Merge Conflict Resolution using Meta-Information

Rajat Goel
Department Computer Science and Engineering
Kanpur Institute of Technology
(Affilated from A.K.T.U)
Kanpur, India

Pronab Kumar Adhikari
Department Computer Science and Engineering
Kanpur Institute of Technology
(Affilated from A.K.T.U)
Kanpur, India

Abstract - Merge conflict often occurs in parallel development when we use version control system like GIT.Resolving a conflict is not an easy task because developers need to first identify the conflict and then resolve conflict by manual changes. In this paper we are proposing a model that will suggest the resolution by using metadata from development history. After a case study of more than 10 projects we have concluded that a number of conflicting lines, conflicting commits and random edits have a high rate of contribution over merge conflict.

Keywords Development History, Merge Conflict, Meta-Information and Machine Learning

I. PREFACE

Currently, in the software developments field, one of the development methods in which a team of multiple developers performs parallel development using a version control system (VCS).when developing in parallel using VCS, a new derivative branch is created from the branch that is the mainstream of development, and after the development is completed, it is integrated (merged) into the mainstream branch. As a result, multiple developers can proceed with development at the same time, and efficient software development can be done. One of the most used VCSs is Git [1].

A common problem when developing with Git is merge conflicts. When we perform merging from a newly created branch to the source branch, if the part edited in the derived branch and the same part in the mainstream branch are edited, there will be a situation where the merge cannot be performed. This is a merge conflict, and it has been clarified that it occurs relatively frequently in software development using Git. In the research by Brun et al As a result of investigating the development history of nine open source software (OSS) including Git and Perl5, merge conflicts occurred in all projects, and the ratio was about 19% on an average and maximum about 42%[2].

It became clear that The problem with merge conflicts takes time and effort to resolve them. When a merge conflict occurs, the developer needs to investigate the cause, devise a solution, and edit it manually. These tasks often take days, which can be very costly for developers and the entire project [3].

Some existing studies have clarified the characteristics of merge conflicts and ways how to resolve them. There is a bias in the number of lines where merge conflicts occur, and it has been clarified that the number of lines where merge conflicts occur is one of the factors that complicate merge conflicts [1]. In addition, research is being conducted to investigate from the commits of parents of merge commits to the common ancestor, and to recommend developers who have the knowledge necessary to resolve merge conflicts from the files edited with the commit creator [5]. Another study of commit authors found that the smaller the commit author's commit rate to a project, the more likely to have bugs or software flaws [6], and merge conflicts. Regarding the method of resolving the merge conflict, it became clear that it is often resolved by adopting one of the edits for the commit pair in which the merge conflict occurred and deleting the other edit. [7]. In this study, we aim to create a model that determines an appropriate method for resolving merge conflicts by utilizing the meta information that is characteristic of these merge conflicts.

In this research, we use machine learning to support the resolution of merge conflicts by creating a model that determines the resolution method from meta information such as the number of lines of merge conflicts, the creation date and time of commits, and the developers who created them.Aim of this experiment is to find, what kind of information contributes to the method of resolving the merge conflict by calculating the correct answer rate of the judgment from the actual development history and measuring the importance of the meta information given as a parameter at the time of model creation.

In this paper, Chapter 2 proposes methods for data collection and model creation, and Chapter 3 evaluates the created model. Chapter 4 discusses the threat to the validity of this study, and Chapter 5 describes future issues. Finally, we will summarize it in Chapter 6.

II. PROPOSAL METHOD

Using machine learning, create a model that determines how to resolve a merge conflict from the meta information related to the commit pair in which the merge conflict has occurred, using the following procedure.

