Accelerating Redundancy-Based Program Repair via Code Representation Learning and Adaptive Patch Filtering

Chen Yang
College of Intelligence and Computing, Tianjin University
Tianjin, China
yangchenyc@tju.edu.cn

ABSTRACT

Automated program repair (APR) has attracted extensive attention and many APR techniques have been proposed recently, in which redundancy-based techniques have achieved great success. However, they still suffer from the efficiency issue mainly caused by the inaccuracy of measuring code similarity, which may produce meaningless patches that hinder the generation and validation of correct patches. To solve this issue, we propose a novel method AccPR, which leverages code representation to measure code similarity and employs adaptive patch filtering to accelerate redundancy-based APR. We have implemented a prototype of AccPR and integrated it with a SOTA APR tool, SimFix, where the average improvement of efficiency is 47.85%, indicating AccPR is promising.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools.*

KEYWORDS

representation learning, patch filtering, automated program repair

ACM Reference Format:
Chen Yang. 2021. Accelerating Redundancy-Based Program Repair via Code Representation Learning and Adaptive Patch Filtering. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3468264.3473496

1 INTRODUCTION

Automated program repair (APR) has attracted extensive attention in recent years [3, 17]. Redundancy-based APR techniques [1, 15] (such as GenProg [12], CapGen [21], SCRepair [5], CRSearcher [19], ssFix [22], SimFix [7] and Refactory [4]) are one of the most important categories in this field and have achieved promising results [6, 7, 22]. Specifically, they first search for a set of code snippets similar to a suspicious faulty code from a code base as references, and then generate a bunch of candidate patches to be validated one by one against a test suite. Although redundancy-based APR can successfully fix a number of faults as demonstrated in existing studies [7, 14, 17, 22], they still suffer from the efficiency issue [13], i.e., requiring much time to repair a bug. For example, a state-of-the-art (SOTA) redundancy-based APR technique (SimFix [7]) sets a time budget of 5 hours for each fault [7]. APR efficiency has important influence on promoting these techniques into practice and significantly affects debugging performance. Therefore, it is important to accelerate redundancy-based APR techniques.

Through deeply investigating redundancy-based APR techniques, there are two major problems affecting their efficiency [11, 13]. First, the searched similar code snippets are not accurate for generating correct patches. Specifically, current redundancy-based APR techniques mainly depend on syntactic features (e.g., AST) to measure code similarity, which cannot capture semantic information and thus the measured similarity is not accurate. This leads to generating many incorrect patches and also wasting much time to validate them [18, 20, 23]. We call it *inaccurate similarity problem*. Second, the order of validating generated patches used currently leads to wasting much time to validate many similar but incorrect patches before the correct one. We call it *(patch) order problem*.

To boost the efficiency of redundancy-based APR techniques, we propose a novel method, called **AccPR**, to overcome the above problems. Regarding the inaccurate similarity problem, AccPR incorporates representation learning to extract deep semantic information from code snippets to improve the accuracy of similarity measurements, inspired by Dantas A et al.[2]. Here, we adopt ASTNN [24], a SOTA code representation learning method, in AccPR. Regarding the order problem, specifically, for a candidate code snippet, a lot of patches can be generated. If a patch is incorrect, patches similar to it are likely to be incorrect, which has never been considered by existing techniques. Therefore, we design an adaptive patch filtering strategy to relieve the order problem. Specifically, according to the validation feedback, AccPR filters out those patches that are highly similar to confirmed incorrect ones. In addition, regarding the set of generated patches for a suspicious faulty code snippet, AccPR ranks patches as the descending order of their similarity since simple patches are more likely to be correct [10, 16].

We have implemented a prototype of AccPR and integrated it with a SOTA redundancy-based APR technique, SimFix [7]. Then, we conducted a preliminary study to investigate its performance on the benchmark, Defects4J [8]. Experimental results show that AccPR improves the efficiency of SimFix by 47.85% on average, demonstrating its effectiveness.

