Some Seeds are Strong: Seeding Strategies for Search-based Test Case Selection

AITOR ARRIETA, Mondragon University
PABLO VALLE, Mondragon University
JOSEBA A. AGIRRE, Mondragon University
GOIURIA SAGARDUI, Mondragon University

The time it takes software systems to be tested is usually long. Search-based test selection has been a widely investigated technique to optimize the testing process. In this paper, we propose a set of seeding strategies for the test case selection problem that generate the initial population of pareto-based multi-objective algorithms, with the goals of (1) helping to find an overall better set of solutions and (2) enhancing the convergence of the algorithms. The seeding strategies were integrated with four state-of-the-art multi-objective search algorithms and applied into two contexts where regression-testing is paramount: (1) Simulation-based testing of Cyber-Physical Systems and (2) Continuous Integration. For the first context, we evaluated our approach by using six fitness function combinations and six independent case studies, whereas in the second context we derived a total of six fitness function combinations and employed four case studies. Our evaluation suggests that some of the proposed seeding strategies are indeed helpful for solving the multi-objective test case selection problem. Specifically, the proposed seeding strategies provided a higher convergence of the algorithms towards optimal solutions in 96% of the studied scenarios and an overall cost-effectiveness with a standard search budget in 85% of the studied scenarios.

CCS Concepts: • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Test Case Selection, Search-based Software Testing, Regression Testing

ACM Reference Format:
Aitor Arrieta, Pablo Valle, Joseba A. Agirre, and Goiuria Sagardui. 2020. Some Seeds are Strong: Seeding Strategies for Search-based Test Case Selection. 1, 1 (May 2020), 49 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Generally, verification and validation activities are time consuming on large software systems. Companies that have a large code-base require a huge number of tests to be executed in many servers, taking hours or even days to complete [15]. In systems like Cyber-Physical Systems (CPSs), testing is time consuming as it requires execution at different levels, even for the same software versions [8, 18]. 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 [17, 80]. 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

Authors' addresses: Aitor Arrieta, Mondragon University, Goiru 2, Mondragon, Spain, 20500, aarrieta@mondragon.edu; Pablo Valle, Mondragon University, Goiru 2, Mondragon, Spain, 20500, pablo.valle@alumni.mondragon.edu; Joseba A. Agirre, Mondragon University, Goiru 2, Mondragon, Spain, 20500, jaagirre@mondragon.edu; Goiuria Sagardui, Mondragon University, Goiru 2, Mondragon, Spain, 20500, gsagardui@mondragon.edu.

XXXX-XXXX/2020/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

validation activities. These activities include several optimization aspects, including automated test case generation [1, 2, 4, 20, 40, 52, 62, 66–68, 74, 87], test case selection/minimization [8, 75, 94, 97–99, 101] and test prioritization [14, 36, 38, 53].

A widely investigated technique for increasing the cost-effectiveness of the verification and validation processes has been regression test selection [35, 100]. The test case selection problem is multi-objective in nature. On the one hand, adding a new test to a given test suite cannot decrease fault detection or the overall test suite adequacy [26], but it increases the overall test execution cost. On the other hand, the test execution cost can be reduced by removing tests from the test set, but this cannot increase fault detection or test set adequacy [26]. For these reasons, multi-objective search algorithms have been widely studied in the last few years to solve the test case selection problem [7, 8, 50, 75, 81, 94, 95, 98]. These approaches have already been successfully deployed in industry [41, 73]. In the last few years, test selection based on evolutionary algorithms have gained attention. Most of them have focused on comparing either (1) which adequacy criteria could fit best for integrating it in the fitness functions [7, 8, 46, 50, 98] or (2) which algorithm performs best when selecting test cases (when having one specific fitness function) [11, 81, 94, 95]. Additionally, most of them compare their approaches with a baseline algorithm, such as, Random Search (RS) [7, 8, 11, 81, 94, 95] or Greedy [98].

It is important to reiterate that applicable regression testing approaches (e.g., search-based test case selection) need to be significantly faster than the run-them-all approach to be beneficial [15]. Faster search-based test case selection approaches could be achieved by developing techniques that allow a faster convergence. 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 [39, 58, 60, 84] and service composition [21, 22]. In the case of test case selection, many studies have proposed either different algorithms or fitness functions [8, 10, 27, 41, 73, 95, 98, 99]. 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 [75]. Nevertheless, the approach presented by Panichela et al. [75] was algorithmic, and the seeding strategy for initializing the population needed to be accompanied by other mechanisms that injected diversity during the search process.

This paper is an extension of our previously published The Genetic and Evolutionary Computation Conference (GECCO) 2020 paper [6]. Specifically, we build upon our conference paper from the following perspectives:

- We propose an additional seeding strategy for the multi-objective test case selection problem [6].
- The original paper integrated our approach in the context of multi-objective black-box test case selection of simulation models of CPSs. Besides this context, our extension involves the integration of the approach within the Continuous Integration (CI) context. Within this new context, we used four new real-world case studies with real faults. Three of these new case studies were industrial case studies (one from Google [34, 54, 82, 83, 91], two from ABB Norway [15, 82, 83, 91]), and an open-source project (i.e., Rails [54]). This allowed us to validate that the proposed seeding strategies could work in different contexts.
- In the original GECCO paper we evaluated the seeding strategies solely in the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) algorithm [32]. In this extension we included three additional multi-objective search algorithms (i.e., IBEA [104], SPEA2 [105] and PESA-II [29]) in the evaluation, which allowed us to validate that the proposed seeding strategies do not only work with a single algorithm.

- We include an additional evaluation metric to study how the seeding algorithms help the search algorithms converge towards optimal solutions.
- We enrich background and related work sections.

In this paper, we present four 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. [75]. The seeding strategies are focused on generating the initial population of the algorithm, which gives a high flexibility when using any state-of-the-art population-based search algorithm. However, for the empirical evaluation, we integrated our seeding strategies with the four multi-objective search algorithms mentioned above. In addition, the evaluation was performed in two contexts where regression test selection might bring significant benefits to the software engineering productivity: (1) the context of simulation-based testing of CPSs, where we used as a base our previous studies [7, 8] and (2) the context of CI development environments. We can summarize the main contributions of this paper as follows:

- We propose a total of four seeding strategies for initializing the population of Pareto-efficient algorithms for the test case selection problem.
- We integrated the approach on top of PlatEMO [93], and instantiated it to solve the multi-objective test case selection into two different application domains: (1) Black-box test case selection of simulation models and (2) regression test selection in continuous integration environments. Within the first application domain, we have integrated the proposed seeding strategies in the framework for test case selection proposed in our previous work [7, 8], which is an open-source framework. Within the second application domain, we used openly available datasets of industrial and real-world case studies [34, 82, 83, 91].
- We perform an empirical evaluation using six case studies in the first application domain and four in the second one. For both domains we derived a total of six fitness combinations. In addition we used the above mentioned four multi-objective search algorithms. For the first domain we used a total of 144 experimental scenarios to compare the different seeding strategies, whereas in the second domain we used a total of 96 experimental scenarios.
- We make all our sources available for replication by other researchers.

To assess the proposed seeding strategies we employed mutation testing within the first context and real faults within the second. This allowed us to determine the fault-revealing capabilities of the solutions. Two evaluation metrics were employed: the Hypervolume (HV) and the Average Convergence, which measures how fast the algorithms converged. The results showed that two of the proposed seeding strategies helped the multi-objective search algorithms produce solutions with higher cost-effectiveness in both application domains, allowing a faster converge towards optimal solutions. Specifically, these two strategies allowed a faster convergence of the algorithms in 96% of the studied experimental scenarios with statistical significance. When the entire search process was considered, these seeding strategies outperformed the non-seeded technique for 85% of the scenarios with statistical significance.

The rest of the paper is structured as follows. General background is presented in Section 2. The proposed seeding strategies are presented in Section 3. The application domains are presented in Section 4. Section 5 explains how we evaluated the proposed seeding strategies. The evaluation results are analysed and discussed in Section 6. Section 7 discusses the threats to validity of our empirical evaluation and how we mitigated them. We position our work with other similar works in Section 8. We conclude the paper in Section 9.

2 BACKGROUND

In this section we present the basic background and terminology related to our paper.

2.1 Multi-objective search algorithms

2.1.1 Search algorithms. Search-based Software Engineeereing (SBSE) aims at formulating a software engineering problem as a mathematical optimization problem [13]. SBSE has been widely applied to solve a wide variety of software engineering problems, including requirements engineering [42], software-effort estimation [88] and software-product line configurations sampling [47]. Software testing is one of the primary areas where SBSE techniques have been applied at several software testing stages (e.g., test generation [1, 4, 40, 68, 84], test case selection/minimization [94, 95, 98, 99, 101] and test case prioritization [13, 36, 53, 90, 96]).

The search algorithms used in this study are evolutionary, meaning they aim at mimicking natural evolution processes [19]. Firstly, these algorithms generate an initial population. This stage is where our seeding strategies are applied. While evolutionary algorithms usually generate the initial population purely randomly, we generate it by using certain seeding strategies applied to the test case selection problem (Section 3). After generating the initial population, evolutionary algorithms apply three operators: (1) selection, (2) crossover and (3) mutation. These operators are applied until the search budget is exceeded (e.g., a time budget is exceeded or the number of fitness evaluations reaches a limit). The first operator (i.e., selection) aims at selecting individuals to be involved in the reproduction, typically by using the fitness functions so as to enable stronger solutions to survive [20]. The crossover operator recombines two individuals (i.e., solutions) based on a randomly selected crossover point (or set of crossover point). [1] The mutation operator changes the genes in each individual with certain probability, typically 1/N, where N is the number of genes in the solution [20].

2.1.2 Pareto optimality. Multi-objective search algorithms are based on the notion of Pareto optimality. This states that with multiple objectives, a solution s_a provided by the search algorithm is better than another solution s_b only when s_a dominates s_b in at least one objective while not being worse than s_b in the rest of the objectives [53]. Based on this, multi-objective search algorithms can be applied to find solutions (s_k) that optimize a set of P objective functions ($OF = \{of_1(s_k), of_2(s_k), ..., of_P(s_k)\}$). These objective functions are usually in conflict with each other [98].

In multi-objective search algorithms, assuming the above mentioned objective functions need to be maximized in order to produce more optimal solutions, a solution s_a is said to dominate the solution s_b ($s_a \succ s_b$) if and only if their objective functions ($of_i(s_a)$ and $of_i(s_b)$) satisfies the following [98]:

$$of_i(s_a) \geq of_i(s_b) \forall i \in \{1, 2, ..., P\}; \text{and } \exists i \in \{1, 2, ..., P\} | of_i(s_a) > of_i(S_b)$$

Unlike single-objective search algorithms, which provide the best single solution based on an individual function, the goal of multi-objective algorithms is to provide a set of solutions that are not dominated by any other in the population. These solutions are said to form the *ParetoOptimalSet* [98]. Their corresponding objective functions form the *ParetoFrontier* [98]. The multi-objective optimization problem is defined as follows [98]. For a given vector of decision variables x, and a set of objective functions $of_i(x)$, where i = 1,2,...,P, maximize $\{of_1(x), of_2(x), ..., of_P(x))\}$ by finding the Pareto optimal set over the feasible set of solutions.

[1] In our study we employ a single-point crossover operator

2.2 Multi-objective test case selection and notation

The multi-objective test case selection problem can be formulated in two ways [97], by using a weighted fitness function, where a multi-objective problem is converted into a single-objective problem [94], or by adopting multiple objectives [98]. In this paper we opted for the second approach, although the proposed seeding strategies can be used with any kind of population-based search algorithm that is applied to solve the test case selection problem, including single-objective search algorithms. The multi-objective test case selection problem has as a goal to select a Pareto efficient subset of the test suite based on multiple criteria [98].

In the remainder of the paper we will use the following notation. $TS = \{tc_1, tc_2, ..., tc_N\}$ 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 [75]. 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 $M \leq N$ [75]. Like most multi-objective test selection studies [7, 8, 75], 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. Conversely, if the i-th digit is 0, it means that the test case tc_i has not been selected.

