Increasing Mutation Testing Effectiveness by Combining Lower Order Mutants to Construct Higher Order Mutants

Quang-Vu Nguyen(✉)

The University of Danang, Vietnam-Korea University of Information and Communication Technology, Da Nang, Viet Nam
nqvu@vku.udn.vn

Abstract. Researching and proposing the solutions in the field of mutation testing in order to answer the question of how to improve the effectiveness of mutant testing is a problem that researchers, who study in the field of mutation testing, are interested. Limitations of mutation testing are really big problems that prevent its application in practice although this is a promising technique in assessing the quality of test data sets. The number of generated mutants is too large and easy-to-kill mutants are two of those problems. In this paper, we have studied and presented our solution, as well as analyzed the empirical results for the purpose of introducing a way to improve the effectiveness of mutant testing. Instead of constructing higher order mutants by using and combining first-order mutants as previous studies, we propose a method to use higher-order mutants for creating mutants. In other words, we have combined two "**lower**" order mutants to construct "**higher**" order mutants, i.e., use two second order mutants to construct a fourth order mutant, guided by our proposed objective and fitness functions. According to the experimental results, the number of generated is reduced and number of valuable mutants is fairly large, we have concluded that our approach seems to be a good way to overcome the main limitations of mutation testing.

Keywords: Mutation testing · Limitations of mutation testing · Overcome · Reducing the cost · Harder to kill · Realistic faults

1 Introduction

In the field of Software Testing, Mutation Testing (MT), including First Order Mutation Testing (FOMT) and Higher Order Mutation Testing (HOMT), is an effectiveness approach (also high automation technique) which has been introduced to assess the quality of test suites.

Mutation Testing technique evaluates the quality of a given test suite by evaluating the test case ability to detect differences between the original program and its mutants. A mutant is another version of the original program, in which one or more different operators have been changed.

N. T. Nguyen et al. (Eds.): ICCCI 2020, LNAI 12496, pp. 205–216, 2020.
https://doi.org/10.1007/978-3-030-63007-2_16

Hamlet [1] and DeMillo et al. [2] are the first authors who present the idea of mutant testing and its related concepts. Whilst, the concepts and descriptions about higher order mutation testing as well as the distinction between First Order Mutants (FOMs) and Higher Order Mutants (HOMs) are firstly introduced by co-authors: Jia et al. [3] and Harman et al. [4].

Hitherto, there are many different interesting researches, such as [3–37], which have been proposed in the field of mutation testing. Some of these researches were developed to apply and improve the effectiveness of mutation testing and some others were proposed to overcome the problems of mutation testing. According to these researches result, mutation testing has a wide range of applications in software testing. It can be used with different programming languages for testing software at the unit level, the integration level or the specification level [5].

However, there are some really big barriers of mutation testing which are the main reasons to explain why mutation testing is not yet widely adopted in practice [5, 6].

- The first of them refers to the problem of high execution cost due to a large number of generated mutants. The execution cost of mutation testing includes not only the execution cost of software under test as well as all generated mutants against the given set of test cases but also the execution cost for their corresponding output results evaluations to determine that the mutant is "killed" or "alive" [1, 2].
- The second one is realism problem of generated mutants, in other words, the mutants do not denote realistic faults.
- And another is generated mutants are simple and so easy to kill. An easy-to-kill mutant is a mutant which is killed by all of given set of test cases.

In Mutation Testing, the quality of a set of test cases is evaluated by MS (Mutation Score) [1, 2] or MSI (Mutation Score Indicator) [11, 26–29] which is calculated based on number of generated mutants, number of **killed mutants**, number of **equivalent mutants** and number of **alive mutants**.

A mutant is called a "**killed mutant**" if its output results differ from the original program when they are executed with the same test case (in a given set of test cases). Conversely, if the mutant has the same output as the original program with all given test cases, it is called "**alive mutant**". **Equivalent mutants** are the mutants which always produce the same output as the original program, so they cannot be killed.

As we mentioned before, there are so many researches, e.g., [3–37], which have been introduced to overcome the main problems of mutation testing including traditional mutation testing (also known as first mutation testing or traditional mutation testing) and higher order mutation testing. With those studies, besides the advantages which make mutation testing get more effectiveness while doing, there are also some disadvantages that need to be considered, i.e., the number of generated mutants is still very large and this leads to a high execution cost of mutation testing.