To sum up, this work makes three major contributions: 1) A novel method to accelerate redundancy-based APR via incorporating representation learning to improve the similarity measurement and an adaptive filtering strategy to save validation time; 2) A prototype

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473496

of AccPR, which has been integrated with a SOTA redundancy-based APR tool, SimFix. 3) A preliminary study on a benchmark Defects4J, demonstrating the proposed method is indeed promising.

2 APPROACH

AccPR follows the workflow of redundancy-based APR techniques. It is designed to solve the inaccurate similarity and patch order problems by optimizing the similarity measurement via code representation and applying an adaptive patch filtering (see Figure 1).

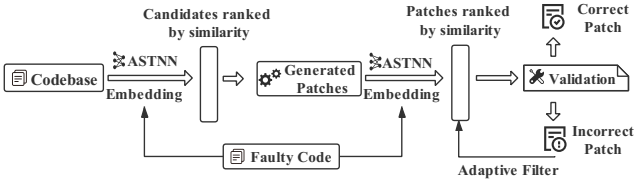

Figure 1: Overview of AccPR

2.1 Similarity Measurement

2.1.1 Code Representation. We incorporate ASTNN [24], a code representation model that can effectively extract semantic information of code snippets in AccPR. It is suitable to our scenario since it is a general-purpose model that is not limited to any specific tasks.

2.1.2 Similarity Measurement. We apply learned embeddings to capture the similarity between code snippets. Given two code snippets m and n, we first leverage ASTNN to embed them into vectors, and then employ *1-Norm* to compute their similarity, which is defined as: $Simi(m,n) = ||astnn(m) - astnn(n)||$, where $astnn(*)$ refers to the embedding result of the given code snippet.

2.1.3 Candidate Code Snippets Ranking. Given a suspicious faulty code snippet n, AccPR identifies a set of similar code snippets $\mathbb{M}$ as candidates for patch generation. That is, for each $m \in \mathbb{M}$, we compute its similarity with n by $Simi(n,m)$. Then, AccPR ranks all candidate code snippets as the descending order of these similarity results, since the code snippets having higher similarities with the faulty code are more likely to generate the correct patch [7, 9, 16].

2.2 Adaptive Patch Filtering

Given a set of similar code snippets, multiple patches $\mathbb{P}$ shall be generated for a faulty snippet n, then AccPR performs a multi-level patch prioritization and filtering strategy according to the following rules to make the correct patch be validated as early as possible.

- R1 (**Similarity**): Patches that have higher similarities with the faulty code (i.e., $Simi(p,n)$ for each $p \in \mathbb{P}$) are ranked higher since patches are usually simple.
- R2 (**Adaptive Filtering**): Patches that are similar to a known incorrect patch under a given threshold will be filtered since they are highly likely to be incorrect as well.

R1 is inspired by syntactic distance proposed by Mechtaev et al. [10, 16], while R2 targets to filter out potential incorrect patches as early as possible with similar inconsequential modifications. AccPR validates patches according to the candidate patch list, which is adaptively updated by applying the above two strategies online.

3 PRELIMINARY STUDY

To investigate the performance of AccPR, we integrated it with a SOTA redundancy-based APR tool, i.e., SimFix, and conducted our experiment on Defects4J [8](v1.2). The experimental environment is Ubuntu 7.5.0 with Oracle JDK 1.8.

In this preliminary study, we only focus on faults that have been fixed, and employ two metrics: 1) *Time Cost*: time for online patch generation, patch embedding and validation; 2) *NPC*: the **N**umber of **P**atch **C**andidates validated before the correct patch[13].

