Seeding Strategies for Multi-Objective Test Case Selection

An Application on Simulation-based Testing

Aitor Arrieta
Mondragon University
Mondragon, Spain
aarrieta@mondragon.edu

Joseba Andoni Agirre
Mondragon University
Mondragon, Spain
jaagirre@mondragon.edu

Goiuria Sagardui
Mondragon University
Mondragon, Spain
gsagardui@mondragon.edu

ABSTRACT

The time it takes software systems to be tested is usually long. This is often caused by the time it takes the entire test suite to be executed. To optimize this, regression test selection approaches have allowed for improvements to the cost-effectiveness of verification and validation activities in the software industry. In this area, multi-objective algorithms have played a key role in selecting the appropriate subset of test cases from the entire test suite. In this paper, we propose a set of seeding strategies for the test case selection problem that generate the initial population of multi-objective algorithms. We integrated these seeding strategies with an NSGA-II algorithm for solving the test case selection problem in the context of simulation-based testing. We evaluated the strategies with six case studies and a total of 21 fitness combinations for each case study (i.e., a total of 126 problems). Our evaluation suggests that these strategies are indeed helpful for solving the multi-objective test case selection problem. In fact, two of the proposed seeding strategies outperformed the NSGA-II algorithm without seeding population with statistical significance for 92.8 and 96% of the problems.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

Test Case Selection, Search-based Software Testing, Regression Testing

ACM Reference Format:
Aitor Arrieta, Joseba Andoni Agirre, and Goiuria Sagardui. 2020. Seeding Strategies for Multi-Objective Test Case Selection: An Application on Simulation-based Testing. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377930.3389810

1 INTRODUCTION

Generally, verification and validation activities are time consuming on large software systems. In systems like Cyber-Physical Systems (CPSs), testing is time consuming as it requires execution at different levels, even for the same software versions [7, 16]. In other contexts, such as Software Product Lines (SPLs), there are a large number of potential configurations, which makes it infeasible to test every single configuration thoroughly [15, 61]. To deal with all these problems, search algorithms have been proposed in the last few years with the goal of increasing the cost-effectiveness of several verification and validation activities. These activities include several optimization aspects, including automated test case generation [1, 2, 4, 17, 32, 40, 47, 51, 52, 58, 64], test case selection/minimization [66, 68, 69, 71] and test prioritization [13, 28, 30, 41].

A widely investigated technique for increasing the cost- effectiveness of the verification and validation processes has been regression test selection [27, 70]. These approaches have already been successfully deployed in industry [33, 57]. In the last few years, test selection based on evolutionary algorithms have gained important attention. Most of them have focused either on comparing (1) which adequacy criteria could fit best for integrating it in the fitness functions [6, 7, 36, 38, 68] or (2) which algorithm performs best when selecting test cases (when having one specific fitness function) [10, 62, 66, 67]. Additionally, most of them compare their approaches with a baseline algorithm, such as, Random Search (RS) [6, 7, 10, 62, 66, 67] or Greedy [68].

A common practice in other search-based software engineering problems has been to seed the initial population with certain seeding strategies. The results of this have been positive in several applications, including test generation [31, 45, 63] and service composition [18, 19]. In the case of test case selection, many studies have proposed either different algorithms or fitness functions [7, 9, 23, 33, 57, 67–69]. However, little attention has been paid to propose seeding strategies for multi-objective test case selection. For instance, Panichella et al. proposed a diversity-based genetic algorithm that seeded the initial population with orthogonal arrays by employing a Hadamard matrix [59]. Nevertheless, the approach presented by Panichela et al. [59] was algorithmic, and the seeding strategy for initializing the population needed to be accompanied by other mechanisms that injected diversity during the search process.

In this paper, we present three seeding strategies designed for population-based search algorithms for test selection (one of which is configurable). In addition, we re-implement the strategy for automatically generating the initial population by Panichella et al. [59]. The seeding strategies are focused on generating the initial population of the algorithm, which gives flexibility in terms of using any state-of-the-art population-based search algorithm. However, for the empirical evaluation, we integrated our seeding strategies with the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [26],

GECCO '20, July 8–12, 2020, Cancún, Mexico
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3389810

which is the most commonly applied Pareto-based search algorithm. In addition, the evaluation was performed in the context of simulation-based testing by reimplementing the algorithms proposed in our previous studies [6, 7]. We can summarize the main contributions of this paper as follows:

- We propose a total of three seeding strategies for initializing the population of Pareto-efficient algorithms for the test case selection problem.
- We integrate them in the context of the multi-objective test case selection of simulation models. To this end, we have integrated the proposed seeding strategies in the framework for test case selection proposed in our previous work [6, 7], which is an open-source framework. We have also developed the population initialization strategy proposed by Panichella et al. as an additional seeding strategy [58].
- We perform an empirical evaluation in the context of simulation-based testing using six case studies, and a total of 21 fitness function combinations integrated within the NSGA-II algorithm.
- We make all our sources available for replication by other researchers.

To assess the proposed seeding strategies we employed mutation testing to determine the fault-revealing capabilities of the solutions. The results showed that two of the proposed seeding strategies helped the NSGA-II algorithm produce solutions with higher cost-effectiveness for the six case studies when compared with a non-seeded initial population generation strategy. For out of a total of 126 problems, two of the proposed seeding strategies outperformed the baseline algorithm (i.e., NSGA-II without seeding strategy) in 117 and 121 problems with statistical significance.

The rest of the paper is structured as follows. The proposed seeding strategies are presented in Section 2. The application domain is presented in Section 3. The empirical evaluation is presented in Section 4. We position our work with other similar works in Section 5. We conclude the paper and propose future research directions in Section 6.

2 SEEDING STRATEGIES

Formalization: $TS = \{tc_1, tc_2, ..., tc_N\}$ is a Test Suite of N test cases (tc). To measure the quality and cost of a test suite, let $OF = \{of_1, of_2, ..., of_p\}$ be a set of p objective functions (of) to be satisfied when selecting test cases [59]. The test case selection algorithm aims at selecting a subset of test cases from TS, such that $TS' = \{tc_1, tc_2, ..., tc_M\}$ is a subset of TS (i.e., $TS' \subseteq TS$), that is Pareto-optimal with respect to the objective functions in OF and $M \leq N$ [59]. Like most multi-objective test selection studies [6, 7, 59], we used a binary coding representation of solutions. In this case, if the i-th digit of the binary string is 1, it means that the test case tc_i from TS is included in the solution. On the contrary, if the i-th digit is 0, it means that the test case tc_i has not been selected.