STEP1 Collecting the history of merge conflicts STEP2 Collecting meta information
STEP3 Obtaining the resolution method
STEP4 Creating the judgment model of the merge conflict resolution method

T

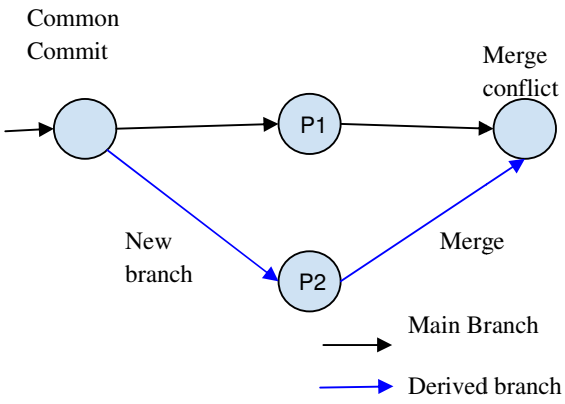


FIG 1

he model determines the resolution method for each commit in the commit pair. As shown in Fig. 1, each commit of the commit pair in which the merge conflict occurred is distinguished. Let commit P1 be the commit on the mainstream branch and commit P2 be the commit on the newly derived branch.

A. Identifying the location of the merge conflict For the commit pair, identify the file path where the merge conflict occurred and the line where the merge conflict occurred. For commits P1 and P2 where a merge conflict has occurred, check out to P1 from the repository and merge into P2, and the result shown in Fig. 2 can be obtained. In the case of Fig. 2, the third line of the output result is the file path in commit P1 of the file in which the merge conflict occurred, and the fourth line is the file path in commit P2 of the file in which the merge conflict occurred. And the set of 3 2 numbers displayed on the 6th line shows the line where the merge conflict occurred. If there is a set of numbers "a, b", a is the first line number where the merge conflict occurred, and b is the number of lines where the merge conflict occurred. (However, when b = 1, b is omitted.) In other words, "10,3" means that a merge conflict occurred from the 10th line to the 10 + 3 (= 13) line in commit P1. , And "11,2" indicates that a merge conflict occurred from the 11th line to the 11 + 2 (= 13) line in commit P2. In addition, "15,6" means that the file after the merge conflict has occurred is saved in the state where the merge conflict has occurred from the 15th line to the 15 + 6 (= 21) line.

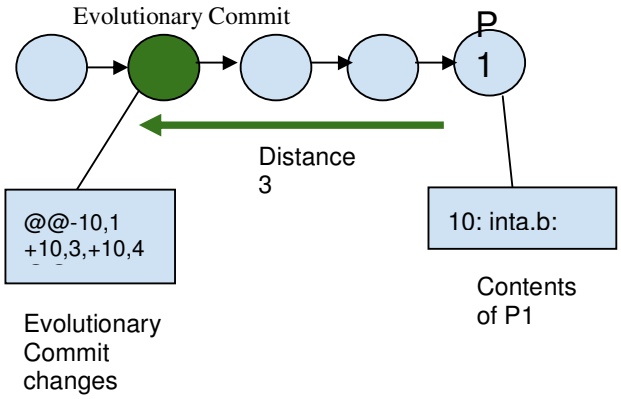
B. Collection of meta information The meta information shown in Table 1 is used as the parameters used to create the model. The meta information to be collected includes the number of lines where merge conflicts occur, the commit creation date and time, the commit rate of the commit creator and the distance between evolutionary commits.

```
git checkout P1
git merge P2
git diff –U0
  1 diff --cc Test.java
  2 index ******
  3 --- a/ Main.java
  4 +++ b/Main.java
  5 @@@ -10,3 -11,2 +15,6 @@@ public class Main
  6 ++<<<<<<< HEAD
  7  +      int a, b;
  8  +      a = 1;
  9  +      b = 2;
 10 ++=======
 11 +       int a = 1;
 12 +       int b = 2;
 13 ++>>>>>>> c3d4c3d4c3d4
```

Table 1 Parameters used to create the model

	Parameter name
P1	lines (P1) Number of lines where merge conflict occurred in P1 time (P1) Commit creation date and time in P1 author (P1) Commit rate of commit creator in P1 evolutionary (P1) Distance between P1 and Evolutionary Commit
P2	lines (P2) Number of lines where merge conflict occurred in P2 time (P2) Commit creation date and time in P2 author (P2) Commit rate of P2 commit creator evolutionary (P2) Distance between P2 and Evolutionary Commit
Difference	lines(d) lines(P2)-lines(P1) time(d) time(P2)-time(P1) author(d) author(P2)-author(P1) evolutionary(d) evolutionary(P2)-evolutionary(P1)

There are four types of distances to the Evolutionary Commit, and these are obtained from the development history for each of P1 and P2. In addition, those differences are also used as parameters. The method of acquiring each parameter from the development history will be described below.