Table 1: Experimental Results on Defects4J

Bug	Time	NPC	Bug	Time	NPC
Ch1	**10.76** (18.32)	**149** (213)	Cl57	**4.22** (4.69)	18 (**15**)
Ch3	Fail (**39.21**)	-	Cl62	**10.55** (66.11)	**121** (534)
Ch7	**9.08** (20.61)	**153** (591)	Cl73	**1.49** (1.65)	**1** (2)
Ch20	0.93 (**0.87**)	17 (**5**)	Cl115	**2.41** (2.78)	**2** (3)
M5	**2.46** (2.79)	**1** (**1**)	L16	**7.62** (Fail)	-
M50	**2.28** (2.50)	**1** (**1**)	L27	**15.56** (26.27)	**569** (985)
M53	6.89 (**4.51**)	14 (**2**)	L33	**1.23** (2.72)	**22** (62)
M57	**2.02** (5.81)	**4** (97)	L39	**28.94** (42.00)	**659** (977)
M59	**2.71** (3.28)	**9** (34)	L41	**2.53** (8.07)	**39** (154)
M63	**2.66** (3.35)	**16** (41)	L50	**12.04** (25.89)	**250** (348)
M70	**0.36** (0.58)	**1** (**1**)	L58	0.35 (**0.32**)	**1** (**1**)
M71	**6.11** (10.80)	**50** (142)	L60	**3.63** (7.82)	**55** (112)
M75	**0.45** (0.89)	**1** (8)	T7	**8.83** (Fail)	
M79	11.64 (**9.58**)	336 (**127**)			
Avg.	Time: **5.91** (11.34)		NPC: **99** (186)		

Abbreviation- Ch: Chart, M: Math, Cl: Closure, L: Lang, T: Time.
X(Y)- X is the result of AccPR, while Y is the result of SimFix.

Table 1 shows the experimental results. We find that AccPR significantly improves the repair efficiency of SimFix, i.e., requiring less time to repair a bug. More concretely, the average repair time is reduced from 11.34 to 5.91 minutes, achieving 47.85% repair time reduction on average. Futhermore, the number of candidate patches is also significantly reduced by 46.48% on average. Particularly, AccPR successfully repaired two more bugs that SimFix failed to repair. Unfortunately, AccPR failed to repair Chart-3 because the desired patch was mistakenly filtered out due to an incorrect similar patch that was validated earlier, which could be solved by a better patch generation or filtering strategy and is worth studying more. AccPR did not yield better results on bugs that could have been fixed quickly by SimFix because of the extra overhead associated with the patch embedding process. However, the initial promising result on most cases demonstrates the effectiveness of AccPR. Overall, the experimental results show that AccPR is promising to accelerate existing redundancy-based APR techniques by overcoming the *inaccurate similarity* and *patch order* problems.

4 CONCLUSION AND FUTURE WORK

To accelerate redundancy-based APR techniques, we proposed a novel method (named AccPR) by leveraging code representation learning for better similarity measurement and designing a novel adaptive patch filtering strategy. To evaluate its effectiveness, we implemented a prototype of it and integrated it with SimFix. The initial experimental results on Defects4J demonstrate the effectiveness of AccPR. In the future, we will further improve AccPR by exploring more advanced code representation learning methods and evaluate it on a wider range of benchmarks and APR tools.

ACKNOWLEDGMENTS

Special thanks to Junjie Chen and Jiajun Jiang for the supervision.

REFERENCES

[1] Zimin Chen and Martin Monperrus. 2018. The remarkable role of similarity in redundancy-based program repair. *arXiv preprint arXiv:1811.05703* (2018).

[2] Altino Dantas, Eduardo F. de Souza, Jerffeson Souza, and Celso G. Camilo-Junior. 2019. Code Naturalness to Assist Search Space Exploration in Search-Based Program Repair Methods. In *Search-Based Software Engineering*, Shiva Nejati and Gregory Gay (Eds.). Springer International Publishing, Cham, 164–170.

[3] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2019. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering* 45, 1 (2019), 34–67. https://doi.org/10.1109/TSE.2017.2755013

[4] Yang Hu, Umair Z. Ahmed, Sergey Mechtaev, Ben Leong, and Abhik Roychoudhury. 2019. Re-Factoring Based Program Repair Applied to Programming Assignments. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 388–398. https://doi.org/10.1109/ASE.2019.00044

[5] Tao Ji, Liqian Chen, Xiaoguang Mao, and Xin Yi. 2016. Automated Program Repair by Using Similar Code Containing Fix Ingredients. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 197–202. https://doi.org/10.1109/COMPSAC.2016.69

[6] Jiajun Jiang, Luyao Ren, Yingfei Xiong, and Lingming Zhang. 2019. Inferring Program Transformations From Singular Examples via Big Code. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 255–266. https://doi.org/10.1109/ASE.2019.00033

[7] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping program repair space with existing patches and similar code. In *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*. 298–309.

