Journal of Intelligent & Fuzzy Systems 32 (2017) 1173–1182
DOI: 10.3233/JIFS-169117 Preprint
IOS Press

Addressing mutation testing problems by applying multi-objective optimization algorithms and higher order mutation

Quang Vu Nguyen [a], Lech Madeyski [b]*
[a] *Vietnam-Korea Friendship Information Technology College, Vietnam*
[b] *Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wyb.Wyspianskiego 27, 50370 Wroclaw, Poland. Tel. +48 71 320 2886, Fax. +48 71 321 1018, E-mail: lech.madeyski@pwr.edu.pl (Corresponding author)*

Abstract. Traditional mutation testing is a powerful technique to evaluate the quality of test suites. Unfortunately, it is not yet widely used due to the problems of a large number of generated mutants, limited realism (mutants not necessarily reflect real software defects), and equivalent mutants problem. Higher order mutation (HOM) testing has been proposed to overcome these limitations of first order mutation testing. We present an empirical evaluation of our approach to higher order mutation testing. We apply different multi-objective optimization algorithms (including one modified by us), as well as our classification of HOMs, proposed objectives and fitness functions. We search for "High Quality and Reasonable HOMs" able to replace all of its constituent FOMs without scarifying test effectiveness and to reflect complex defects requiring more than one change to correct them. Our approach leads to: 1) reduced cost of mutation testing due to reduced number of HOMs, 2) harder to kill mutants (which mimic harder to find defects), 3) reduced cost of mutation testing as it does not waste resources for creating easy-to-kill mutants. Furthermore, we establish a relevant upper bound on mutation order in higher order mutation testing and thus reduce the cost of mutation even further.

Keywords: Mutation testing, Higher Order Mutation Testing, Higher Order Mutants, Multi-objective optimization algorithm, Upper bound order

1. Introduction

Mutation testing (called also First Order Mutation Testing (FOMT)) is used to evaluate the fault detection capability of the test suits by inserting changes into the original program to generate mutants, and then checking whether the given set of test cases is good enough to detect the difference between original program and its mutants or not. **Mutants** are the different versions of an original program generated by inserting, via a **mutation operator**, only one semantic change (or fault) into the original program. Mutation operators depend on programming languages. For example, there are some traditional mutation operators like deletion of a statement; replacement of Boolean expressions; replacement of arithmetic expression; or replacement of a variable. If a test case distinguishes between the mutant and the original program, it is said to kill the mutant. In other words, the mutant **is killed** by the test case. If a mutant was killed by all of given TCs, it is named "Easy to kill". Conversely, a mutant is said to be **alive** if no test case detects the injected fault.

In mutation testing, we can use Mutation Score (MS) or Mutation Score Indicator (MSI) to evaluate quality of a set of test cases. MS is the ratio of killed mutants to the difference between all generated mutants and equivalent mutants [4,1], while MSI is the ratio of killed mutants to all generated mutants [12,13,14,16]. Value of above mentioned scores lies between 0 and 1. A low score means that the majority of faults cannot be detected accurately by the test set. A higher score indicates that most of the injected faults have been identified with this particular test set. When score is close to 1, we can say that the given set of test cases is a good one. If score is zero, there is no test case that can kill the mutants.

Although Mutation Testing (MT) has been introduced as an automated technique to assess the quality of the test cases, the problem is a large number of generated first order mutants (FOMs). Furthermore, most of mutants are simple, often easily to detect and do not denote realistic faults [8,21]. Besides, one of mutation testing problems is the generation of too many equivalent mutants [16,7,5,21], which have the same semantic meaning as the original program under test.

There are many different approaches which have been proposed for overcoming the problems of MT [21] including second order mutation testing [16,19, 26,30,28] and higher order mutation testing [7,5,11,6] in general. The main goal of higher order mutation testing is to generate higher order mutants (HOMs) that can be used to improve the effectiveness of mutation testing. Instead of using only one simple change as traditional mutation testing, higher order mutation testing uses more complex changes to generate mutants by applying two or more mutation operators. Higher Order Mutation Testing (HOMT), an idea of Jia and Harman [7,5], not only is the unique approach able to address all of the three mutation testing problems simultaneously, but also is the only one to deal with the problem of realism of injected defects as there is an empirical evidence that real defects require more than one change to fix them [29].

Still there is little research in the field of applying multi-objective optimization algorithms to search for valuable HOMs, e.g., Strongly Subsuming and Coupled HOMs [7,5], although this is a promising approach. It allows to find higher order mutants that represent more realistic complex faults, which are harder to kill than first order mutants. There are three papers, by Harman et al. [5] and Langdon et al. [10,11], which focused on multi-objective higher order mutation testing with genetic programming. Haider et al. [3] proposed fuzzy based optimization approach in combination with all path coverage criterion to safely reduce a test suite to a single solution and they found that the approach significantly reduces the test suite to a precise test suite.