Let us consider as an example the code snippet in Listing 1 aiming to return the maximum value of two input parameter (i.e., n and m). Let us assume that two objective functions are used by a test selection algorithm: (1) the quality objective is line coverage (i.e., number of lines executed by the selected test cases in the TS') and (2) the cost objective is the number of selected test cases. Thus, a test selection algorithm would aim at increasing the line coverage, while reducing the number of selected test cases. Let us also assume that we have an initial Test Suite TS of four test cases (i.e., $TS = \{tc_1, tc_2, tc_3, tc_4\}$), where $tc_1 = \{n = 2, m = 1\}$, $tc_2 = \{n = 1, m = 1\}$, $tc_3 = \{n = 5, m = 1\}$, $tc_4 = \{n = 2, m = 5\}$.

The search algorithm returns a series of solutions represented a binary coding. For instance, the following solution $s_k = \{1, 0, 0, 0\}$ selects only tc_1 from TS; assuming lines 2-5 are considered to compute the line coverage, such test will cover a 75% of line coverage (i.e., line 4 is not covered), and the cost function, which refers to number of test cases, will be 1. In terms of line coverage, one could argue that the following solution $s_j = \{0, 0, 0, 1\}$ is better, since test case tc_4 from TS would exercise line 4, which is not covered by any other test case in TS, achieving this way a line coverage of 100%. Meanwhile, the test cost would remain the same (i.e., number of test cases equal to 1). Thus, for this specific objective function, solution s_j dominates solution s_k (i.e., $s_j \succ s_k$).

Naturally, one could argue that s_j is not an appropriate test suite, because it does not test the situation where n is higher than m (i.e., the condition in line 3 is not negated). Thus, if the quality metric would be changed to condition coverage, both s_k and s_j would have the same coverage percentage (i.e., 50%). This condition coverage could be further improved by selecting, for instance, test cases tc_1, and tc_4. The algorithm would encode this in solution s_l as follows: $s_l = \{1, 0, 0, 1\}$. In such a case, the condition in line 3 is set both to false (when executing tc_1) and true (when executing tc_2); therefore, solution s_l would obtain a condition coverage of 100%, while its cost would increase to 2 test cases. s_l would not dominate neither solution s_j nor s_k, since the cost for executing the test suites represented by s_j and s_k are lower. Therefore, for the first example (line coverage) the Pareto optimal set is s_j, whereas the second example (condition coverage) the Pareto optimal set would then be s_j, s_k and s_l.

```
int max (int n, int m){
  int max = n;
  if(m>n)
    max = m;
  return max;
}
```

Listing 1. Code snippet of a function for returning the highest value from two integer inputs

The search space for test case selection is 2^N, which means that it grows exponentially with the number of tests in the initial test suite. For example, for an initial test suite of 100 test cases, there can be a total of 1.27×10^{30} solutions. It is impracticable to computationally determine the objective functions for such a large search space. Therefore, using search algorithms as an alternative to brute force in order to solve the test case selection problem is a sensible option.

3 SEEDING STRATEGIES

In this section we explain the proposed four seeding strategies and the seeding strategy proposed by Panichella et al. [75].

3.1 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 [75], 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. There is a high likelihood that when i is low, there might be solutions with no test cases, which results in no test execution (i.e., no cost), but also not testing anything (i.e., no effectiveness). However, this is not considered an issue. Firstly, from the algorithmic perspective, this can be positive because during the generations of the search algorithm, these type of solutions can help other when doing the crossover, removing a large amount of test cases from the suite. From the practical aspect, this might be useful in different contexts. For instance, in the context of Continuous Integration (CI), if two subsequent commits are performed fastly (e.g., due to having forgotten to add something), the developer might not want to test the first commit. Another example could be when using a fitness function measuring coverage, and having no test cases covering a newly added line. Furthermore, it is important to highlight that the targeted search algorithms are multi-objective, what means that there will very likely be more than one solution in the Pareto-frontier which can be selected and which will have test cases.

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 ← 1 to nPop do
2     for j ← 1 to N do
3         if rand() ≤ i /nPop then
4             initialPop(i,j) = 1;
5         else
6             initialPop(i,j) = 0;
7         end
8     end
9 end
```

Figure 1 shows the example of the objective space covered in two random runs of the seeding strategies (i.e., without running the rest of the algorithm), one by the Dynamic Test Suite Size-based Random Seeding strategy proposed in this section (the left figure) and a non-seeded approach (the right figure). As can be seen, the proposed seeding strategy covers a wider area in the objective space compared to the non-seeded approach. This is due to a wider variety in the test suite sizes.

In the remainder of the paper, we refer to this seeding strategy as "Dynamic".

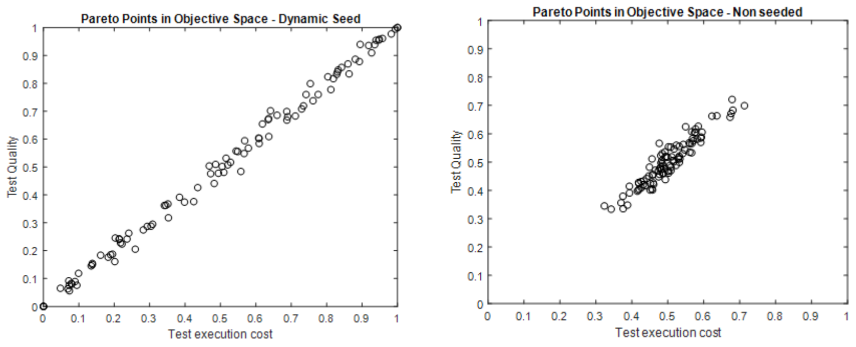


Fig. 1. Example of two random runs of the population execution for the proposed dynamic test suite size-based random seeding strategy and a non-seeded strategy

3.2 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.

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 $i \leftarrow 1$ to $nPop$ do
2     for $j \leftarrow 1$ to $N$ do
3         if $rand() \leq desiredTestSuiteSize /100$ then
4             initialPop(i,j) = 1;
5         else
6             initialPop(i,j) = 0;
7         end
8     end
9 end
```

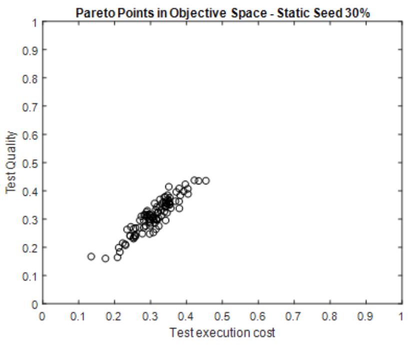
Figure 2 shows how the solutions are distributed for two independent runs of the static test suite size-based random seeding that was configured to have a test suite size of 30 and 70%. As can be seen, the objective area covered by the 30% test suite size strategy covers an area of lower cost but also lower test quality as compared with the 70% test suite size strategy.

In the remainder of this paper, we refer to this strategy as Static30 or Static70, being 30 and 70 two independent configurations of this strategy for the desiredTestSuiteSize parameter.

3.3 Adaptive Random Population Generation

In previous studies, it has been shown that injecting diversity into the population improves performance of test case selection [75], as it leads the algorithm to have lower probability of being trapped in some local optimum [31]. Inspired by the Adaptive Random Testing (ART) algorithm [25], 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 [25]. This algorithm has been widely used to generate test cases [23–25], 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).



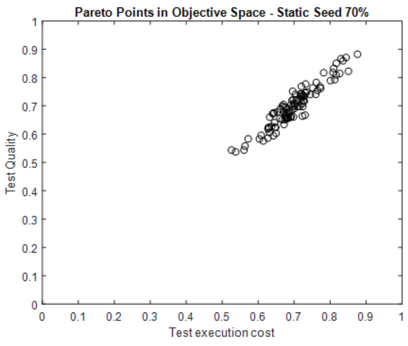


Fig. 2. Example of two random runs of the population execution for the proposed Static Test Suite Size-based Random Seeding strategies (configured to the 30% and the 70% of the test suite size)

Typically, the size of this set of candidates is 10 [24, 25], 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. [25]. 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.

An 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 [7, 8, 98], which makes this strategy applicable in practice. In the remainder of the paper, we refer to this strategy as ARPG. Some distance measures might have also a strong influence in the execution time of this seeding strategy in those cases where the number of tests is long. We therefore recommend distance functions who have linear execution time based on the number of test cases, as we do with the Hamming Distance.

3.4 Adaptive Random Population Generation with Dynamic Test Suite Size

This algorithm complements the Dynamic Test Suite Size-based Random Seeding with the Adaptive Random Population Generation Seeding strategy. We conjecture that having solutions in the population with a different test suite size and the more diverse as possible will increase the performance of the multi-objective test case selection approach.

Algorithm 4 shows the procedure to generate the initial population by following this strategy. In a first step, an initial solution is generated and included in the initial population, where the probability of including a test case in this solution is $1/nPop$ (Lines 1-7). 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, where the probability of selecting a test case in a candidate solution is $i/nPop$, i being the index of the solution being generated in the population (Lines 9-17). 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 and included in the solution (Lines 18-29). As with the previous strategy, we use the Hamming Distance to measure the distance

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 $i \leftarrow 2$ to $nPop$ do
3      for $j \leftarrow 1$ to $nCandidate$ do
4          candidateSets(j,:) = randSol(N);
5      end
6      for $j \leftarrow 1$ to $nCandidate$ do
7          minDist = 1;
8          for $k \leftarrow 1$ to $i-1$ do
9              distance = measureDistance(initialPop(k,:), candidateSet(j,:))
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
```

between a candidate set and a solution of the population. This process is repeated until the entire initial population is generated.

Similarly to the Dynamic seeding strategy, in this case there is high likelihood that the initial populations have no test cases selected. However, as previously explained in Section 3.1, this is not considered an issue.

As can be seen, the difference between this strategy and the ARPG is the way the first solution to be included in the population is generated and the way the candidate sets are generated. For such generation, the probability of selecting a test case incrementally varies as the population is being generated. This process for changing the size might make the generation of candidates slightly slower when compared with the ARPG, because the algorithm needs to assign a test or not based on a problability (Lines 2-6 and 11-15). Nevertheless, this cost is marginal when compared to the cost of the entire execution of the algorithms. In the remainder of the paper, we refer to this strategy as DynamicARPG.

3.5 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 Panichella et al. [75]. 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 [71] that uses the Hadamard matrices to build orthogonal arrays. The underlying idea by Panichella et al. [75] was to distribute and diversify the initial population in order the exploration to be

Algorithm 4: Algorithm for the ARPG seeding with dynamic test suite size