In the above-mentioned studies [3–37], the authors have focused on two main methods for constructing higher order mutants: (1) Insert n faults ($n = 1..70$) [18, 19, 23] to create a n-order-mutant; (2) Combine n first-order-mutants to produce a n-order-mutant [20–25].

Different from the 2 above-methods, we will propose the another: Combine two "**lower"** order mutants to construct "**higher"** order mutants (guided by our proposed objective and fitness functions) as follows:

- Use two first order mutants to construct a second order mutant.
- Use two second order mutants to construct a fourth order mutant.
- And so on.

In addition, based on our previous research results [20–25], we will only use **not-easy-to-kill mutants** (details are presented in Sect. 2) to construct the higher-order-mutants.

The purpose of this investigation is to focus on the way to overcome the aforementioned limitations as well as to improve the effectiveness of mutation testing by combining the lower order mutants to construct higher order mutants. In this study of the paper, we have focused on reducing the number of mutants generated while improving the quality of mutations.

In the next section, we will present in details the backgrounds as well as the hypotheses of research problem and from that introduce the idea for this study. Section 3 presents and analyzes empirical results to demonstrate the effectiveness of our solution. And the last section is used to give a conclusion and some ideas for future work.

2 Background and the Idea for Study

In our previous works [20–25], we have proposed and proved (through empirical results) the effectiveness of applying multi-objective optimization algorithms into higher order mutation testing based on our mutant classification as well as our objective and fitness functions.

According to our researches, there are some positive results that we will reuse as "hypotheses" for the study in this paper, specifically as follows:

- Our mutant classification consists of eleven kinds (**H1–H11**) of HOMs and can cover all of the available cases of generated HOMs [20–24]. The idea of our HOMs classification approach is based on the combination of a set of test cases which can kill HOM, and sets of test cases which can kill its constituent FOMs.
- **H1** is group of alive (potentially equivalent) mutant. Alive mutants can be "really equivalent mutants" or "difficult-to-kill mutant". This mean that, the mutants cannot be killed by the given set of test cases which are included in the project under test, but they perhaps could be killed by other new, better test cases in terms of fault detecting. Reduction of H1 mutants leads to reducing of mutation testing execution cost of the given set of test cases on those live mutants [20–24].
- **H4–H9** are groups of "**Reasonable mutants"**. They are harder to kill and more realistic (reflecting real, complex faults) than their constituent FOMs [20–24].
- **H7** is group of "**High quality and reasonable mutants"** [20–24]. These mutations, in our study, are the best ones which can be used to replace the set of their constituent FOMs while ensuring the effectiveness of the test. In this case, the set of test cases can kill both HOMs and their constituent FOMs.

- **eNSGAII algorithm** is the best multi-objective optimization algorithm in terms of constructing the "H7 - High Quality and Reasonable HOMs" [24]. Approximately 12% of "reasonable HOMs" in our studies, which were found by eNSGAII algorithm, are classified as "high quality and reasonable HOMs".
- **Five** (5) a relevant upper bound on mutation order in higher order mutation testing [23, 24]. This conclusion of us based on the relationships between the order of mutation testing and the properties of mutants.
- The **not-easy-to-kill mutants** should be used to generate higher order mutants because it seems to be a promising method to improve the quality of test cases [22, 24].

Easy-to-kill mutants are the mutants that are killed by all given test cases of project under test. The not-easy-to-kill mutants are the remaining ones of the set of all generated mutants after deleting all easy-to-kill mutants.

From these hypotheses, in this study, we have proposed the solution for increasing the effectiveness (in terms of reducing the cost, generating valuable mutants and do not waste computational resources for creating mutants, which are easy to kill by most test cases) of mutation testing by combining two lower order mutants to construct higher order mutants (5 is upper bound order [23, 24]) as follows:

- Firstly, generate all possible FOMs, calculate number of generated FOMs, number of live (potentially equivalent) FOMs and MSI.
- Delete easy-to-kill-FOMs from the set of generated FOMs.
- From set of remaining-FOMs (The not-easy-to-kill FOMs), generate Second Order Mutants (SOMs) by combining two FOMs guided by our objective and fitness functions;
- Calculate number of H1, H4–H9, H7 and MSI of second order mutation testing.
- Delete easy-to-kill-SOMs from the set of generated SOMs.
- From set of remaining-SOMs (The not-easy-to-kill SOMs), generate Fourth Order Mutants (FOOMs) by combining two SOMs guided by our objective and fitness functions;
- Calculate number of H1, H4–H9, H7 and MSI of fourth order mutation testing.
- Evaluate and compare the obtained results.