In this paper, we investigate applying multi-objective optimization algorithms for finding "High Quality and Reasonable HOMs" (one of 11 HOM categories in our HOMs classification described in Section 2). Our main goal is to carry out an empirical evaluation of the proposed approach, which can be used to construct difficult to kill and more realistic HOMs, especially "High Quality and Reasonable HOMs", as well as to reduce the number of generated higher order mutants. We want to bring out some useful findings for applying multi-objective optimization algorithms in the area of higher order mutation testing for overcoming the limitations of traditional mutation testing. As a result, it could be used to improve the mutation testing effectiveness in general.

The rest of the paper is organized as follows. Section 2 presents the landscape of the reported research on higher order mutation testing applying multi-objective optimization algorithms. The kinds of higher order mutants based on our HOMs classification also are included in this section. Section 3 describes the posed research questions that the study will answer. The next section is used to shortly present the selected algorithms, programs under test and supporting tool. Section 5 shows the experimental results needed to answer the posed research question as well as to bring out some useful findings. The last section presents conclusions, discussions of threats to validity and proposition of future works.

2. Background

As we mentioned in Section 1, still there is little research on applying multi-objective optimization algorithms in higher order mutation testing.

In fact, there are many optimization problems, which have more than one objective function and the objective functions are conflicting, to some extent, preventing simultaneously the simple optimization of each objective. Hence, multi-objective optimization algorithms have been devised for solving optimization problems and making the decisions that satisfy multiple objectives.

Harman et al. [5] and Langdon et al. [10,11] are the first authors who focused on applying multi-objective optimization algorithm with genetic programming in HOMT. In order to produce better mutants, they suggested inserting "semantically close" faults instead of inserting "syntactically close" faults to the original program under test. They then applied a multi-objective optimization algorithm (*NSGAII*) with genetic programming in the area of higher order mutation testing and the obtained results demonstrated that this approach is able to find harder to kill higher order mutants that represent more realistic complex faults.

In our previous works [22,23,24,25], we proposed a new classification of HOMs to cover all of the available cases of generated HOMs, as well as we described our objectives and fitness functions. Our HOMs classification consists of eleven kinds of HOMs which are illustrated in Table 1. The notations are explained below:

- H1: Live (potentially equivalent) Mutant)
- H2: Non-Quality, Un-Reasonable and With New TCs
- H3: Non-Quality, Un-Reasonable and With Mixed TCs
- H4: Non-Quality, Reasonable and With New TCs
- H5: Non-Quality, Reasonable and With Mixed TCs
- H6: Non-Quality, Reasonable and With Old TCs
- H7: **High Quality and Reasonable**
- H8: Quality, Reasonable and With Mixed TCs
- H9: Quality, Reasonable and With Old TCs
- H10: Quality, Un-Reasonable and With Old TCs
- H11: Quality, Un-Reasonable and With Mixed TCs

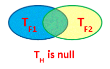
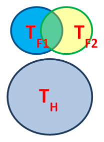
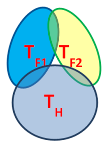
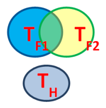
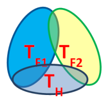
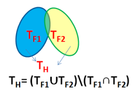
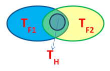
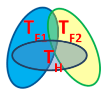
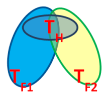
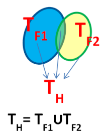
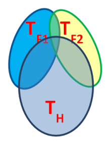
We performed the empirical evaluation of multi-objective optimization algorithms in HOMT. Different from Langdon et al. [5,10,11], who have suggested directly inserting more than one fault into a program under test to generate higher order mutants, we introduced another method as follows. First, we generate the set of available first order mutants and then generate valuable higher order mutants by combining two or more first order mutants.

Our experimental results are as follows: the total number of generated HOMs is small (about 30% in comparison to number of FOMs), the mean ratio of reasonable HOMs (subsuming HOMs) to all found HOMs is about 54%, while the mean ratio of "High Quality and Reasonable HOMs" (strongly subsuming and coupled HOMs) to all found reasonable HOMs (subsuming HOMs) is fairly high (around 8%) [23].

It is worth emphasizing that higher order mutation testing in general and the presented approach in particular is able to address all of the three main problems of mutation testing (a large number of generated mutants, limited realism of artificial defects and equivalent mutant problem) simulataneously [20]. In this paper, we build upon our previous research [22,23,24,25] and collect empirical evidence to answer new research questions (e.g., RQ1, RQ2, RQ3.2, RQ3.5) posed in

Table 1

Eleven categories of HOMs based on the combination of sets of test caces

HOM category	Illustration
H1	
H2	
H3	
H4	
H5	
H6	 T_{F1} T_{F2} T_H $T_H = (T_{F1} \cup T_{F2}) \setminus (T_{F1} \cap T_{F2})$
H7	
H8	
H9	
H10	
H11	

Section 3 on a basis of extended number of programs under test (e.g., in [23] we used three programs under test, while in this paper we use five, see Table 2). These answers are essential to extend our understanding of higher-order mutation testing as a viable alternative to traditional first order mutation testing.