```
Input: N //Number of test cases
nPop //Population size
nCandidate //Number of candidate solutions
Result: initialPop // initial population
for $i \leftarrow 1$ to $N$ do
    if $rand() \leq 1 /nPop$ then
        initialPop(1,i) = 1;
    else
        initialPop(1,i) = 0;
    end
end
for $i \leftarrow 2$ to $nPop$ do
    for $j \leftarrow 1$ to $nCandidate$ do
        for $k \leftarrow 1$ to $N$ do
            if $rand() \leq i /nPop$ then
                candidateSet(j,k) = 1;
            else
                candidateSet(j,k) = 0;
            end
        end
    end
    for $j \leftarrow 1$ to $nCandidate$ do
        minDist = 1;
        for $k \leftarrow 1$ to $i-1$ do
            distance = measureDistance(initialPop(k,:), candidateSet(j,:))
            if $distance < minDist$ then
                minDist = distance;
            end
        end
        distArray(j) = minDist ;
    end
    distIndx = max(distArray);
    initialPop(i,:) = candidateSets(distIndx,:);
end
```

more effective. They conjecture that this diversification favours the search algorithm (a Genetic Algorithm in their case) towards global optima.

Algorithm 5 shows the procedure to generate this population following the orthogonal population generation method [75]. Given a population size (*nPop*) and the number of test cases in the initial test suite (N), Lines 1-4 aim at generating a Hadamard matrix of dimensions $K \times K$. A Hadamard matrix H of order n is an $n \times n$ matrix with the following property:

$$H \times H^T = H^T \times H = nI \tag{1}$$

where all the elements in H are either +1 or -1 and I is the identity matrix. This matrix, except for the first row and the first column, have the same number of +1 and -1 elements. The following is a Hadamard matrix of $n = 4$:

$$H_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix} \tag{2}$$

As proposed by Panichella et al. [75], we generate this matrix using the *hadamard* function available in MATLAB. To generate K, Panichella et al., used $K = \lceil (L+1)/4 \rceil \times 4$, where $\lceil (L+1)/4 \rceil$ refers to the smallest integer number greater or equal to $K = \lceil (L+1)/4 \rceil$, L being the highest number of *nPop* (i.e., number of population) or N (i.e., number of test cases). However, in the MATLAB version we used (i.e., MATLAB 2019b), K had to be K/12 or K/20 or power of 2. Subsequently, we implemented the function *getHadamardK* which returned the lowest value equal or higher than $\lceil (L+1)/4 \rceil \times 4$ that met that condition. As proposed by Panichella et al. [75], we remove the first column of the matrix (Line 5) because it is the only one that contains only +1 elements, which means that the first test case in TS will be selected by all the solutions in the population. Lines 6 to 14 convert the generated matrix into the population following the encoding we used (i.e., binary coding (see Section 2.2)). In the remainder of the paper we refer to this seeding strategy as "Orthogonal".

Algorithm 5: Algorithm for the Orthogonal Population seeding

