HomoTR: Online Test Recommendation System Based on Homologous Code Matching

Chenqian Zhu [1], Weisong Sun [1], Qin Liu [1] *, Yangyang Yuan [1], Chunrong Fang [1], Yong Huang [2]
[1] State Key Laboratory for Novel Software Technology, Nanjing University, China
[2] Mooctest Inc., Nanjing, China
*qinliu@nju.edu.cn

ABSTRACT

A growing number of new technologies are used in test development. Among them, automatic test generation, a promising technology to improve the efficiency of unit testing, currently performs not satisfactory in practice. Test recommendation, like code recommendation, is another feasible technology for supporting efficient unit testing and gets increasing attention. In this paper, we develop a novel system, namely HomoTR, which implements online test recommendations by measuring the homology of two methods. If the new method under test shares homology with an existing method that has test cases, HomoTR will recommend the test cases to the new method. The preliminary experiments show that HomoTR can quickly and effectively recommend test cases to help the developers improve the testing efficiency. Besides, HomoTR has been integrated into the MoocTest platform successfully, so it can also execute the recommended test cases automatically and visualize the testing results (e.g., Branch Coverage) friendly to help developers understand the process of testing. The demo video of HomoTR can be found at https://youtu.be/_227EfcUbus.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Online Programming, Homologous Code Matching, Test Recommendation

ACM Reference Format:
Chenqian Zhu [1], Weisong Sun [1], Qin Liu [1] *, Yangyang Yuan [1], Chunrong Fang [1], Yong Huang [2]. 2020. HomoTR: Online Test Recommendation System Based on Homologous Code Matching. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3324884.3415296

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3415296

1 INTRODUCTION

Nowadays, software has become an indispensable part of people's life. With the variety of application scenarios, software is increasingly complex, which makes it more prone to errors. Fatal errors may lead to serious economic loss, or even cause the demise of software. Therefore, software testing is needed to ensure its correctness and availability and becomes more important.

Unit testing is one of the most basic and important software testing methods, it will directly affect the subsequent testing, and ultimately have a significant impact on the quality of the software. However, developers tend to focus on production code development, thus ignoring software testing development and the training of test case writing [3]. Besides, writing test cases is a boring and time-consuming task. Therefore, some advanced test technologies (e.g., automatic test generation), are born to improve the testing efficiency. Automatic test generation seems promising, but it still has many limitations in practical large-scale applications [1]. Thus, other technologies, such as test recommendation, are needed to help developers improve the testing efficiency.

However, the existing test recommendation technologies[2, 4, 5], have some shortcomings. For example, only the existing test cases in the project which developers participate in can be recommended [4], or recommendation accuracy declines due to the limitation of metric (e.g., only use method signature as the metric) [2, 5]. If we can use more test cases from more projects as the corpus or use more effective metrics, the test recommendation accuracy will be improved.

In this paper, we develop a novel system, namely HomoTR, which implements online test recommendations by measuring the homology of two methods. Homology means different methods share the same or similar structure or functionality. HomoTR is now integrated into the MoocTest[1] platform, which is a software testing education platform. When developers want to write test cases, they can trigger HomoTR. It will recommend test cases automatically. When developers run the test cases, it will visualize the testing results (e.g., Branch Coverage) friendly to help developers understand the process of testing. At present, HomoTR has been put into practical application and obtained more positive feedback.

2 METHODOLOGY

2.1 Overview

As shown in Figure 1, HomoTR contains two parts: test recommendation and testing results visualization. In test recommendation, HomoTR searches for test cases from corpus by homologous method

[1]http://www.mooctest.net

matching. HomoTR then executes the recommended test cases automatically, and the testing results are visualized friendly, aiming at helping developers understanding testing processes.