2.1 Dynamic Test Suite Size-based Random Seeding

Pareto-efficient test case selection algorithms aim at producing a set of non-dominated solutions that provide a trade-off between effectiveness (e.g., achieve certain degree of coverage) and cost (e.g., the time it takes a test suite to be executed or its size). Usually, solutions encompassing test suites with a larger number of test cases have higher probabilities of detecting faults [59], whereas solutions with a lower number of test cases are less costly. With this seeding strategy, we aim at providing solutions over the entire initial population with different number of test cases. To this end, we propose to uniformly distribute the test suite size of the solutions over the population. This would allow for several advantages. Firstly, solutions that are very effective (but costly) will be produced, along with solutions that are less effective but with lower cost. Secondly, this could allow for exploring solutions in broader directions within the search space, leading overall solutions to be fitter.

The pseudocode in Algorithm 1 shows how we implemented this seeding strategy. As an input, the Number of test cases (N) and the population size (nPop) is given. As an output, the algorithm returns the initial population, which is a two-dimensional array (each row being a solution). We build the initial population (initialPop) as follows: for each solution in the population (Line 1), the probability of a test being included in the solution is i/nPop (Line 3), i being the index of a solution in the initial population. A test is included in the j-th position of the i-th solution, if the function $rand()$ returns a number lower or equal to i/nPop. If the contrary is observed, the test case will not be included in the test suite. Subsequently, the probabilities of a test being selected increases as the solution index of the initial population increases. This way, the solutions at the beginning of the population will have a low number of test cases selected, whereas the solutions in the last positions of the initial population will have a high number of test cases. This algorithm permits a generation of an initial population of solutions that include test suites of a different variety of sizes. This strategy is coined in Section 4 as "Dynamic_seed".

Algorithm 1: Algorithm for the dynamic test suite size-based random seeding

```
Input: N //Number of test cases
nPop //Population size
Result: initialPop(nPop,N) // initial population
1 for $i \leftarrow 1$ to $nPop$ do
2     for $j \leftarrow 1$ to $N$ do
3         if $rand() \leq i/nPop$ then
4             initialPop(i,j) = 1;
5         else
6             initialPop(i,j) = 0;
7         end
8     end
9 end
```

2.2 Static Test Suite Size-based Random Seeding

Test engineers who have domain knowledge might guess the typical size required for detecting all faults. To this end, we propose a strategy where a predefined test suite size could be selected by the test engineer, in order for the initial population to include solutions that will have sizes that are close to the predefined test suite size. This would allow the search algorithm to exploit the search in an area predefined by the test engineer. Algorithm 2 shows how we implemented this seeding strategy, which is similar to the algorithm explained in the previous subsection but with the difference that the probability of a test case to be included is static. As can be seen in Line 3, the probability of a test case to be included is of the desired test suite size (desiredTestSuiteSize), which is a value (in percentages) provided as input to the algorithm. This strategy is coined in Section 4 as "Static_seed_XX%", being XX the assigned desiredTestSuiteSize.

Algorithm 2: Algorithm for the static test suite size-based random seeding

```
Input: N //Number of test cases
nPop //Population size
desiredTestSuiteSize // Percentage of the test suite size
Result: initialPop // initial population
1 for i ← 1 to nPop do
2     for j ← 1 to N do
3         if rand() ≤ desiredTestSuiteSize /100 then
4             initialPop(i,j) = 1;
5         else
6             initialPop(i,j) = 0;
7         end
8     end
9 end
```

2.3 Adaptive Random Population Generation

In previous studies, it has been shown that injecting diversity into the population improves performance of test case selection [59], as it leads the algorithm to have lower probability of being trapped in some local optimum [25]. Inspired by the Adaptive Random Testing (ART) algorithm [22], we propose the Adaptive Random Population Generation (ARPG) algorithm to generate an initial population that promotes diversity between solutions. The hypothesis behind the ART algorithm is that the higher the diversity of test cases, the higher the probability of detecting faults [22]. This algorithm has been widely used to generate test cases [20–22], and thus, we believe it can also be appropriate to generate an initial population considering diversity.

Algorithm 3 shows the pseudocode of the implemented ARPG algorithm to generate the initial population. Initially, a first solution is randomly generated and included in the first index of the initial population (Line 1). For the remaining solutions in the population, the process works as follows. A set of candidate solutions to be included in the initial population is generated (Lines 3-5). Typically, the size of this set of candidates is 10 [21, 22], so we used this number in our evaluation. Among these set of candidate solutions, the one which is farthest (i.e., the most dissimilar) from the solutions that are already included in the initial population is selected (Lines 6-17), as proposed by Chen et al. [22]. We used the Hamming Distance to measure the distance between a candidate set and a solution of the population due to its simplicity. This process is repeated until the entire initial population is generated.

A major issue this seeding strategy could have involves its running time, especially when the number of population tends to increase. Its big O complexity for this strategy would be $O[(nPop - 1) * (nCand + (nCand(nPop^2 - nPop)/2))]$, which makes the running time exponential to the population size. However, usually the population size in the context of test case selection is around 100 [6, 7, 68], which makes this strategy applicable in practice.

Algorithm 3: Algorithm for the ARPG seeding

```
Input: N //Number of test cases
nPop //Population size
nCandidate //Number of candidate solutions
Result: initialPop // initial population
1  initialPop(1,:) = randSol(N);
2  for i ← 2 to nPop do
3      for j ← 1 to nCandidate do
4          candidateSets(j,:) = randSol(N);
5      end
6      for j ← 1 to nCandidate do
7          minDist = 1;
8          for k ← 1 to i-1 do
9              distance = measureDistance(initialPop(j,:),
                candidateSet(k,:))
10             if distance < minDist then
11                 minDist = distance;
12             end
13         end
14         distArray(j) = minDist ;
15     end
16     distIndx = max(distArray);
17     initialPop(i,:) = candidateSets(distIndx,:);
18 end
```

2.4 Orthogonal Population Generation

To the best of our knowledge, the only work in the context of search-based test case selection where an initial population is not generated in the standard way is that of Panichela et al. [59]. To this end, we reimplemented their initial population generation approach to compare it with the seeding strategies proposed in this paper. This strategy is based on the orthogonal arrays methodology [55] that uses the Hadamard matrices to build orthogonal arrays. This strategy is coined in Section 4 as "Orthogonal_seed".