```
   Input: nPop //Population size
   N //Number of Test cases
   Result: initialPop // initial population
1  L=max(N,nPop);
2  K= getHadamardK(⌈(L+1)/4⌉ × 4)
3  $H_k$= hadamard(K);
4  $H_K$ = sortRowsAscendingOrder($H_k$);
5  $H_K$ = deleteFirstColumn($H_k$);
6  for $i \leftarrow 1$ to $nPop$ do
7      for $j \leftarrow 1$ to $N$ do
8          if $H_k(i,j) == 1$ then
9              initialPop(i,j) = 1;
10         else
11             initialPop(i,j) = 0;
12         end
13     end
14 end
```

4 APPLICATION DOMAINS

The proposed seeding strategy can be applied in any population-based search algorithm tackling the test case selection problem. Test case selection has been applied in multiple software applications, including C++ software [86], Java software [44] and Software Product Lines [94, 95]. This section describes the two selected domains where we applied the proposed seeding strategies.

4.1 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) [18, 66]. 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 [68]. However, simulating some of these systems is commonly time-consuming, where a single simulation may take hours to complete in some systems [68]. 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 [8, 13], 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 [8, 18]. Consequently, test optimization is paramount, and recent approaches have proposed black-box testing methodologies, including test case generation [65, 66, 68] and test case selection [10, 13]. 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) [8].

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 [8]. These adequacy criteria can be categorized into two main parts: (1) test quality metrics, which measured a quantitative degree of certain anti-patterns defined in previous works [63], 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 [37, 45, 46]. Four anti-patterns for simulation-based testing were selected based on previous studies [7, 8, 63]: (1) instability, (2) discontinuity, (3) growth to infinity and (4) the output minimum-maximum difference. Figure 3 illustrates the first three. Instability refers to the anti-pattern where an output signal shows quick and frequent oscillations [63]. Discontinuity is an anti-pattern in which an output signal shows a short duration pulse [63]. The growth to infinity anti-pattern measures how an output signal grows to an infinite value. The output minimum-maximum difference measures the difference between the minimum and maximum output values [8]; this metric was inspired to overcome simulation-based specific problems (e.g., not having specification for bounds and predicting how thoroughly a model has been exercised) [8]. As for the similarity measures, two measures were used: (1) input-based test similarity and (2) output-based test similarity. The former measures the similarity of a set of selected test cases by considering the signals used to stimulate the model under test. The latter measures the similarity of a set of selected test cases by considering the output signals of the model. The Euclidean distance is used to measure this similarity as proposed in previous studies [64, 65]. All the details of the fitness functions are explained in a prior work [8].

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 [8] 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

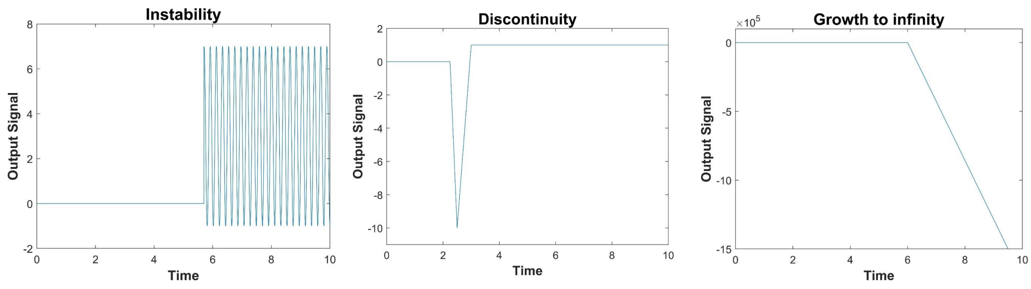


Fig. 3. Anti-patterns for simulation models [8]

with different fitness functions. Based on the aforementioned anti-patterns and similarity metrics, we derived six fitness function combinations, summarized in Table 1. We used only two objective functions because we figured out that the provide a higher cost-effectiveness than when using more objective functions [6, 8]. Furthermore, we evaluate more case studies and more algorithms and having a larger amount of fitness function combinations would significantly increase the time for executing experiments.

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

Fitness configuration ID	Objetcive 1	Objective 2
c1	Discontinuity	Test execution time
c2	Growth to infinity	
c3	Instability	
c4	Input similarity	
c5	Output similarity	
c6	MinMax	

Fitness functions for the black-box test case selection context

We now explain how the fitness functions where defined in the first application context.

Test Execution Time (TET): Define a set of test cases $TS_{select} = \{tc_{s_1}, tc_{s_2}, ..., tc_{s_N}\}$, that are selected from the initial test suite $TS_{init} = \{tc_{i_1}, tc_{i_2}, ..., tc_{i_M}\}$. Let TET_{s_a} denote the test execution time of the test case tc_{s_a} from TS_{select}, and TET_{i_b} be the test execution time of test case tc_{i_b} in TS_{init}. The total Test Execution Time (TET) of a set of selected test cases is:

$$TET_{select} = \frac{\sum_{a=1}^{N} TET_{s_a}}{\sum_{b=1}^{M} TET_{i_b}} \tag{3}$$

The search algorithm aims at $\underline{\text{minimizing}}$ this metric.

Discontinuity (D): Based on the above definition, let D_{s_a} denote the discontinuity level of the test case tc_{s_a} in TS_{select}, and D_{s_b} the discontinuity level of test case tc_{s_b} in TS_{init}, normalized from 0 to 1. The total discontinuity score of a set of selected test cases is:

$$D_{select} = \frac{\sum_{a=1}^{N} D_{s_a}}{\sum_{b=1}^{M} D_{s_b}} \tag{4}$$

The search algorithm aims at $\underline{\text{maximizing}}$ this metric. More details of how the discontinuity level of a test case is measured and its normalization procedure is explained in our prior study [8].

Growth to infinity (G): Based on the above definition, let G_{s_a} denote the growth to infinity level of the test case tc_{s_a} in TS_{select}, and G_{s_b} the discontinuity level of test case tc_{s_b} in TS_{init}, normalized from 0 to 1. The total growth to infinity score of a set of selected test cases is:

$$G_{select} = \frac{\sum_{a=1}^{N} G_{s_a}}{\sum_{b=1}^{M} G_{s_b}} \tag{5}$$

The search algorithm aims at maximizing this metric. More details of how the growth to infinity level of a test case is measured and its normalization procedure is explained in our prior study [8].

Instability (I): Based on the above definition, let I_{s_a} denote the instability level of the test case tc_{s_a} in TS_{select}, and I_{s_b} the instability level of test case tc_{s_b} in TS_{init}, normalized from 0 to 1. The total growth to infinity score of a set of selected test cases is:

$$I_{select} = \frac{\sum_{a=1}^{N} I_{s_a}}{\sum_{b=1}^{M} I_{s_b}} \tag{6}$$

The search algorithm aims at maximizing this metric. More details of how the instability level of a test case is measured and its normalization procedure is explained in our prior study [8].

MinMax (MM): Based on the above definition, let MM_{s_a} denote the minimum-maximum difference level of the test case tc_{s_a} in TS_{select}, and MM_{s_b} the minimum-maximum difference level of test case tc_{s_b} in TS_{init}, normalized from 0 to 1. The total growth to infinity score of a set of selected test cases is:

$$MM_{select} = \frac{\sum_{a=1}^{N} MM_{s_a}}{\sum_{b=1}^{M} MM_{s_b}} \tag{7}$$

The search algorithm aims at maximizing this metric. More details of how the minimum-maximum difference level of a test case is measured and its normalization procedure is explained in our prior study [8].

Input similarity (disInput): Based on the above definition, let $disIn(tc_{s_a}, tc_{s_b})$ be the input distance function between test cases tc_{s_a} and tc_{s_b} from TS_{select}, and let $disIn(tc_{s_c}, tc_{s_d})$ be the input distance function between test cases tc_{s_c} and tc_{s_d} from TS_{init}. The distance metric is based on the Euclidean distance [8], and it ranges from 0 to 1, where 0 means that the two test cases are the same and 1 means that they are completely different (based on the input signals). The total input distance score of a set of selected test cases is:

$$disInput_{select} = \frac{\sum_{a=1}^{N} \sum_{b=a+1}^{N} disIn(tc_{s_a}, tc_{s_b})}{\sum_{c=1}^{M} \sum_{d=c+1}^{M} disIn(tc_{s_c}, tc_{s_d})} \tag{8}$$

The search algorithm aims at maximizing this metric. More details of how the distance between two test cases is measured is explained in our prior study [8]. It is assumed that there are dissimilar test cases and therefore, the Equation does not divide by zero.

Output similarity (disOutput): Based on the above definition, let $disOut(tc_{s_a}, tc_{s_b})$ be the output distance function between test cases tc_{s_a} and tc_{s_b} from TS_{select}, and let $disOut(tc_{s_c}, tc_{s_d})$ be the output distance function between test cases tc_{s_c} and tc_{s_d} from TS_{init}. The distance metric is based on the Euclidean distance [8], and it ranges from 0 to 1, where 0 means that the two test cases are the same and 1 means that they are completely different (based on the output signals). The total output distance score of a set of selected test cases is:

$$disOutput_{select} = \frac{\sum_{a=1}^{N}\sum_{b=a+1}^{N} disOut(tc_{s_a}, tc_{s_b})}{\sum_{c=1}^{M}\sum_{d=c+1}^{M} disOut(tc_{s_c}, tc_{s_d})} \tag{9}$$

The search algorithm aims at maximizing this metric. More details of how the distance between two test cases is measured is explained in our prior study [8]. It is assumed that there are dissimilar test cases and therefore, the Equation does not divide by zero.

4.2 Regression test selection in continuous integration environments

Continuous Integration (CI) environments allow software engineers to continually integrate and test their code [54]. This allows for reducing the amount of code rework required to evolve the system by speeding up development time [34]. In this context, it is paramount to ensure that enough testing is performed prior to code submission [34]. However, the codebase can be extremely large, and it might not be possible to execute the entire test suite. Subsequently, regression test selection approaches have been investigated to make continuous integration testing more cost-effective [34, 49, 82, 83, 91, 102].

CI environments can provide a large amount of historical information in relation to the execution of tests (e.g.., tested versions, verdict information, test case duration). Inspired by previous works [34, 82, 83, 95], we have defined the following objectives functions adapted to the multi-objective test case selection context:

- Fault Detection Capability (FDC): CI environments allow for extracting information related to the failure rate obtained by each test case. Based on information of the failure rate of each test case, we define the FDC of a given $TS' \subseteq TS$ as the sum of the failure rate of each of the selected test cases in TS'. The search algorithm aims at maximizing this objective function. This metric has shown a high effectiveness in several regression test optimization approaches [11, 13, 82, 83, 94, 95].
- Time Since Last Failure (TSLF): This metric was proposed by Elbaum et al. [34] to redefine test case selection and prioritization in CI environments. In this paper we adapt it to the search-based test case selection context. Specifically, it aims at favouring those test cases that have failed in a recent version. Elbaum et al. [34] conjecture that those tests that have failed in a recent version are "proxies" for code changes, targeting code that is churning. The search algorithm aims at minimizing this objective function.
- Time Since Last Execution (TSLE): This metric was also proposed by Elbaum et al. [34], and aims at favouring those tests that have not been executed in a while. The hypothesis is that despite having minor effects initially, if a test suite is repeatedly ignored over time, it can have a negative effect in the regression testing effectiveness. The search algorithm aims at maximizing this objective function to favour those test cases that have not been executed in recent versions.

Other common metrics in the context of regression testing are those related to white-box coverage. However, we could not use this information because they were not available in the used dataset. Conversely, the defined metrics have been used in other studies (e.g., [34, 82, 83, 95]) and these have been found to be effective. For instance, the FDC metric directly obtains information of the failures detected by the test cases, which is an information that is highly accurate in the context of regression testing.

