Test Data Generation Techniques for Mutation Testing: A Systematic Mapping

Francisco Carlos M. Souza[†], Mike Papadakis[‡],
Vinicius H. S. Durelli[†], and Marcio Eduardo Delamaro[†]

[†]Universidade de São Paulo-USP,
Instituto de Ciências Matemáticas e de Computação (ICMC),
São Carlos – SP, Brazil
{fcarlos,durelli,delamaro}@icmc.usp.br
[‡]University of Luxembourg
Interdisciplinary Center for Security, Reliability and Trust (SnT),
Luxembourg – Luxembourg
michail.papadakis@uni.lu

Abstract. Mutation testing is regarded as an effective testing criterion. However, its main downside is that it is an expensive and time consuming technique, which has hampered its widespread adoption in industrial settings. Automatic test data generation is an option to deal with this problem. The successful automation of this activity requires the generation of a small number of tests that can achieve good mutation score. In this paper, we describe the process, results and discussions of a systematic mapping conducted to gather and identify evidence with respect to the techniques and approaches for test data generation in mutation testing. We selected 19 primary studies that focus on techniques based in search, code coverage, restrictions, and hybrid techniques. Our results indicate that most of the reported approaches are not industry-ready, which underscores the importance of further research in the area so that automatic test data generation can reach an acceptable level of maturity and be used in realistic settings.

Keywords: Mutation Testing, Test Data Generation, Systematic Mapping

1 Introduction

Software testing is a fundamental activity for achieving and assessing the quality of a software product [1]. The goal is to identify potential faults in order to increase the quality of software products. However, in general, it is not possible to fully automate software testing activities because of constraints related to undecidable problems.

Test criteria have been defined to help testers design sound test cases. An important test criterion that has been around for more than forty years is mutation testing [2,3]. The driving principle of mutation testing is using faults that mimic mistakes that a competent programmer would make. To simulate this, simple syntactic changes are applied to the original program, these changes yield faulty versions of the program called mutants. If the original program and a mutant

generate different outputs for a given test case, then the mutant is regarded as "dead". Therefore, the goal of mutation testing is to find a test set capable of killing a significant number of mutants.

In mutation testing, mutants can be categorized into First Order Mutants (FOMs) and Higher Order Mutants (HOMs) due to types and quantity of faults seeded. FOMs are generated by applying a mutation operator only once [4] whereas HOMs, introduced by Jia and Harman [5], are generated by applying mutation operators in the programs more than once.

Other important aspects of mutation testing are the concepts of strong and weak mutation that refer to way in each mutants are considered dead. For instance, strong mutation (traditional mutation testing) amounts to creating test cases that causes the program state to propagate to the output, where failures are observable [6]. Then, a test case is said to "strongly" kill a mutant when it causes the given mutant to produce an output that differs from the one generated by the original program [4]. As for weak mutation, the focus is shifted from entire programs to components, the idea is to kill mutants that lead the program to an erroneous state. Instead of checking mutants after the execution of the entire program, mutants are checked immediately after the execution of the mutated component.

One of the major challenges in mutation testing is generating test data. This challenge consist in identifying a set of test data that maximizes the number of mutants killed. The fundamental problem is to identify test data (x_1, x_2, x_3,..., x_k), so that, when the test data set is executed over the set of mutants (M_1, M_2, M_3, ..., M_n) it is obtained the highest number of mutants killed [7].

Despite that test sets can be generated and selected manually by a tester, this practice is complex and time-consuming, so there is a need for automated generation. Although test data generation is known to be complex a lot of effort has been put to automate the test data generation. In this context, the aim of this study is to describe a Systematic Mapping (SM) whose purpose is to identify test data generation approaches based on mutation testing. Also, this study identifies promising approaches that can be investigated for the same purpose.

It is important to highlight that there is no SM in the field of test data generation for mutation testing. The survey of Jia and Harman [4], which addresses mutation testing in general, mentions some related work in this context, but without further investigation as we report in this paper.

The rest of this paper is organized as follows: in Section 2 outlines how the SM herein described was planned and conducted. Section 3 describes results and threats to validity. Finally, Section 4, concludes with a summary of our findings and a discussion regarding the future directions for research in this area.