3 APPLICATION DOMAIN: BLACK-BOX TEST CASE SELECTION OF SIMULATION MODELS

Simulation models are typically used by engineers to model and simulate complex systems, such as Cyber-Physical Systems (CPS) [16, 51]. This technology is largely employed as it supports engineers in several activities, including automated test generation and early testing of CPSs without requiring an initial prototype [53]. However, simulating some of these systems is commonly time-consuming, where, a single simulation may take hours to complete in some systems [53]. Furthermore, testing a CPS requires several test levels, even with the same software version being tested. Usually, testing starts at the Model-in-the-Loop (MiL) level, following with the Software-in-the-Loop (SiL) level and lasting with the Hardware-in-the-Loop (HiL) level [7, 12], this being a real-time simulation.

Testing these systems by using simulation-based testing also poses several other difficulties, such as the use of co-simulation, human test oracle, or several fidelity levels of the models [7, 16]. Consequently, test optimization is paramount, and recent approaches have proposed black-box testing methodologies, including test case generation [50, 51, 53] and test case selection [9, 12]. With black-box testing we refer to focusing only on the inputs and outputs of the system, without requiring either external data related to historical failures or white-box coverage. This technique has been found to be appropriate to solve the test case selection problem, showing improvement over traditionally employed white-box coverage techniques (i.e., Decision Coverage, Condition Coverage, and Modified Condition/Decision coverage) [7]. Furthermore, this technique is largely multi-objective, as having larger test suites increases the probability of detecting faults, but it also increases the test execution time. The algorithm, thus, needs to find a trade-off between cost and effectiveness.

In a previous work, we proposed a set of test adequacy criteria for multi-objective test case selection adapted to the context of simulation-based testing of CPSs [7]. These adequacy criteria could be categorized into two main parts: (1) adequacy criteria that measured a quantitative degree of certain anti-patterns defined in previous works [48], and (2) a measure of distances between test cases based on the Euclidean distance adapted to the context of simulation-based testing. The hypothesis behind the former is that the higher the quantitative degree of an anti-pattern, the higher the probability of finding faults. As for the latter's, the hypothesis is that the more dissimilar the selected test cases are, the higher the probability of finding faults. This hypothesis has been widely investigated, showing positive results [29, 35, 36].

Besides multi-objective test case selection for simulation-based testing of CPSs being an important challenge, we integrated our seeding strategies with our test case selection framework [7] for different reasons. Firstly, the availability of an open source benchmark on Github along with experimental material (including case studies, test cases, mutants, execution scripts, etc.). Secondly, we provided more than one derived fitness function, which allows us to perform a more comprehensive empirical evaluation, not only evaluating our seeding strategies with specific case studies, but also with different fitness functions. The derived fitness functions are the same as the ones proposed in our previous work [7], and are summarized in Table 1. A maximum of three objective functions are selected and integrated within the NSGA-II algorithm [26]. The maximum number of objective functions was three because NSGA-II suffers from scalability issues when solving optimization problems with more than three objectives [58]. Additionally, many-objective algorithms with a larger number of objective functions have shown worse results on this benchmark [7], with the NSGA-II being the best algorithm.

Table 1: Fitness function combinations integrated on the NSGA-II

Fitness Configuration ID	Objective 1	Objective 2	Objective 3
c1	Discontinuity		
c2	Growth to infinity		
c3	Instability	None	
c4	Input similarity		
c5	Output similarity		
c6	MinMax		
c7	Discontinuity	Growth to infinity	
c8	Discontinuity	Instability	
c9	Discontinuity	Input similarity	
c10	Discontinuity	Output similarity	Test execution time
c11	Discontinuity	MinMax	
c12	Growth to infinity	Instability	
c13	Growth to infinity	Input similarity	
c14	Growth to infinity	Output similarity	
c15	Growth to infinity	MinMax	
c16	Instability	Input similarity	
c17	Instability	Output similarity	
c18	Instability	MinMax	
c19	Input similarity	Output similarity	
c20	Input similarity	MinMax	
c21	Output similarity	MinMax	

4 EMPIRICAL EVALUATION

To evaluate the proposed seeding strategies, we defined the following two Research Questions (RQ):

RQ1: How do the proposed seeding strategies perform when compared with non-seeded multi-objective search algorithms? With this RQ we aimed at answering whether the seeding strategies do actually perform better than search algorithms without having the initial population seeded. To this end, we compared a total of 21 combinations of fitness functions within the NSGA-II algorithm over six case studies (i.e., a total of 126 combinations). To answer this RQ, for each of the combinations, four seeding strategies were compared with a non-seeded approach. The non-seeded NSGA-II generates initial populations purely randomly.

RQ2: Among the proposed seeding strategies, which one fares best? The intention behind the second RQ was to see whether there is a best seeding strategy for test case selection to recommend to practitioners. To this end, we compared each of the seeding strategies with one another for each of the 126 combinations.

It should be noted that in the previous studies [6, 7] there was a sanity check done by comparing the NSGA-II algorithm with Random Search (RS), demonstrating significant improvement. Thus, we did not compare the seeding strategies along with the NSGA-II algorithm with respect to RS.

4.1 Experimental setup

For the experimental setup, we replicated the experiment proposed in our previous work [7] by adapting it to answer the RQs presented in this empirical evaluation.

4.1.1 Case Studies. Six case studies involving Simulink models of different sizes, complexities and domains were employed, which provides a wide heterogeneity to the experiment. Table 2 provides a summary of the selected case studies in terms of (1) number of Simulink blocks, (2) number of inputs, (3) number of outputs, (4) number of test cases used in the study, (5) the initial number of mutants and (6) the final number of mutants. It is noteworthy that one of these case studies, i.e., The Electro-Mechanical Braking (EMB) system, was an industrial case study developed by Bosch engineers [65]. This case study was previously used in other evaluations [48]. The remaining case studies involve (1) CW, a model of four car windows with its control software, (2) CC, the software in charge of automatically controlling the speed of the car, (3) Tiny, a toy Simulink model, (4) AC Engine, an Alternating Current Engine with its control software involving safety functionalities, and (5) Two Tanks, a case study involving a Simulink model of Two Tanks. All these case studies have been previously used to evaluate Simulink testing methods [6–8, 34, 48, 50, 51, 54, 56]. We used the same test cases provided by in our benchmark [7] in order for the results to be compared with their approach.

