Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2019-IW-003

2019-IW-003

Improving Automated Program Repair with Retrospective Fault Localization

Tongtong Xu

International Conference on Software Engineering: Companion Proceedings 2019

2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

Improving Automated Program Repair with Retrospective Fault Localization

Tongtong Xu[1,2]
[1] Department of Computing, The Hong Kong Polytechnic University, China
[2] State Key Laboratory for Novel Software Technology, Nanjing University, China

***Abstract*—Although being recognized as a critical step in automated program repair, fault localization has been only loosely coupled into the fixing process in existing program repair approaches, in the sense that fault localization has limited interactions with other activities in fixing.**

We propose in this paper to deeply integrate fault localization into the fixing process to achieve more effective and efficient program repair. Our approach introduces a feedback loop in fixing between the activities for locating the fault causes and those for generating and evaluating candidate fixes. The feedback loop enables partial evaluation results of candidate fixes to be used to locate fault causes more accurately, and eventually leads to fixing processes with improved effectiveness and efficiency.

We have implemented the approach into a tool, named RESTORE, based on the JAID program repair system. Experiments involving faults from the DEFECTS4J standard benchmark indicate that the integrated fault localization can boost automated program repair: RESTORE produced valid fixes to 63 faults and correct ones to 38 faults, outperforming any other state-of-the-art repair tool for Java while taking 36% less running time compared with JAID.

I. INTRODUCTION

In the past few years, research on automatically proposing simple fixes to program faults, i.e., automated program repair (APR), attracted a lot of attention. Given a fault manifested by some failing test(s), APR techniques aim to suggest valid fixes that can make the program pass the failing test(s) and a regression test suite. An important category of APR techniques propose fixes in a generate-and-validate (G&V) fashion, i.e., they first generate and then check a large amount of candidate fixes to find out if any condidate can make all the tests pass. Given the search-based nature of G&V APR techniques, good knowledge about the fault's whereabouts is critical for their success, and those techniques typically apply fault localization to gather that knowledge.

Despite its enormous importance, fault localization has been only loosely coupled into the fixing process with G&V APR so far. In most cases, fault localization is a one-off activity that takes place at the beginning of fixing [1], [2], [3], [4], employing spectrum-based techniques [5], [6], [7] to decide the order in which locations are used for generating fixes. On the one hand, spectrum-based fault localization, while easy to implement, has only limited effectiveness [8]; On the other hand, they miss the opportunity to make use of the extra information produced during fixing to facilitate locating the fault. The negative consequences of that include correct program elements being attempted in vain during fixing and suboptimal fixing performance in the end. Approaches like GenProg [9], [10], [11] and AE [12] are based on genetic programming. In such approaches, variants of the faulty program that pass more tests stand a better chance to survive to the next generation, essentially tuning the fixing process to focus on more promising locations on-the-fly. These approaches, however, did not perform well in evaluations, partly because the number of tests passed does not provide enough guidance and the search "can easily devolve into random search" [13].

We propose in this paper a novel technique to deeply integrate fault localization into the fixing process so that G&V APR can become more effective and more efficient. Specifically, the new technique combines spectrum-based and mutation-based techniques into a feedback loop for more accurate fault localization. Spectrum-based fault localization (SBFL) is computationally inexpensive but with limited effectiveness; Mutation-based fault localization (MBFL) can be more effective but is considerably more expensive. To benefit from the high effectivenss while not suffering from the potential high cost of MBFL, instead of generating mutants of the faulty program by applying mutation operators, the new technique drives MBFL using higher-order mutants derived from candidate fixes.

We have implemented the technique into a tool named RESTORE. RESTORE is based on the JAID APR tool for Java, which abstracts the states of a faulty program during its execution using a rich set of boolean typed expressions and uses the state information to guide fault localization and fix generation. JAID was able to propose valid fixes to 31 faults, and correct fixes to 25 faults, from the DEFECTS4J benchmark [14]. In comparison, RESTORE was able to produce valid fixes to 63 faults and correct fixes to 38, outperforming all existing repair tools for Java while taking 36% less running time than JAID.

While RESTORE is based on the existing tool JAID, our technique is applicable to any program repair tools that use fault localization and validate patches through tests. Since JAID remains one of the most effective automated repair tools for Java at the time of this writing, the experimental results indicate the potential of the new fault localization technique to be integrated into other program repair tools.