3. Research questions

We pose the following Research Questions (RQs) that the study will answer.

RQ1. What is the ratio of the number of generated HOMs to the number of generated FOMs?

RQ2. What is MSI of HOMT by applying multi-objective optimization algorithm?

RQ3. How popular are higher order mutants (HOMs) in the identified mutant categories?

We split RQ3 into 5 sub-questions to answer:

- *RQ3.1.* What are the ratios of the number of HOMs in the identified mutant categories to all generated HOMs?
  The goal of the RQ3.1 is to investigate the number (distribution) of HOMs in the identified mutant categories.
- *RQ3.2.* What are the ratios of "not good" HOMs that are generated?
  "Not good" HOM mean "Non-quality and Unreasonable HOM" (H2 and H3) which are easier to kill than their constituent FOMs and there is lack of test cases that can kill simultaneously HOM and its constituent FOMs.
- *RQ3.3.* What are the ratios of the number of HOMs in the mutant order to all generated HOMs?
  By means of this question, we want to obtain the percentage of generated HOMs in the mutant order.
- *RQ3.4.* What are the ratios of "High Quality and Reasonable HOMs" (H7) to all generated HOMs in the mutant order?
  This question focuses on a kind of mutants being of special interest. The aim is to obtain the frequency of generating "High Quality and Reasonable HOMs" (H7). Such mutants not only reflect harder to kill mutants, which reflect realistic, complex faults but also could be used to replace all of its constituent FOMs.
- *RQ3.5.* What are the ratios of "live (potentially equivalent) mutants" (H1) to all generated HOMs in the mutant order?
  Live mutants can be "really-equivalent mutants" or "difficult-to-kill mutant". They cannot be killed by the given set of test cases which are included in the selected project under test (but they perhaps could be killed by some new test cases). Reduction of Live (potentially equivalent) mutants leads to reduced mutation testing execution cost of the given set of test cases on those live mutants. Answering this question (together with the answering the question RQ3.4) may help to find the relevant order of higher order mutation testing.

RQ4. Which multi-objective optimization algorithms are more suitable for searching for "Reasonable HOMs" and "High Quality and Reasonable HOMs"? How can one improve the quality of the mentioned algorithms in terms of mutant reduction, generating difficult-to-kill mutants and constructing "High Quality and Reasonable HOMs"?

RQ5. Which strategies using the set of first order mutants to generate higher order mutants are useful in driving new quality test cases development?

By the means of RQ5, we want to evaluate which of the proposed approaches using the set of first order mutants to generate higher order mutants is better suited to drive development of new test cases by using MSI and, as a result, to improve the software quality.

4. Algorithms, projects under test and supporting tool

Five selected multi-objective optimization algorithms [23] are briefly described below.

NSGA-II is the second version of the Non-dominated Sorting Genetic Algorithm for solving non-convex and non-smooth single and multi-objective optimization problems. NSGA-II uses an elitist principle, emphasizes non-dominated solutions, and uses an explicit diversity preserving mechanism. **NSGA-III** is the extension of NSGA-II which is based on the supply of a set of reference points and demonstrated its advantages in three to 15-objective optimization problems. The **eNSGA-II** extends NSGA-II's concepts by adding e-dominance, adaptive population sizing, and self-termination to minimize the need for parameter calibration. E-dominance is a concept where a user is able to specify the precision with which he wants to obtain the Pareto-optimal solutions to a multi-objective problem, in essence giving him the ability to assign a relative importance to each objective. The **eMOEA** is a steady state multi-objective evolutionary algorithm that co-evolves both an evolutionary algorithm population and an archive population by randomly matching individuals from the population and the archive to generate new solutions. The **Random search** is used to generate random solutions, which are evaluated and all non-dominated solutions are retained. The result is the set of all non-dominated solutions.

There are five selected Projects Under Test (PUTs): BeanBin, Barbecue, JWBF, CommonsChain 1.2 and CommonsValidiator 1.4.1, which are five real-world, open source projects downloaded from SourceForge (https://sourceforge.net/). Table 2 shows the projects selected for the experiment along with their number of classes (NOC), lines of code (LOC) and number of test cases (NOT). For each project under test, we run the HOMT process 5 times for each selected algorithm.