We selected this second context as an extension to our conference-version paper [6] due to several reasons. Firstly, it is a context that has been widely investigated during the last few years in the context of regression testing [15, 34, 49, 54, 61, 82, 83, 91, 102]. This is, to a large extent, due to the importance that CI environments pose in current software engineering companies to develop their software systems. Secondly, we found available dataset from large projects and industrial companies, including Google [34, 54, 82, 83] and ABB [82, 83, 91]. Lastly, the context is very different to the one used in the conference version paper [6] and presented in the previous version, which helps validate that the proposed seeding strategies could work in other contexts. We derived six fitness functiopn combinations (Table 2) by combining each of the objective functions defined above with the test execution time by forming a bi-objective fitness combination. In addition, we used up to three objectives by combining each of them with one another and with the test execution time.[2]

Table 2. Fitness function combinations integrated on the Regression test case selection algorithms

Fitness configuration ID	Objetcive 1	Objective 2	Objective 3
c1	FDC		Test execution time
c2	TSLF		
c3	TSLE		
c4	TSLF	TSLE	
c5	FDC	TSLE	
c6	FDC	TSLF	

Fitness functions for the regression test selection in continuous integration environments

We now explain how the fitness functions where defined in the second application context.

Test Execution Time (TET): Define a set of test cases $TS_{select} = \{tc_{s_1}, tc_{s_2}, ..., tc_{s_N}\}$, that are selected from the initial test suite $TS_{init} = \{tc_{i_1}, tc_{i_2}, ..., tc_{i_M}\}$. Let TET_{s_a} denote the average test execution time of the test case tc_{s_a} for all the CI cycles from TS_{select}, and TET_{i_b} be the average test execution time of test case tc_{i_b} for all the CI cycles in TS_{init}. The total Test Execution Time (TET) of a set of selected test cases is:

$$TET_{select} = \frac{\sum_{a=1}^{N} TET_{s_a}}{\sum_{b=1}^{M} TET_{i_b}} \tag{10}$$

The search algorithm aims at minimizing this metric.

Fault Detection Capability (FDC): Based on the above definition, let FDR_{s_a} denote the failure detection ration of test case tc_{s_a} in TS_{select}, and FDR_{s_b} the failure detection ratio of test case tc_{s_b} in TS_{init}, normalized from 0 to 1. The total FDC score of a set of selected test cases is:

$$FDC_{select} = \frac{\sum_{a=1}^{N} FDR_{s_a}}{M} \tag{11}$$

The Failure Detection Ration (FDR) of a test case refers to the number of times it has failed (i.e., triggered a failure) with respect to the number of times the test has been executed [82, 83]. The search algorithm aims at maximizing this metric.

Time Since Last Failure (TSLF): Based on the above definition, let $LFCD_{s_a}$ denote the time difference since the test case tc_{s_a} in TS_{select} triggered a failure in terms of cycle difference (e.g., if the test is aimed to be selected at cycle 300, and the last time it triggered a fault was in cycle 250, the difference will be 50). Let NoC be the total number of cycles. If the test case has never failed, the $LFCD_{s_a}$ is set to NoC. The total $TSLF_{select}$ of a set of selected test cases is:

[2]The selected algorithms for the evaluation were multi-objective. Therefore it is not appropriate to use more than three objective functions.

$$TSLF_{select} = \frac{\sum_{a=1}^{N} LFCD_{s_a} + NoC \times |M - N|}{M \times NoC} \tag{12}$$

The search algorithm aims at minimizing this metric.

Time Since Last Execution (TSLE): Based on the above definition, let $LECD_{s_a}$ denote the time difference since the test case tc_{s_a} in TS_{select} was executed in terms of cycle difference (e.g., if the test is aimed to be selected at cycle 300, and the last time it was executed was in cycle 299, the difference will be 1). Let NoC be the total number of cycles. The total $TSLE_{select}$ of a set of selected test cases is:

$$TSLE_{select} = \frac{\sum_{a=1}^{N} LECD_{s_a}}{M \times NoC} \tag{13}$$

The search algorithm aims at maximizing this metric.

5 EMPIRICAL EVALUATION

In this section we discuss the experimental set-up we carried out to empirically evaluate the proposed seeding strategies.

5.1 Research questions

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 that did not have the initial population seeded. To this end, we compared, for the first application context, a total of six fitness combinations along with four search algorithms (i.e., NSGA-II, IBEA, SPEA2 and PESA-II) over six case studies (i.e., a total of 144 combinations). Additionally, for the second application context, a total of 6 fitness combinations within four search algorithms over four case studies (i.e., a total of 96 combinations) were compared. To answer this RQ, for each of all these 240 combinations, five seeding strategies (the static seed had 2 independent configurations) were compared with a non-seeded approach. The non-seeded approach generated the initial populations purely randomly.

RQ2: Among the proposed seeding strategies, which one fares best? The intention behind the second RQ was to assess whether there is a seeding strategy that stands out over the rest in order to recommend it to practitioners. To this end, we compared each of the seeding strategies with one another for each of the 240 combinations (i.e., 144 combinations for the first application context and 96 combinations for the second one).

RQ3: Do the proposed seeding strategies produce significant execution time overhead? The third RQ aims at comprehensively analysing the overhead caused by the proposed seeding strategies. To this end, we measure the time taken by the different algorithms with and without the seeding strategies.

5.2 Case studies

This section briefly explains the details of the selected case studies in both of the selected application contexts.

5.2.1 Black-box test case selection case studies. Six case studies involving Simulink models of different sizes, complexities and domains were employed, which provide a wide heterogeneity to the experiment. Table 3 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 [92]. This case study was previously used in other evaluations [63]. 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 [7–9, 43, 63, 65, 66, 69, 72]. We used the same test cases provided by in our benchmark [8] in order for the results to be compared with their approach.

Table 3. Key characteristics of the selected case studies in the first application context

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

5.2.2 Continuous-integration test case selection case studies. Four case studies involving real test execution historical information of CI projects were used. Three of them were industrial case studies from ABB Robotics Norway and Google. The ABB Robotics Norway's case studies relate to (i) ABB Paint Control and (ii) ABB IOF/RO, and provided information related to the tests for testing robotic systems. These case studies have been previously used in other regression testing studies [15, 82, 83, 91]. The case study from Google relates to the GSDTSR data set, which contains test suite results executed over a sample of Google products. The dataset, originally used in the paper [34] has also been used in other regression testing studies (e.g., [54, 82, 83]). The last case study related to Rails, an open-source case study written in Ruby and hosted on Travis CI and used in a previous work [54]. Table 4 provides a summary of the characteristics for the selected case studies, which includes (1) the number of test cases that each case study has, (2) the number of CI cycles, (3) the total number of verdicts (e.g., a test case might have been executed more than once) and (4) the percentage of failed test cases (i.e., percentage of verdicts that returned a "fail").

Table 4. Key characteristics of the selected case studies in the second application context

Case Study	# of Test Cases	# of CI Cycles	# of Verdicts	% of Failed Test Cases
ABB Paint Control	89	352	25,594	19.36%
ABB IOF/ROL	1,941	320	3,019	28.42%
Google GSDTSR	5,555	336	1,260,617	0.25%
Rails	2010	3,528,911	313,859,138	0.0051%

5.3 Evaluation metrics

We used the revisited Hypervolume (HV) metric to measure the effectiveness of our approach, as employed in similar works [7, 8, 75]. For the first application context, for a set of solutions from the Pareto-frontier, we used (1) the cost and (2) the percentage of faults revealed by each solution as external utility functions. For the second application context, for a set of solutions from the Pareto-frontier, we used (1) the cost and (2) the percentage of failures revealed by each solution 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 percentage of faults/failures detected and the cost (i.e., the test execution time). By using this information, a new Pareto-frontier was obtained aiming at maximizing the percentage of faults detected 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 percentage of faults detected 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.

For the first application context (i.e., black-box test case selection of simulation models) 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 [48]. To this end, a set of mutants was 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., [76], (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 [8], which employed the mutation operators proposed by Hann et al., [43] for Simulink models. Information related to the number of mutants in the initial set and the final set are available in Table 3. The distribution of the resulting mutants with respect to the test cases can be found in Figure 4. As can be seen, there are different kind of distributions. For instance, the ACEngine case study has a large number of test cases not detecting any single mutant, and only a small portion of the test cases in the test suite detects mutants. However, it is noteworthy that in many cases, there are test cases which are very long in time, subsequently, leading to detecting more mutants. Therefore, in occasions, it is better to select a few test cases that have a small test execution time than a single one having a long execution time. For the second application context (i.e., regression test selection in continuous integration environments), similar to the other studies using these datasets [82, 83], we used the information available for the evaluation cycle that was used related to the real failures detected by the executed tests. Figure 5 depicts a histogram with the relation between the number of test cases and the percentage of cycles that these have failed. As can be seen, the type of failure distribution is different in the four case studies. In the Google dataset, a high amount of failures are triggered by a few test cases. A similar pattern can also be appreciated in the Rails case study, although not as exacerbated as in the Google's dataset. In these both datasets, there is a high amount of test cases that do not fail during the cycles. However, notice that the distribution of both IOF and Paint datasets are completely different. It can be observed that in the Paint dataset there is a high amount of tests that have failed between 75 and 85% of the cycles. Conversely, in the IOF dataset, it can be appreciated that there are around 100 test cases that have never failed, and there is a large amount of test cases that have failed in only a few cycles. There are however, also, a large amount of test cases that have failed in most of the cycles; for instance, there is a high number of test cases that have failed in 75% or more test cycles, unlike the cases of Google and Rails. Subsequently, we believe that there is a high heterogeneity in the context of failure distributions. More importantly, all this data comes from real-world and industrial case studies. Unfortunately, in this second application context, information of the differences in faults was not available. This means that a fault could be revealed by two different test cases, including a bias in our evaluation (i.e., one can argue that a single test is enough to detect more than one fault, and the second test is useless to execute). Nevertheless, maximizing the number of detected failures is an important aspect of test selection too, and this metric has been also used in previous works [82, 83, 91] using these case studies. Furthermore, we believe that there might be a strong correlation between the percentage of failures detected and the number of faults detected.