C. Commit meta information

For the commit pairs P1 and P2 where the merge conflict occurred, get the commit creation date and time and the commit ratio of the commit creator for the entire project.

First, the method of obtaining the commit rate is, Find out who created the commit, for every commit in the project for each commit creator calculate the ratio of the commit creator's commits to the total project commits. This is the commit rate of a commit creator.

Next, for the commit pairs P1 and P2 where the merge conflict occurred, the commit creation date and time and the commit creator are extracted. Get the commit rate of each commit creator by comparing it with the commit rate information you got earlier.

D. Number of lines where merge conflicts occur

The number of lines where a merge conflict occurs can be obtained from the output when the location where the merge conflict occurs is specified from Fig. 2.

E. Distance to Evolutionary Commit

In each of the commits P1 and P2 where the merge conflict occurred, the commit in which the location where the merge conflict occurred was edited immediately before is called the Evolutionary Commit [8].

As a parameter for model creation, get the distance between the Evolutionary Commit and the commit where the merge conflict occurred. Here, the distance between commits means the number of commits that exist from one commit to the other, as shown in Fig. 3.

Since all the information about when a certain line was edited is recorded in the development history, it is possible to find the Evolutionary Commit by tracing the history from P1 and P2 to the oldest one and calculate the number of commits up to that point.

F. Collection of solutions

As a method of resolving the merge conflict, for each commit of the commit pair in which the merge conflict occurred, Table 2 shows.

Table 2. Merge conflict resolution method definition

ADOPT	Adopt all edits
DELETE	Delete all edits
EDIT	Adopt part of the edit, make additional edits, or both
ZERO	The edited part is 0 lines

Four types are defined. The output by the model is a set of resolution methods (P1 resolution method, P2 resolution method).

Find out what edits were made to the merge commit by comparing it to the situation where the merge conflict occurred. In merge commit M, if a conflict occurs between its parent commit pairs P1 and P2, the resolution method can be specified by analyzing the difference between the result of merging P1 and P2 and M. it can. Here, the lines shown in green in Fig. 4 are the lines that were deleted when the merge conflict was resolved. It can be seen that the 15th line of Test.java was deleted from the 5th line of the execution result, and the 19th to 23rd lines of Main.java were deleted from the 7th line. Comparing this result with the result in Fig. 2, it can be judged that the contents of the 16th to 18th lines edited by P1 are adopted. Therefore, it can be said that the elimination method for P1 is ADOPT and the elimination method for P2 is DELETE.

Figure 5 shows an example of EDIT as a method for resolving merge conflicts. The 8th line "-int c = a + b;" means that it was newly added when the merge conflict was resolved. In other words, in this case, the resolution method for P1 is EDIT, and the resolution method for P2 is DELETE. In the judgment model of the solution method this time, what kind of editing is performed is not specified.

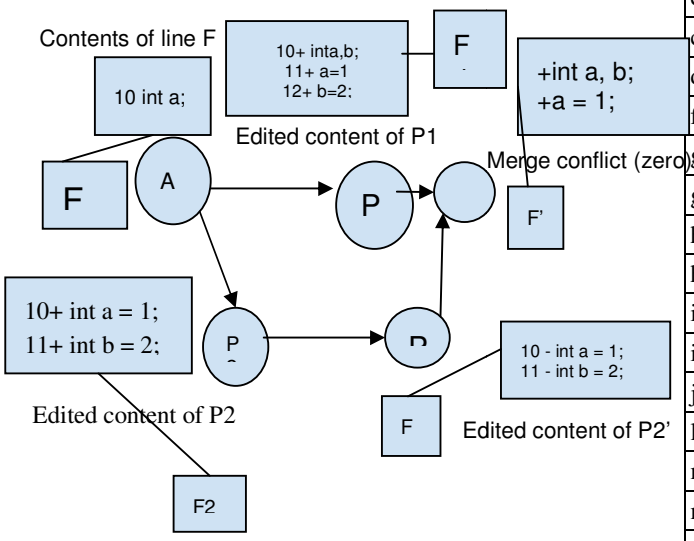
A case where the method for resolving a merge conflict is ZERO is that the location where the merge conflict occurs is edited once, the edit is deleted, and an attempt is made to merge. As shown in Figure 6, a branch is created from the common ancestor A, edited once in commit B, and then edited in commit P2.

```
git checkout P1
git merge P2
git diff –U0 M
  1 diff --git Test.java
  2 index ******
  3 --- a/ Main.java
  4 +++ b/Main.java
  5 @@@ -14,0 -15 @@@ public class Main
  6 +<<<<<<< HEAD
  7 @@@ -18,0 -19,4 @@@ public class Main
  8 +=======
  9 +       int a = 1;
 10 +       int b = 2;
 11 +>>>>>>> a1b2a1b2a1b2
```