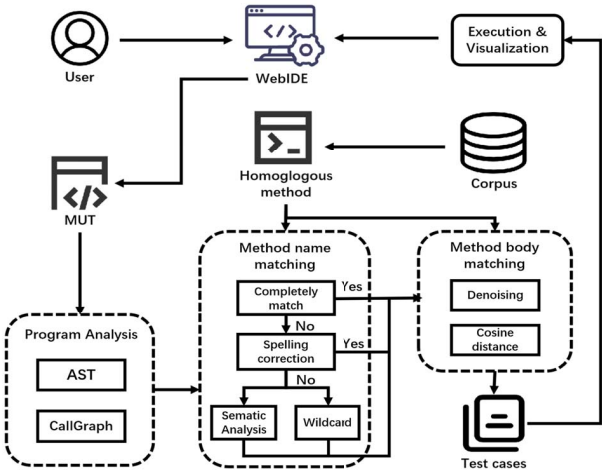

Figure 1: Overview of HomoTR

2.2 Test Recommendation

The corpus has been stored in ElasticSearch on MoocTest. Test recommendation process can be triggered when the developer wants to test the codes which are not covered by test cases.

The corpus stores the information of the methods, including class name, method name, method input/output (I/O) structure, method code, corresponding test case, and so on. Method input structure includes the quantity, sequence and types of the parameters. Method output structure includes the type of the return type of the method (a java method only has one return value).

This test recommendation process is implemented in the following steps:

1) Do program analysis for method under test (MUT) to extract the name, parameters, implementation, and other information.

2) Use the name and I/O structure of MUT to match methods in the corpus strictly. If successful, return the matching results (up to 10 methods). Go to step 6.

3) If step 2 fails, based on matching I/O structure, use Levenstain distance to do spelling correction for the method name and do matching. If successful, return the matching result (up to 20 methods). Go to step 6.

4) If step 3 fails, based on matching I/O structure and the help of word2vec-GoogleNews-vectors, semantic similarity analysis is carried out between the names of methods in the corpus and MUT. Then 10 methods with the highest similarity and reaching the threshold are selected. The threshold is selected manually based on hundreds of results of semantic similarity analysis experiments.

5) Based on matching I/O structure, add wildcard characters to the method name to get at most 10 methods. Return these methods together with the methods selected in step 4. Go to step 6.

6) Use cosine distance to measure the text similarity between the code of MUT and each returned method. Rank all the cosine distance. The method with the highest code similarity will be considered as homologous with MUT, and the test case of the method will be recommended to test MUT.

2.3 Testing Results Visualization

The functionality of the testing results visualization is to color the code lines according to the test coverage of the code. It is aimed at making the users clearly see whether the codes are covered by test cases. After users running test cases, OpenClover[2] will measure code coverage. According to the code coverage information, code coloring will be triggered by WebIDE. If lines are not covered, the area beside the code line number will be paint in red. If lines are partially covered, for example, there are two MUTs in one line but only one of them has been tested, the color will be yellow. If lines are completely covered, the color will be green.

3 EXPERIMENT

To evaluate the effectiveness of HomoTR, we need to prove that the performance of HomoTR is better than SENTRE [2].

Experiment subjects. In our experiment, we try to verify the performance of HomoTR is better than SENTRE. SENTRE is a test search engine which is not open-source and publicly available. Thus, we implement the test recommendation strategy based on the signature matching and relaxation algorithm proposed by SENTRE.

Table 1: List of experimental items.

Project No.	Project Name	# method	# TC	Project Usage
1	interviewcoder/leetcode	608	2083	build corpus
2	rekinyz/LeetCode	152	560	build corpus
3	interviewcoder/interviewbit	131	339	build corpus
4	interviewcoder/lintcode	47	30	build corpus
5	interviewcoder/geeksforgeeks	48	50	build corpus
6	gouthampradhan/leetcode	1025	0	project under test
7	nagajyothi/InterviewBit	633	0	project under test

As shown in Table 1, we conduct the experiment on seven open-source projects from GitHub, including the realization of nearly a thousand programming problems on the programming website, such as Leetcode[3], Lintcode[4], InterviewBit[5], and GeeksForGeeks[6]. Different platforms and different people lead to different implementations of similar programming problems. Project No.1 - 5 are used as raw materials to build corpus with the help of TeSRS [5] and MAF [6]. Project No.6 - 7 only include realization, which are used as MUT.

