バージョン管理システムにおける コンフリクト自動解消への試み

七海 龍平　伊藤 恵

今日のソフトウェア開発においてバージョン管理システム (以下，VCS) は重要な技術である．複数人での開発現場において VCS の利用は必要不可欠なものとなっている．バージョン管理とは，ソースコードなどを含むファイルの変更をスナップショットとして保存し記録することで，それらをバージョンとして管理する概念である． VCS の操作において，変更履歴をブランチと呼ばれる系列に分岐させる操作と，分岐させたものを併合するマージと呼ばれる操作が存在する．これらの操作によって複数人での開発が容易になっている．しかし，マージ中にコンフリクトが発生すると，ブランチをマージすることができない．コンフリクトとは，分岐させた各ブランチ上の同一ファイルの同一箇所を編集した時に発生する競合問題である．マージを進めるには，コンフリクトを手動で解消する必要がある．この作業には，時間とコストがかかることが多い．本研究では，VCS 利用時におけるコンフリクトの問題に焦点をあて，発生したコンフリクトに対して機械学習を用いて自動解消する手法の提案を試みる．本稿では OSS の開発履歴から得られた学習データについての報告をする．

Version control system (VCS) is one of the most important technology for today's software development. The use of VCS is a essential at the software development site of multiple persons. Version control means saving and recording changes in files including source code as snapshots. So, it is a concept of managing them as versions. Operations of VCS include an operation that are to branch the change of develop history as a series called a branch and an operation called merge to merge the branch ones. Through these operations, development by multiple persons is facilitated. However, if conflicts occur during a merge operation, branches can not be merged. The conflict is a problem that occurs when editing the same part of the same file on each of the branches. If you want to proceed with the merge operation, it is necessary to repair the conflict manually. Usually, this work takes a lot of time and cost. In this research, we focus on the problem of the conflict in VCS, and try to propose a method to automatically repair conflicts that using machine learning. In this paper, we report on data sets obtained from OSS development history.

1　はじめに

今日のソフトウェア開発においてバージョン管理システム（以下，VCS）は重要な役割を担っている．バージョン管理とは，ある時点でのソースコードなどを含むファイルのスナップショット（これをバージョンと呼ぶ）を保存し，各バージョンを系列として管理する概念である [5]．VCS を利用したソフトウェア開発では，この概念により複数人で並行して開発を行うことができる．

ソフトウェア開発のプロジェクト内では並行に行うべき作業が多々発生する．例えば，あるバージョンのソースコード内で，既存の機能Aに存在するバグの修正と新機能Bの実装が同時に必要になったときなどが挙げられる．この場合，現バージョンから並行に作業を進め変更履歴を表現することができると便利である．それを実現するために VCS ではプロジェクト内の変更履歴をブランチと呼ばれる系列に分ける操作がある．このブランチによって VCS を利用した開発では，複数人で並列開発を行うことが可能となっている．

最終的には上記で説明したブランチを，1 つにまとめる必要が出てくる．ブランチで並行開発したものを 1 つにまとめる操作をマージと呼ぶ．このマージ操作

Attempt to Automatically Repair Conflicts in Version Control System.
Nanaumi Ryuhei, 公立はこだて未来大学　システム情報科学部　情報アーキテクチャ学科, School of Systems Information Science, Future University Hakodate.

では，コンフリクトと呼ばれる重大な問題が発生する可能性がある．コンフリクトとは，マージを行おうとしているブランチ同士で重なりあう部分を同時に変更したときに発生する問題である．コンフリクト発生時にどのブランチの変更を採用するのか，またはどちらとも採用するのかは機械的に正しいと判断することはできていない．そのため，現状では最終的にコンフリクトの修正を手動で解消する必要がある [2]．Listing 1 は実際にコンフリクトが発生した時のソースコードの状態である．この例では，int y の値を 2 または 3 のどちらにするかを判断し手動で直す必要がある．Listing 2 は Listing 1 を修正した後の状態である．例の修正は単純で時間のかからないものであるが，プロジェクトが大きいほどコンフリクトは多く発生する傾向があり [5]，コンフリクトの修正には多くの時間とコストが必要となる．Phillips らの研究によると，ブランチとマージに関してプロジェクト管理者にアンケート調査をした結果，回答者のうち 54 %がマージ操作における最重要課題はコンフリクトであると回答している [3]．この結果から示されるようにコンフリクトはソフトウェア開発において大きな課題の一つとなっている．

本稿では，オープンソースソフトウェア（以下，OSS）の開発履歴を学習データとし，機械学習を用いてこれらの現在手動で解消されているコンフリクトを自動で解消するための提案の試みについて報告する．

Listing 1 コンフリクトしているソースコード

```
 int x = 1;
<<<<<<< HEAD
 int y = 2;
=======
 int y = 3;
>>>>>>> 2d26451821e024c2
```

Listing 2 コンフリクトを修正したソースコード

```
 int x = 1;
 int y = 2;
```

2 関連研究

バージョン管理システムにおけるコンフリクトに関しては様々な研究が行われている．また，プログラムの実行結果がテストケースに対して期待された結果を返さない状態であるバグを自動解消する研究もまた，数多くされている．バグの自動解消に関する研究結果は，ソースコードの変更を予測するという点で，本研究でも活用することができると考えている．関連研究については次節以降に詳細を述べる．