In our solution, we have focused on eliminating the easy-to-kill mutants before constructing the higher-order mutants due to many studies have been demonstrated that the majority (about 90%) of real faults of software are complex faults [18, 19].

We have used (and extended) Judy [29] as a support tool to generate and evaluate FOMs and HOMs (SOMs and FOOMs) with the same full set of mutation operators of Judy. This is a tool that can be used to generate mutations, perform mutant testing, evaluate results and produce reports as well as support both first order and high order mutation testing.

As described in Sect. 2, eNSGAII algorithm is the best multi-objective optimization algorithm in terms of constructing valuable mutants. That is why we continue to use this algorithm in our research of this paper.

Selected projects under test, five open source projects (Barbecue, BeanBin, JWBF, Common Chain 1.2 and Common Validator 1.4.1) downloaded from SourceForge

(https://sourceforge.net/), for experiment along with number of classes (NOC), number of lines of code (LOC) and number of build-in test cases (NOT) are shown in Table 1.

Table 1. Projects under test

PROJECT	NOC	LOC	NOT
Barbecue	57	23.996	190
BeanBin	72	5.925	68
JWBF	51	13.572	305
CommonChain 1.2	103	13.410	17
CommonValidator 1.4.1	144	25.422	66

These projects have a fairly large number of lines of code, as well as built-in test cases that are considered to be quality. Besides, these projects also include a test suite with a high number of test cases.

3 Empirical Results and Analysis

We have produced First Order Mutants, Second Order Mutants, Fourth Order Mutants as well as analyzed and calculated the corresponding parameters for each order of the generated mutants.

Experimental results of First Order Mutation Testing (FOMT), Second Order Mutation Testing (SOMT) and Fourth Order Mutation Testing (FOOMT) are shown in Table 2, 3 and 4 respectively.

Table 2. FOMT results

PROJECT	Number of FOMs	% of live (potentially equivalent) FOMs to all FOMs	MSI(%)
Barbecue	3.084	84,21%	15,79%
BeanBin	1.330	84,89%	15,11%
JWBF	1.482	87,04%	12,96%
CommonChain 1.2	1.476	57,35%	42,65%
CommonValidator 1.4.1	2.981	52,90%	47,10%

In this study, we have used the following parameters to confirm our proposed solution is good or not to improve the effectiveness of mutation testing (in terms of reducing the cost, generating valuable mutants and do not waste computational resources for creating mutants which are easy to kill by most of the test cases):

Table 3. SOMT results

PROJECT	Number of SOMs	% of H1 to all SOMs	% of H4–H9 to all SOMs	% of H7 to all SOMs	MSI(%)
Barbecue	678	37,61%	54,24%	6,25%	62,39%
BeanBin	297	37,89%	53,18%	6,12%	62,11%
JWBF	329	42,22%	40,09%	6,08%	57,78%
CommonChain 1.2	320	39,85%	51,12%	6,21%	60,15%
CommonValidator 1.4.1	656	36,10%	63,47%	6,25%	63,90%

Table 4. FOOMT results

PROJECT	Number of FOOMs	% of H1 to all FOOMs	% of H4–H9 to all FOOMs	% of H7 to all FOOMs	MSI
Barbecue	99	32,15%	66,19%	6,13%	67,85%
BeanBin	68	33,01%	58,24%	6,01%	66,99%
JWBF	71	34,83%	54,41%	6,05%	65,17%
CommonChain 1.2	78	38,95%	45,80%	6,14%	61,05%
CommonValidator 1.4.1	92	25,12%	60,94%	6,12%	74,88%

- Number of generated mutants;
- Number of "**Live**" mutants;
- Number of "**Reasonable**" mutants;
- Number of "**High Quality and Reasonable**";

In Table 2, the main parameters are followings:

- Number of generated first order mutants;
- Proportion of live (potentially equivalent) mutants (H1) to all generated mutants;
- MSI;

Whilst, in Table 3 and 4, the main parameters are followings:

- Number of generated second/fourth order mutants,
- The proportions of Live (potentially equivalent) mutants (H1) to all generated mutants;
- The proportions of group of reasonable mutants (H4–H9) to all generated mutants;
- "High Quality and Reasonable" mutants (H7) to all generated mutants.
- MSI;

From the obtained results shown in Tables 2, 3 and 4 above, we can see that the number of constructed mutations tends to significantly reduce in order from lower to higher (See Fig. 1 below).