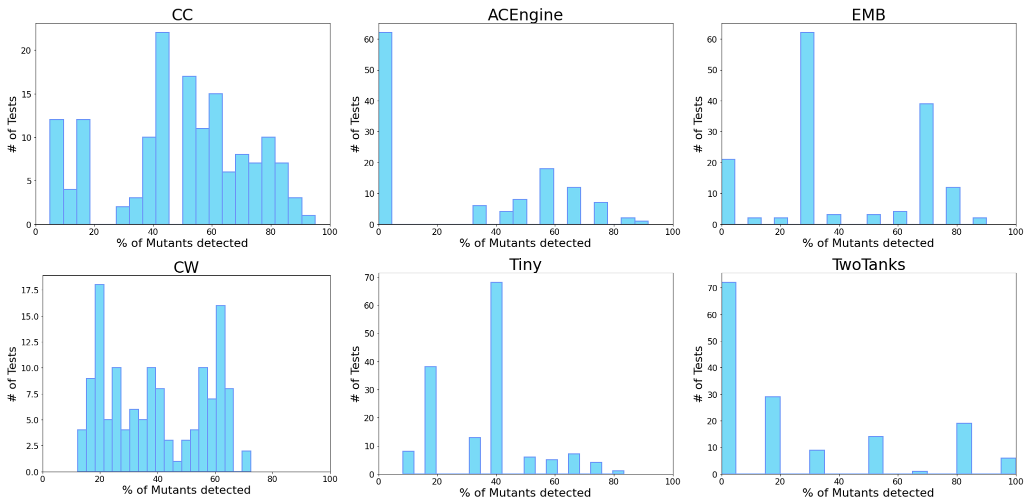


Fig. 4. Histogram showing the distribution of faults among the test cases for the first application context

The HV was measured both, within the last set of Pareto-frontier solutions provided by the search algorithms as well as during the search (every 500 generations). This not only provided us information on how the proposed seeding strategies impacted on the final result but also allowed us to observe how fast the algorithms converged. We measured how fast our algorithms converged by using the Average Convergence metric, as proposed in other studies where the convergence of different seeding strategies for search algorithms was measured [77–79]:

$$Avg.Convergence = 1 - \frac{OptimalFitness - AverageFitness}{OptimalFitness} \tag{14}$$

where, for each algorithm run, *AverageFitness* refers to the average HV values obtained for the ten measured iterations (i.e., every 500 iterations of the algorithm the modified HV is measured) and *OptimalFitness* refers to the known highest HV value for that case study (i.e., for each of the case studies we checked for the highest HV obtained by our algorithms). A higher value of *Avg.Convergence* means that the algorithm has found a set of optimal solutions faster, meaning that it has converged faster.

5.4 Experimental scenarios and statistical tests

The evaluation was divided into different experimental scenarios for each of the selected application domains. Each experimental scenario included:

- Case study: Six case studies in the first application domain and four case studies in the second application domain

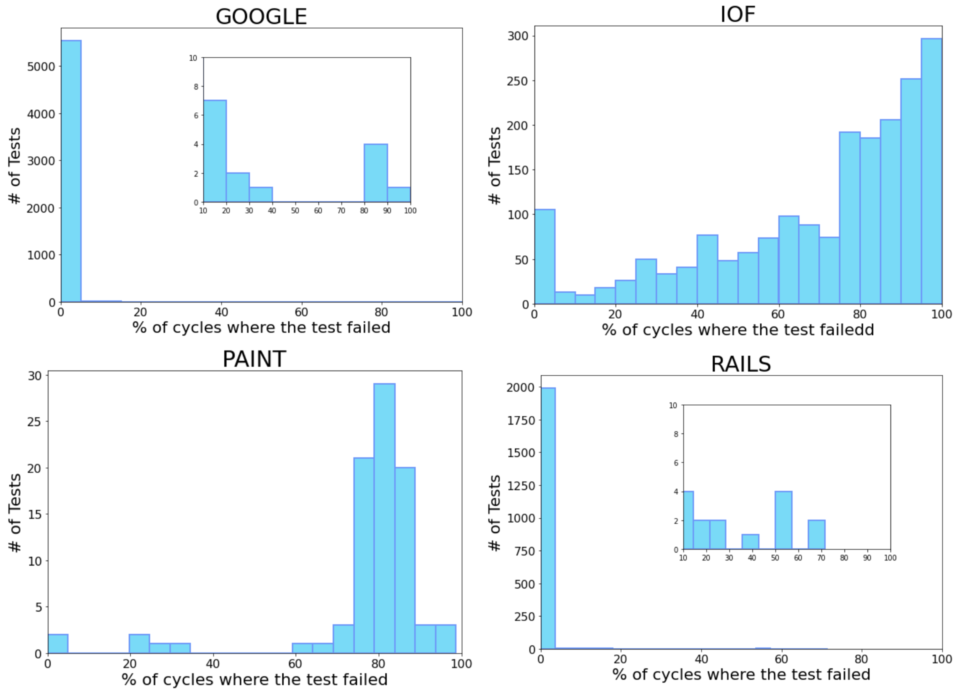


Fig. 5. Histogram showing the distribution between the test cases and the number of cycles where the test failed for the second application context

- Multi-objective search algorithm: Four algorithms (NSGA-II, IBEA, SPEA2, PESA-II)
- Objective functions: A total of six fitness functions combinations (c1 to c6)

Subsequently, a total of $6 \times 4 \times 6 = 144$ experimental scenarios were derived for the first application domain and $4 \times 4 \times 6 = 96$ for the second application domain. For all these scenarios, we applied six seeding strategies and the non-seeded approach, which was the baseline of our study.

As suggested by Arcuri and Briand [3], for each of the experimental scenarios, the algorithm along with the seeding strategy was run 50 times to account for random variations.

For each experimental scenario, we first executed the Kruskal-Wallis statistical test, which performed a one-factor many-levels test. We assessed whether there was statistical differences among the selected seeding strategies. If we found there were statistical differences (i.e., p-value ≤ 0.05), we used a post-hoc analysis for assessing the statistical difference between each of the seeding strategy pairs. For the post-hoc analysis , we used the Holm's method in order to avoid risks with alpha inflation by applying corrections to the p-values. Furthermore, to assess the difference between the seeding strategies, we employed the Vargha and Delaney $\hat{A}_{12}$ value. The $\hat{A}_{12}$ value measures the probability that running the seeding strategy A yields better performance than running the seeding strategy B. Thus, we used this to classify which of the seeding strategies were better when there was statistical significance.

5.5 Algorithms setup

We implemented our seeding algorithms on top of PlatEMO [93], an open-source multi-objective optimization platform that implements over 100 multi and many-objective search algorithms.

For the selected algorithms, 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 chose these parameter values based on other studies related to multi-objective test case selection [7, 8, 98] as well as guidelines [3]. As in our previous work [7, 8], for the NSGA-II selection operator, we used the binary tournament selection operator [16, 32]. The remaining algorithms' configurations parameters were the default ones provided in PlatEMO [93].

As for the seeding strategies, the number of candidate sets in the ARPG strategy was set to 10 based on the original paper [25]. In the case of the static seeds, we experimented with two instance configurations, where we set one of the algorithm's configuration for a desired test suite size of 30% (coined as Static30), and the other one of 70% (coined as Static70). 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).

6 ANALYSIS OF THE RESULTS AND DISCUSSION

To keep the paper at a reasonable size, we generated four tables summarizing the statistical results. On the one hand, Tables 5 and 6 report the summary of the statistical tests for the first application domain both for the HV metric as well as for the average convergence. On the other hand, Tables 7 and 8 report the summary of the statistical tests for the second domain. For each experimental scenario, we first executed the Kruskal-Wallis test for a one-factor many-levels test, which assessed whether there was statistical differences among the selected seeding strategies. If we found there were statistical differences (i.e., p-value ≤ 0.05), we used a post-hoc analysis. This post-hoc analysis was based on the Holm's method to avoid risks with alpha inflation by applying corrections to the p-values.

For each case study in each of the application contexts, 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 experimental scenarios where the seeding strategy in column "Strategy A" outperformed the seeding strategy in column "Strategy B" with statistical significance. This meant that $\hat{A}_{12} > 0.5$ and the hypothesis was rejected based on the Holm's method, for either of the metrics (i.e., HV or Avg. Convergence).[3] The second number (i.e., the one in the middle) represents the number of experimental scenarios 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 the hypothesis was rejected based on the Holm's method). The third number (i.e., the one in the right) represents the number of experimental scenarios where there was no statistical significance between both strategies (i.e., it failed to reject the hypothesis, or the hypothesis was accepted based on the Holm's method). The column "Total" represents a summary of all the results obtained for each of the case studies.

Besides, we analyzed the difference existing between two different seeding strategies by categorizing their difference based on the $\hat{A}_{12}$ values, as suggested by Romano et al. [85]. We categorized the difference existing between the "Strategy A" and the "Strategy B" as *negligible* if $d < 0.147$, as *small* if $d < 0.33$, as *medium* if $d < 0.474$ and as *large* if $d >= 0.474$, where $d = 2|\hat{A}_{12} - 0.5|$. These values were obtained based on the study by Romano et al. [85]. These results are summarized in Tables 12, 13, 14 and 15. For each case study, seven values are shown (i.e., A++, A+, A, =, B, B+ and B++), divided by a slash (i.e., /). The first value (A++) means the number of experimental scenarios

[3]Note that the higher the value of these metrics, better the performance of the seeding strategy

that the difference between Strategy A and B was large, in favour of strategy A. The second value (A+) means the number of experimental scenarios that the difference between Strategy A and B was medium, in favour of strategy A. The third value (A) means the number of experimental scenarios that the difference between Strategy A and B was small, in favour of strategy A. The fourth value (=) means the number of experimental scenarios that the difference between Strategy A and B was negligible. The fifth value (B) means the number of experimental scenarios that the difference between Strategy A and B was small, in favour of strategy B. The sixth value (B+) means the number of experimental scenarios that the difference between Strategy A and B was medium, in favour of strategy B. Lastly, the seventh value (B++) means the number of experimental scenarios that the difference between Strategy A and B was large, in favour of strategy B.

Besides, a higher level of details of the obtained results (e.g., statistical tests results, such as specific results of the $\hat{A}_{12}$ values between all the seeding strategies for all the experimental scenarios, boxplots and other figures) is available in the following webpage: https://sites.google.com/alumni.mondragon.edu/tosem-some-seeds-are-strong

Table 5. Summary of the performed statistical tests for the Hypervolume metric within the first application context

		CW	EMB	CC	Tiny	TwoTanks	ACEngine	Total
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non Seeded	Dynamic	1/22/1	0/24/0	0/22/2	0/24/0	0/23/1	2/7/15	3/122/19
	Static30	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	2/10/12	2/129/13
	Static70	23/1/0	24/0/0	24/0/0	24/0/0	24/0/0	17/0/7	136/1/7
	ARPG	0/3/21	1/0/23	1/1/22	0/1/23	0/2/22	0/0/24	2/7/135
	Orthogonal	5/1/18	16/0/8	5/0/19	13/0/11	13/0/11	0/0/24	52/1/81
	DynamicARPG	1/22/1	0/24/0	0/22/2	0/24/0	0/24/0	2/8/14	3/124/17
Dynamic	Static30	16/2/6	19/2/3	16/3/5	21/0/3	20/3/1	0/4/20	92/14/38
	Static70	23/1/0	24/0/0	24/0/0	24/0/0	24/0/0	17/2/5	136/3/5
	ARPG	22/1/1	24/0/0	22/0/2	24/0/0	22/0/2	6/2/16	120/3/21
	Orthogonal	22/1/1	24/0/0	22/0/2	24/0/0	23/0/1	6/3/15	121/4/19
	DynamicARPG	1/0/23	0/2/22	0/0/24	0/1/23	1/1/22	0/0/24	2/4/138
Static30	Static70	23/1/0	24/0/0	24/0/0	24/0/0	24/0/0	22/1/1	141/2/1
	ARPG	23/1/0	24/0/0	24/0/0	24/0/0	24/0/0	10/2/12	129/3/12
	Orthogonal	22/1/1	24/0/0	24/0/0	24/0/0	24/0/0	7/2/15	125/3/16
	DynamicARPG	1/16/7	1/18/5	3/17/4	2/22/0	3/20/1	3/0/21	13/93/38
Static70	ARPG	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	0/18/6	0/137/5
	Orthogonal	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	0/18/6	0/137/5
	DynamicARPG	1/23/0	0/24/0	0/24/0	0/24/0	0/24/0	2/17/5	3/136/5
ARPG	Orthogonal	10/0/14	17/0/7	4/0/20	17/0/7	11/0/13	0/0/24	59/0/85
	DynamicARPG	1/22/1	0/24/0	0/22/2	0/24/0	0/24/0	2/7/15	3/123/18
Orthogonal	DynamicARPG	1/22/1	0/24/0	0/22/2	0/24/0	0/24/0	3/6/15	4/122/18

6.1 RQ 1 – Comparison with the baseline

In general, both Dynamic and Dynamic ARPG seeding strategies outperformed the non-seeded approach in most cases for both application domains. Specifically, when considering the modified HV metric, both seeding strategies outperformed with statistical significance the non-seeded approach in most of the cases. The Dynamic seeding strategy outperformed it in 85% of experimental scenarios, whereas the Dynamic ARPG for 85.8% of the experimental scenarios (Tables 5 and 7). When considering the $\hat{A}_{12}$ values and the classification performed by Romano et al. [85], the differences were large for 80% of the experimental scenarios for the Dynamic seeding strategy and for 79.5% for the Dynamic ARPG as compared with the non-seeded approach. When considering the convergence metric, the results were even stronger. Both strategies outperformed with statistical significance the non-seeded approach in 96.25% of experimental scenarios (Tables 6 and 8). This difference was large for 95.8% of the experimental scenarios according to the classification performed

Table 6. Summary of the performed statistical tests for the Average Convergence metric within the first application context

		CW	EMB	CC	Tiny	TwoTanks	ACEngine	Total
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non Seeded	Dynamic	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	2/22/0	2/141/1
	Static30	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/144/0
	Static70	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	144/0/0
	ARPG	2/0/22	0/0/24	1/0/23	1/1/22	1/2/21	1/0/23	6/3/135