Evaluation metrics. In our experiment, we employ the number of matching methods and unmatched methods, the number of successful and failed test cases, the test accuracy, and test coverage adequacy [7] which includes class coverage (CC), method coverage (MC), and line coverage (LC) as metrics. The number of matching or unmatched methods is the number of methods under test (MUTs) that succeed or fail to match similar methods. The number of failed and successful test cases is the statistics of execution results obtained by running test cases after simple manual adjustment after

[2]https://openclover.org
[3]https://leetcode-cn.com/
[4]https://www.lincode.com
[5]https://www.interviewbit.com
[6]https://www.geeksforgeeks.org

test recommendation. The test accuracy is the percentage of successful test cases and the total number of recommended test cases. CC, MC, and LC are the results of the collection of test coverage of project source code.

Experiment results. In our experiment, HomoTR and SENTRE are respectively used to recommend test cases for all the public methods in Project No.6 - 7. The results are shown in Table 2.

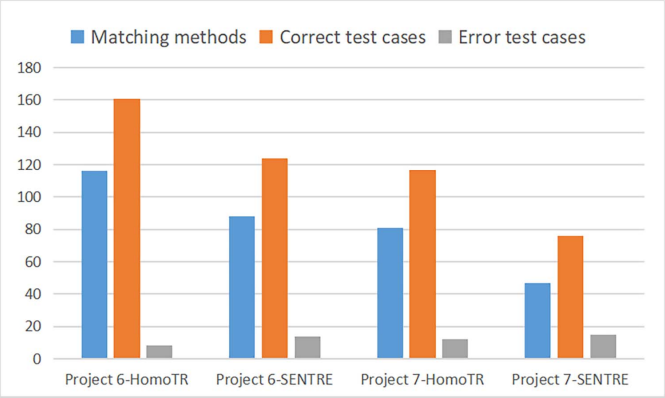

Figure 2: Comparison of the number of matching results and test cases.

As is shown in Figure 2 and Figure 3, for Project No.6 - 7, the number of matching methods of HomoTR is significantly more than that of SENTRE, thus more test cases are recommended. Besides, from the data in Table 2, it can be seen that the test execution results of HomoTR are obviously superior in CC, MC, and LC.

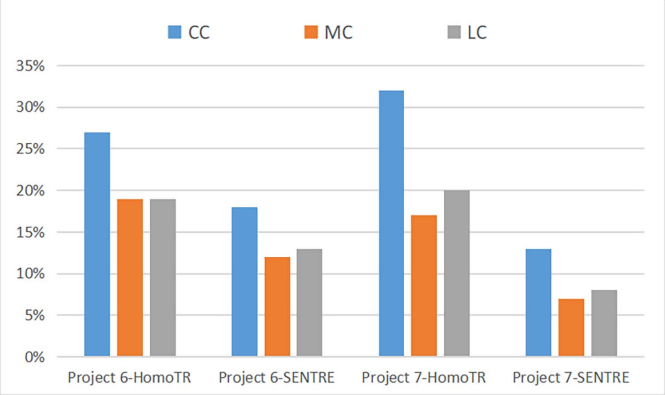

Figure 3: Comparison of coverage.

The test accuracy column in Table 2 is used to measure whether the test case can correctly test MUT, that is, the accuracy of test recommendation. It can be seen that the result of SENTRE is between 80% - 90%, while HomoTR can maintain 90% - 95% accuracy on the premise of recommending more test cases.

The defect of SENTRE is that when the class name of matching method is similar to that of MUT, and if the I/O structure of the method is same with that of MUT, it may determine that the method is similar to MUT and recommend wrong test cases even though their purpose is different. HomoTR can comprehensively consider the similarity between method signature and method body code to recommend the test cases of the homologous method.