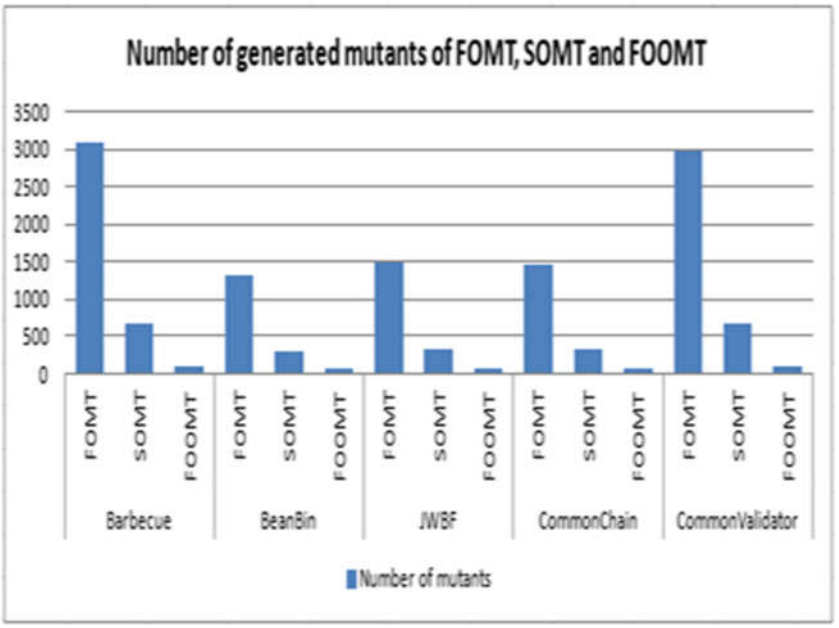

Fig. 1. Number or generated mutants of 1st, 2nd and 4th order mutation testing

Number of generated of FOMs (1st order) is about 4,5 times more than number of generated of SOMs (2nd order), and number of generated of SOMs (2nd order) is about 5,5 times more than number of generated of FOOMs (4th order).

In other words, the number of higher order mutants is reduced at least 70% compared to the adjacent lower order mutants. This is because we only use the set of not-easy-to-kill lower mutants instead of using all lower generated mutants to construct the higher mutants. In addition, we focus on constructing the valuable (H7 - High Quality and Reasonable) mutants instead of generating all possible mutants by applying our proposed objectives and fitness functions [20–24].

Reducing the number of generated mutants leads to reduce the execution cost of mutation testing. Therefore, it can be said that our solution in this paper is a promising method for overcoming the big problem (a very large number of generated mutants) of mutation testing in general and higher order mutation testing in particular.

Figure 2 shows that the number of H1 mutants decreases in orders (from 73.28% of FOMT to 38.73% of HOMT and 25.12% of FOOMT). This demonstrates that the higher order mutants are more complex and more realistic than their constituent lower order mutants. As mentioned above, H1 is the group of alive (potentially equivalent) mutants.

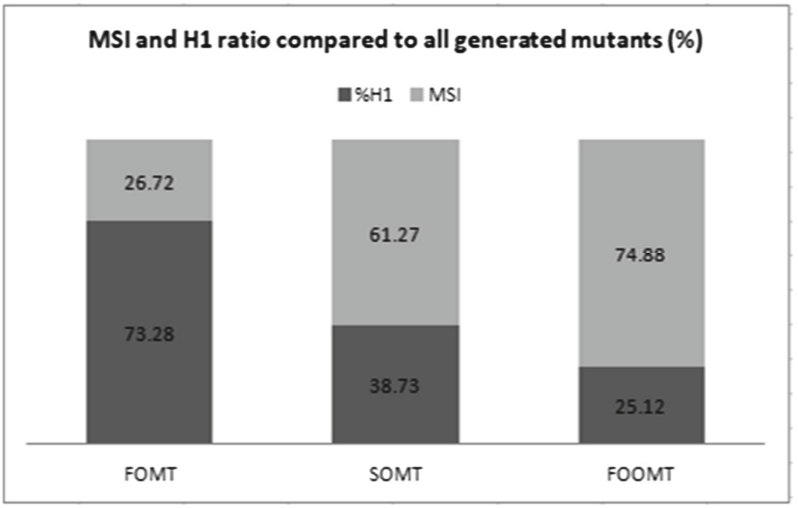

Fig. 2. The mean MSI and H1 proportion to all generated mutants

The second conclusion drawn from the experimental results is: Mutant combination does not decrease the quality of set of test cases due to increasing of MSI in order (from 26.72% of FOMT to 61.27% of HOMT and 74.88% of FOOMT).