Table 2: Key characteristics of the selected case studies

Case Study	# of Blocks	# of Inputs	# of Outputs	# of Test Cases	Initial set of mutants	Final set of mutants
CW	235	15	4	133	250	96
EMB	315	1	1	150	40	18
CC	31	5	2	150	60	20
Tiny	15	3	1	150	20	9
AC Engine	257	4	1	120	20	12
Two Tanks	498	11	7	150	34	6

4.1.2 Evaluation metrics. Mutation testing was employed to assess the fault revealing capability of a solution, as it has shown to be a good substitute of real faults [37]. To this end, a set of mutants were generated for each of the case studies, and the relation between test cases and killed mutants was obtained. With this information, we removed (1) duplicated mutants (i.e., mutants equivalent to one another but not to the original program) as recommended by Papadakis et al., [60], (2) mutants that were killed by all test cases (as we considered them to be too weak mutants) and (3) mutants that were not killed by any test case (to avoid the inclusion of equivalent mutants). We used the mutants available in the framework [7], which employed the mutation operators proposed by Hann et al., [34] for Simulink models. Information related to the number of mutants in the initial set and the final set are available in Table 2.

As in the case of [6, 7, 59], we used the revisited Hypervolume (HV) metric to measure the effectiveness of our approach. In this case, for a set of solutions from the Pareto-frontier, we used (1) the cost and (2) the percentage of faults revealed by each solution (i.e., mutation score) as external utility functions. Thus, to compute this revisited HV function, for the Pareto-frontier returned by the search algorithm, each solution was individually assessed by obtaining their mutation score and the cost (i.e., the test execution time). By using this information, a new Pareto-frontier was obtained aiming at maximizing the mutation score and minimizing the test execution time. The derived Pareto-frontier was later employed to obtain the HV measure, by having as a reference point 0% for the mutation score and the time to execute the entire test suite for the cost. Notice that the higher the HV measure, the better the performance of the algorithm.

4.1.3 Statistical tests. As suggested by Arcuri and Briand [3], each algorithm along with the seeding strategy was run 50 times to account for the random variations. When we obtained the experimental data, the Shapiro-Wilk test was applied to assess how the data was distributed. As the data was not normally distributed, we employed the Mann-Whitney U-test to obtain the statistical significance between two different combinations. If the p-value was lower than 0.05, we considered that there was statistical significance. To assess the difference between the seeding strategies, we employed the Vargha and Delaney $\hat{A}_{12}$ value.

4.1.4 Algorithms setup. In order for the approach to be compared with the approach proposed in our previous work [6, 7], we configured the NSGA-II as in their evaluation. The population size was set to 100 and the total number of fitness evaluations was 25,000. The crossover rate was set to 0.8, and a standard single point crossover operator was employed. The mutation of a variable was done with the standard probability 1/N, N being the number of test cases in the initial test suite. We considered these parameter values based on other studies related to multi-objective test case selection [6, 7, 68] as well as guidelines [3]. As in our previous work [6, 7], for the selection operator, we used the binary tournament selection operator [14, 26]. As for the seeding strategies, the number of candidate sets in the ARPG strategy was set to 10 based on the original paper [22]. In the case of the static seeds, we experimented with two instance configurations, where we set one of the algorithm configuration for a desired test suite size of 30% (coined as Static_seed_30%), and the other one of 70% (coined as Static_seed_70%). These configurations were selected based on an initial preliminary experiment. Additionally, we wanted to analyze configurations with larger and shorter test suites than 50% (as the non-seeded generated initial population typically provides solutions that include the 50% of test cases).

4.2 Analysis of the Results and Discussion

To keep the paper at a reasonable size, Table 3 reports the summary of the results after applying the statistical tests. Each selected seeding strategy was compared with one another for each case study and each fitness combination from Table 1. For each case study, we provide inside the column $A/B/=$ the number of times each seeding strategy outperformed another one. The first number (i.e., the one in the left) represents the number of fitness combinations where the seeding strategy in column "Strategy A" outperformed the seeding strategy in column "Strategy B" with statistical significance (i.e., $\hat{A}_{12} > 0.5$ and p-value < 0.05). The second number (i.e., the one in the middle) represents the number of fitness combinations where the seeding strategy in column "Strategy B" outperformed the seeding strategy in column "Strategy A" with statistical significance (i.e., $\hat{A}_{12} < 0.5$ and p-value < 0.05). The third number (i.e., the one in the right) represents the number of fitness combinations where there was no statistical significance between both strategies. The column "Total" represents a summary of all the results obtained for each of the case studies. In addition, Figure 1 shows the distribution of the obtained HV results. As we could not report the results for the 126 fitness functions combinations, we report for each of the selected case studies the obtained results for the best fitness functions combination (obtained from [7]).[1]

The first RQ aimed at answering whether the proposed seeding strategies outperformed the NSGA-II algorithm without an initial seeding strategy. The summary of the results show that overall, there were two strategies that outperformed the NSGA-II with a non-seeded population generation whereas others performed similarly to the non-seeded algorithm. For five out of six case studies, the Dynamic Test Suite Size-based Random Seeding strategy (coined as *Dynamic*) and the Static Test Suite Size-based Random Seeding

[1]The remaining boxplots, along with the scripts used for the statistical analysis as well as the results are included in the replication package.

Table 3: Summary of the performed statistical tests

			CW	EMB	CC	Tiny	AC Engine	TwoTanks	Total
	Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
RQ1	Non-Seeded	Dynamic	0/21/0	0/21/0	0/21/0	0/21/0	0/12/9	0/21/0	0/117/9
		Static30	0/21/0	0/21/0	0/21/0	0/21/0	0/16/5	0/21/0	0/121/5
		Static70	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	126/0/0
		ARPG	0/3/18	9/0/12	3/0/18	0/0/21	0/1/20	3/1/17	15/5/106
		Orthogonal	2/0/19	1/0/20	0/0/21	1/0/20	0/1/20	1/1/19	5/2/119
RQ2	Dynamic	Static30	8/3/10	3/1/17	6/9/6	2/5/14	0/5/16	11/2/8	30/25/71
		Static70	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	126/0/0
		ARPG	21/0/0	21/0/0	21/0/0	21/0/0	11/0/10	21/0/0	116/0/10
		Orthogonal	21/0/0	21/0/0	21/0/0	21/0/0	12/0/9	21/0/0	117/0/9
	Static30	Static70	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	21/0/0	126/0/0
		ARPG	21/0/0	21/0/0	21/0/0	21/0/0	13/0/8	21/0/0	118/0/8
		Orthogonal	21/0/0	21/0/0	21/0/0	21/0/0	13/0/8	21/0/0	118/0/8
	Static70	ARPG	0/21/0	0/21/0	0/21/0	0/21/0	0/21/0	0/21/0	0/126/0
		Orthogonal	0/21/0	0/21/0	0/21/0	0/21/0	0/21/0	0/21/0	0/126/0
	ARPG	Orthogonal	4/0/14	0/4/17	0/1/20	1/0/20	0/3/18	0/3/18	5/11/110