2 Systematic Mapping

SMs aim to establish gaps, clusters, and categorize existing evidence from a set of studies on a given search area [8]. SMs comprise a planning phase, which has to do with devising research questions and defining inclusion and exclusion criteria. The planning phase is followed by search and screening of primary studies. Usually, the data extraction step of an SM is broader than the data extraction process for an Systematic Literature Review (SLR).

Moreover, the analysis of a given SM does not include any sort of meta-analysis because data is often summarized through graphic representations [9]. We followed the systematic approach proposed by Kitchenham [10] to conduct a SM that gives an overview of the field of test data generation for mutation testing. Next section further describes the topics that our SM aims to shed light on by outlining the research questions we devised.

2.1 Research Questions

Research questions embody the purpose of a SM. An approach commonly used to formulate SM research questions is to follow the PIOC criteria. Using these criteria research questions are broken down to reflect four aspects: (i) population; (ii) intervention; (iii) outcome; and (iv) context [11]. The PIOC criteria utilized by us for creation of the research questions are shown in Table 1.

Table 1. PIOC criteria applied to our SM.

PIOC Criteria	
Population	mutation testing
Intervention	test data generation techniques
Outcomes	test data generation techniques for mutation testing
Context	with a focus on mutation testing

In this context, the aim this paper is to obtain an overview of test data generation techniques that have been used in the mutation testing domain. Towards this end, we devised two research questions:

- **RQ$_1$:** What techniques/approaches have been used for test data generation for mutation testing?
- **RQ$_2$:** How can these test data generation techniques/approaches be classified?

2.2 Search Strategy

To identify relevant primary studies we followed a search strategy that encompassed two steps: definition of the search string and selection of the databases to be used. The search string was created based on the following keywords: *"mutation testing"* and *"test data generation"*. For each keyword, their synonyms and terms with similar meaning were identified. These keywords and their synonyms were combined by using the boolean operators *AND* and *OR*. Such a search string was then adapted according to the characteristics of each search engine. An overview of the resulting search string is shown below.

```
(''test data generation'' OR ''test data generator'' OR
''data generation'' OR ''test case generation'' OR ''test
case generator'' OR ''case generation'') AND (''mutation
test'' OR ''mutation testing'' OR ''mutation analysis'')
```

The next step was to define which databases the search string should be applied to. We selected seven databases *(IEEE Xplore, ACM Digital Library, Compendex, ScienceDirect, Scopus, Springer, Web of Knowledge)* which are deemed as the most relevant scientific sources [12] and therefore, likely to contain important primary studies.

2.3 Inclusion and Exclusion Criteria

To assist in the selection of relevant primary studies that address our research questions we have applied a set of inclusion and exclusion criteria. Studies were included in our SM only if they met the following inclusion criteria (IC):

- **IC_1:** to be included into our SM, a study has to describe at least one test data generation for mutation testing technique/approach;

With regard to the exclusion criteria (EC), studies were excluded if they met one of the following criteria:

- **EC_1:** study is not available in electronic format;
- **EC_2:** duplicate studies reporting similar results;
- **EC_3:** study is not written in English;
- **EC_4:** study is not a full paper or short paper (e.g., posters, tutorials, technical reports, thesis, and dissertations).

2.4 Study Selection

The search string was used on the selected electronic databases to identify candidate primary studies. At first, duplicate papers were excluded, resulting in 154 studies. Afterwards, titles, abstracts, introductions and conclusions were read and the inclusion and exclusion criteria were applied, which resulted in 24 studies. These 24 candidate primary studies were read in full and our inclusion and exclusion criteria were applied again. Eventually, five primary studies were removed from the set of selected papers because they not presented test data generation techniques/approaches for mutation testing which resulted in a final set containing 19 primary studies.

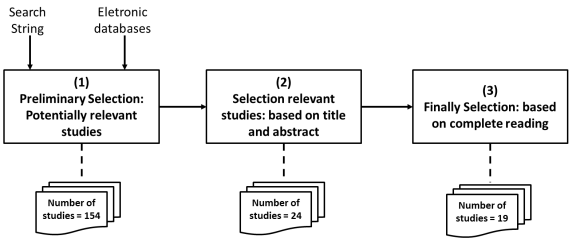

Fig. 1. Selection Process of Primary Studies

Thus, we have identified 19 relevant studies from the five electronic databases that meet the RQ addressed in this SM. All primary studies are presenting

techniques or approaches to generate test data for mutation testing and they are listed in the next section.