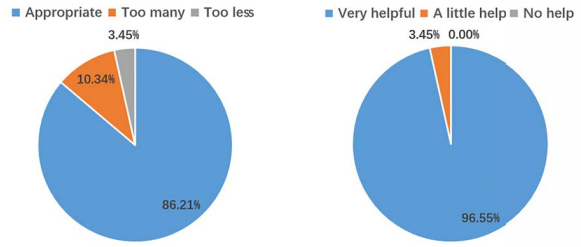

Figure 4: The adequacy and helpfulness of recommended test cases.

Besides, we invite 29 students who major in software engineering and have test experience to solve test problems by using HomoTR, and fill in feedback.

The statistics of feedback results are shown in Figure 4, 86.21% of them think that, on the basis of meeting the test coverage goals, the number of test cases recommended by HomoTR is appropriate. 96.55% of them say that HomoTR is very helpful in improving the efficiency of test writing, which indicates that HomoTR obtains high user satisfaction.

The experiment also finds that most developers can complete the problem in 10-30 minutes and reach coverage requirements after using HomoTR, while it takes 1-2 hours to achieve similar results without using HomoTR according to the former exam results of software testing courses. To sum up, HomoTR can recommend test cases that are easy to understand, which will bring great help to test writing.

In brief, the experiment shows that, compared with SENTRE, HomoTR can match more MUTs, recommend more test cases, and maintain more than 90% of the test accuracy. It also has the expected performance for CC, MC, and LC of source code.

4 USAGE

In this section, we will show the usage of HomoTR.

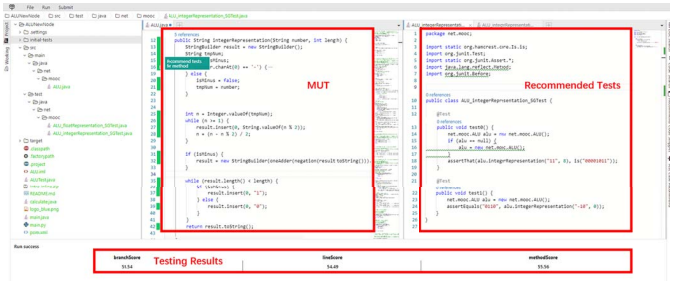

Figure 5: Test recommendation.

Test recommendation. Figure 5 shows the functionality of test recommendation. We can see the area of MUT. When the user enters the WebIDE and wants HomoTR to recommend test cases, make the mouse hover over the source code line number for 2

Table 2: Comparative experimental result.

Project No.	Algorithm	matching methods	unmatched methods	failed cases	successful cases	Test accuracy	CC	MC	LC
6	HomoTR	116	909	8	161	95%	27%	19%	19%
	SENTRE	88	937	14	124	90%	18%	12%	13%
7	HomoTR	81	525	12	117	91%	32%	17%	20%
	SENTRE	47	586	15	76	84%	13%	7%	8%

seconds to trigger test recommendation button "Recommend tests for method". Click this button, HomoTR will recommend test cases for the method in this line. We can see the recommended test cases in the area of *Recommended Tests*. Move the mouse away for 2 seconds, and the button will disappear automatically. Run the test cases, we can see the test results at the bottom.

Testing Results Visualization. Figure 6 shows an example of testing results visualization. The color of the area beside the line number represents the current coverage of each line of source code. Red indicates that they are not covered, yellow means only some branches are covered, green indicates that they have been completely covered, and invalid lines, such as empty lines and comments, are not colored.

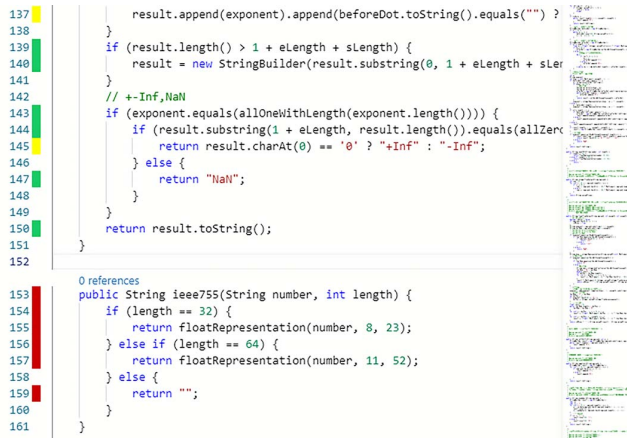

Figure 6: Testing results visualization.