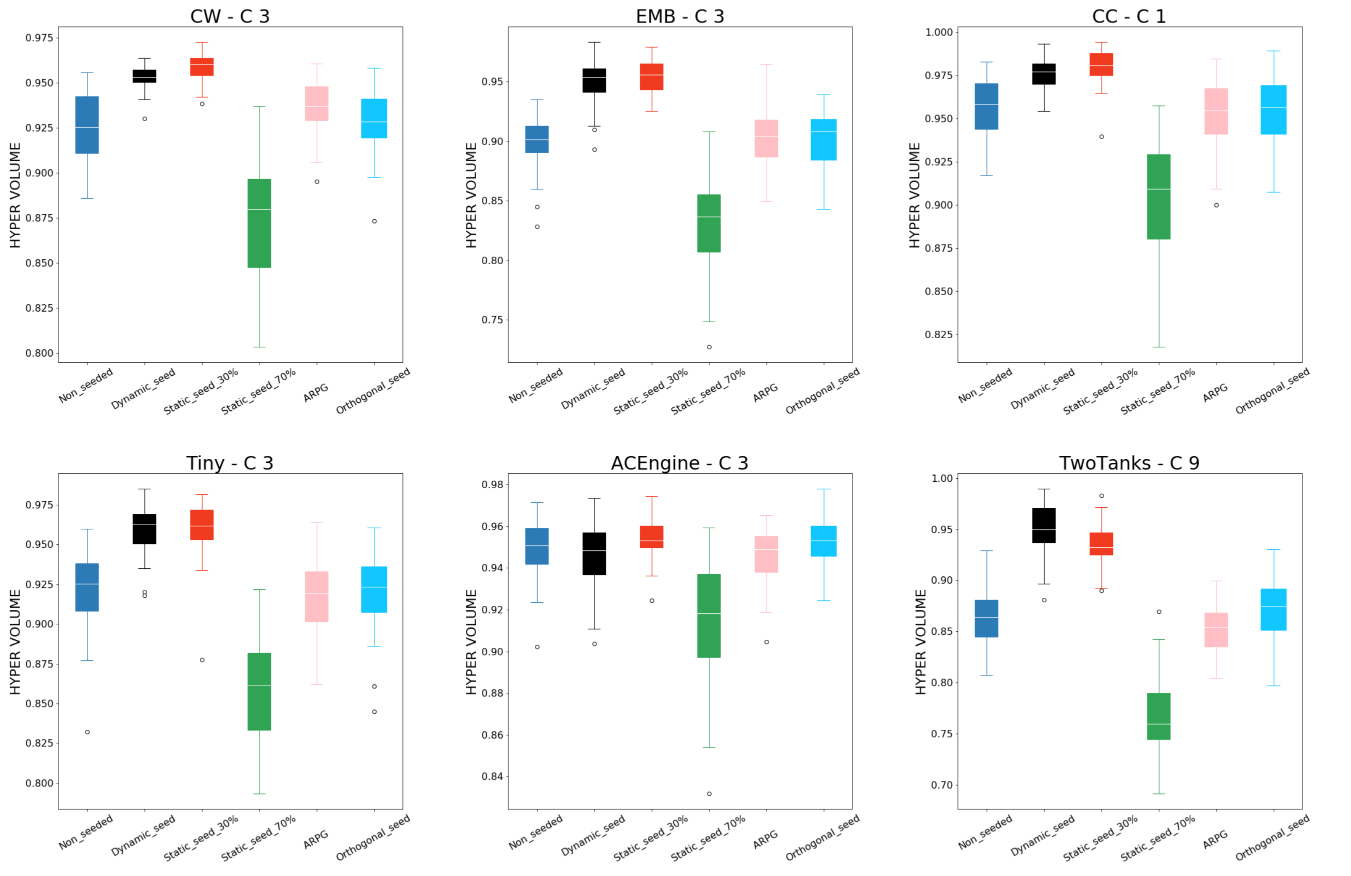

Figure 1: Distribution of the obtained HV metrics for each seeding strategy for the best fitness combination of each case study

strategy, configured to include around 30% of the test cases in the test suite (coined as Static_seed_30%), significantly outperformed the non-seeded NSGA-II algorithm for all the fitness combinations. Additionally, for the AC Engine case study, although in some of the fitness combinations the seeding strategies performed similarly (i.e., in 9 out of 21 fitness function combinations for the dynamic seeding and 5 out of 21 fitness combinations for the static seeding), in the remaining strategies, the seeding strategies outperformed the NSGA-II that did not have the initial population seeded with statistical significance. This means that the non-seeded algorithm did not outperform these two seeding strategies for any of the 126 cases studied in this paper. Thus, at least two of the proposed techniques performed positively, giving solutions with higher cost-effectiveness in the context of multi-objective test case selection for simulation-based testing.

In the case of the ARPG seeding strategy and the orthogonal population generation strategy, both strategies performed similarly to the non-seeded strategy. In fact, for 106 out of 126 cases, there was no statistical significance between the ARPG seeding strategy and the algorithm that did not include any seeding strategy. Similarly, for the orthogonal population generation strategy, which was proposed by Panichella et al. [59], there was no statistical significance for 119 out of 126 cases. Thus, overall, we did not find striking differences between these two seeding strategies and the NSGA-II algorithm without seed. In previous studies [25, 59], it has been demonstrated that injecting diversity in the population shows a positive effect on multi-objective test case selection. The injected diversity in the studies proposed in other studies [25, 59] were through the entire search process. Our study shows, however, that injecting diversity solely for generating the initial population is not enough for test case selection, meaning that techniques, such as the Orthogonal population seeding strategy would require further algorithmic treatment, as proposed by Panichella et al. [59].

Unlike for the aforementioned seeding strategies, the static seeding strategies configured to have around 70% of the test suite size was found to be much worse than the non-seeded strategy. In fact, the non-seeded algorithm outperformed this strategy with statistical significance for all the 126 cases (i.e., all fitness function combinations for the six case studies). A potential reason for this could be that the population produced with such seeding strategy converges soon, getting stuck on local regions. Furthermore, the solutions that contained around the 70% of the test cases from the entire test suite could be too costly for the context of simulation models. The first RQ can be answered as follows:

Two of the selected techniques (i.e., the Dynamic_seed and the Static_seed_30%) outperformed with statistical significance the NSGA-II algorithm without the initial population seeded, outperforming it in 93 and 96% of the problems with statistical significance. The strategies ARPG and the Orthogonal performed similarly to the non-seeded strategy (there were not statistical significance for 84 and 94% of the problems). Conversely, the Static_seed_70% strategy performed worse than the NSGA-II algorithm without the initial population seeded for the 126 problems.

