2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Beyond GumTree: A Hybrid Approach to Generate Edit Scripts

Junnosuke Matsumoto, Yoshiki Higo and Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
{j-matumt, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—On development using a version control system, understanding differences of source code is important. Edit scripts (in short, ES) represent differences between two versions of source code. One of the tools generating ESs is GumTree. GumTree takes two versions of source code as input and generates an ES consisting of *insert*, *delete*, *update* and *move* nodes of abstract syntax tree (in short, AST). However, the accuracy of *move* and *update* actions generated by GumTree is insufficient, which makes ESs more difficult to understand. A reason why the accuracy is insufficient is that GumTree generates ESs from only information of AST. Thus, in this research, we propose to generate easier-to-understand ESs by using not only structures of AST but also information of line differences. To evaluate our methodology, we applied it to some open source software, and we confirmed that ESs generated by our methodology are more helpful to understand the differences of source code than GumTree.**

***Index Terms*—Difference, Edit Script, GumTree**

I. Introduction

It is inevitable to use a version control system when developing software systems. Understanding code differences is required before a variety of activities in software development and maintenance. Version control systems provide a functionality to retrieve any past version of source code. This is useful to compare two versions of source code. Edit scripts (in short, ES) represent differences between two versions of source code. In general, to generate ESs, `diff` command is used. `diff` command is based on Myers algorithm [1] and generates ESs on a line-based granularity, and line-based ESs consist of two actions, *insert* and *delete*.

`diff` command has two problems. The first problem is, line-based differences are coarse-grained and do not consider syntax information. For example, a line, `int foo = 0;` is edited to `final int foo = 0;`, `diff` command regards that all tokens of the line are edited. The second problem is, `diff`'s ESs include only two actions. Thus, they cannot sufficiently present developer's intent.

To solve those problems, GumTree [2] was developed. GumTree generates abstract syntax tree (in short, AST)-based ESs. They consist of four actions, *insert*, *delete*, *move* and *update*. GumTree's ESs are used in many higher level applications or further research [3]–[6].

However, GumTree has problems as well. One of them is that occasionally GumTree cannot appropriately detect *move* and *update* actions [7]. We applied GumTree to many revisions of some open source software (in short, OSS), to check whether the ESs were correct or not. As a result, we found that many *move* actions were not correct and many pairs of *delete* and *insert* should be *update* actions. For example, GumTree outputs *move* actions even though the code is not edited, moreover, GumTree outputs *delete* and *insert* actions even though the code is updated. Such inappropriate action generating makes ESs unnecessarily longer. The longer ESs are, the more difficult it is for developers to understand them [2], [8], [9].

In this paper, by shortening ESs, we propose to generate easier-to-understand ESs. To improve accuracy of *update* and *move* actions, it is necessary to enhance the algorithm of matching nodes in two ASTs [10]. While GumTree matches the AST nodes with only information of tree structure, our methodology matches the AST nodes with also information of `diff` command. Our methodology divides AST nodes into two groups. The first group is a set of AST nodes in lines not included in line-based differences. The second group is a set of nodes in line-based differences. Then, our methodology matches AST nodes within the same groups. This strategy can avoid the aforementioned problems of inappropriate ESs generating.

We applied it to many revisions of some OSS, and we confirmed that it generated shorter ESs. We also experimented with research participants, and we confirmed that our methodology is more helpful to understand code differences than GumTree.

II. Differences of Abstract Syntax Tree

A. Abstract Syntax Tree

An AST represents a tree structure of source code. An AST node consists of the following information.

- A parent node: a reference to its parent. However, the root node has no parent node.
- A type: a kind of a node (e.g., if statement, variable declaration).
- A value: information of the node other than the type (e.g., name of class).