	Orthogonal	21/0/3	22/0/2	16/0/8	21/0/3	21/0/3	0/2/22	101/2/41
	DynamicARPG	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	2/22/0	2/141/1
Dynamic	Static30	22/2/0	22/1/1	21/0/3	22/0/2	20/0/4	4/3/17	111/6/27
	Static70	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	144/0/0
	ARPG	23/1/0	24/0/0	24/0/0	24/0/0	24/0/0	22/2/0	141/3/0
	Orthogonal	23/0/1	24/0/0	24/0/0	24/0/0	24/0/0	22/2/0	141/2/1
	DynamicARPG	0/0/24	0/1/23	0/0/24	0/0/24	0/1/23	0/0/24	0/2/142
Static30	Static70	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	144/0/0
	ARPG	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	144/0/0
	Orthogonal	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	144/0/0
	DynamicARPG	1/21/2	0/22/2	0/21/3	0/22/2	0/22/2	3/5/16	4/113/27
Static70	ARPG	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/144/0
	Orthogonal	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/144/0
	DynamicARPG	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	0/144/0
ARPG	Orthogonal	22/0/2	23/0/1	16/0/8	20/0/4	20/0/4	0/1/23	101/1/42
	DynamicARPG	0/23/1	0/24/0	0/24/0	0/24/0	0/24/0	2/22/0	2/141/1
Orthogonal	DynamicARPG	0/24/0	0/24/0	0/24/0	0/24/0	0/24/0	2/21/1	2/141/1

Table 7. Summary of the performed statistical tests for the Hypervolume metric within the second application context

		Google	IOF	Paint	Rails	Total
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non Seeded	Dynamic	4/20/0	0/24/0	0/22/2	1/16/7	5/82/9
	Static30	7/0/17	15/0/9	1/11/12	9/6/9	32/17/47
	Static70	16/4/4	5/0/19	17/0/7	6/3/15	44/7/45
	ARPG	0/1/23	0/1/23	0/0/24	0/0/24	0/2/94
	Orthogonal	3/5/16	1/1/22	2/0/22	4/0/20	10/6/80
	DynamicARPG	4/20/0	0/24/0	0/22/2	1/16/7	5/82/9
Dynamic	Static30	20/0/4	24/0/0	20/0/4	21/1/1	85/1/10
	Static70	20/4/0	24/0/0	23/0/1	27/1/6	84/5/7
	ARPG	20/4/0	24/0/0	22/0/2	28/0/6	84/4/8
	Orthogonal	20/1/3	24/0/0	22/0/2	19/1/4	85/2/9
	DynamicARPG	0/0/24	0/0/24	0/0/24	0/0/24	0/0/96
Static30	Static70	10/4/10	0/5/19	20/1/3	4/13/5	34/23/37
	ARPG	2/9/13	0/16/8	8/1/15	9/8/7	19/34/43
	Orthogonal	1/10/13	0/14/10	15/1/8	6/7/11	22/32/42
	DynamicARPG	0/20/4	0/24/0	0/20/4	0/22/2	0/86/10
Static70	ARPG	4/16/4	0/7/17	0/18/6	4/3/17	8/44/44
	Orthogonal	4/19/1	0/9/15	0/12/12	9/4/11	13/44/39
	DynamicARPG	4/20/0	0/24/0	0/23/1	1/17/6	5/84/7
ARPG	Orthogonal	4/4/16	2/1/21	2/0/22	5/0/19	13/5/78
	DynamicARPG	4/20/0	0/24/0	0/22/2	0/17/7	4/83/9
Orthogonal	DynamicARPG	1/20/3	0/24/0	0/22/2	1/19/4	2/85/9

by Romano et al. [85]. This means that the proposed seeding strategies help on obtaining a faster convergence, which can be further observed in Figures 6 and 7.[4] All these results suggest that both of these seeding strategies are appropriate and outperform the non-seeded approach when dealing with the test case selection problem.

Results differed when considering other seeding strategies against the non-seeded approach. For instance, the static seed configured to build test suites of the 30% outperformed the non-seeded

[4]More figures of this type can be found in the website: https://sites.google.com/alumni.mondragon.edu/tosem-some-seeds-are-strong

Table 8. Summary of the performed statistical tests for the Average Convergence metric within the second application context

		Google	IOF	Paint	Rails	Total
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non Seeded	Dynamic	3/20/1	0/24/0	0/24/0	0/22/2	3/90/3
	Static30	12/1/11	17/0/7	18/1/5	17/0/7	64/2/30
	Static70	18/4/2	20/0/4	22/0/2	4/3/17	64/7/25
	ARPG	0/1/23	0/1/23	0/0/24	1/0/23	1/2/93
	Orthogonal	12/1/11	6/1/18	24/0/0	8/0/16	50/1/45
	DynamicARPG	3/20/1	0/24/0	0/24/0	0/22/2	3/90/3
Dynamic	Static30	21/0/3	24/0/0	24/0/0	24/0/0	93/0/3
	Static70	20/4/0	24/0/0	24/0/0	22/1/1	90/5/1
	ARPG	20/2/2	24/0/0	24/0/0	23/0/1	91/2/3
	Orthogonal	20/0/4	24/0/0	24/0/0	22/0/2	90/0/6
	DynamicARPG	0/0/24	0/0/24	0/0/24	0/0/24	0/0/96
Static30	Static70	14/4/6	1/4/19	17/5/2	1/18/5	33/31/32
	ARPG	1/12/11	0/18/6	1/17/6	0/18/6	2/65/29
	Orthogonal	6/8/10	0/11/13	8/6/10	0/11/13	14/36/46
	DynamicARPG	0/21/3	0/24/0	0/24/0	0/24/0	0/93/3
Static70	ARPG	3/19/2	0/19/5	0/22/2	4/2/18	7/62/27
	Orthogonal	4/11/8	0/12/12	2/14/8	12/0/12	18/38/40
	DynamicARPG	4/20/0	0/24/0	0/24/0	0/22/2	4/90/2
ARPG	Orthogonal	12/3/9	6/1/17	23/0/1	9/0/15	50/4/42
	DynamicARPG	2/20/2	0/24/0	0/24/0	0/23/1	2/91/3
Orthogonal	DynamicARPG	0/20/4	0/24/0	0/24/0	0/22/2	0/90/6

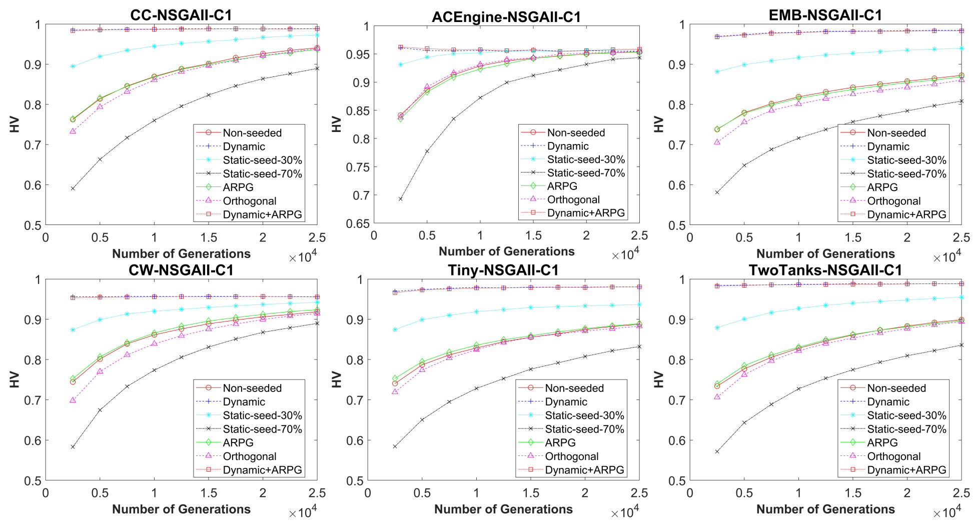


Fig. 6. Median HV obtained for the 50 algorithm runs at every 500 generations for the NSGA-II algorithm configured with the C1 fitness combination within the first application domain

approach in the first application domain when considering both the modified HV metric (Table 5) as well as the convergence metric (Table 6). However, in the second application domain, the non-seeded approach performed slightly better than this strategy, suggesting that this strategy might only be applicable in some application domains, but not in others. This could be due to the fact that in the first application domain, where simulation models of CPSs are involved, the test cases take a long time to execute, whereas in the second domain, the tests take less time.

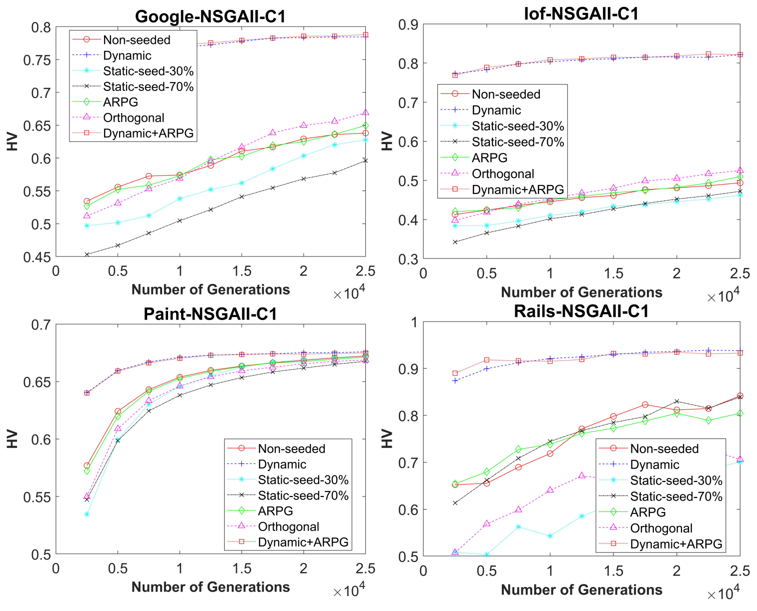


Fig. 7. Median HV obtained for the 50 algorithm runs at every 500 generations for the NSGA-II algorithm configured with the C1 fitness combination within the second application domain

Subsequently, in those cases where the time may be a relevant factor, the static seed configured to build test suites of the 30% (or lower) could be an appropriate technique.

The ARPG strategy performed similarly to the non-seeded approach. On the one hand, in 95.4% of all the experimental scenarios there was no statistical significance when considering the HV metric. On the other hand, in 95% of the cases, there was no statistical significance when considering the average convergence metric. This means that the proposed ARPG strategy did not have any impact in the search algorithm, which might be due to three reasons. First, having diversity in the population in terms of selected and not selected test cases might not have an impact when selecting test cases. Second, the selected distance function (i.e., Hamming distance) might not be appropriate for this context. Third, the number of candidate solutions, which was configured to have 10, did not produce sufficiently distinct populations, and this number might need to be increased.

Unexpectedly, the orthogonal population strategy performed worse than the non-seeded approach. While the approach performed similarly when considering the HV metric, where there was no statistical significance between the strategy and the baseline for 67% of the scenarios, when considering the Average Convergence, the baseline approach outperformed the orthogonal seed in 63% of the experimental scenarios with statistical significance. This strategy, which was originally proposed by Panichela et al. [75], aims at promoting diversity during the search process. Probably, in order this technique to be effective, it requires additional mechanisms that inject diversity during the search, as proposed in the original work [75]. Another technique that performed worse than the baseline was the static seed configured to select around 70% of the test cases. For most experimental scenarios, the baseline outperformed this technique, which means that the number of tests selected in the initial population by this technique might be too large.

In any application of search algorithms, the appropriate selection of the fitness function is fundamental. In our previous study [8], where different fitness function combinations were analyzed for the context of search-based test case selection of simulation models, we noticed that some fitness function combinations were stronger than others. For instance, we found that those fitness functions that had some anti-patterns as effectiveness measures were stronger than those that were based on test case similarity. Upon a closer look, we noticed that the cases where the non-seeded approach outperformed the Dynamic and Dynamic ARPG strategies could be the result of the fitness function not being appropriate enough. For instance, this was the case of the C3 fitness function combination in the Google case study, which combines the TSLE fitness function as effectiveness metric with the test execution time. As can be seen in Figure 8, in such case, the only seeding strategy that converges towards better HV values is the Static70, while the remaining techniques' performance degrade over the search. In a similar line, some search algorithms, in combination with some fitness function combinations and the proposed seeding strategies that performed best (i.e., Dynamic and Dynamic ARPG) might not be appropriate to solve the test case selection problem. For instance, this was the case in the Rails case study when using the IBEA algorithm and the C3 combination, or in the case of the ACEngine case study when using the IBEA algorithm alongside the C4 combination (Figure 9).

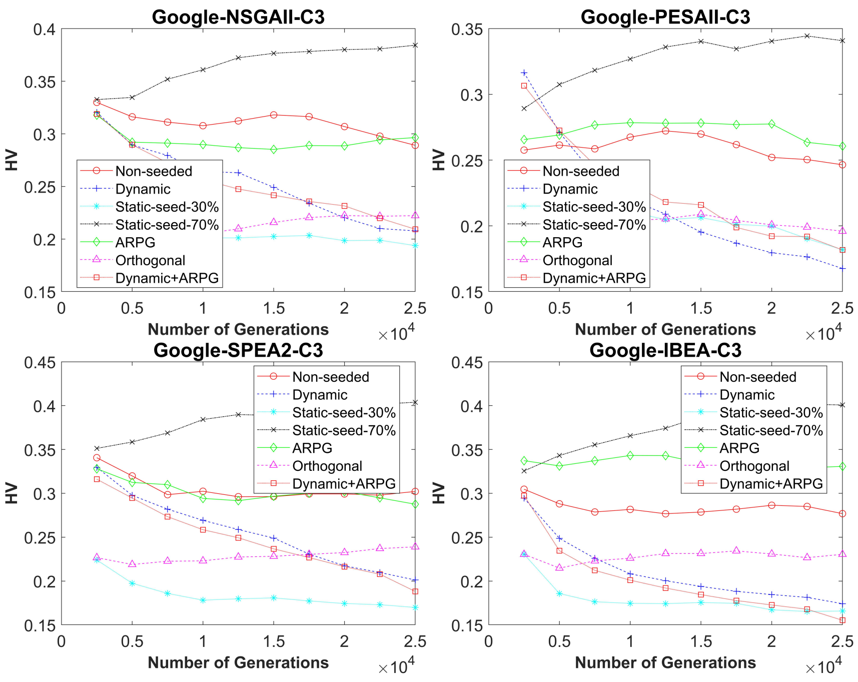


Fig. 8. Median HV obtained for the 50 algorithm runs at every 500 generations of the fitness function combination C3 along with the Google case study, where the seeding strategies had negative effects due to the fitness function not working correctly

The combination of some seeding strategies, search algorithm and fitness function was not particularly well suited. For instance, this was the case of the PESA-II algorithm with the Dynamic and DynamicARPG seeding strategies along with the C4 fitness combination in the first application domain, as can be seen in Figure 10. In this case, C4 combination combines the TET with the input similarity. While the input similarity showed a low performance when compared to other anti-patterns (e.g., discontinuity) [8], in our experiments with the PESA-II algorithm it showed

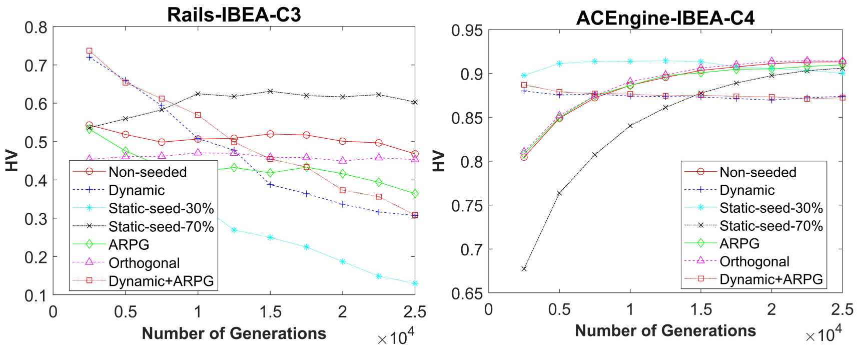


Fig. 9. Median HV obtained for the 50 algorithm runs at every 500 generations of two cases where the seeding strategies had a negative effect

how the HV degraded over time. Conversely, the other algorithms (e.g., NSGA-II) along with the C4 combination were able to maintain or improve the overall HV indicator. Subsequently, it can be seen that in some cases the selection of the algorithm is as important as the selection of an appropriate fitness function.

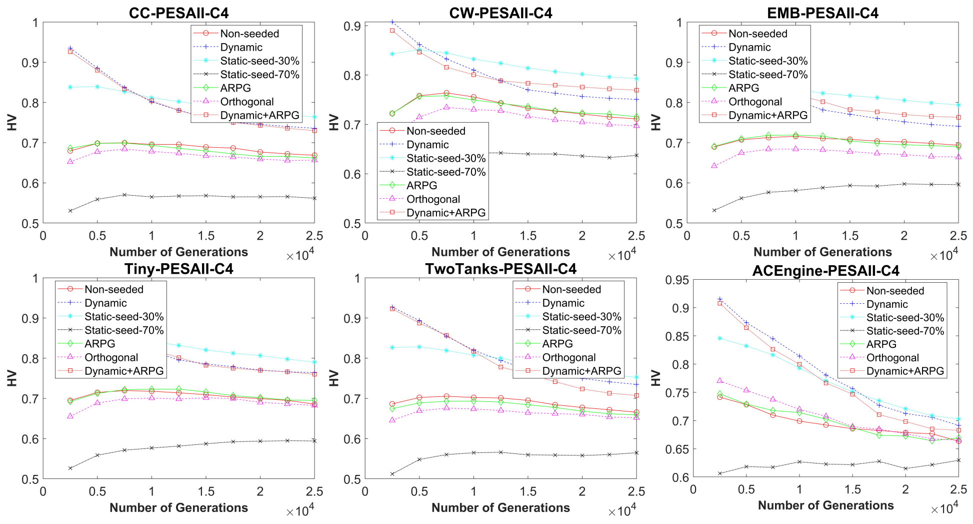


Fig. 10. Median HV obtained for the 50 algorithm runs at every 500 generations of the PESA-II algorithm in the first application domain with the C4 fitness function combinations. As can be seen, the HV indicator degrades as the search advances in number of generations.

Table 9 reports which were the combinations that for the 50 independent runs obtained highest average HV values. As can be seen, in all of them, NSGA-II was the algorithm providing the highest HV. Interestingly, for the fitness function combination, in the first application context, the combination of Discontinuity and TET objectives were the ones performing best in all the case studies, which is in line with the results obtained in our previous work [8]. In the second application context, the combination of FDC and TET objectives where the ones performing best in terms of the highest average HV indicator.

Table 9. Summary of the combinations showing highest average HV indicator

Case Study	**Algorithm**	**Fitness configuration**	**Seeding Strategy**
ACEngine	NSGA-II	C1	DynamicARPG
CC	NSGA-II	C1	DynamicARPG
CW	NSGA-II	C1	Dynamic
EMB	NSGA-II	C1	Dynamic
Tiny	NSGA-II	C1	Dynamic
TwoTanks	NSGA-II	C1	Dynamic
Google	NSGA-II	C1	DynamicARPG
IOF	NSGA-II	C1	DynamicARPG
Paint	NSGA-II	C1	Dynamic
Rails	NSGA-II	C1	Dynamic