The second RQ aimed at answering the difference existing between the proposed seeding strategies. From the previous RQ, it was easy to determine that the Dynamic Test Suite Size-based Random seeding strategy (coined as “Dynamic_seed”) and the Static Test Suite Size-based Random Seeding (coined as “Static_seed_30%”) stood out over the rest of the strategies. When comparing these two approaches, as can be seen in Table 3, the Dynamic_seed outperformed the Static_seed_30% strategy with statistical significance in 30 out of 126 cases. Conversely, the Static_seed_30% outperformed the Dynamic_seed in 25 out of 126 cases. Subsequently, there were 71 cases out of 126 where no statistical significance was appreciated according to the performed statistical tests.

When having a closer view, we saw that the Dynamic_seed strategy was slightly better than the Static_seed_30% in the CW, EMB and TwoTanks case studies, whereas the Static_seed_30% slightly outperformed the Dynamic_seed strategy in the CC, Tiny and AC Engine case studies. We noticed that the case studies where the Dynamic_seed was the best seeding strategy were more complex than the ones from the Static_seed_30%.[2] Although further empirical evidence would be required to generalize these findings, our intuition is that the Dynamic_seed strategy could be more positive in larger simulation models as it generates initial population with test suites of all sizes, and thus, it explores wider areas in the search space. In large case studies, it is noteworthy that finding bugs could be more time consuming as they might require more test cases to achieve higher test coverage degrees. Furthermore, a drawback of the Static_seed_30% is that it is configurable, which means that its good performance could be system-specific (i.e., 30% could be good in some systems, whereas it could be bad in other systems). We could answer the second RQ as follows:

Both the Dynamic_seed and the Static_seed_30% performed similarly. The Dynamic_seed performed slightly better in larger case studies whereas the Static_seed_30% performed better in simpler ones. The main drawback of Static_seed_30% is that as it is configurable it could work properly in some systems but not so in others.

4.3 Threats to validity

Internal validity: An internal validity threat in our study could be related to the generated mutants. In Simulink models, employing mutation testing is highly expensive due to the physical layer encompassing complex mathematical equations. Subsequently, a large set of mutants was not feasible to generate. To reduce this threat, we employed the same mutants as generated in our previous studies [6, 7]. Furthermore, notice that the amount of mutants used in this study was similar to those used in other empirical evaluations of testing methods for Simulink models [10–12, 34, 39, 42–44, 49–51]. Additionally, we removed duplicated mutants as recommended by Papadakis et al. [60] to further mitigate this threat. Another internal validity threat in this study is referred to the parameters of the algorithms (e.g., population size), which were not changed. To mitigate this threat we configured the algorithm considering related guidelines and works that included Pareto-based search algorithms for test case selection [68]. Also, some of our seeding strategies are configurable. The number of candidate sets in the ARPG strategy was set to 10. Another value could have changed our results, but we selected this number based on the original ART paper [22]. Additionally, for the static seeding strategies, we selected two instances of this algorithm (i.e., the test suite size parameter at 30 and 70%) based on initial preliminary algorithm runs.

External validity: As in any search-based software engineering evaluation, an external validity threat is related to the generalization of results. We tried to mitigate this threat by using six case studies involving Simulink models of different sizes and characteristics. As for the sizes, according to a study of 391 public Simulink models, more than half of the analyzed models had less than 100 blocks, and around 75% of models had less than 300 blocks [24]. Four of the used case studies had from 235 to 498 blocks, which means that most of our case studies are larger than most of the public subject models. In addition, for each of the case studies, we used a total

[2]Notice that although CW had fewer blocks than the AC Engine, the CW has four complex statecharts that are counted as one block each

of 21 different combinations of fitness functions to compare each seeding strategy with one-another.

Another external validity threat in our study is that we only tested the strategies within the NSGA-II algorithm. This algorithm, however, has been largely employed for multi-objective test case selection [7, 68], and it has shown better performance than two many-objective algorithms in the framework used in this paper [7].

Conclusion validity: A conclusion validity threat in our study might be related to the non deterministic nature of evolutionary algorithms. This threat was mitigated by running each algorithm 50 times to account for the random variations, as recommended in guidelines [3]. Additionally, we carefully analyzed the results by applying appropriate statistical tests.

Construct validity: In randomized algorithms, construct validity threats arise when the measures used are not comparable across the algorithms. We mitigated this threat by using the same stopping criterion for all the algorithms (i.e., we set the total number of fitness evaluations at 25,000).

5 RELATED WORK

Test case selection has been widely studied in the current literature. Yoo et al., identified and analysed the positive and negative aspects of 12 different approaches based on an extensive analysis of the state-of-the-art [70]. Engströem et al. identified 28 techniques for regression test selection [27]. Multi-objective algorithms to solve the test case selection problem was first proposed by Yoo et al. [68]. They evaluated the use of the NSGA-II algorithm integrated with objective functions that included (1) coverage, (2) historical information related to faults and (3) testing costs. The same authors later extended this study, where they proposed an hybrid approach [69]. In the last few years, several new approaches have been proposed to adapt the test case selection problem to different emergent areas, including defence software [33, 57], or compute-intensive CPSs [6, 7]. Most multi-objective test selection approaches aim at proposing effective objective functions and study whether they act as a reasonable surrogate for fault detection capabilities [6, 7, 36, 38, 68]. Unlike all these studies, our approach aims at comparing how different strategies for seeding the initial population perform in the context of multi-objective test case selection algorithms.

Other approaches compare the performance of evolutionary algorithms for selecting test cases in different context, such as time-constrained scenarios [62], or product lines [10, 66, 67]. However, all these approaches consider non-seeded initial population generation approaches. Other approaches have proposed algorithms where the initial population is particularly seeded following a specific strategy. For instance, Panichella et al., proposed including diversity in genetic algorithms to improve optimality of multi-objective test case selection. To this end, they proposed mechanisms of orthogonal design and orthogonal evolution with the aim of increasing diversity during the search process [59]. A similar approach was proposed by De Lucia et al., which proposed enhancing the NSGA-II algorithm by increasing population diversity in the obtained Pareto frontiers during the search [25]. The difference between these approaches and the one we propose in this paper is that in their case, diversity is included during the entire search process, whereas our solution is intended solely to seed the initial population. As previously commented, this provides high flexibility in practice, as our seeding strategies can be applied in any population-based algorithms, including the ones proposed in test case selection specific multi-objective search algorithms [25, 59].