Furthermore, to assist in the extraction process, information concerning the analysis of all studies was stored using JabRef. We designed a data extraction form to support the data collection and synthesis steps. In addition to the standard information, the data extraction form also included information about the type of techniques employed, the number of programs used in the evaluation, the programming language in which the approach was implemented, the mutation scores achieved, execution time of the proposed approach. Thus, to keep track of the information reported in the of the selected primary studies, we used a Microsoft Excel spreadsheet and R (which is a statistical environment) for graph generation.

3 Analysis and Discussion of Results

In this section the techniques described in the primary studies are detailed (RQ_1) and categorized (RQ_2). Also, some results of the primary studies are discussed. Table 2 shows the 19 primary studies we selected. The fourth and fifth column presents how the primary studies were evaluated and the databases which they were identified. The last column indicates the venue in which the primary study was published: conference proceedings (C), journal (J), symposium (S), and workshop (W).

Table 2: Overview of the primary studies.

Study	Author	References	Validated	Database	Type
S1	Papadakis-2013	[13]	Experiment	Scopus	J
S2	Louzada-2012	[14]	Experiment	IEEE	C
S3	Haga-2012	[15]	Experiment	Compendex	C
S4	Papadakis-2012	[16]	Experiment	Compendex	J
S5	Harman-2011	[17]	Experiment	ACM	S
S6	Nica-2011	[18]	Experiment	IEEE	C
S7	Papadakis-2010	[19]	Experiment	ACM	W
S8	Rad-2010	[20]	Experiment	IEEE	C
S9	Fraser-2010	[21]	Experiment	ACM	S
S10	Papadakis-2010	[22]	Experiment	Compendex	S
S11	Mishra-2010	[23]	-	Compendex	S
S12	Zhang-2010	[24]	Experiment	Scopus	C
S13	Papadakis-2009	[25]	Experiment	Scopus	C
S14	Ayari-2007	[7]	Experiment	ACM	C
S15	May-2007	[26]	Experiment	Springer	C
S16	Liu-2006	[27]	Experiment	IEEE	C
S17	Masud-2005	[28]	-	IEEE	C
S18	Demillo-1991	[6]	Experiment	IEEE	J
S19	Offutt-1990	[29]	-	IEEE	C

As shown in Table 2, most primary studies were published in conference proceedings. Upon further analysis, we found that 89% (17 out of 19) of the selected studies present some sort of empirical evaluation aimed at validating the underlying test data generation technique.

3.1 Description Test Data Generation Techniques/Approaches

Ant Colony Optimization – ACO [30]: describes an approach based on intelligence of ant colony, formalized to solve combinatorial optimization problems, inspired by the foraging behavior of some ant species. The process of obtaining the solution with the ACO can adopt a representation by graphs, on which the ants are agents that process the construction of the solution by probabilistic selection of the node.

Bacteriological Algorithms – BA [31]: this study is inspired by the concept of evolutionary ecology[1] to improve test sets. In this context, BA aims at the mutation in the initial population (test cases) to adapt itself to a particular environment. The adaptation is based on small changes in individuals, which are called bacteria, and they are evaluated using a particular function based on the problem. BA receives, as input, an initial test cases set and generates, as output, test cases, supposedly better than the initial set.

Genetic Algorithm – GA [33]: consists of an iterative process aiming to identify the best solution from a set of solutions (population) for a given problem. The process starts with a population of candidate solutions (test cases) typically randomly generated. Then, the structure of the individuals, named chromosomes, are evaluated with a fitness function predefined and those that are more suitable to achieve the solution of the problem are sent to reproduction and consequently, generate new individuals for population. This population evolves on each iteration until the algorithm reaches a stopping criterion, such as a fixed number of generations.

Hill Climbing – HC [34]: this search technique combines a general search method (generate-and-test) with the use of functions for evaluating the states generated by the method. This technique aims to identify the best path to be followed in the search and then, it returns a satisfactory result for a given problem. Therefore, the algorithm consists in selecting an initial solution randomly, evaluating it and improving it step by step, from the investigation of the neighborhood of the current solution.