Table 2
Projects under test

Project	NOC	LOC	NOT
BeanBin	72	5925	68
Barbecue	57	23996	190
JWBF	51	13572	305
CommonsChain 1.2	103	13410	17
CommonsValidiator 1.4.1	144	25422	66

Our supporting tool is Judy [16,17], a mutation testing tool, which has been written in Java and for Java. Judy is easy to configure and use. It allows to configure and launch mutation testing via command line in Windows, Linux and Mac OS X. Judy supports a large set of mutation operators for first order mutation testing, as well as higher order mutation testing such as: mutants generation, mutants execution and mutation analysis. Judy also supports build-in cluster for distributed computations. The list of some mutation operators available in Judy is presented in [16,17] as well as in [23].

5. Results and analysis

5.1. Answer to RQ1

Table 3 shows the mean number of FOMs which were generated by applying all Judy operators to produce mutants, and the mean number of total generated HOMs (from the set of generated FOMs) for different PUTs by applying the selected multi-objective optimization algorithms.

Table 3
The mean number of generated FOMs and HOMs

Number of FOMs	Number of HOMs
1965	416

The first step of our experimental procedure is to generate all possible FOMs, and our purpose is to compare the number of generated FOMs with the number of generated HOMs. From the set of FOMs, valuable HOMs were generated and evaluated by applying multi-objective optimization algorithms, guided by our objectives and fitness functions. According to the results which are presented in Table 3, The total number of generated HOMs of order 2 to 15 was reduced to about 22% of the number of FOMs. While, according to results of Langdon et al. [10,11], the number of generated HOMs grows exponentially with order. This is because we use the proposed objectives and fitness functions to focus on constructing the valuable (high quality and reasonable) HOMs instead of generating all possible HOMs.

Finding 1. *Our approach leads to reduced cost of mutation testing by reducing the number of HOMs.*

5.2. Answer to RQ2

The mean values of mutation score indicator (MSI) in our experimentation and in Langdon et al.'s [10,11] experimentation after higher order mutation testing execution are presented in Table 4.

Table 4
The mean MSI values of HOMT (%)

Our experimentation (of order 2 to 15)	Langdon et al. (of order 2 to 4)
68	99

The experimental results indicate that our approach seems to be better than the approach of Langdon et al. [10,11] in terms of generating difficult-to-kill higher order mutants by applying multi-objective optimization algorithm. This is because the number of live (potentially equivalent) mutants, which cannot be killed by the given test suite (albeit could be killed by new and high quality test cases), in our approach (about 32%) is much larger than in Langdon et al.'s (about 1%).

Finding 2. *Our approach leads to hard to kill mutants, which mimic harder to find defects and drive development of higher quality test cases.*

5.3. Answer to RQ3

Answer to RQ3.1

The results presented in Figure 1 indicate that the majority of generated HOMs are **H6** (Non-Quality, Reasonable and With Old TCs) and **H1** (Live (potentially equivalent) mutants). A high number of **H6** shows that there are many generated HOMs which are more difficult to kill than FOMs and can only be killed by TCs belonging to the union of sets of TCs that can kill their constituent FOMs, but not all of their constituent FOMs. A large number of **H1** indicates that we need more new test cases (with better quality than the given test cases) to kill them.

The mean proportion of **H7** in total of generated HOMs is similar to **H3**, **H4** and **H10**, around from 4.2% to 5.5%, whilst the numbers of **H2**, **H5**, **H8**, **H9** and **H11** are very small.

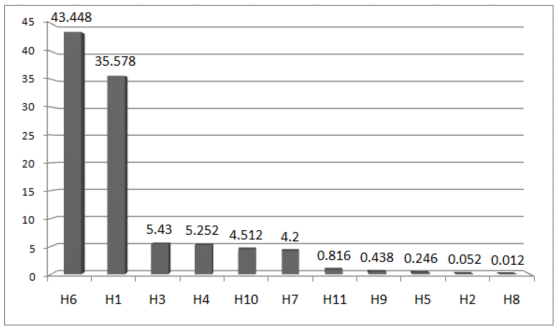


Fig. 1. The mean proportion of 11 categories of HOMs in total of HOMs(%)

Answer to RQ3.2

According to the experimental results (see Figure 1), the mean percentage of Non-quality and Unreasonable HOMs (H2 and H3, which are easier to kill than their constituent FOMs and there are no any test cases which can kill them and simultaneously all their constituent FOMs) is not large, around 5.5% in total.

Finding 3. *Our approach do not waste computational resources for creating mutants, which are easy to kill by most test cases, and addresses the problem of cost of mutation testing.*

Answer to RQ3.3, RQ3.4 and RQ3.5

Table 5 shows the mean ratios of number of HOMs (H1-H11) of a particular order (2-15) to all generated HOMs (Column 2) as well as the mean ratios of High quality - Reasonable HOMs (H7) and live HOMs (H1) to all produced HOMs in each individual order (Column 3 and 4). The results indicate that generally for lower orders, the number of generated HOMs is larger than for higher orders.