Related work. Recently, Timperley et al. [15] investigated the application of MBFL in APR. In their study, single-edit changes were used to mutate the faulty program, then both the original and mutated programs were run against all tests, and the test results were compared to compute the suspiciousness

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00066

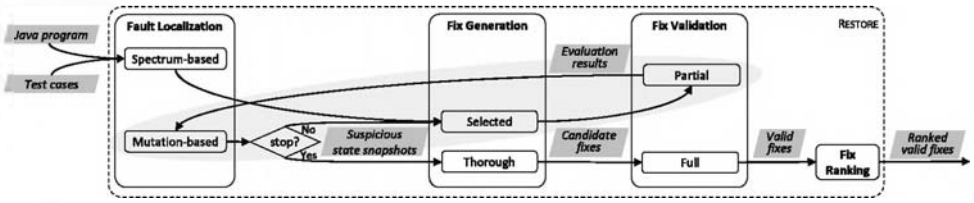

Fig. 1: An overview of how RESTORE works.

values for the locations. No significant improvement on the final repair performance was observed in the study, supposedly because simple single-edit mutations were not enough to reveal the subtle differences among programs [15]. Compared with using first order mutants generated solely for the purpose of fault localization, the design in RESTORE offers two advantages. First, as candidate fixes, higher-order mutants stand a better chance for affecting the test results. Second, results from running the mutants against the tests are directly useful for both fault localization and fix validation, reducing the performance overhead.

II. How RESTORE Works

Fig. 1 depicts an overview of the fixing process using RESTORE. RESTORE expects as input a faulty Java program P, a faulty method `fixme`, and a set T of test cases for P, where at least one test fails due to the fault under repair.

Fixing starts with first identifying a preliminary list of suspicious state snapshots[1] through SBFL. In the feedback loop that follows, a small number of candidate fixes are first generated from selected state snapshots and then partially validated, i.e., evaluated using only a subset of tests from T, to assess their impacts on the faulty program. Next, the results of partial validation are used to calculate the suspiciousness values of program locations based on the formula given in [16]. Afterwards, if some suspicious locations are found or all candidates have been validated, the loop terminates and the suspicious scores of snapshots are revised to become more discriminatory; Otherwise the loop will continue and partially validate more fix candidates. In the end, all candidate fixes at the suspicious locations are generated and fully validated, i.e., checked against all tests from T, and valid fixes that can make all tests pass are ranked by their desirability before reported to the user.

With the integrated fault localization, the top-ranked snapshots have a *higher chance* of leading to *valid fixes* when used in the following phases of the repair technique. The higher-precision fault localization technique means that *fewer candidates* need to be generated and (fully) validated, leading to an overall faster process. In turn, RESTORE's more efficient search of the original fix space allows it to explore a *larger space* in comparable time, ultimately leading to the potential of valid fixes that are outside JAID's fix space.

[1]In JAID, a state snapshot $s = \langle l, e, v \rangle$ captures the evaluation result v of a predicate e at a program location l.

TOOL	VALID	CORRECT	UNIQUE	PRECISION	RECALL
RESTORE	63	38	5	60%	11%
ACS [17]	23	18	12	78%	5%
CapGen [18]	25	22	3	88%	6%
Elixir [19]	41	26	?	63%	7%
HDA [20]	?	23	3	?	6%
JAID [4]	36	25	1	69%	7%
SimFix [21]	56	34	11	61%	10%
SketchFix [22]	26	19	2	73%	5%
	?	34	0	?	10%
ssFix [23]	60	20	1	33%	6%

TABLE I: Comparison between RESTORE and state-of-the-art program repair tools for Java.

III. Experimental Evaluation

As it has become customary to evaluate automated program repair tools for Java, our experiments use real-world faults in the DEFECTS4J collection [14]. DEFECTS4J includes 357 faults from 5 open-source Java projects; each fault comes with at least one test triggering the failure in addition to other passing or failing tests.

Effectiveness. Tab. I reports on the number of faults that RESTORE was able to repair with a valid fix (VALID) or a correct fix (CORRECT), the number of faults *no other tool* can currently fix (UNIQUE), and the corresponding precision (CORRECT/VALID) and recall (CORRECT/357) achieved. The table also compares the results of RESTORE with those of the other APR tools for Java. RESTORE was able to repair, with a valid or a correct fix, the most faults from DEFECTS4J. RESTORE, ACS, and SimFix can exclusively repair 5, 12, and 11 faults, respectively, which suggests RESTORE's fix space is somewhat *complementary* to other repair tools for Java.

