変更があるマージコミットではマージの共変更が将来行われる確率がブランチ上の共変更よりも高くなっており、追加された変更がないマージコミットでは逆になっていた。マージコミットで新たに変更が追加された場合、マージコミットで扱うほうが良いかもしれない。

4.5 インプリケーション

変更推薦では、基本的にブランチ上のコミットをマージコミットでまとめず、ブランチ上のコミットで個別に扱うほうが良いことが示唆される。ブランチが長くなるほどブ

ランチ上のコミットで個別に扱うほうが良くなる傾向があり、ブランチの長さが 1 のような短いブランチではマージコミットで扱うほうが良いが、その差は非常に小さい。また、長いブランチは様々なファイルへの変更を含むので、1 つのブランチでも変更推薦に影響を与える機会が多い。そのため、一律にブランチ上のコミットで個別に扱うか、マージコミットで扱うかを決める場合、マージコミットで長いブランチを扱うことで推薦可能範囲が広がる以上に、将来行われない共変更が作られるデメリットが大きく、ブランチ上のコミットで個別に扱うほうがリスクが低い。

ただし、変更された回数が多いファイルをクエリとして変更推薦する場合やブランチをあまり使われず、使用されているブランチも短いリポジトリでは、履歴戦略を変えても影響が小さいため、ブランチの扱いを注意する必要性は低くなる。

4.6 妥当性の脅威

内的妥当性. 変更推薦手法は、Kovalenko らの実装 [9] を基に作成した。その手法は、推薦が非常に限定される実装であり、ブランチの影響が小さくなっている可能性がある。

本論文では主なブランチの用途から、ブランチ上の変更は単一タスクであるという仮定のもと分析を行ったが、ブランチ上の変更は複数のタスクが行われていることがある。実際、apache/cassandra では異なるイシューのコミットをまとめてマージしていた。この論文の結果は、Miura ら [6] の結果を否定するものではない。

外的妥当性. Kovalenko ら [5] と Miiura ら [6] が使用した Apache と Eclipse のリポジトリのみが対象で偏りがあり、その開発スタイルに影響を受けている可能性がある。例えば、マージを多用されていた apache/cassandra では、開発の過程を記録せずにイシューに対応する変更が 1 コミットにまとめられていた。ブランチ上に開発の過程が記録されているリポジトリでは、マージで扱うほうが良い結果になる割合が増えるかもしれない。

5. 関連研究

Moonen らは変更推薦に影響を与える様々な要素の研究を行った。相関ルールの順位付けに使う *Confidence* や *Support* などの尺度の影響や推薦ルール作成に使用するコミットでのファイル変更数の制限の影響 [3]、使用する履歴の長さや直近の履歴を使用する影響 [4] を調べた。

ブランチの調査として、Zou ら [7] は、Github に公開されている 2923 のリポジトリでの使い方の調査を行った。また、ブランチの与える影響の研究として、Shihab ら [11] は、ソフトウェア品質に与えるブランチの影響を調べた。本論文では変更推薦に与えるブランチの特徴ごとの影響を分析した。

6. まとめ

本論文では、Kovalenko ら [5] と Miura ら [6] の結果の違いに端を発し、変更推薦結果に与えるブランチの影響の分析を行った。Kovalenko ら [5] のブランチ上のコミットを個別に扱う履歴戦略の性能がわずかに高くなるという結果は比較相手でブランチ上の変更が欠落していることが影響していた。それを修正するとマージコミットでまとめる履歴戦略が高い性能になったものが均衡した。

また、変更推薦にブランチが与える影響の分析の結果、変更推薦では基本的にブランチ上のコミットを個別に扱うほうが良いという結果になった。ブランチ上のコミットをマージコミットでまとめて扱うことで、共変更の漏れを防げる一方、大きいブランチでそれをやると将来変更されない共変更のノイズの悪影響が大きくなる。ブランチ上のコミットで共変更を扱うほうがリスクが小さい。

今後の課題として、より広い範囲のリポジトリを対象に TARMAQ [1] といった最新の変更推薦手法で実験することでの一般性の向上がある。

謝辞 本論文の一部は日本学術振興会科学研究費補助金（JP18K11238）の助成を受けた.

参考文献

[1] Rolfsnes, T., Di Alesio, S., Behjati, R., Moonen, L. and Binkley, D. W.: Generalizing the analysis of evolutionary coupling for software change impact analysis, *Proc. SANER*, pp. 201–212 (2016).
[2] Zimmermann, T., Weisgerber, P., Diehl, S. and Zeller, A.: Mining version histories to guide software changes, *IEEE Trans. Softw. Eng.*, Vol. 31, No. 6, pp. 429–445 (2005).
[3] Moonen, L., Di Alesio, S., Binkley, D. and Rolfsnes, T.: Practical guidelines for change recommendation using association rule mining, *Proc. ASE*, pp. 732–743 (2016).
[4] Moonen, L., Rolfsnes, T., Binkley, D. and Di Alesio, S.: What are the effects of history length and age on mining software change impact?, *Empir. Softw. Eng.*, Vol. 23, No. 4, pp. 2362–2397 (2018).
[5] Kovalenko, V., Palomba, F. and Bacchelli, A.: Mining file histories: Should we consider branches?, *Proc. ASE*, pp. 202–213 (2018).
[6] Miura, K., McIntosh, S., Kamei, Y., Hassan, A. E. and Ubayashi, N.: The impact of task granularity on co-evolution analyses, *Proc. ESEM*, pp. 1–10 (2016).
[7] Zou, W., Zhang, W., Xia, X., Holmes, R. and Chen, Z.: Branch Use in Practice: A Large-Scale Empirical Study of 2,923 Projects on GitHub, *Proc. QRS*, pp. 306–317 (2019).
[8] Bird, C. and Zimmermann, T.: Assessing the value of branches with what-if analysis, *Proc. FSE*, pp. 1–11 (2012).
[9] Kovalenko, V.: Mining File Histories: Should We Consider Branches?—Online Appendix, `https://github.com/vovak/branches`.
[10] Pugh, S., Binkley, D. and Moonen, L.: The Case for Adaptive Change Recommendation, *Proc. SCAM*, pp. 129–138 (2018).
[11] Shihab, E. and Zimmermann, T.: The effect of branching strategies on software quality, *Proc. ESEM*, pp. 301–310 (2012).