Obtained empirical results also show that we can find many "High Quality and Reasonable HOMs" (H7) from the $2nd - order$ to the $5th - order$. For the $6th - order$, as well as for higher orders, generated H7 are rare. There is lack of H7 in many cases (see Table 5 and Figure 2). As a result, we may conclude that higher order mutation up to the $5th - order$ can be rewarding wrt. searching for H7 by applying multi-objective optimization algorithms. While the ratio of Live (potentially equivalent) mutants (H1) to total number of HOMs is large, 22% to 55%, for every order (see Table 5 and Figure 2).

Table 5

The mean percentage of number of total generated HOMs (H1-H11), H7 and H1 in the mutant order (%)

Order	H1-H11	H7	H1
2	11.50	14.90	54.34
3	14.80	8.10	44.40
4	14.90	5.60	30.21
5	14.00	2.50	25.28
6	9.20	0.30	25.23
7	7.60	0.39	32.25
8	5.00	0.25	33.11
9	4.50	0.67	33.12
10	3.60	0.41	36.11
11	3.30	0.00	39.77
12	3.60	0.54	37.46
13	3.10	0.00	35.11
14	2.40	0.00	34.94
15	2.50	0.00	34.80

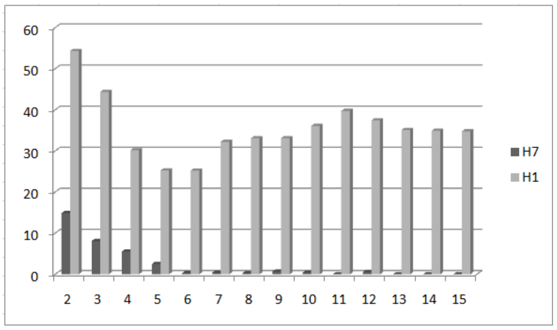


Fig. 2. The ratios of H1 and H7 to all generated HOMs per order (%)

Finding 4. *Five (5) is a relevant upper bound on mutation order in higher order mutation testing as the ratio of "High Quality and Reasonable HOMs" (H7) to total number of HOMs is relatively high for orders between 2 and 5. This ratio is low, close to zero, for orders higher than 5, while the ratio of live (and potentially equivalent) mutants to the total number of HOMs is large for every order. Our finding suggests that a possible way to reduce the cost of mutation is by a limited order of HOMs.*

5.4. Answer to RQ4

Figure 3 shows the percentage of "High Quality and Reasonable HOMs" (H7) compared with total generated HOMs and the number of Reasonable HOMs (H4-H9). According to the results of five applied algorithms, the mean ratio of H7 to all found Reasonable HOMs (H4-H9) is about 8%. This number is relatively high because the proportion of all Reasonable HOMs to all generated HOMs is quite a large (see Table 6), while the ratios of the number of H1 to the total number of HOMs are more or less the same for five algorithms.

Table 6
The ratios of Reasonable HOMs to generated HOMs

Algorithm	%Reasonable HOMs	%Live HOMs
eMOEA	55.10	35.58
NSGAII	54.93	34.16
eNSGAII	52.24	37.61
NSGAIII	54.44	34.26
Random	51.27	36.28

This indicates that we can find the mutants that are harder to kill and more realistic (reflecting real, complex faults) than FOMs by applying multi-objectives optimization algorithm. Especially "High Quality and Reasonable HOMs" can be used to replace the set of their constituent FOMs without lost of testing effectiveness. In addition, our experimental results indicate that the mean ratio of the number of TCs which kill generated HOMs to the number of TCs which kill their constituent FOMs is about **0.6**. This means that, basically, the HOMs which were generated by applying multi-objective optimization algorithms based on our approach are harder to kill than the generated FOMs in terms of number of test cases that can kill the mutants.

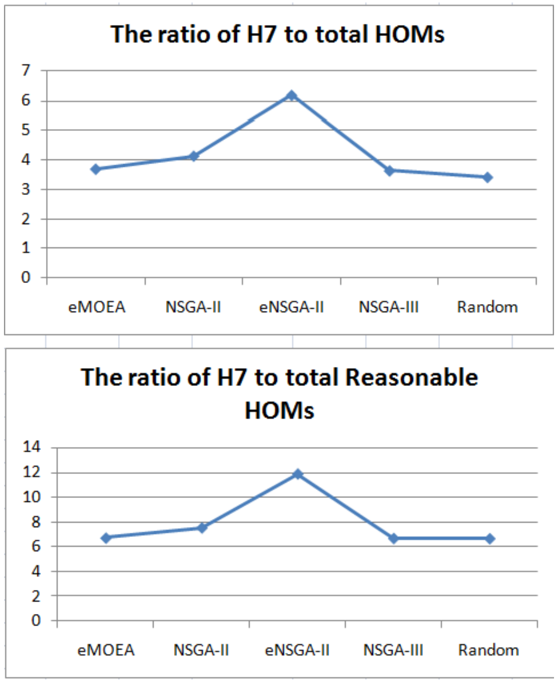


Fig. 3. The ratios of H7 to total HOMs and Reasonable HOMs(%)