[8] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. San Jose, CA, USA, 437–440.

[9] A. Koyuncu, K. Liu, Tegawendé F Bissyandé, D. Kim, J. Klein, M. Monperrus, and Y. L. Traon. 2020. FixMiner: Mining Relevant Fix Patterns for Automated Program Repair. *Empirical Software Engineering* 25, 3 (2020), 1980–2024.

[10] Xbd Le, D. H. Chu, D. Lo, C. L. Goues, and W. Visser. 2017. S3: Syntax-and Semantic-Guided Repair Synthesis via Programming by Examples. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*.

[11] Claire Le Goues, Stephanie Forrest, and Westley Weimer. 2013. Current challenges in automatic software repair. *Software quality journal* 21, 3 (2013), 421–443.

[12] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering* 38, 1 (2012), 54–72. https://doi.org/10.1109/TSE.2011.104

[13] Kui Liu, Shangwen Wang, Anil Koyuncu, Kisub Kim, Tegawendé F. Bissyandé, Dongsun Kim, Peng Wu, Jacques Klein, Xiaoguang Mao, and Yves Le Traon. 2020. On the Efficiency of Test Suite Based Program Repair: A Systematic Assessment of 16 Automated Repair Systems for Java Programs *(ICSE '20)*. New York, NY, USA, 615–627. https://doi.org/10.1145/3377811.3380338

[14] M. Martinez and M. Monperrus. 2016. ASTOR: A Program Repair Library for Java. In *International Symposium on Software Testing and Analysis*.

[15] Matias Martinez, Westley Weimer, and Martin Monperrus. 2014. Do the Fix Ingredients Already Exist? An Empirical Inquiry into the Redundancy Assumptions of Program Repair Approaches *(ICSE Companion 2014)*. New York, NY, USA, 492–495. https://doi.org/10.1145/2591062.2591114

[16] S. Mechtaev, J. Yi, and A. Roychoudhury. 2015. DirectFix: Looking for Simple Program Repairs. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*.

[17] Martin Monperrus. 2018. Automatic Software Repair: A Bibliography. *ACM Comput. Surv.* 51, 1, Article 17 (Jan. 2018), 24 pages. https://doi.org/10.1145/3105906

[18] Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. 2015. An Analysis of Patch Plausibility and Correctness for Generate-and-Validate Patch Generation Systems *(ISSTA 2015)*. 24–36. https://doi.org/10.1145/2771783.2771791

[19] Yingyi Wang, Yuting Chen, Beijun Shen, and Hao Zhong. 2017. CRSearcher: Searching Code Database for Repairing Bugs. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware* (Shanghai, China) *(Internetware'17)*. Association for Computing Machinery, New York, NY, USA, Article 16, 6 pages. https://doi.org/10.1145/3131704.3131720

[20] Westley Weimer, Zachary P Fry, and Stephanie Forrest. 2013. Leveraging program equivalence for adaptive program repair: Models and first results. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 356–366.

[21] Ming Wen, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. 2018. Context-Aware Patch Generation for Better Automated Program Repair. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 1–11. https://doi.org/10.1145/3180155.3180233

[22] Qi Xin and Steven P. Reiss. 2017. Leveraging syntax-related code for automated program repair. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 660–670. https://doi.org/10.1109/ASE.2017.8115676

[23] Yingfei Xiong, Xinyuan Liu, Muhan Zeng, Lu Zhang, and Gang Huang. 2018. Identifying Patch Correctness in Test-Based Program Repair *(ICSE '18)*. 789–799. https://doi.org/10.1145/3180155.3180182

[24] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu. 2019. A Novel Neural Source Code Representation Based on Abstract Syntax Tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*.