Immune Inspired Algorithm – IIA [35]: Castro and Timmis employ models, theories, and principles of human immune system as metaphors to help develop new systems. The IIA consists of an iterative process that evolves from a population of candidate solutions called antibodies. Each antibody corresponds to a simple test data, seeking those that can kill at least one mutant program that has not been killed by a test data existing. The antibodies identified as useful are added to a test set and at the end of process a suitable data test set is obtained.

[1] Evolutionary Ecology is the study of living organisms in the context of their environment in order to discover how they adapt [32].

Enhanced Control Flow Graph – ECFG [25]: this technique considers the problem of killing mutants in branch coverage problem and aims to select the best paths of the program for test activity, i.e., paths that have higher probability to facilitate this activity, either by its complexity, testability, effectiveness or applicability, avoiding trapping, spending too much time on equivalent mutants or on those hard to kill.

Improved Iterative Relaxation Method – IIRM [36]: is the improvement of the Iterative Relaxation Method proposed by Gupta [37]. The method IIRM aims at test data generation to kill multiple mutants in mutation testing. The test data are generated based on information about reachability and necessity conditions of mutants proposed by Demillo and Offutt [38].

Constraint-Based Testing – CBT [6]: is a technique proposed by DeMillo and Offutt that uses an algebraic constraint system to detect specific types of defects in the program under test. CBT was the inspiration to define the three conditions that must be satisfied so that a test case can kill a mutant: *i)* Reachability; *ii)* Necessity; and *iii)* Sufficiency.

Distinguishing Test Cases – DTC [18]: allows to discriminate two programs (original program and mutant) through a input. Both the original program and its alive mutants are converted to a constraint satisfaction problem (CSP) [2]. Then, ask the constraint solver to search an identical input for the two programs, such that they differ by at least one output value.

Dynamic Symbolic Execution – DSE [40]: also known as concolic execution. This technique combines the concrete execution (real) with symbolic execution (identification of concrete values for the input parameters by constraints solver) of a program for test data generation. Thus, during the execution of a specific program, along with its execution path, a set of symbolic constraints is generated. The restrictions must be resolved so that the adequate test data can be generated to guide the program execution [40].

It is worth highlighting that, some of the techniques described were used by more than one primary study. Figure 2 shows the distribution of studies according to the respective techniques and year of publication.

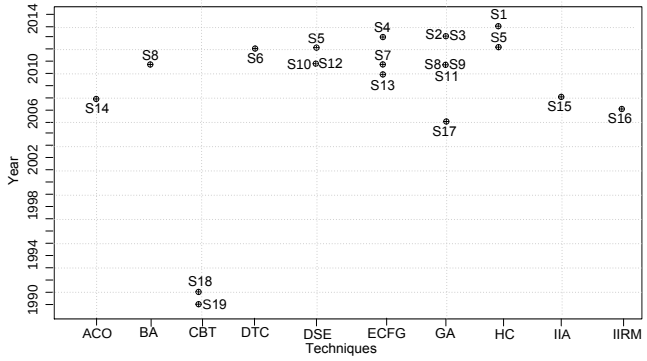

Fig. 2. Evidence of primary studies by techniques

[2] CSP consists of a set of variables with domains on which allowed value combinations are specified as constraints [39]

Each marker represents the primary study that reported a certain technique. We observed that two studies ($S5$ and $S8$) used two techniques simultaneously. As it can be seen, the most used technique was the GA, which was mentioned in six studies ($S2$, $S3$, $S8$, $S9$, $S11$, and $S17$).

3.2 Classification of the Techniques

After analyzing the 19 primary studies, we identified 10 test data generation techniques. These techniques were classified into four broad categories: C1-search based techniques; C2-code coverage based techniques; C3-restriction based techniques; and C4-hybrid techniques.

The aforementioned categories were derived based on the characteristics of each technique. C1 comprises techniques that use heuristics (34% of the studies) to identify the best solution in a search space of a given problem. C2 groups techniques (14% of the studies) that use code coverage, analysis of the data flow and control flow, to generate test data. C3 includes studies (11%) describing techniques using resolution of restriction systems and in C4 11% of the studies combine one or more techniques to generate test data. Furthermore, the study S5 is grouped in the categories C1 and C4 because it uses the HC and DSE techniques respectively (Figure 3).