Finding 5. *Our approach leads to reduced cost of mutation testing, increases realism of mutants, generates harder-to-kill mutants than FOMs and drives development of higher quality test cases at the same time.*

According to our experimental results, the *eMOEA* algorithm is the best in terms of constructing Reasonable HOMs (H4-H9) (see Table 6), while the *eNSGAII* algorithm is the best for searching for "High Quality and Reasonable HOMs". Approximately 12% of Reasonable HOMs, which were found by *eNSGAII* algorithm, are classified as "High Quality and Reasonable HOMs" (see Figure 3).

In 2004, McConnell, who has been two-time winner of the Software Development Magazine Jolt Award, came to the following conclusions based on the results of his study [18]: *"Industry average experience is about 1–25 errors per 1000 lines of code for delivered software"* and *"The Applications Division at Microsoft experiences about 10–20 defects per 1000 lines of code during in-house testing and 0.5 defects per 1000 lines of code in released product"*. Hence, we may speculate that in the complete versions of the software projects, a single line of code rarely has more than one error. From that, we propose an approach to modify the multi-objective optimization algorithm applied to construct higher order mutants. Instead of creating a random initial list of HOMs (parent population) from list of FOMs, we create an initial list of HOMs by combining n FOMs guided by the rule *"apply no more than one mutation operator to each line of code"*. We choose the *eNSGAII* algorithm to modify, because it is the best algorithm in terms of constructing the "High Quality and Reasonable HOMs". The modified algorithm is named **eNSGAII-DiffLOC** algorithm.

Results of the empirical comparison of two algorithms (eNSGAII and eNSGAII-DiffLOC) are shown in Table 7. The notations are explained below:

- NoM is the ratio of the number of generated HOMs to the number of FOMs.
- NoT is the ratio of the number of TCs which kill generated HOMs to the number of TCs which kill their constituent FOMs.
- NoR is the ratio of generated "reasonable HOMs" to all generated HOMs.
- NoH7 is the ratio of H7 ("High Quality and Reasonable HOMs") to all generated HOMs.
- NoH1 is the ratio of H1 (live (potentially equivalent) HOMs) to all generated HOMs.

Table 7
A comparison of two algorithms (%)

Algorithm	NoM	NoT	NoR	NoH7	NoH1
eNSGAII	15.33	62.4	58.03	6.14	31.79
eNSGAII-DiffLOC	14.72	60.01	59.25	6.72	33.01

The obtained results indicate that *eNSGAII-DiffLOC* is slightly better than the original *eNSGAII* algorithm in terms of mutant reduction (slightly smaller NoM of *eNSGAII-DiffLOC*), generates harder-to-kill mutant (slightly smaller NoT and bigger NoR of *eNSGAII-DiffLOC*) and is able to construct "High Quality and Reasonable HOMs" (slightly higher NoH7 of *eNSGAII-DiffLOC*).

On the other hand, slightly larger number of Live (potentially equivalent) mutants (NoH1) can be seen as a disadvantage, due to the fact that H1 mutants include, to some extent, equivalent mutants. However, H1 mutants include also "difficult-to-kill mutants", which are valuable.

Finding 6. *The proposed eNSGAII-DiffLOC seems to be slightly better than original eNSGAII algorithm in terms of mutant reduction, generating harder-to-kill mutant and constructing "High Quality and Reasonable HOMs".*

5.5. Answer to RQ5

To answer the RQ5, we have performed an empirical study, in which HOMs are generated in three ways. In the first way (HOMT1), HOMs are generated by combining FOMs from the set of all generated FOMs. In the second way (HOMT2), we delete first live FOMs from set of generated FOMs, and then create HOMs by combining FOMs from the set of killed FOMs. And the last (HOMT3), first delete all of easy-to-kill FOMs, which were killed by all of given TCs, from set of generated FOMs, then create HOMs by combining FOMs from the remaining set of FOMs (named set of not-easy-to-kill FOMs). Then we use mutation score indicator (MSI) as the indicator of usefulness of higher order mutation in driving development of TCs. The live mutants, which cannot be killed by the given test suite, make up a significant part of generated mutants and may drive the development of new test cases. The experimental results are shown in Table 8, in which *FOMT* is implementation of first order mutation testing.