B. GumTree

Our methodology is based on GumTree, thus, we show GumTree's algorithm in this subsection. GumTree takes two versions of source code as input. GumTree makes ASTs from each source code and generates an ES that

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00082

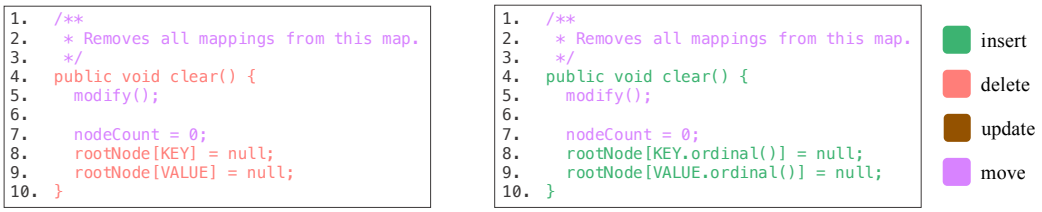

Fig. 1. An edit script generated by GumTree. The colored code presents the differences between the two versions (the length is 22).

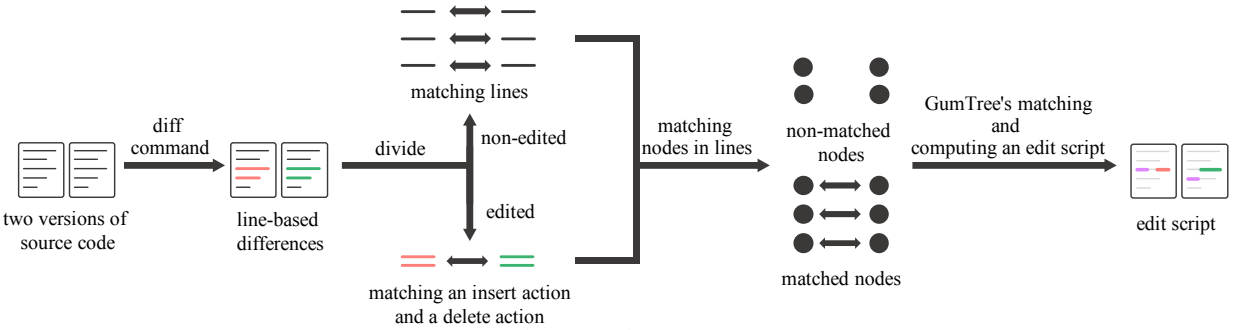

Fig. 2. Overview of our methodology

presents AST-based differences. The length of an ES is the number of edit actions included in it.

GumTree's algorithm consists of the following two steps:

1. matching similar AST nodes from both ASTs, and
2. computing an ES from matching results

In the first step, a node in an AST can be matched only with a node that has the same type in the other AST. In the second step, from matching results, GumTree computes information about (1)which nodes are inserted, (2)which nodes are deleted, (3)which nodes are moved to where, and (4)which nodes are updated to what. The second step was sufficiently optimized [11], thus we focus on the matching algorithm.

GumTree's matching algorithm consists of two phases. In the first phase, GumTree matches subtrees of both ASTs. In the second phase, nodes in the matched subtrees are matched if the nodes have the same type and the Jaccard similarity between both subtrees of the nodes is over the threshold.

However, GumTree's matching algorithm has a problem [10]. GumTree often mismatches nodes against developer's intent. As a result, GumTree generates unnecessarily longer ESs. The longer ESs are, the more difficult it is for developers to understand them [2], [8], [9]. We applied GumTree to a revision of Apache Commons Collections. Figure 1 visualizes a part of the results. While only the lines 8 and 9 are edited, GumTree outputs some actions for all lines. For example, while `public` in the line 4 is not edited, GumTree outputs *delete* and *insert* actions. The reason is that `public` in both ASTs are not matched.

III. Related Work

A. The Imprecisions of GumTree

Guillermo et al. discussed the imprecisions of GumTree [10]. In the paper, they applied GumTree to a C# software system, and they found that 27% of the 86 file version pairs is not optimal. They found that there were issues in matching phase. They indicated that GumTree treated source code as "just an AST", and did not consider many language features.