Seeding strategies have been proposed for solving other search-based software engineering problems. From the testing perspective, several approaches have been proposed in the past. Fraser and Arcuri proposed three seeding strategies (e.g., seeding of constants extracted from source code) for search-based test generation of unit testing, showing a positive impact when compared to non-seeded algorithms [31]. This study was further extended, confirming the positive impact of the seeding strategies in further subject programs [63]. Seeding was also applied on SAPIENZ [46], a test generation tool for testing Android programs. Specifically, SAPIENZ statically analyses some files to extract strings to seed the multi-objective search algorithm in charge of generating test cases. Lopez-Herrejon et al., proposed a total of three seeding strategies for pairwise software product lines testing [45], showing a positive impact both, in the final solutions returned by the search algorithms as well as the time it takes the algorithm to converge. Besides search-based testing, seeding has also been successfully applied to solve other software engineering problems, including service composition problems [18, 19] and software improvement [5]. In contrast to all these studies, the proposed seeding strategies proposed in this paper are designed for the test case selection problem. To the best of our knowledge, there are no previous papers that have proposed and studied the impact of seeding strategies for multi-objective test selection.

6 CONCLUSION AND FUTURE WORK

In this paper we propose three different seeding strategies for initializing the population of multi-objective test case selection algorithms. We integrate them within the context of simulation-based testing and the NSGA-II. We empirically evaluated four instances of the proposed strategies and an additional strategy proposed in another study [59], by employing six case studies and 21 fitness function combinations in each case study. Two instances of the proposed seeding strategies (i.e., *Dynamic_seed* and *Static_seed_30%*) outperformed the NSGA-II algorithm without seeding strategies for 92.8 and 96% of the cases with statistical significance. In the future we would like to extend this work from several perspectives, such as, including seeding strategies in genetic operators (e.g., mutation and crossover operators).

Replication package: For the sake of replicability, we make all our code, experimental scripts and results available at http://doi.org/10.5281/zenodo.3739219

ACKNOWLEDGEMENTS

This work is part of the TESEO project (KK2019/00026), funded by the Department of Education, Universities and Research of the Basque Government. This work was carried out by the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Government.

REFERENCES

[1] Shaukat Ali, Lionel C Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. 2010. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering* 36, 6 (2010), 742–762.

[2] M Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Jānis Benefelds. 2017. An industrial evaluation of unit test generation: Finding real faults in a financial application. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*. IEEE Press, 263–272.

[3] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 1–10.

[4] Andrea Arcuri and Gordon Fraser. 2014. On the effectiveness of whole test suite generation. In *International Symposium on Search Based Software Engineering*. Springer, 1–15.

[5] Andrea Arcuri, David Robert White, John Clark, and Xin Yao. 2008. Multi-objective improvement of software using co-evolution and smart seeding. In *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 61–70.

[6] Aitor Arrieta, Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2018. Multi-objective Black-box Test Case Selection for Cost-effectively Testing Simulation Models. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. ACM, New York, NY, USA, 1411–1418. https://doi.org/10.1145/3205455.3205490

[7] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. 2019. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Information & Software Technology* 114 (2019), 137–154. https://doi.org/10.1016/j.infsof.2019.06.009

[8] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2017. Search-Based Test Case Generation for Cyber-Physical Systems. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*. 688–697.

[9] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2018. Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics* 14, 3 (2018), 1055–1066.

[10] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2016. Search-based Test Case Selection of Cyber-physical System Product Lines for Simulation-based Validation. In *Proceedings of the 20th International Systems and Software Product Line Conference*. 297–306.

[11] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2016. Test Case Prioritization of Configurable Cyber-Physical Systems with Weight-Based Search Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 1053–1060. https://doi.org/10.1145/2908812.2908871

[12] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2019. Search-Based test case prioritization for simulation-Based testing of cyber-Physical system product lines. *Journal of Systems and Software* 149 (2019), 1 – 34. https://doi.org/10.1016/j.jss.2018.09.055

[13] Wesley Klewerton Guez Assunção, Thelma Elita Colanzi, Silvia Regina Vergilio, and Aurora Pozo. 2014. A multi-objective optimization approach for the integration and test order problem. *Information Sciences* 267 (2014), 119–139.

[14] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. 12.

[15] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615 – 636.

[16] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. 2016. Testing the Untestable: Model Testing of Complex Software-intensive Systems. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*. ACM, 789–792. https://doi.org/10.1145/2889160.2889212

[17] Jose Campos, Yan Ge, Gordon Fraser, Marcello Eler, and Andrea Arcuri. 2017. An Empirical Evaluation of Evolutionary Algorithms for Test Suite Generation. In *Symposium on Search-Based Software Engineering*.

[18] Tao Chen, Miqing Li, and Xin Yao. 2018. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 1419–1426.

[19] Tao Chen, Miqing Li, and Xin Yao. 2019. Standing on the Shoulders of Giants: Seeding Search-based Multi-Objective Optimization with Prior Knowledge for Software Service Composition. *Information and Software Technology* (2019).

[20] Tsong Yueh Chen, F-C Kuo, Robert G Merkel, and Sebastian P Ng. 2004. Mirror adaptive random testing. *Information and Software Technology* 46, 15 (2004), 1001–1010.

[21] Tsong Yueh Chen, Fei-Ching Kuo, Robert G Merkel, and TH Tse. 2010. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software* 83, 1 (2010), 60–66.

[22] Tsong Yueh Chen, Hing Leung, and IK Mak. 2004. Adaptive random testing. In *Annual Asian Computing Science Conference*. Springer, 320–329.

[23] Ankur Choudhary, Arun Prakash Agrawal, and Arvinder Kaur. 2018. An effective approach for regression test case selection using pareto based multi-objective harmony search. In *Proceedings of the 11th International Workshop on Search-Based Software Testing*. ACM, 13–20.

[24] Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T Johnson, and Christoph Csallner. 2018. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 981–992.

[25] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Annibale Panichella. 2012. On the role of diversity measures for multi-objective test case selection. In *Proceedings of the 7th International Workshop on Automation of Software Test*. IEEE Press, 145–151.

[26] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[27] Emelie Engström, Per Runeson, and Mats Skoglund. 2010. A systematic review on regression test selection techniques. *Information and Software Technology* 52, 1 (2010), 14–30.

[28] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. 2015. Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. ACM, New York, NY, USA, 234–245.