Table 8
The mean value of MSI for each project under test (%)

PUT	Bar	Bean	Chain	Validator	JWBF
FOMT	15.79	15.11	41.65	47.10	12.96
HOMT1	69.82	38.23	87.28	92.69	92.35
HOMT2	98.72	100	100	99.77	100
HOMT3	63.94	63.59	46.69	85.91	82.90

The results presented in Table 8 indicate that the given sets of TCs of PUTs have lower MSI in first order mutation testing. It means that there are many live FOMs and the given sets of TCs are not good enough to detect the difference between original program and their mutants and, therefore, need to be improved following the results of mutation analysis based on the FOMT strategy. The numbers of live FOMs makes up from 53% to 87% of generated mutants. Only a small number of FOMs were killed by the given sets of TCs. In the case of live FOMs, we have to check whether the live FOMs are equivalent mutants or not, but it often involves additional human effort [16]. If mutants are not equivalent, developers or testers need to create new TCs, as in Test-Driven Development (TDD) or Continuous Test-Driven Development (CTDD) [15], and check whether they are able to kill live FOMs or not. If live FOMs are equivalent mutants, TCs, which can kill them, do not exist. The most striking result is that the HOMT2 strategy appeared to be useless as it gives a false impression that TCs are of high quality (MSI is equal or close to 100%) and the usefulness of HOMT2 is strongly limited, i.e., opportunities of test case improvement guided by results of HOMT2 mutation analysis are rare if any. Almost all of higher degree mutants, which were constructed by combining the killed FOMs, are also killed. This indicates that, combining first order killed mutants to create higher degree mutants is not a good way to evaluate and improve the quality of given set of test cases because the generated HOMs are easy to kill. The HOMT1 and HOMT3 strategies seem to be better and offer more opportunities to improve the quality of given set of test cases, as MSI (and the number of killed mutants) decreased in comparison to HOMT2.

Finding 7. *We should not use first order killed mutants to create difficult (but possible) to kill higher order mutants. Furthermore, using not-easy-to-kill mutants to generate higher order mutants seems to be a promising method to improve the quality of TCs.*

6. Conclusions and future work

In this paper, we have performed the experimental evaluation of the effect of applying multi-objective optimization algorithms in the area of higher order mutation testing based on our proposed HOMs classification, objectives and fitness functions using Judy mutation testing tool for Java and open source projects. The results indicate that our approach can be used to construct the difficult to kill and more realistic HOMs, especially "High Quality and Reasonable HOMs", as well as to reduce the number of generated HOMs. As a result, the approach can be used to improve the mutation testing effectiveness in general.

We have empirically evaluated our approach to HOMT presented some findings, which can be useful from the point of view of the effectiveness of mutation testing. Our approach can lead to reduced cost of mutation testing, can produce hard to kill mutants (which mimic hard to find defects and drive development of high quality test cases), suggests that there is an upper bound on the order of mutants, which allows to reduce the cost of mutation even further and do not waste computational resources for creating mutants, which are easy to kill by most test cases. The proposed approach, being a HOMT approach, also addresses the problem of realism of mutants. eNSGAII-DiffLOC, being a modified version of eSNGAII, seems to be a good starting point for further research and development of algorithms, which are better in terms of mutant reduction, harder-to-kill mutant generation and "High Quality and Reasonable HOMs" construction.

Further research is recommended because using only 5 selected projects under test (PUTs) may not be a representative sample of all Java programs. In addition, the obtained results may also depend on programming language, applied mutation tool, mutation operators and algorithms. Beside to this, the number of analyzed projects under test is too small to derive more firm conclusions using, even using robust statistical methods [9]. Hence, further research should focus on collecting more data. Furthermore, the obtained differences in the analyzed projects under test are small so new modifications and algorithms should be searched for and empirically evaluated. It would be also interesting to combine the proposed approach with a technique that utilizes a data-flow analysis to decrease the number of mutation points and consequently to reduce the number of higher order mutants [2], as well as to compare the outcomes of software reliability models applied to tests [27] with mutation testing.

References

[1] R.A. DeMillo, R.J. Lipton, and F.G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer*, **11**(4) (1978), 34–41.

[2] A.S. Ghiduk. Reducing the number of higher-order mutants with the aid of data flow. *e-Informatica Software Engineering Journal*, **10**(1) (2016), 31–50.

[3] A.A. Haider, A. Nadeem, and S. Rafiq. Multiple objective test suite optimization: A fuzzy logic based approach. *Journal of Intelligent and Fuzzy Systems*, **27**(2) (2014), 863–875.

[4] R.G. Hamlet. Testing Programs with the Aid of a Compiler. *IEEE Transactions on Software Engineering*, **3**(4) (1977), 279–290.

[5] M. Harman, Y. Jia, and W.B. Langdon. A Manifesto for Higher Order Mutation Testing. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '10, pp. 80–89, Washington, DC, USA, 2010. IEEE Computer Society.