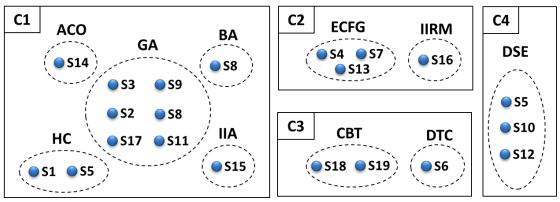

Fig. 3. Studies organized according to their categories.

Despite the fact that C1 is the category containing more studies, most approaches in this category have concentrated in using genetic algorithms (6 studies). Usually, search-based techniques rely on fitness functions or objective functions. In software testing, these functions are defined to measure how well a test satisfies a specific criteria, i.e, they determine the problem to be solved. However, if fitness functions are not well defined, the results obtained can fall short of their intended goal. On the other hand, search-based techniques have been successfully used in many optimization problems and the utilization for test data generation is a promising possibility to solve these problems.

3.3 Discussion Primary Studies

Based on the results obtained in the 19 primary studies, we found that several authors have tried to reduce computational costs and optimize the test data generation process for mutation testing. The main goal is to generate test data that can detect non-equivalent mutants in a shorter time frame, by reducing the costs of test data generation process and improving the quality of the resulting test data set.

By analyzing our SM results we observed that most of the studies are related to the use of search-based algorithms or propose techniques based on CBT and conditions that must be satisfied to kill a given mutant as described by DeMillo and Offutt [6], such as studies S5 and S10. Furthermore, we have collected information regarding whether comparative evaluations (column 3) were carried out (Table 3) in order to analyze the quality of the empirical evaluations conducted by the selected studies.

Table 3: Primary Studies Analysis I

Studies	**Technique**	**Comparative Evaluations**
S1 Papadakis-2013	HC / Fitness Squema	Fitness
S2 Louzada-2012	Elist GA	GA with and without elitism
S3 Haga-2012	GA	With and without GA / manual and automatic generation
S4 Papadakis-2012	ECFG	-
S5 Harman-2011	DSE and HC	DSE variations
S6 Nica-2011	DTC	-
S7 Papadakis-2010	ECFG	JPF-SE, Concolic and eTOC tools
S8 Rad-2010	BA/GA	BA and GA
S9 Fraser-2010	GA	Manual and Automatic generation
S10 Papadakis-2010	DSE	-
S11 Mishra-2010	Elist GA	-
S12 Zhang-2010	DSE	Pex tool for structural testing
S13 Papadakis-2009	ECFG	Branch testing
S14 Ayari-2007	ACO	HC, GA and Random Search
S15 May-2007	IIA	GA
S16 Liu-2006	IIRM	Tool that generates test data for killing a single mutant
S17 Masud-2005	GA	Not performed
S18 Demillo-1991	CBT	Not performed
S19 Offutt-1990	CBT	-

Most studies have reported satisfactory results for the empirical evaluations that were conducted, but most evaluations did not take into account variables such as time, computational cost, and equivalent mutants. When a large number of mutants is considered, the time required for performing mutation analysis, generating test data, and executing mutants is longer. Moreover, in the current state of the art, most of the identified studies use "toy" programs that barely resembled real-world programs as subjects in their evaluations. Thus, for techniques and approaches to reach a level of performance that would be valuable in practice it is necessary that future studies in the area perform empirical evaluations using real-world programs that are used in industrial settings.

Additionally, it is important to highlight the costs and resources required for the application of techniques and approaches. Five studies ($S3$, $S5$, $S12$,

$S16$, and $S18$) have reported the time for test data generation, but not the time spent in the whole test data generation process. Regarding studies that used metaheuristic and search-based techniques, they provide information on the amount of iterations determined by the authors, but not the time required for the process. Such a measure is important for the application of the technique.

Table 4 presents some information related to studies (Std.) in column 1; the number of programs (Prog. Numb.) used to perform the empirical evaluation in the studies in column 2; programming language (Prog. Lang.) in column 3; the size of the largest program used in the evaluation (Largest Prog.) in column 4; Mutation Score (MS) obtained in column 5; and the last two columns present information regarding the time for generating test data and the number of iterations/fitness assessment that has been defined in studies that used search-based approach and metaheuristics.