Figure 4 Determining how to resolve merge conflict

```
git checkout P1
git merge P2
git diff –U0 M
  1 diff --git Test.java
  2 index ******
  3 --- a/Main.java
  4 +++ b/Main.java
  5 @@@ -14,0 -15 @@@ public class Main
  6 +<<<<<<< HEAD
  7 @@@ -18,0 -19,4 @@@ public class Main
  8 -       int c = a + b;
  9 +=======
 10 +       int a = 1;
 11 +       int b = 2;
 12 +>>>>>>> a1b2a1b2a1b2
```

Figure 5 Merge Conflict resolution using EDIT



Suppose that the damaged part is deleted. At this time, there is a fact that the same part was edited on Git, but there is a situation where no change is seen on the file, and a merge conflict occurs where the edited part is 0 lines. If the location where the merge conflict occurs is 0 lines, ADOPT, DELETE, or EDIT cannot be determined from the merge commit M after resolution, so ZERO was defined as one of the merge conflict resolution methods.

G. Creating a merge conflict resolution method judgment model In determining the merge conflict resolution method, the meta information of the merge conflict extracted from the development history is input, and what kind of commit conflict occurs for the commit pair Random forest is used as an algorithm to create a model that outputs a combination of resolution methods to see if the resolution method is appropriate.

III. Evaluation of the proposed method

In order to evaluate the proposed method, the OSS development history that is actually open to the public is used to evaluate from the following points.

RQ1: How accurate is the model judgment?

RQ2: What is the meta information that contributes to the decision? The evaluation is performed according to the following procedure. STEP1: Collect merge conflicts from the project development history.

STEP2: Divide the collected merge conflict into teacher data and test data.

STEP3: Extract meta information and resolution method from teacher data and create a model. STEP4: Extract meta information from test data and input it to the model.

STEP5: Check if the judgment result of the model matches the actual solution method.

Table 3 Number of commits and merge conflicts

Project Name	Merge Conflict
beam	981
camel	37
cassandra	13132
cordova-android	700
curator	255
dubbo	426
flink	2152
geode	427
groovy	294
hbase	108
hive	4809
ignite	1703
incubator-heron	56
jmeter	394
lucene-solr	1606
mahout	105
maven	248
nifi	591
nutch	442
rocketmq	46

As a test target, we collected 20 Java projects from the OSS provided by Apache. Table 3 shows the number of merge conflicts extracted from the project used for the test.

A. Correct answer rate of 1 model

On calculating the correct answer rate of the judgment of the resolution method by the model, the collected merge conflict is divided into 5 parts, one is used as test data, and the rest is used as teacher data. The average correct answer rate was

resolution method determined by the model in both the commit P1 on the mainstream branch and the commit P2 on the newly created derived branch in the test data. Table 4 shows the percentage of correct answers as a result of the test.

As a result of the test, a high correct answer rate of up to 94.43% was obtained, and there were 6 out of 20 projects with a correct answer rate of 80% or more. From this result, it can be seen that the model created by the method of this study is a useful judgment model for merge conflicts in some projects. In addition, the average correct answer rate for 20 projects was 66.41%, but considering that there are 16 combinations of merge conflict resolution methods that are output, it is considered that the resolution methods have been sufficiently determined. Is done.

On the other hand, the project with the lowest correct answer rate was incubator-heron, and the correct answer rate was 23.83%. As the cause of the decrease in the correct answer rate, Table 3 shows that the number of incubator-heron merge conflicts is 56, which suggests insufficient learning of the model.