Another noteworthy result, shown in Fig. 3, that is proportion of group H4–H9 mutants to total generated HOMs is high, about 52,42% (in SOMT) to 57,12% (in FOOMT).

As we mentioned in Sect. 2, H4–H9 are groups of reasonable mutants which are harder to kill and more realistic (reflecting real, complex faults) than their constituent. The term "Reasonable mutant" (harder to kill mutant), in our study, means that "*the set of test cases which kills mutant is smaller than the set of test cases which kills its constituent*".

Reducing number of needed test cases leads to reducing testing costs without loss of test effectiveness. Because, in mutation testing (including higher order mutation testing), the mutation testing cost covers not only the original program but also all of its mutants against all given set of test cases.

The number of test cases is smaller but still ensuring the quality is what we want in mutation testing. It can be said that overcoming the problem of mutation testing costs will lead to the widespread application of mutation testing techniques to reality.

According to experimental results, in both of SOMT and FOOMT, the mean ratio of H7 mutants to all generated mutants is about 6,2% and to all reasonable mutants (H4–H9) is about 11%.

This is a fairly high number because H7 mutant is a best one (**High Quality and Reasonable Mutant**) in our proposed approach [20–25] for constructing the quality mutants. In fact, they can be used to replace the set of their constituent mutants without lost of testing effectiveness. In other words, it helps to reduce the mutation testing cost

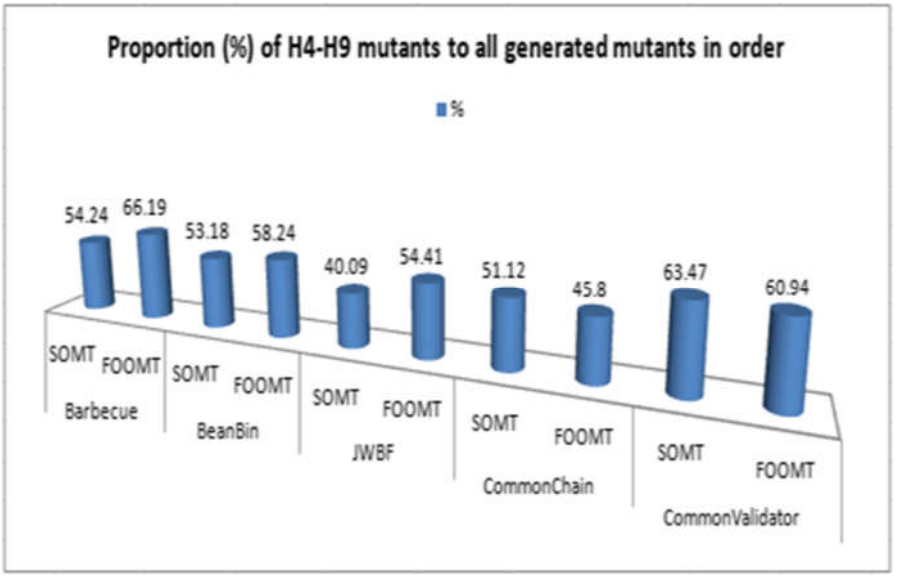

Fig. 3. Proportion (%) of H4–H9 mutants to all generated mutants in order

effectively. In our solution, although the number of higher-order mutants is always lower than the number of lower-order mutants, the rate of H7 is still high.

This, once again, demonstrates that this is a promising method that may need to be further researched and applied to not only reduce the number of produced mutants but also to construct a fairly large number of **High Quality and Reasonable Mutant**.

4 Conclusions

In this paper, through the approach of combining two lower order mutants to construct higher order mutants, we have investigated and presented a solution which can be used to overcome the main limitations of mutation testing. That are problems of large number of mutants and this also leads to have a very high execution cost; generated mutants are simple, easy to kill and do not denote realistic faults in practice.