Table 4: Primary Studies Analysis II

Std.	**Prog. Numb.**	**Prog. Lang.**	**Largest Prog.**	**MS**	**Time**	**Iteration**
S1	8 laboratory programs	Java	90 LoC	-	-	50000 fitness assessment
S2	3 commercial benchmarks	Java	-	92.2%	-	300/ 500/ 1000 generations
S3	1 laboratory program	C	93 LoC	96%	137 seconds average	-
S4	30 programs units	Various	96 LoC	-	-	1/ 50/ 300/ 5000/ 50000 paths
S5	8 industrial programs / 10 laboratory programs	C	9564 LoC	First Order: 69% Second Order: 71%	307/ 301 minutes	-
S6	8 classes	Java	100 LoC	-	-	-
S7	10 laboratory programs	Java	-	86.3%	-	-
S8	1 laboratory program	Java	-	87.5% /90.6%	-	16/ 17 generations
S9	2 non-trivial examples	Java	413 classes	72%	-	50 generations
S10	2 non-trivial examples + 3 small Siemens suite examples	C	500 LoC	63%	-	8395 average iterations

Continuation on next page

Table 4 – *Primary Studies Analysis II*

Std.	Prog. Numb.	Prog. Lang.	Largest Prog.	MS	Time	Iteration
S12	5 data structure and algorithm programs	.Net	472 LoC	Weak mutation score: 89.83% / Strong mutation score: 83.40% (i.e., greater than 1 second)	1/ 5/ 10/ 20/ 40 seconds	-
S13	8 java program units	Java	45 LoC	90.2%	-	-
S14	2 laboratory programs	Java	72 LoC	88.5%	-	8000/ 10000 iterations
S15	4 laboratory programs	-	-	89.2%	-	500 iterations
S16	5 laboratory programs	-	-	94.2%	59.02 minutes	-
S18	5 laboratory programs	Fortran	55 LoC	98.4%	7 minutes	-

In this SM, we also identified some studies [6, 17, 22] performed experiments and concluded that it is possible to achieve good mutation scores on strong and weak mutation using test data that satisfy at least the reachability and infection conditions. The main problem to satisfy the Propagation condition is that from the infection point there may be a infinity of paths which can propagate a fault and to test all paths is unwieldy.

During the analysis of the selected studies, we noticed that in addition to using approaches for test data generation, the aforementioned concepts (FOMs, HOMs, Strong and Weak Mutation) were also used to reduce the number of mutants generated and the computational cost of the mutation testing such as HOMs (study $S5$), weak mutation (studies $S4$, $S7$, $S10$, $S12$, and $S13$), strong mutation (study $S12$) and mutant schemata technique (study $S7$). We observed that such strategies may have been used in order to balance the computational cost necessary for the use of the tool or technique for data generation with the number of executions associated to the high number of mutants produced.

3.4 Threats to Validity

The most common limitations in a SM are the possible biases introduced in the selection process and inaccuracies of the data extraction. These limitations were identified as potential threats to the validity of this SM and its results. Whether our proposed classification is coherent also represents a threat to validity. It is also worth mentioning that since a limited set of search engines was used, we cannot rule out the possibility that we might have missed some papers. We tried

to mitigate this threat by selecting the search engines that are most likely to include all relevant papers.

Internal validity: the selection process, was performed by a single researcher, however, the included and excluded studies were checked by another researcher, without compromising the quality of the SM. Regarding the extraction process, some papers omit important information, which forced the researcher carrying out the SM to seek such information in related papers. Another internal validity threat stems from research that had outcomes that were not made available. In addition, carrying out a manual search could have improved coverage of the primary studies, thus we acknowledge that the possible inclusion of studies would have contributed in increasing internal validity. Furthermore, in order to ensure a fair selection process and avoid bias we developed a research protocol based on the guidelines proposed by Kitchenham [10].

External validity: we can relate it to the degree to which the primary studies are representative of the overall goal of the review. We believe that our review protocol helped us achieve a representative set of studies.