Efficiency. RESTORE often runs faster than JAID, even though it explores a larger fix space. Compared with that of JAID, the overall running time of RESTORE is 36% shorter and the required time for RESTORE to produce a valid fix is 14% shorter.

To better understand how fault localization integration helps to improve APR, we also compare *the number of fixes that are checked until the first correct fix is found* (C2C) by JAID and RESTORE: The smaller the C2C, the more efficient the fixing process. It turns out that RESTORE needs to check 51% fewer fixes than JAID until it finds the first correct fix.

Acknowledgment This research was partly supported by the Hong Kong RGC General Research Fund (GRF) (PolyU 152703/16E) and The Hong Kong Polytechnic University internal fund (1-ZVJ1 and G-YBXU).

REFERENCES

[1] Y. Pei, C. A. Furia, M. Nordio, Y. Wei, B. Meyer, and A. Zeller, "Automated Fixing of Programs with Contracts," *IEEE Transactions on Software Engineering*, vol. 40, no. 5, pp. 427–449, 2014.

[2] F. Long and M. Rinard, "Staged Program Repair with Condition Synthesis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015, New York, NY, USA, 2015, pp. 166–178.

[3] ——, "Automatic patch generation by learning correct code," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '16. New York, NY, USA: ACM, 2016, pp. 298–312.

[4] L. Chen, Y. Pei, and C. A. Furia, "Contract-based Program Repair Without the Contracts," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 637–647.

[5] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 273–282.

[6] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 15–26.

[7] W. Eric Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *J. Syst. Softw.*, vol. 83, no. 2, pp. 188–208, Feb. 2010.

[8] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA '11. New York, NY, USA: ACM, 2011, pp. 199–209.

[9] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *Proceedings of the IEEE 31st International Conference on Software Engineering*, 2009, pp. 364–374.

[10] W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen, "Automatic program repair with evolutionary computation," *Communications of the ACM*, vol. 53, no. 5, pp. 109–116, 2010.

[11] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *2012 34th International Conference on Software Engineering (ICSE)*, ser. ICSE '12, Jun. 2012, pp. 3–13.

[12] W. Weimer, Z. Fry, and S. Forrest, "Leveraging program equivalence for adaptive program repair: Models and first results," in *2013 IEEE/ACM 28th International Conference on Automated Software Engineering*, Nov. 2013, pp. 356–366.

[13] Z. Qi, F. Long, S. Achour, and M. Rinard, "An analysis of patch plausibility and correctness for generate-and-validate patch generation systems," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: ACM, 2015, pp. 24–36.

[14] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 437–440, http://defects4j.org.

[15] C. S. Timperley, S. Stepney, and C. Le Goues, "An Investigation into the Use of Mutation Analysis for Automated Program Repair," in *International Symposium on Search Based Software Engineering*. Paderborn: York, Aug. 2017, pp. 99–114.

[16] M. Papadakis and Y. Le Traon, "Metallaxis-FL: Mutation-based Fault Localization," *Software Testing, Verification, and Reliability*, vol. 25, no. 5-7, pp. 605–628, August 2015.

[17] Y. Xiong, J. Wang, R. Yan, J. Zhang, S. Han, G. Huang, and L. Zhang, "Precise condition synthesis for program repair," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 416–426.

[18] M. Wen, J. Chen, R. Wu, D. Hao, and S. Cheung, "Context-aware patch generation for better automated program repair," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 1–11.

[19] R. K. Saha, Y. Lyu, H. Yoshida, and M. R. Prasad, "Elixir: Effective object oriented program repair," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 648–659.

[20] X. D. Le, D. Lo, and C. Le Goues, "History driven program repair," in *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*. Osaka, Japan: IEEE Computer Society, 2016, pp. 213–224.

[21] J. Jiang, Y. Xiong, H. Zhang, Q. Gao, and X. Chen, "Shaping program repair space with existing patches and similar code," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, 2018, pp. 298–309.

[22] J. Hua, M. Zhang, K. Wang, and S. Khurshid, "Towards practical program repair with on-demand candidate generation," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 12–23.

[23] Q. Xin and S. P. Reiss, "Leveraging syntax-related code for automated program repair," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 660–670.