B. An Extension of GumTree

As an extension of GumTree, IJM was developed [7]. To improve accuracy of matching algorithm, IJM uses the features of Java. IJM consists of three approaches: Partial matching, Name-aware matching and Merging name nodes. Partial matching decreases the amount of nodes that are matched between different methods. Name-aware matching takes the names and values of nodes into account. Merging name nodes decreases the AST size by merging some node types with their respective simple name nodes.

IV. Methodology

Our methodology is based on GumTree. In GumTree, calculating ESs after matching AST nodes is sufficiently optimized. Thus, we propose to improve the matching algorithm. Figure 2 gives an overview of our methodology. In order to improve the matching algorithm, we use not only AST information but also line-diff information calculated by `diff` command. Before applying GumTree's matching algorithm, our methodology matches AST nodes based on the line-diff information.

From the line-diff information, lines in the source code are divided into *edited* lines and *non-edited* lines. AST nodes in the *edit* lines are matched with only nodes in the *edited* lines in the other AST. AST nodes in the *non-edited* lines are matched in the same way.

Since not all AST nodes are matched, non-matched AST nodes are matched by GumTree's algorithm after our

```
1.  /**
2.   * Removes all mappings from this map.
3.   */
4.  public void clear() {
5.    modify();
6.
7.    nodeCount = 0;
8.-   rootNode[KEY] = null;
9.-   rootNode[VALUE] = null;
10. }
```
original source code

```
1.  /**
2.   * Removes all mappings from this map.
3.   */
4.  public void clear() {
5.    modify();
6.
7.    nodeCount = 0;
8.+   rootNode[KEY.ordinal()] = null;
9.+   rootNode[VALUE.ordinal()] = null;
10. }
```
edited source code

Fig. 3. The results of `diff` command

matching algorithm. GumTree's architecture is divided to some modules (e.g., matcher, action generater). By giving matching information to GumTree's matcher, it tries to match unmatched nodes. From the matching results of our methodology and GumTree, an action generator of GumTree calculates an ES.

A. Approach to non-edited Lines

We assume that lines which are judged as *non-edited* by `diff` command are not edited by developers. Each *non-edited* line has a perfectly matching line between two versions of source code. In *non-edited* lines, AST nodes that share the same type get matched.

Figure 3 shows the results that `diff` command has been applied to the source code in Figure 1. For example, the line 4 in the both source code is *non-edited* line because the line is a perfectly matching line. Thus, our methodology matches each node in the line 4. By this matching algorithm, `public` node is matched, while GumTree's algorithm cannot match the nodes.

B. Approach to edited Lines

We assume that many nodes which should be matched are slightly moved between two versions. In this assumption, the nodes in the deleted and inserted lines are matched with each other.

There are *non-edited* lines above and below *edited* lines, thus, based on this information, our methodology makes corresponding relationships between the deleted and inserted chunks.

For example, in the original source code, *delete* actions are generated for the lines 8 and 9. The lines 7 and 10 which are above and below the lines, are *non-edited* lines. In the edited source code, there is a line which is completely matched with the line 7 of the original source code, and then, the line is the line 7 of the edited source code. In the same way, the line 10 of the both source code is matched with each other. From those information, our methodology matches a deleted chunk between the lines 7 and 10 of the original source code and an inserted chunk between the lines 7 and 10 of the edited source code. After matching deleted chunks and inserted chunks, our methodology matches the nodes within the deleted chunks and the inserted chunks.

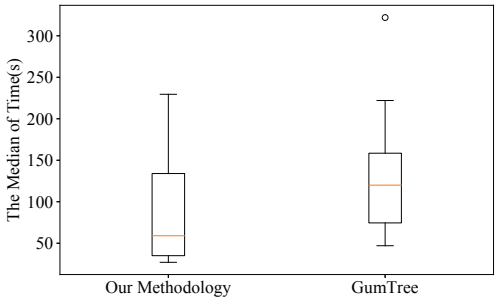

Fig. 4. Results of Exp-2: the medians of time for each difference

GumTree calculates the Jaccard similarity of two versions of subtrees for matching nodes. Our methodology also calculates the Jaccard similarity for matching nodes.