4 Concluding Remarks

Mutation testing is considered a promising and powerful criteria. However, it is not widely used in practice due to its intricacies: for instance, it is not straightforward to efficiently generate test data based on mutants.

Test data generation has been viewed as a complex activity, because it is necessary to find a trade-off between a set of test data that is appropriate with the amount of data required to kill the mutants. Thus, it is possible to achieve successful reduction of costs, consequently allowing the mutation testing to be more useful and manageable.

The results of this SM indicate that there is still a research gap in this area. Despite years of research, we identified 19 studies with 10 different techniques for test data generation and only few studies provide a comprehensive evaluation including industrial scale programs. Additionally, no study determines the level of complexity to apply the techniques and measure their feasibility. In contrast, our analysis highlighted some trends like the use of search-based algorithms and metaheuristics, which seem to have achieved encouraging results in the selected studies.

In this context, even with the performance increase of mutation testing and the use of search-based techniques, code coverage based techniques, restrictions and hybrid techniques, test data generation are challenging problems that need investigation. According to the results described in the primary studies, there is still room for improvement so that the presented techniques can be employed in industrial settings achieving satisfactory performance.

Although the approaches reported in the studies are steps towards advancing this theme, there is a need to perform experiments on real programs, as well as in industrial environments, in order to obtain convincing evidence that the techniques and approaches can be used in practice and thus, motivate the use of mutation testing in industrial environments. Another gap identified is related to the application of the techniques, since the cost of application can be prohibitive for the test project, considering variables such as time and processing for the test data generation and cost to confront them with the mutants.

As future work we intend to evaluate the feasibility of these techniques, verify their complexity, and cost. We believe that this can be regarded as a first step towards automating the generation of test data for mutation testing.

References

1. Myers, G.J., Sandler, C., Badgett, T.: The Art of Software Testing. 3rd edn. Wiley (2011)
2. Demillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Journal Computer **11** (1978) 34–41
3. DeMillo, R.A., Sayward, F.G., Lipton, R.J.: Program mutation: A new approach to program testing. In: Infotech International State of the Art Report: Program Testing. (1979) 107–126
4. Harman, M., Jia, Y.: Analysis and survey of the development of mutation testing. Journal IEEE Transactions on Software Engineering **37** (2009) 649–678
5. Jia, Y., Harman, M.: Higher order mutation testing. Journal of Information and Software Technology **51** (2009) 1379–1393
6. DeMillo, R.A., Offutt, A.J.: Constraint-based automatic test data generation. IEEE Transactions on Software Engineering **17** (1991) 900–910
7. Ayari, K., Bouktif, S., Antoniol, G.: Automatic mutation test input data generation via ant colony. In: The Genetic and Evolutionary Computation Conference, ACM (2007) 1074–1081
8. Budgen, D., Turner, M., Brereton, P., Kitchenham, B.: Using mapping studies in software engineering. In: Proceedings of the 20th Annual Meeting of the Psychology of Programming Interest Group (PPIG), Lancaster, United Kingdom, Lancaster University (2008) 195–204
9. Kitchenham, B.: Procedures for performing systematic reviews. Technical report, Technical Report TR/SE-0401. Department of Computer Science, Keele University and National ICT. Australia. (2004)
10. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. In: EBSE Technical Report. (2007)
11. Petticrew, M., Roberts, H.: Systematic Reviews in the Social Sciences: A Practical Guide. Blackwell Publishing (2006)
12. Dyba, T., Dingsoyr, T., Hanssen, G.K.: Applying systematic reviews to diverse study types. In: International Symposium on Empirical Software Engineering and Measurement (ESEM '07). (2007) 225–234
13. Papadakis, M., Malevris, N.: Searching and generating test inputs for mutation testing. Journal SpringerPlus **2** (2013) 1–12
14. Louzada, J., Camilo-Junior, C.G., Vincenzi, A., Rodrigues, C.: An elitist evolutionary algorithm for automatically generating test data. In: World Congress on Computational Intelligence (WCCI'12), IEEE (2012) 1–8
15. Haga, H., Suehiro, A.: Automatic test case generation based on genetic algorithm and mutation analysis. In: International Conference on Control System, Computing and Engineering (ICCSCE'12), IEEE (2012) 119–123
16. Papadakis, M., Malevris, N.: Mutation based test case generation via a path selection strategy. Journal Information and Software Technology **54** (2013) 915–932
17. Harman, M., Jia, Y., Langdon, W.B.: Strong higher order mutation-based test data generation. In: ACM SIGSOFT Symposium on the Foundations of Software Engineering. (2011) 212–222
18. Nica, S.: On the improvement of the mutation score using distinguishing test cases. In: International Conference on Software Testing, Verification and Validation Workshops (ICSTW '11), IEEE Computer Society (2011) 423–426
19. Papadakis, M., Malevris, N., Kallia, M.: Towards automating the generation of mutation tests. In: 5th Workshop on Automation of Software Test. (2010) 111–118
20. Rad, M., Akbari, F., Bakht, A.: Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing. In: Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on. (2010) 1–6