[29] Robert Feldt, Simon M. Poulding, David Clark, and Shin Yoo. 2016. Test Set Diameter: Quantifying the Diversity of Sets of Test Cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016*. 223–233. https://doi.org/10.1109/ICST.2016.33

[30] Javier Ferrer, Peter M. Kruse, Francisco Chicano, and Enrique Alba. 2012. Evolutionary Algorithm for Prioritized Pairwise Test Data Generation. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO '12)*. ACM, New York, NY, USA, 1213–1220. https://doi.org/10.1145/2330163.2330331

[31] Gordon Fraser and Andrea Arcuri. 2012. The seed is strong: Seeding strategies in search-based software testing. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 121–130.

[32] Gordon Fraser and Andrea Arcuri. 2013. Whole test suite generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291.

[33] Vahid Garousi, Ramazan Özkan, and Aysu Betin-Can. 2018. Multi-objective regression test selection in practice: An empirical study in the defense software industry. *Information and Software Technology* 103 (2018), 40–54.

[34] Le Thi My Hanh, Nguyen Thanh Binh, and Khuat Thanh Tung. 2016. A Novel Fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis. *Journal of Systems and Software* 120, C (2016), 17–30.

[35] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology* 22, 1 (2013), 6:1–6:42.

[36] Hadi Hemmati and Lionel Briand. 2010. An industrial investigation of similarity measures for model-based test case selection. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 141–150.

[37] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 654–665.

[38] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. 2017. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 1311–1318.

[39] Khuat Thanh Le Thi My Hanh and Nguyen Thanh Binh Tung. 2014. Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing. *International Journal of Computer and Information Technology* 3, 04 (2014), 763–771.

[40] Xuelin Li, W. Eric Wong, Ruizhi Gao, Linghuan Hu, and Shigeru Hosono. 2017. Genetic Algorithm-based Test Generation for Software Product Line with the Integration of Fault Localization Techniques. *Empirical Software Engineering* (2017), 1–51. https://doi.org/10.1007/s10664-016-9494-9

[41] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software Engineering* 33, 4 (2007), 225–237.

[42] Bing Liu, Lucia, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2016. Simulink fault localization: an iterative statistical debugging approach. *Software Testing, Verification and Reliability* 26, 6 (2016), 431–459.

[43] Bing Liu, Lucia Lucia, Shiva Nejati, and Lionel Briand. 2017. Improving Fault Localization for Simulink Models using Search-Based Testing and Prediction Models. In *24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2017)*.

[44] Bing Liu, Shiva Nejati, Lionel C Briand, et al. 2018. Effective fault localization of automotive Simulink models: achieving the trade-off between test oracle effort and fault localization accuracy. *Empirical Software Engineering* (2018), 1–47.

[45] Roberto E Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Alexander Egyed, and Enrique Alba. 2014. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 387–396.

[46] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 94–105.

[47] Reza Matinnejad, Shiva Nejati, Lionel Briand, Thomas Bruckmann, and Claude Poull. 2015. Search-based automated testing of continuous controllers: Framework, tool support, and case studies. *Information and Software Technology* 57 (2015), 705 – 722.

[48] Reza Matinnejad, Shiva Nejati, and Lionel C. Briand. 2017. Automated Testing of Hybrid Simulink/Stateflow Controllers: Industrial Case Studies. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 938–943. https://doi.org/10.1145/3106237.3117770

[49] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2015. Effective test suites for mixed discrete-continuous stateflow controllers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 84–95.

[50] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2016. Automated Test Suite Generation for Time-continuous Simulink Models. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 595–606.

[51] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2019. Test Generation and Test Prioritization for Simulink Models with Dynamic Behavior. *IEEE Trans. Software Eng.* 45, 9 (2019), 919–944. https://doi.org/10.1109/TSE.2018.2811489

[52] Phil McMinn. 2004. Search-based software test data generation: a survey. *Software testing, Verification and reliability* 14, 2 (2004), 105–156.

[53] Claudio Menghi, Shiva Nejati, Lionel C Briand, and Yago Isasi Parache. 2020. Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification. In *International Conference on Software Engineering (ICSE)*.

[54] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C. Briand. 2019. Generating automated and online test oracles for Simulink models with continuous and uncertain behaviors. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. 27–38. https://doi.org/10.1145/3338906.3338920

[55] Douglas C Montgomery. 2017. *Design and analysis of experiments*. John wiley & sons.

[56] Shiva Nejati, Khouloud Gaaloul, Claudio Menghi, Lionel C. Briand, Stephen Foster, and David Wolfe. 2019. Evaluating model testing and model checking for finding requirements violations in Simulink models. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. 1015–1025. https://doi.org/10.1145/3338906.3340444

[57] Ramazan Özkan, Vahid Garousi, and Aysu Betin-Can. 2017. Multi-objective regression test selection in practice: an empirical study in the defense software industry. In *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.

[58] Annibale Panichella, Fitsum Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* (2017).

[59] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, and Andrea De Lucia. 2015. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering* 41, 4 (2015), 358–383.

[60] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 936–946.

[61] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2012. Pairwise testing for software product lines: Comparison of two approaches. *Software Quality Journal* 20, 3-4 (2012), 605–643.

[62] Dipesh Pradhan, Shuai Wang, Shaukat Ali, and Tao Yue. 2016. Search-Based Cost-Effective Test Case Selection Within a Time Budget: An Empirical Study. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 1085–1092. https://doi.org/10.1145/2908812.2908850

[63] José Miguel Rojas, Gordon Fraser, and Andrea Arcuri. 2016. Seeding strategies in search-based unit test generation. *Software Testing, Verification and Reliability* 26, 5 (2016), 366–401.

[64] Alireza Salahirad, Hussein Almulla, and Gregory Gay. 2019. Choosing the fitness function for the job: Automated generation of test suites that detect real faults. *Software Testing, Verification and Reliability* 29, 4-5 (2019), e1701.

[65] T. Strathmann and J. Oehlerking. 2015. Verifying Properties of an Electro-Mechanical Braking System. In *In 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*. 49–56.

[66] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. 2013. Minimizing test suites in software product lines using weight-based genetic algorithms, In Proceedings of the 2013 Genetic and Evolutionary Computation Conference. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 1493 – 1500.

[67] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. 2015. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software* 103, 0 (2015), 370 – 391.

[68] Shin Yoo and Mark Harman. 2007. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 140–150.

[69] Shin Yoo and Mark Harman. 2010. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software* 83, 4 (2010), 689–701.

[70] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Softw. Test. Verif. Reliab.* 22, 2 (March 2012), 67–120.

[71] Shin Yoo, Mark Harman, and Shmuel Ur. 2011. Highly scalable multi objective test suite minimisation using graphics cards. In *International Symposium on Search Based Software Engineering*. Springer, 219–236.