V. Experiment

Toevaluate our methodology, we compared it with GumTree in the following two experiments:

Exp-1: investigating whether ESs of our methodology are shorter than the ones by GumTree, and

Exp-2: investigating whether ESs of our methodology are more helpful to understand code differences than the ones by GumTree.

We did not compare our methodology with IJM [7] because IJM changes the structure of ASTs. It is difficult to compare fairly ES generation techniques that use ASTs of different structures.

A. Exp-1

We experimented to investigate whether our methodology generates shorter ESs than GumTree. Firstly, GumTree was applied to all Java files in all commits of some Java OSS in Table I. For the code differences including 50 or longer ESs, our methodology was also applied. Code differences of short ESs have less room for improvement. Thus, we did not compare them.

The comparison results are shown in Table I. In all the OSS, the sum and the median of the ESs which were generated by our methodology are smaller than the ones by GumTree. Moreover, for 30∼50% of the differences, our methodology generated shorter ESs than GumTree.

TABLE I
RESULTS OF EXP-1

OSS	commits	differences*1	sum		median		ratio of shorter ESs	
			GumTree	Ours	GumTree	Ours	GumTree	Ours
activemq	10,021	5,326	1,001,361	981,435	110.0	108.0	8%	33%
commons-collections	3,050	1,640	426,383	394,574	122.0	115.0	3%	50%
commons-io	2,116	782	180,821	176,596	124.5	117.5	8%	34%
commons-lang	5,263	2,375	627,934	600,855	130.0	126.0	5%	37%
commons-math	6,317	4,435	1,276,316	1,223,448	131.0	128.0	7%	39%
hibernate-search	7,163	4,364	766,425	750,287	105.0	103.0	13%	38%
spring-roo	6,132	4,880	1,296,941	1,271,632	130.0	129.0	12%	42%

*1 In this table, term "difference" corresponds to changes in a Java source file in a commit.

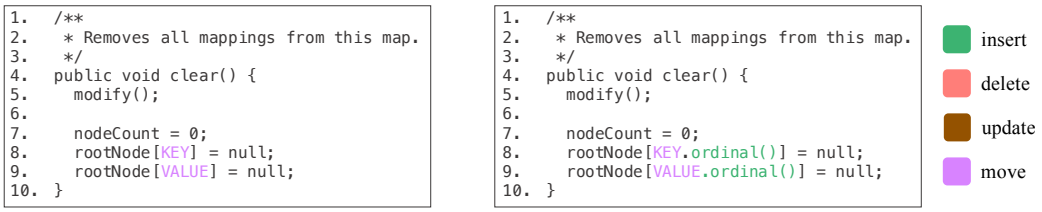

Fig. 5. An edit script generated by our methodology (the length is 6)

The results do not follow normal distribution. Thus, we used the Wilcoxon signed rank test to check whether the results are statistically significant. Since we obtained p-values ≤ 0.05 from all the results, the results of our methodology were significantly different from GumTree.

B. Exp-2

We experimented with 14 research participants to evaluate whether ESs of our methodology are more helpful to understand code differences than the ones by GumTree. The research participants consist of four undergraduate students, nine graduate students, and one professor. All of the participants are accustomed to using Git and Java programming.

We chose the 15 code differences from commons-math. The differences were selected by the following conditions.

- The size of GumTree's ESs is from 50 to 200.
- It is the top 15 code differences where GumTree and our methodology generated most different size of ESs.

The reason why we limited the size is that we did not want to make a big burden for the participants. We used the tool included in GumTree to confirm the ESs. The tool visualizes ESs via a Web browser.

At first, to learn how to use the tool, the research participants checked three code differences other than the 15 target code differences. Then, they checked the 15 code differences, with measuring the time and then they answered how the source code had been changed.

The participants were divided into two groups, X and Y. Each group checked each of the 15 code differences by either of the GumTree or our methodology. X group used GumTree for the odd-numbered code differences while Y group used our methodology for them. For even-numbered code differences, the two groups used the other tool.