Our empirical results indicate that the solution presented in this study is fairly good way to address the main limitations of mutation testing in terms of reducing the cost, generating valuable mutants and do not waste computational resources for creating mutants, which are easy to kill by most test cases.

Overcoming the mentioned main limitations of mutation testing will contribute to promote applying mutation testing into the field of software testing more widely and effectively. This applying will be very helpful for those who want to improve the quality of test cases (especially test data sets) for software testing because, it can be said up, mutation testing is considered as an automatic, powerful and effective technique to evaluate the quality of test suites.

Our research results, though only a preliminary study, are positive for us to continue to research and from that to promote the actual implementation of mutation testing. Of course, to achieve that desire, we need further investigation with more realistic and larger software to demonstrate that our proposing solution is really effective. Working with software testing companies or software companies is also a good way to validate and improve the effectiveness of this approach in the future.

References

1. Hamlet, R.G.: Testing programs with the aid of a compiler. IEEE Trans. Softw. Eng. (SE) **3**(4), 279–290 (1977)
2. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: help for the practicing programmer. IEEE Comput. **11**(4), 34–41 (1978)
3. Jia, Y., Harman, M.: Higher order mutation testing. Inform. Softw. Technol. **51**, 1379–1393 (2009)
4. Harman, M., Jia, Y., Langdon, W.B.: A manifesto for higher order mutation testing. In: Third International Conference on Software Testing, Verification, and Validation Workshops (2010)
5. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. **37**(5), 649–678 (2011)
6. Nguyen, Q.V., Madeyski, L.: Problems of mutation testing and higher order mutation testing. In: van Do, T., Thi, H.A.L., Nguyen, N.T. (eds.) Advanced Computational Methods for Knowledge Engineering. AISC, vol. 282, pp. 157–172. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06569-4_12
7. Offutt, A.J.: Investigations of the software testing coupling effect. ACM Trans. Softw. Eng. Methodol. **1**, 5–20 (1992)
8. Polo, M., Piattini, M., Garcia-Rodriguez, I.: Decreasing the cost of mutation testing with second-order mutants. Softw. Test. Verification Reliab. **19**(2), 111–131 (2008)
9. Kintis, M., Papadakis, M., Malevris, N.: Evaluating mutation testing alternatives: a collateral experiment. In: Proceedings 17th Asia Pacifc Software Engineering. Conference (APSEC) (2010)
10. Papadakis, M., Malevris, N.: An empirical evaluation of the first and second order mutation testing strategies. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ser. ICSTW'10, Computer Society, pp. 90–99. IEEE (2010)
11. Madeyski, L., Orzeszyna, W., Torkar, R., Józala, M.: Overcoming the equivalent mutant problem: a systematic literature review and a comparative experiment of second order mutation. IEEE Trans. Softw. Eng. **40**(1), 23–42 (2014)
12. Omar, E., Ghosh, S.: An exploratory study of higher order mutation testing in aspect-oriented programming. In: IEEE 23rd International Symposium on Software Reliability Engineering (2012)
13. Jia, Y., Harman, M.: Constructing subtle faults using higher order mutation testing. In: Proceedings Eighth Int'l Working Conference Source Code Analysis and Manipulation (2008)
14. Omar, E., Ghosh, S., Whitley, D.: Constructing subtle higher order mutants for java and aspectJ programs. In: International Symposium on Software Reliability Engineering, pp. 340–349 (2013)
15. Omar, E., Ghosh, S., Whitley, D.: Comparing search techniques for fnding subtle higher order mutants. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1271–1278 (2014)