[6] Y. Jia and M. Harman. Constructing subtle faults using higher order mutation testing. In *Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 249–258, 2008.

[7] Y. Jia and M. Harman. Higher order mutation testing. *Information and Software Technology*, **51** (2009), 1379–1393.

[8] Y. Jia and M. Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, **37**(5) (2011), 649–678.

[9] B.A. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering*, (in press) (2016). DOI: 10.1007/s10664-016-9437-5.

[10] W.B. Langdon, M. Harman, and Y. Jia. Multi objective higher order mutation testing with genetic programming. In *Testing: Academic and Industrial Conference - Practice and Research Techniques. TAIC PART '09.*, pp. 21–29, Sept 2009.

[11] W.B. Langdon, M. Harman, and Y. Jia. Efficient multi-objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, **83**(12) (2010), 2416–2430.

[12] L. Madeyski. On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests. In J. Münch and P. Abrahamsson, editors, *Product-Focused Software Process Improvement*, **4589** of *Lecture Notes in Computer Science*, pp. 207–221. Springer Berlin Heidelberg, 2007.

[13] L. Madeyski. Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites. *Software Process: Improvement and Practice*, **13**(3) (2008), 281–295.

[14] L. Madeyski. The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. *Information and Software Technology*, **52** (2010), 169–184.

[15] L. Madeyski and M. Kawalerowicz. Continuous Test-Driven Development—A Novel Agile Software Development Practice and Supporting Tool. In J. Filipe and L. Maciaszek, editors, *ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 260–267, 2013.

[16] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation. *IEEE Transactions on Software Engineering*, **40**(1) (2014), 23–42.

[17] L. Madeyski and N. Radyk. Judy – A Mutation Testing Tool for Java. *IET Software*, **4**(1) (2010), 32–42.

[18] S. McConnell. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA, 2004.

[19] E.S. Mresa and L. Bottaci. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, **9**(4) (1999), 205–232.

[20] Q.V. Nguyen. *Searching for high quality higher order mutants*. Wroclaw University of Science and Technology, PhD Thesis, 2016.

[21] Q.V. Nguyen and L. Madeyski. Problems of mutation testing and higher order mutation testing. In T. Do, H.A. Le Thi, and N.T. Nguyen, editors, *Advanced Computational Methods for Knowledge Engineering*, **282** of *Advances in Intelligent Systems and Computing*, pp. 157–172. Springer, 2014.

[22] Q.V. Nguyen and L. Madeyski. Searching for Strongly Subsuming Higher Order Mutants by Applying Multi-objective Optimization Algorithm. In H.A. Le Thi, N.T. Nguyen, and T.V. Do, editors, *Advanced Computational Methods for Knowledge Engineering*, **358** of *Advances in Intelligent Systems and Computing*, pp. 391–402. Springer, 2015.

[23] Q.V. Nguyen and L. Madeyski. Empirical evaluation of multi-objective optimization algorithms searching for higher order mutants. *Cybernetics and Systems*, **47**(1-2) (2016), 48–68.

[24] Q.V. Nguyen and L. Madeyski. Higher order mutation testing to drive development of new test cases: An empirical comparison of three strategies. In N.T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong,, editors, *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I*, **9621** of *Lecture Notes in Artificial Intelligence*, pp. 235–244. Springer, Berlin Heidelberg, 2016.

[25] Q.V. Nguyen and L. Madeyski. On the relationship between the order of mutation testing and the properties of generated higher order mutants. In N.T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong, editors, *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Da Nang, Vietnam, March 14-16, 2016, Proceedings, Part I*, **9621** of *Lecture Notes in Artificial Intelligence*, pp. 245–254. Springer, Berlin Heidelberg, 2016.

[26] M. Papadakis and N. Malevris. An empirical evaluation of the first and second order mutation testing strategies. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pp. 90–99, April 2010.

[27] H. Pham. A generalized fault-detection software reliability model subject to random operating environments. *Vietnam Journal of Computer Science*, **3**(3) (2016), 145–150.

[28] M. Polo, M. Piattini, and I. Garcia-Rodriguez. Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification, and Reliability*, **19**(2) (2008), 111–131.

[29] R. Purushothaman and D.E. Perry. Toward Understanding the Rhetoric of Small Source Code Changes. *IEEE Transactions on Software Engineering*, **31**(6) (2005), 511–526.

[30] A.M.R. Vincenzi, E.Y. Nakagawa, J.C. Maldonado, Márcio Eduardo Delamaro, and Roseli Aparecida Francelin Romero. Bayesian-Learning Based Guidelines to Determine Equivalent Mutants. *International Journal of Software Engineering & Knowledge Engineering*, **12**(6) (2002), 675–689.