2.1 機械学習によるコンフリクトの発生予測

Zirgler の研究結果から，同時に変更されたファイルの数，ブランチ上のコミットの数などのブランチに関わるデータがコンフリクトを予測するためのよい指標であることが分かる [4]．Zirgler の研究では機械学習を用いてソフトウェア開発プロジェクト内でブランチ間の潜在的なコンフリクト発生を予測し，早期マージなどの対策を行うことを目標とした．結果として，コンフリクトの可能性を予測するツールである GITCoP を開発し，4 種類の機械学習アルゴリズムと，2 種類のアンサンブル学習について評価した．結果として特定の条件下でC 言語によって書かれたリポジトリに対して，ランダムフォレストで学習したモデルが 94.8 %の確率でコンフリクトを予測することができたと報告されている．

2.2 機械学習を用いたバグの自動修正

Long らはオープンソースのソフトウェア開発リポジトリから得られたバグの修正履歴を利用し，機械学習を用いたバグ修正パッチ自動生成システムである Prophet を開発した [1]．結果として，実際のオープンソース開発上で発生した 69 のバグのうち，既存の研究で 1〜 2 個を自動で修正できていた結果に対して，Prophet では同数中の 15〜18 個のバグを自動で修正したことでその有用性を大いに示した．

3 自動解消への試み

現在，コンフリクトの発生を予測するための研究は多くされているが，発生したコンフリクトを自動で解消するための研究は十分に行われていない．本研究では，機械学習を用いたコンフリクトの自動解消を試みる．使用する機械学習の手法は複数選択することを想定しており，各手法で予測モデルを作成した後，

自動解消の精度を比べる予定である．本研究では，ランダムフォレスト，ロジスティック回帰，単純ベイズ分類器の 3 種類で比較する想定である．

3.1 特徴量の設計

学習に使用するデータセットを作成するために，OSS の開発プロジェクトが数多く存在する GitHub を利用する．既存の公開されている開発プロジェクトのマージ履歴をもとに実際にコンフリクトを再現し，データを集める．データセット作成に関する詳細は次節に述べる．データからは特徴量としては以下の 7 種類を抽出する．

1. 各ブランチのコミットメッセージの長さ
2. 各親ブランチに関わる人数
3. 各親ブランチの author の総コミット回数
4. 各親ブランチの最終コミット時刻
5. 各親ブランチのマージ回数
6. 各親ブラントのコミット回数
7. 各親ブランチのコードの行数

上記の項目のうち，1〜3 に関しては Zirgler の研究を参考に決定した [4]．4〜7 に関しては本研究独自の特徴量となっている．これらの項目から，分かるように本研究では特徴量としてプログラム意味論は想定しておらず，マージ履歴から抽出することができるデータのみを利用しコンフリクトを自動解消することを目指す．そのため，データとして集めるソースファイルはプログラミング言語の種類に依存することはない．現状では，コードの行数が多いほど新しいコードが含まれていると考えており,7 番目の各親ブランチのコードの行数の項目が最も寄与度が高いと仮定している．

3.2 教示データについて

Git の開発履歴からコンフリクトを発生させたファイルだけではなく，コンフリクトを解消した後のファイルも取得することができる．コンフリクトが発生したファイルと解消した後のファイルの差分を判定することで，学習データに修正の正解データとして以下の 5 種類の教示データを与える．

I マージ元のソースコードを採用する
II マージ先のソースコードを採用する
III どちらも採用する
IV どちらも採用しない
V その他

コンフリクトの解消方法としては基本的に，開発者がどのブランチのソースコードを採用するかを決定し，それ以外のコードを不採用とする方法がとられる．しかし，可能性としては上記の Ⅲ,Ⅳ,V の方法がとられる可能性も大いにありうる．V のその他は，ファイルが削除されている場合や親ブランチのソースコードとは全く違う新たなコードが採用されている状態など上記の 4 種に当てはまらない状態を示す．本研究では，コンフリクト発生時にファイルに対する操作を網羅するため上記の 5 種の方法に分類する．

4 データセットについて

本研究の機械学習で利用するデータセットには，OSS の開発プロジェクトの開発履歴を使用する．GitHub 上に存在する OSS の開発リポジトリに対して，フォークと呼ばれる操作を行うことで，自分のリポジトリとしてコピーすることができる．フォークにより自分のリモートリポジトリへコピーした開発データを，ローカルへダウンロードすることでクローンを作成することができる．この時クローンされるデータはソースコードなどのファイルデータだけではなくこれまでの開発履歴のデータもクローンされる．この開発履歴のデータを利用することで，プロジェクト内で今までに行った全マージ操作を再現することができる．そのため過去に起こったコンフリクトの再現も可能であることになる．再現されたコンフリクトからはデータセットとしては以下の 5 つのデータを抽出する．