16. Belli, F., Güler, N., Hollmann, A., Suna, G., Yıldız, E.: Model-based higher-order mutation analysis. In: Kim, T., Kim, H.K., Khan, M.K., Kiumi, A., Fang, W., Ślęzak, D. (eds.) ASEA 2010. CCIS, vol. 117, pp. 164–173. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17578-7_17
17. Akinde, A.O.: Using higher order mutation for reducing equivalent mutants in mutation testing. Asian J. Comput. Sci. Inform. Technol. **2**(3), 13–18 (2012)
18. Langdon, W.B., Harman, M., Jia, Y.: Multi-objective higher order mutation testing with genetic programming. In: Proceedings Fourth Testing: Academic and Industrial Conference Practice and Research (2009)
19. Langdon, W.B., Harman, M., Jia, Y.: Efficient multi-objective higher order mutation testing with genetic programming. J. Syst. Softw. **83**(12), 2416–2430 (2010)
20. Nguyen, Q.V., Madeyski, L.: Searching for strongly subsuming higher order mutants by applying multi-objective optimization algorithm. In: Le Thi, H.A., Nguyen, N.T., Do, T.V. (eds.) Advanced Computational Methods for Knowledge Engineering. AISC, vol. 358, pp. 391–402. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17996-4_35
21. Nguyen, Q.V., Madeyski, L.: Empirical evaluation of multi-objective optimization algorithms searching for higher order mutants. Cybern. Syst. Int. J. **47**(1-2), 48–68 (2016)
22. Nguyen, Q.V., Madeyski, L.: Higher order mutation testing to drive development of new test cases: an empirical comparison of three strategies. In: Nguyen, N.T., Trawiński, B., Fujita, H., Hong, T.P. (eds.) ACIIDS 2016. LNCS (LNAI), vol. 9621, pp. 235–244. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49381-6_23
23. Nguyen, Q.V., Madeyski, L.: On the relationship between the order of mutation testing and the properties of generated higher order mutants. In: Nguyen, N.T., Trawiński, B., Fujita, H., Hong, T.P. (eds.) ACIIDS 2016. LNCS (LNAI), vol. 9621, pp. 245–254. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49381-6_24
24. Nguyen, Q.V., Madeyski, L.: Addressing mutation testing problems by applying multi-objective optimization algorithms and higher order mutation. J. Intell. Fuzzy Syst. **32**, 1173–1182 (2017). https://doi.org/10.3233/jifs-169117
25. Nguyen, Q.V., Pham, D.T.H.: Is higher order mutant harder to kill than first order mutant. An experimental study. In: Nguyen, N.T., Hoang, D.H., Hong, T.P., Pham, H., Trawiński, B. (eds.) ACIIDS 2018. LNCS (LNAI), vol. 10751, pp. 664–673. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75417-8_62
26. Madeyski, L.: On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests. In: Münch, J., Abrahamsson, P. (eds.) PROFES 2007. LNCS, vol. 4589, pp. 207–221. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73460-4_20
27. Madeyski, L.: The impact of pair programming on thoroughness and fault detection effectiveness of unit tests suites. Softw. Process Improv. Pract. **13**(3), 281–295 (2008). https://doi.org/10.1002/spip.382
28. Madeyski, L.: The impact of test-first programming on branch coverage and mutation score indicator of unit tests: an experiment. Inform. Softw. Technol. **52**(2), 169–184 (2010). https://doi.org/10.1016/j.infsof.2009.08.007
29. Madeyski, L., Radyk, N.: Judy - a mutation testing tool for java. IET Softw. **4**(1), 32–42 (2010). https://doi.org/10.1049/iet-sen.2008.0038
30. Zhu, Q., Panichella, A., Zaidman, A., An investigation of compression techniques to speed up mutation testing. In: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Vasteras, pp. 274–284 (2018)
31. Jimenez, M., et al.: Are mutants really natural. A study on how naturalness helps mutant selection. In: The International Symposium on Empirical Software Engineering and Measurement (ESEM) (2018)

32. Saber, T., Delavernhe, F., Papadakis, M., O'Neill, M., Ventresque, A.: A hybrid algorithm for multi-objective test case selection. In: 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, pp. 1–8 (2018)
33. Abuljadayel, A., Wedyan, F.: An investigation of compression techniques to speed up mutation testing. In: International Journal of Intelligent Systems and Applications, January 2018
34. Lewowski, T., Madeyski, L.: Mutants as patches: towards a formal approach to mutation testing. Found. Comput. Decis. Sci. **44**(4), 379–405 (2019)
35. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: survey, landscapes and horizons. In: IEEE Transactions on Software Engineering (2019). https://doi.org/10.1109/tse.2019.2962027
36. Bokaei, N.N., Keyvanpour, M.R.: A comparative study of whole issues and challenges in mutation testing. In: *2*019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), Tehran, Iran, pp. 745–754 (2019). https://doi.org/10.1109/kbei.2019.8735019
37. Naeem, M.R., Lin, T., Naeem, H., Ullah, F., Saeed, S.: Scalable mutation testing using predictive analysis of deep learning model. IEEE Access **7**, 158264–158283 (2019). https://doi.org/10.1109/access.2019.2950171