5 RELATED WORK

With the development of software testing, test case recommendation [2, 4, 5] gets more and more attention. *Test Recommender* [4] recommends test cases from the project which developers participate in. It is useful for inexperienced developers to learn writing test cases, but the project should have many test cases. So it do not work for a new project. *TeSRS* [5] gets test snippets from superior crowdsourcing test scripts by program slicing, and recommends test cases by method signature matching to assist test novice in learning unit testing. Werner et al. [2] built a test case search engine SENTRE which contains lots of test cases collected from the open web. Compared with the work [4], their technique can recommend test cases from different projects. SENTRE uses method signatures and relaxation algorithm to recommend test cases. However, both TeSRS and SENTRE will cause a drop in accuracy when facing the change of the method name.

6 CONCLUSION

The paper proposes an online test recommender HomoTR. HomoTR recommends the test cases from the method in our corpus to the new method under test which shares homology with the former. Besides, HomoTR provides testing results visualization to show the testing details. The preliminary results show that HomoTR can help developers to write basic test cases quickly and effectively.

6.1 Future work

The system still needs to be improved. Future work will mainly focus on the semantic similarity measurement of method names, similarity measurement of the code, and the automatic program slicing technology. First, we just use word2vec-GoogleNews-vectors as the word vector model and manually select the threshold of semantic similarity. We may use other artificial intelligence technologies to improve semantic similarity measurement of methods names, and dynamically adjust the similarity threshold according to the change of corpus to achieve a better matching effect. Second, even though the existing code body similarity measurement by using cosine distance seems to have good effects, we may compare it with the code body similarity measurement by using abstract syntax tree, control flow graph, or program dependency analysis to find whether there is a better way. Third, the code corpus of HomoTR is limited by the existing automatic slicing technology. We need to check the test case slice manually to ensure its correctness. In the future, we may put program dependencies into use to improve slicing technology and build a better code corpus.

ACKNOWLEDGEMENTS

This work is supported partially by National Key R&D Program of China (2018YFB1403400), National Natural Science Foundation of China(61802171), and Fundamental Research Funds for the Central Universities(14380021,14913413). Qin Liu is the corresponding author.

REFERENCES

[1] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2015. Does Automated Unit Test Generation Really Help Software Testers? A Controlled Empirical Study. *TOSEM* 24, 4 (2015), 23:1–23:49.
[2] Werner Janjic and Colin Atkinson. 2013. Utilizing software reuse experience for automated test recommendation. In *Proceedings of the 8th AST Workshop*. 100–106.
[3] Raphael Pham, Stephan Kiesling, Olga Liskin, Leif Singer, and Kurt Schneider. 2014. Enablers, Inhibitors, and Perceptions of Testing in Novice Software Teams. In *Proceedings of the 22th FSE*. 30–40.
[4] Raphael Pham, Yauheni Stoliar, and Kurt Schneider. 2015. Automatically recommending test code examples to inexperienced developers. In *Proceedings of the 10th ESEC/FSE*. 890–893.

[5] Ruixiang Qian, Yuan Zhao, Duo Men, Yang Feng, Qingkai Shi, Yong Huang, and Zhenyu Chen. 2020. Test recommendation system based on slicing coverage filtering. In *Proceedings of the 29th ISSTA*. 573–576.
[6] Weisong Sun, Xingya Wang, Haoran Wu, Ding Duan, Zesong Sun, and Zhenyu Chen. 2019. MAF: method-anchored test fragmentation for test code plagiarism detection. In *Proceedings of the 41th ICSE(SEET)*. 110–120.
[7] Hong Zhu, Patrick A. V. Hall, and John H. R. May. 1997. Software Unit Test Coverage and Adequacy. *CSUR* 29, 4 (1997), 366–427.