1. コンフリクト発生中のファイル
2. コンフリクト解消済のファイル
3. マージ先親ブランチのファイル
4. マージ元親ブランチのファイル
5. 3 章で示した特徴を抽出したファイル

また，データ収集に使う開発プロジェクトの選択については，プロジェクトの中で人気が高いもの（スターの数が多いもの）から順に選択する．GitHub では，機能の 1 つとして利用者がお気に入りのリポジトリに対してスターを付与することができ，ここで言

う人気の高いプロジェクトとは，スターの数が多いプロジェクトのことを指している．

5 収集したデータ

現状の収集量の目標値はコンフリクト数で 10,000 回分に相当するデータ量である．収集量が目標値に達した時点で機械学習によるコンフリクトの判別プログラムを作成し，その性能を確かめる．結果によって，収集する特徴量，データの収集量などを再度検討する．表 1 は現在収集した一部のデータから，総マージ数とそれに対するコンフリクト数を示したものだ．現状では 20 プロジェクト，1957 回のコンフリクトに対するデータが集まっている．ただし，すべての再現データが完全にマージを再現できているかは現状では確かめられていない．現状集まっているデータ量では，特徴や相関は見えてこないが，表 1 から少なくとも人気の高いプロジェクトでは，ほぼ確実にコンフリクトが発生しているのではないかと考えることができる．また，現状で収集が完了しているデータセットのコンフリクトの修正状態をランダムに確認したところ，修正時にコンフリクト発生箇所以外の部分に新たな修正を加えて再度マージを行っている修正済みファイルが多く存在した．この結果から，コンフリクトが発生しているファイルに教示データを付与する際，付与する教師データを単純にファイル同士の差分が存在するかで決定することは不適切である可能性が考えられ，コンフリクトが発生した部分に限定した差分の比較が必要であることが分かった．

今後，データ量が目標値に到達時に収集したデータの加工をする際，上記の問題を考慮すること以外にも外れ値などの検出のためデータの相関や特徴の可視化を行い確認する必要もあると考える．

6 機械学習によるコンフリクトの判別

本章では学習データが用意できたあと，どのようにして機械学習による判別を行っていくのかを述べる．第 3 章で述べたように，本研究ではランダムフォレスト，ロジスティック回帰，単純ベイズ分類器の順に予測モデルを作成していく．学習データ用意後はまず，ランダムフォレストによる予測モデル作成を行う．ランダムフォレストによる予測モデルを作成するために，特徴量として収集したデータを説明変数とし，決定木を作成する．木の数としては現在検討中であるが，基本的には少ない本数から可能な限り増やしていく予定である．また，学習を行う際は過学習の対策としてデータセットをクロスバリデーション用 90%と過学習対策テスト用の 10%に分ける．クロスバリデーション用の 90%はさらに訓練データとテストデータとして分けてスコア調整を行う．スコアとして十分なものができた場合，過学習対策テスト用の 10%をテストデータとして使う．スコアが十分なものとならない場合は計画していた説明変数を増やし再度学習を行う．

現状ではランダムフォレストの計画を進めているが，今後はロジスティック回帰，単純ベイズ分類器の計画を進め，順次予測モデルの作成を行っていく．

表 1　収集データのマージ回数に対するコンフリクト回数

プロジェクト名	マージ回数	コンフリクト回数
freeCodeCamp	3162	155
bootstrap	4099	464
free-programming-books	1590	58
react	2431	17
tensorflow	2205	218
You-Dont-Know-JS	272	3
vue	7	2
oh-my-zsh	1509	42

7 おわりに

本稿では，VCS におけるコンフリクトを機械学習を用いて自動で解消するための試みに関して，学習に必要なデータの収集状況と今後の見通しについて報告した．データを収集する過程で，実社会の大多数の OSS 開発プロジェクトでコンフリクトが発生していることが分かった．また，収集したデータから学習データに教示データを与える際の判断をファイル全体での差分により決定することは不適切である可能性があることが分かった．今後はまず，データを目標数まで収集することに専念する．その後は，集まっているデータが正確なデータであるのか検証し，学習データの教示データについてはファイル全体ではなくコンフリクトが発生している部分のみの差分により判断をす

る．学習データが完成次第，ランダムフォレストでの予測モデル作成を進めていく見通しである．課題として，機械学習による判別実装後に十分なスコアが得られない場合は再度特徴量について検討することで大幅な手戻りが生じる可能性があることが挙げられる．

参 考 文 献

[1] Fan Long, M. R.: Automatic Patch Generation by Learning Correct Code, *POPL'16*, (2016), pp. 20–22.

[2] 浜野純: コミット家系図とマージ, 株式会社 秀和システム, 2009.

[3] Shaun Phillips, Jonathan Sillito, R. W.: Branching and Merging: An Investigation into Current Version Control Practices, *ICSE'11*, (2011), pp. 21–28.

[4] Ziegler, T.: GITCoP: A Machine Learning Based Approach to Predicting Merge Conflicts from Repository Metadata, 2017.

[5] 松本 健一湯月 亮平: 開発者はどのようにしてコンフリクトを解消しているのか：コンフリクト解消の自動化に向けて, No. 2014, 2014, pp. 31–32.