Further investigation will be conducted on the projects for which the judgment accuracy was very good. Beam is one of the projects whose judgment accuracy exceeds 90%.

Table-4 Correct answer rate for each pair based on judgement model

	Correct answer rate
beam	90.92%
camel	43.06%
cassandra	48.49%
cordova-android	50.70%
curator	66.44%
dubbo	74.01%
flink	65.66%
geode	57.48%
groovy	65.57%
hbase	44.23%
hive	63.58%
ignite	67.88%
incubator-heron	23.83%
jmeter	94.43%
lucene-solr	63.42%
mahout	82.15%
maven	80.03%
nifi	92.94%
nutch	70.20%
rocketmq	83.29%

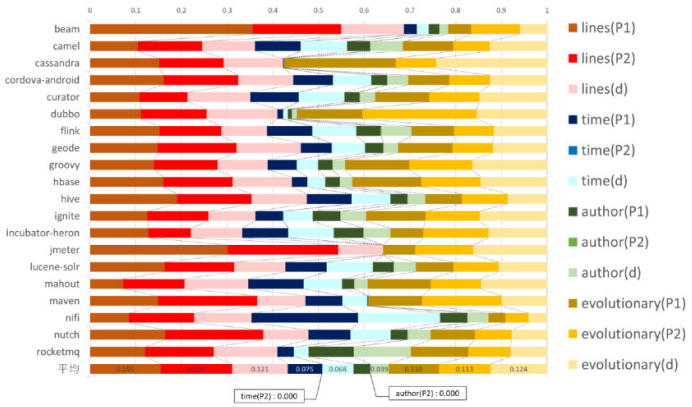
Table 5. Judgement for resolution method (DELETE, ADOPT)in beam

		Judgement Result	
		TRUE	FALSE
Actual Solution	TRUE	112	5
	FALSE	1	X

Approximately 60% of the adoption was resolved by one-sided adoption of (DELETE, ADOPT). If there is a bias in the solution method, the model created by machine learning may answer only the biased solution method. Table 5 shows the merge conflicts in which the actual resolution method was (DELETE, ADOPT) and the merge conflicts in which the judgment model was judged to be (DELETE, ADOPT). From this table, in the test shown in Fig. 5, Recall = 0.99, Precision = 0.96, F1 value = 0.97 for the resolution method (DELETE, ADOPT), and this model makes an accurate judgment on the bias of the resolution method in the project, From the above results, the answer to RQ1 is as follows.

RQ1. How accurate is the determination of how to resolve merge conflicts based on meta information? When tested on 20 projects, the average was 66.41%. In some cases, an accuracy of 90% or more is obtained, and such projects tend to have a bias in the method of resolving merge conflicts, but the model can judge without overreacting to the bias.

For projects with a small number of merge conflicts, it may not be possible to learn sufficiently and the accuracy may decrease.



B. Parameter importance

An index called importance is used to find out which parameter the created model classifies the solution method. The importance is the ratio of each parameter contributing to the classification in the created random forest model, and the sum of the importance of all parameters is 1.

For each project, 80% of the collected merge conflicts were given as teacher data to create a model, and the importance of each parameter was examined. The result is as shown in Fig. 7. Here, the average in Fig. 7 is the average of 20 projects for each parameter.

From Fig. 7, the number of lines (P1) and lines (P2) at the location where the merge conflict occurs are both as large as 0.15 or more, and there is no commit whose importance becomes extremely small. Also, regarding lines (d), there are many projects whose importance exceeds 0.1. Therefore, it can be said that the number of lines at the location where the merge conflict occurs is meta information that greatly contributes to the determination of the method for resolving the merge conflict in any commit. In addition, the average value of the distance from the commit in which the conflict occurred to the Evolutionary Commit, the evolutionary (P1) and evolutionary (P2), and the difference evolutionary (d) is 0.11 or more, which is a large value.

On the other hand, the commit creation date and time and the commit rate of the commit creator are values that cannot be said to contribute significantly to the judgment. In particular, from Table 7, it can be seen that the meta information of commits on the newly created branch from the mainstream branch, such as time (P2) and author_ratio (P2), does not contribute to the determination of the merge conflict resolution method. However, regarding the commit creation date and time, the information time (P1) of commit A on the mainstream branch is 0.1 or more in some projects, so it can be said that it contributes to the judgment of merge conflict.