21. Fraser, G., Zeller, A.: Mutation-driven generation of unit tests and oracles. In: International Symposium on Software Testing and Analys (ISSTA'10). (2010) 147–157
22. Papadakis, M., Malevris, N.: Automatic mutation test case generation via dynamic symbolic execution. In: 21st International Symposium on Software Reliability Engineering (ISSRE'10). (2010) 121–130
23. Mishra, K.K., Tiwari, S., Kumar, A., Misra, A.: An approach for mutation testing using elitist genetic algorithm. In: International Conference on Computer Science and Information Technology, IEEE (2010) 426–429
24. Zhang, L., Xie, T., Tillmann, N., Halleux, J., Mei, H.: Test generation via dynamic symbolic execution for mutation testing. In: Internacional Conference on Software Maintenance (ICSM'10). (2010)
25. Papadakis, M., Malevris, N.: An effective path selection strategy for mutation testing. In: Asia-Pacific Software Enginneering Conference ASPEC'09, IEEE (2009) 422–429
26. May, P., Timmis, J., Mander, K.: Immune and evolutionary approaches to software mutation testing. In: Proceedings of the 6th international conference on Artificial immune systems. ICARIS'07, Springer-Verlag (2007) 336–347
27. Liu, M.H., Gao, Y.F., Shan, J.H., Liu, J.H., Zhang, L., Sun, J.S.: An approach to test data generation for killing multiple mutants. In: Proceedings of the 22nd IEEE International Conference on Software Maintenance. ICSM '06, IEEE Computer Society (2006) 113–122
28. Masud, M., Nayak, A., Zaman, M., Bansal, N.: Strategy for mutation testing using genetic algorithms. In: Electrical and Computer Engineering, 2005. Canadian Conference on. (2005) 1049–1052
29. Offutt, A.J.: An integrated system for automatically generating test data. In: Proceedings of the first international conference on systems integration on Systems integration '90. ISCI '90, Piscataway, NJ, USA, IEEE Press (1990) 694–695
30. M., D., Di Caro, G.: The Ant Colony Optimization meta-heuristic. New ideas in optimization (1999)
31. Baudry, B., Flrurey, F., Jean-Marc, J., Le Traon, Y.: From genetic to bacteriological algorithms for mutation-based testing. Software Testing Verification Reliability **15** (2005) 73–96
32. Pianka, E.R.: Evolutionary Ecology. Benjamin-Cummings Publishing Company (1999)
33. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1989)
34. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. 2 edn. Pearson Education (2003) Chapter 4.
35. Castro, L.N., Timmis, J.: Artificial Immune Systems A New Computational Intelligence Approach. 1st ed. Springer (2002)
36. Shan, J.H., Wang, J., Qi, Z.C., Wu, J.P.: Improved method to generate path-wise test data. Journal of Computer Science and Technology **18** (2003) 235–240
37. Gupta, N., Mathur, A.P., Sofia, M.L.: Automated test data generation using an iterative relaxation method. In: International Symposium on Foundations of Software Engineering. (1998)
38. Offutt, A.J.: An integrated automatic test data generation system. Journal of Systems Integration **1** (1991) 391–409
39. Soininen, T., Gelle, E.: Dynamic constraint satisfaction in configuration. Technical report (1999)
40. Clarke, L.A., Richardson, D.J.: Applications of symbolic evaluation. Journal of Systems and Software **5** (1985) 15–35