Figure 4 shows the boxplot of the time that the participants took to understand each code difference. From those results, the median for all the code differences is less than a half of GumTree. We found that there is a significant difference of execution between our methodology and GumTree by using the Mann-Whitney U test.

VI. Discussion

We found that our methodology generates shorter ESs than GumTree from Exp-1. The main reason is that our methodology appropriately matches AST nodes in *non-edited* lines. We discuss the differences of the source code in Figure 1. In GumTree, the node of method declaration in the line 4 is not matched, so that GumTree outputs an ES that the node was deleted and inserted. Moreover, because the node of method declaration is not matched, GumTree recognizes that the parent node of the node in the line 5 gets changed. Thus, GumTree outputs an ES that the node in the line 5 was moved. Figure 5 shows the results that our methodology is applied to the source code in Figure 1. The ES is more helpful to understand than Figure 1. For example, the line 4 has been regarded as *non-edited* by `diff` command, then, our methodology matches the node in the line 4. As a result, our methodology can find that the node in the line 5 has the same parent between two versions, then, our methodology generates an ES that the node is not moved. In this way, matching a node affects other nodes, then, we found that our methodology generated shorter ESs.

From the results of Exp-2, we found that ESs of our methodology are more helpful to understand code differences. The code differences which took a shorter time to understand with a statistical significance, consist of simple changes such as insertion of `@Override`. On the other hand, the code differences which we could not obtain a statistical significance consist of complicated changes. From those results, our methodology is more helpful for simple changes than complicated changes.

VII. Threats to Validity

We conducted the experiments on Java OSS. While we expect the same results for other programming languages, we did not confirm that our methodology is more helpful than GumTree.

VIII. Conclusion

In this paper, by using the line-based differences, we proposed a methodology which generates shorter and more helpful ESs than GumTree. To evaluate it, it was applied to 7 OSS projects, and then, we succeeded in generating shorter ESs. We also experimented with 14 research participants, and then we confirmed that ESs of our methodology are more helpful to understand code differences.

As a future work, we are going to apply our methodology other programming languages than Java.

Acknowledgment

This work was supported by MEXT/JSPS KAKENHI 17H01725.

References

[1] E. W. Myers, "Ano(nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 1, pp. 251–266, Nov 1986. [Online]. Available: https://doi.org/10.1007/BF01840446

[2] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing," in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, 2014, pp. 313–324. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642982

[3] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig, "Api code recommendation using statistical learning from fine-grained changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, 2016, pp. 511–522. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950333

[4] C. Macho, S. Mcintosh, and M. Pinzger, "Extracting build changes with builddiff," in *Proceedings of the 14th International Conference on Mining Software Repositories*, ser. MSR '17, 2017, pp. 368–378. [Online]. Available: https://doi.org/10.1109/MSR.2017.65

[5] J. Yi, U. Z. Ahmed, A. Karkare, S. H. Tan, and A. Roychoudhury, "A feasibility study of using automated program repair for introductory programming assignments," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, 2017, pp. 740–751. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106262

[6] Q. Hanam, F. S. d. M. Brito, and A. Mesbah, "Discovering bug patterns in javascript," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, 2016, pp. 144–156. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950308

[7] V. Frick, T. Grassauer, F. Beck, and M. Pinzger, "Generating accurate and compact edit scripts using tree differencing," *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 264–274, 2018.

[8] G. Dotzler and M. Philippsen, "Move-optimized source code tree differencing," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016, 2016, pp. 660–671. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970315

[9] Y. Higo, A. Ohtani, and S. Kusumoto, "Generating simpler ast edit scripts by considering copy-and-paste," in *The 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE2017)*, 10 2017, pp. 532–542.

[10] G. de la Torre, R. Robbes, and A. Bergel, "Imprecisions diagnostic in source code deltas," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18, 2018, pp. 492–502. [Online]. Available: http://doi.acm.org/10.1145/3196398.3196404

[11] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96, 1996, pp. 493–504. [Online]. Available: http://doi.acm.org/10.1145/233269.233366