RQ2: What parameters influence the judgment? Depending on the importance of the model, the number of lines in which the merge conflict occurred (lines (P1), lines (P2), lines (d)) and the distance from the commit in which the merge conflict occurred to the evolutionary Commit (evolutionary (P1), evolutionary (P2) and evolutionary (d)) contribute significantly to the method of resolving merge conflicts.

Also, the commit creation date and time (time (P1)) of the commits that exist on the mainstream branch is not large, but it contributes to the judgment. On the other hand, it can be said that the commit creation date and time (time (P2)) of the commit on the newly created branch from the mainstream and the commit rate (author (P2)) of the commit creator do not contribute much to the judgment.

IV. Validity threats

A. Problems related to parameter selection In this study, the meta information used for model creation parameters is the commit creation date and time, the commit rate of the commit creator, the number of lines in which merge conflicts occurred, and the evolutionary Commit. Although the distance to the development history also hides meta information that is not used this time. Therefore, the combination of meta information utilized by the model created in this study is not always optimal. However, based on the percentage of correct answers obtained in this study, we think that the model in this study may be useful for resolving merge conflicts.

B. Problems related to experimental subjects

All the projects used in this study are OSS provided by Apache, and there is no certainty that the results of this study will be similar to those of other OSS and commercial projects. However, since the development scale and domain of the projects targeted for research are not extremely biased, we believe that we were able to investigate a wide range of projects.

V. In addition to the meta information used to create the model for this research, there are commit messages and tags as information left on Git. The commit message and tag attached to a commit are large pieces of information that represent the edits made in that commit. For example, whether a commit that has a merge conflict or its Evolutionary Commit message contains words related to bug fixes or feature additions may affect how you resolve the merge conflict when it occurs. Therefore, a future task is to investigate commit messages and tags and increase the parameters used to determine how to resolve merge conflicts using the model. In addition, if the number of merge conflicts in the project development history is small, it is not possible to make an appropriate judgment, which is a problem to be solved in order to utilize the model of this research in actual development. The large number of merge conflicts that occur in the development history means that a certain period of time has passed since the project was launched. In other words, the judgment by this model is most frequently edited by multiple developers, and cannot be applied at the start-up of a project where there is concern about the occurrence of merge conflicts. As a solution to this problem, we aim to create a versatile model that does not depend only on the development history of the applied project by creating a model from the meta information of merge conflicts collected from the development history of multiple projects.

VI. 6. In summary, we proposed a method to support the resolution of merge conflicts in software development by creating a model that determines the resolution method using the meta information of the development history. As a result of the test using the OSS that is actually published, it became clear that the correct answer rate of the judgment model created by the method of this research is about 65% on average and 94% at the maximum, and it is possible to judge the solution method with high accuracy. It was. However, as a drawback of using machine learning, in a project where the number of merge conflicts that can be obtained from the development history is small, it is expected that the accuracy of judgment will decrease due to insufficient learning of the model. From this result, in a project where many

merge conflicts occur, it is considered to be useful as a guideline for determining how to resolve the merge conflicts when the developer faces them.

REFRENCES

[1] Anita Sharma,Shane McKee, Nicholas Nelsonand resolutions.
[2] Reid Holmes,YuriyBrun, Michael D. Ernst, and David Notkin. Early detection of collaboration conflicts and risks.
[3] Bakhtiar Khan Kasi and Anita Sarma. Cassandra: Proactive conflict minimization through optimized task scheduling. In International Conference on Software Engineering(ICSE), pp. 732–741, 2017.
[4] Iftekhar Ahmed, Caius Brindescu, UmmeAydaMannan, Carlos Jensen, and Anita Sarma
[5] Catarina Costa, Leonardo Murta, and Anita Sarma.
[6] Jon Eyolfson, Lin Tan, and Patrick Lam.
[7] JairFigueiredo
[8] MehranMahmoudi, Sarah Nadi, and NikolaosTsantalis. Are refactorings to blame?