For these cases, we analyzed the Pareto-frontier returned by the search algorithms, and took the ones with highest mutation score (for the first application context) and the highest failure detection ratio (for the second application context), and analyzed their TET. In the first application context, both the non-seeded approach and the one with the seeding strategy obtained the maximum mutation score, except for some outliers in the case study of the CW (for the case of the non-seeded approach). Figure 11 shows the distribution of the TET obtained for each of the case studies by the seeding strategy obtaining highest HV and the non-seeded approach. As can be seen, in the most complex case studies in terms of number of blocks (i.e., EMB and TwoTanks), the seeding strategy helped significantly in the reduction of the TET. For the TwoTanks case study, the test suites obtained with the Dynamic seed took between 2 and 6% of the original TET to execute, whereas for the non-seeded approach, the test suites took between 6 and 13% of the original TET. This means that in this specific case study, the Dynamic seed could take less than half of the time taken by the test suites provided non-seeded approach while detecting the same amount of mutants. In the case of the EMB, the median value of the distribution for the seeded approach was around 7.5% of the TET of the original test suite, whereas the median value for the non-seeded approach was around 13%; thus, the reduction in the overall TET was quite significant in this case too. In the remaining four case studies, the TET values were similar between the seeded and non-seeded approaches. In the Tiny and ACEngine case studies, the seeded approaches showed a lower TET median value, but the distributions were slightly larger. In the CC case study, the median was similar, but the distribution for the seeded approach was larger. Finally, the median TET value of the CW case study was slightly lower, in addition to the overall distribution. However, in this specific case, the distribution of the TET values were large in both cases. This might be due to a larger amount of mutants being used. When having a closer look at the results, we noticed that for obtaining the modified HV values, the number of non-dominated solutions in the seeded approaches were much higher than those in the non-seeded approach. This means that in practice, where decision makers are needed to select one specific solution from the Pareto-frontier returned by the algorithm (e.g., based on some time budget), when using the seeded approaches, the probabilities for selecting a better solution is higher.

Results where quite different in the second application domain. This could be because in these case studies, instead of measuring the percentage of faults detected, we focused on failures.[5] As can be seen in Figure 12, the seeding strategies favour a high failure detection ratio, selecting in most of the solutions at least one solution that detects all the failures that are detectable by the

[5]Notice that the dataset do not have access to the faults

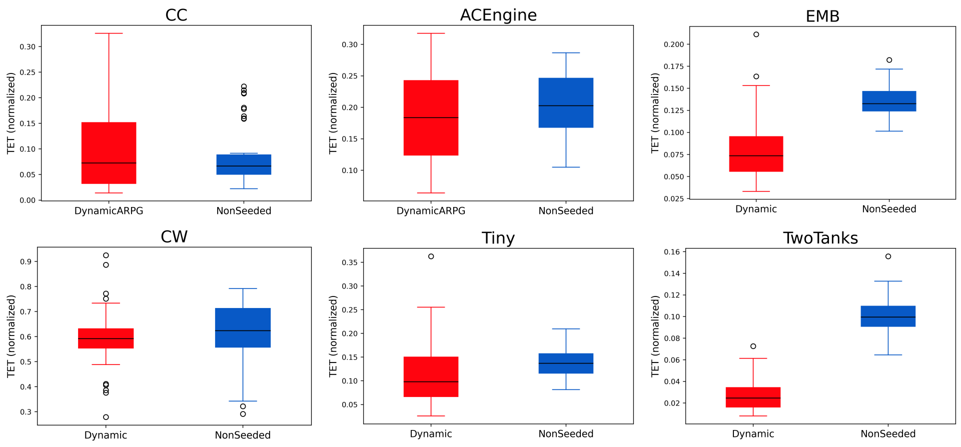


Fig. 11. Test Execution Time taken to obtain maximum mutation score of individual solutions for the configuration from Table 9. In most of these cases, the mutation score was 100%, except for some outliers in the case study of the CW in the case of the non-seeded approach

test suite. Instead, the non-seeded approach did not detect all failures in most of the cases. This has a direct impact on the overall TET of the approaches, as can be appreciated in Figure 13. It can be concluded that the seeded approaches favour the fault detection ratio, while the non-seeded approach favours cost reduction. When having a closer look, a similar pattern to that from the first domain was observed. The number of solutions in the derived Pareto frontier was longer for the case of the seeded approach when compared with the non-seeded approach. This means that the decision maker has more choices and better chances of selecting an adequate solution in the case of the seeded approach.

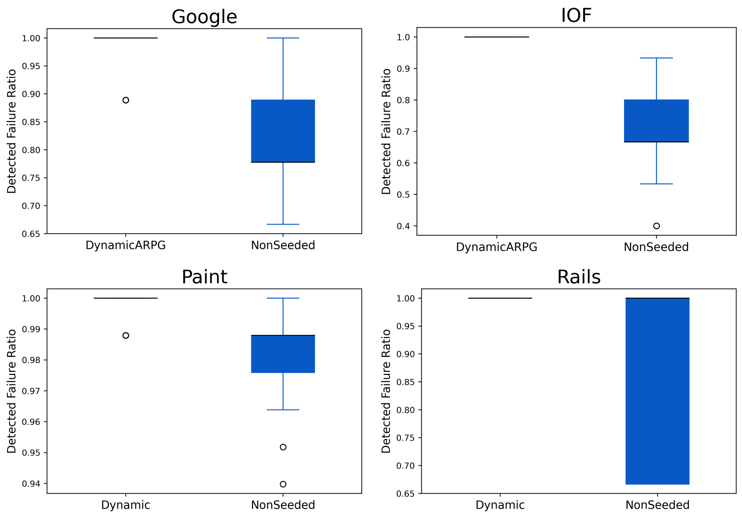


Fig. 12. Maximum percentage of failures detected by individual solutions for the configuration from Table 9

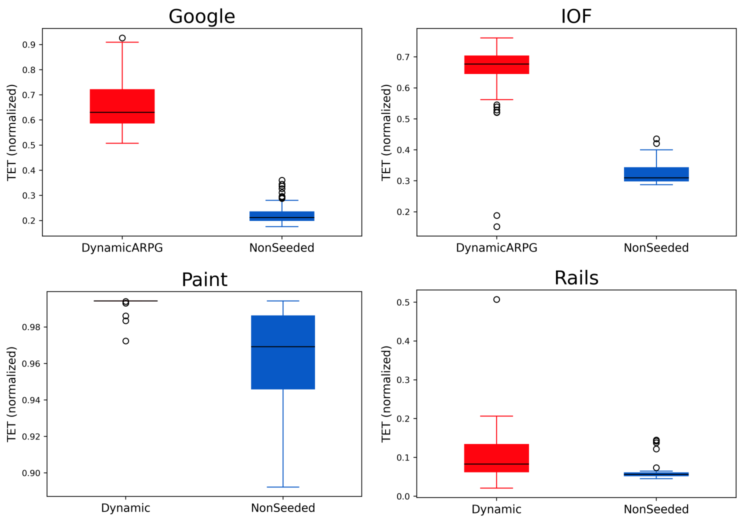


Fig. 13. Maximum Test Execution Time (TET) by individual solutions to obtain the maximum failure ratio for the configuration from Table 9

For both application domains, it could be appreciated that both the Dynamic and the DynamicARPG seeding strategies outperformed the non-seeded strategy. Additionally, in the first application domain, the Static30 also outperformed the non-seeded strategy. This means that from the practical perspective, exceptions aside, the use of some of the proposed strategies can be beneficial for the test case selection problem. However, when analyzing individual solutions with highest mutation scores and highest failure rates, we saw that the execution time of the test cases is not always beneficial for the seeded approach. While there were cases where the TET was reduced down to the 50% when compared with the non-seeded approach while maintaining the mutation score (e.g., EMB and TwoTanks case studies), this analysis suggest that effective decision makers are required when selecting solutions from the Pareto-frontier, which remains for future work. We therefore recommend the use of either the Dynamic or the DynamicARPG strategies for the test case selection problem. Subsequently, the first RQ can be answered as follows:

The Dynamic and DynamicARPG seeding strategies outperformed the non-seeded approach with statistical significance for around 85% of the experimental scenarios when considering the modified HV metric and for around 96% of the experimental scenarios when considering the Average Convergence.

6.2 RQ 2 – Best seeding strategy

In the previous RQ, three of the selected strategies outperformed the baseline (non-seeded approach) with statistical significance for most of the experimental scenarios when considering the first application domain: Dynamic, Static30 and DynamicARPG. Conversely, the Static30 seeding strategy did not outperform the baseline technique in the second application context, while both Dynamic and DynamicARPG strategies did. When considering the first application scenario, both the Dynamic and DynamicARPG outperformed the Static30 strategy with statistical significance for most of the experimental scenarios too. Specifically, when considering the modified HV metric, the Dynamic seed outperformed the Static30 technique with statistical significance for 63.8% of the experimental scenarios, whereas the DynamicARPG for 64.5% of the experimental scenarios. These results were even stronger when considering the Average Convergence metric (i.e., 77% of the experimental scenarios for the Dynamic seed and 78% of the cases for the DynamicARPG). This means that even in an application domain where the Static30 technique outperformed the baseline, both the Dynamic and DynamicARPG strategies are better. As for the second application domain, the Dynamic strategy outperformed this seeding strategy in 88% of the scenarios when considering the HV metric, and in 96.9% of the scenarios when considering the Average Convergence metric. Similarly, the DynamicARPG outperformed the Static30 seed in 89.6% of the experimental scenarios when considering the modified HV metric and in 96.9% of the scenarios when considering the Average Convergence metric. Overall, all this means that both the Dynamic and DynamicARPG techniques are stronger than the Static30 seed.

As expected from the results of RQ1, both the Dynamic and DynamicARPG performed better than the Static70, ARPG and Orthogonal seeding strategies for most of the cases. There were only some few experimental scenarios where these three approaches outperformed the Dynamic and DynamicARPG techniques with statistical significance. This could be because the combination between some search algorithm with some fitness function combinations might not be appropriate for test case selection. For instance, this was the case in the Rails case study when using the IBEA algorithm and the C3 combination, or in the case of the ACEngine case study when using the IBEA algorithm alongside the C4 combination. These two examples are shown in figure 9, and have been discussed in the previous RQ.

When comparing the Dynamic and the Dynamic ARPG strategies, results were generally indistinguishable. On the one hand, when considering the modified HV metric, there was no statistical significance in 97.5% of the experimental scenarios. On the other hand, when considering the Average Convergence, there was no statistical significance in 99.2% of the experimental scenarios. It is noteworthy that in the second application domain, there was no statistical significance in any of the experimental scenarios. All this means that the cost-effectiveness of both techniques is mostly the same.

From the practical perspective, generally both the Dynamic seed and the DynamicARPG techniques outperformed the remaining seeding strategies. When compared among them, the results were in indistinguishable. Nonetheless, the DynamicARPG seed combines both, the ARPG seed and the dynamic seed strategies. While the DynamicARPG seed is competitive when compared with the non-seeded approach, the fact that the ARPG performed close to the non-seeded approach (see discussion on RQ1) suggests that it is the varying test suite size factor that makes this algorithm perform better than the non-seeded approach. Furthermore, computationally, the Dynamic seed is faster than the DynamicARPG, although both techniques are fast enough to be applicable in practice. Having this discussed, we can answer the second RQ as follows:

The Dynamic and DynamicARPG seeding strategies outperformed the remaining seeding strategies in at least 73.75% of the experimental scenarios when considering the modified HV metric and in 85% of the experimental scenarios when considering the Average Convergence metric. Despite not having significant differences between themselves, the Dynamic seed is recommended for use by practitioners dealing with the multi-objective test case selection problem.

6.3 RQ 3 – Algorithms Running Times and Overhead

The third RQ aims at analyzing the effects of the seeding strategies in the execution time of the search algorithms. To this end, in each run, we measured the time required by the algorithms to execute and compared it with the non-seeded approach. We run the same statistical tests as in the cases of RQ1 and RQ2. Table 10 summarizes the results of the statistical tests for the first application domain whereas Table 11 does the same for the second application domain.

Table 10. Summary of the performed statistical tests for the execution time within the first application context

		ACEngine	CC	CW	EMB	Tiny	TwoTanks	TOTAL
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non seeded	Dynamic	11/5/8	11/8/5	14/1/9	7/7/10	24/0/0	23/0/1	90/21/33
	Static30	0/24/0	0/24/0	0/24/0	0/22/2	0/20/4	15/8/1	15/122/7
	Static70	24/0/0	24/0/0	24/0/0	24/0/0	24/0/0	23/0/1	143/0/1
	ARPG	20/2/2	6/1/17	5/4/15	1/13/10	16/0/8	20/0/4	68/20/56
	Orthogonal	7/4/13	3/8/13	1/11/12	8/0/16	12/1/11	17/0/7	48/24/72
	DynamicARPG	17/1/6	21/1/2	17/0/7	11/3/10	24/0/0	23/0/1	113/5/26

Table 11. Summary of the performed statistical tests for the execution time within the second application context

		Google	IoF	Paint	Rails	Total
Seeding Strategy A	**Seeding Strategy B**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**	**A/B/=**
Non seeded	Dynamic	8/16/0	9/15/0	5/12/7	11/12/1	33/55/8
	Static30	0/24/0	0/24/0	0/21/3	0/24/0	0/93/3
	Static70	24/0/0	24/0/0	23/0/1	24/0/0	95/0/1
	ARPG	24/0/0	24/0/0	22/0/2	23/0/1	93/0/3
	Orthogonal	24/0/0	11/9/4	2/0/22	1/20/3	38/29/29
	DynamicARPG	8/16/0	17/7/0	20/1/3	15/7/2	60/31/5

As can be seen, there was statistical significance between some of the seeding strategies and the non-seeding approach. For instance, the Static30 seed outperformed the non-seeding approach, whereas Static70 was outperformed by the non-seeded approach. This suggests that the number of test cases that have been selected in the solution has an impact in the running time, probably because it takes more time to compute the fitness of the solutions with a higher amount of test cases being selected.

From RQ1 and RQ2, we derived that the Dynamic and DynamicARPG seeding strategies were the ones performing best. In the case of Dynamic, the non-seeded approach seemed to perform better in the first application domain, outperforming it with statistical significance in 62.5% of the experimental scenarios; in such application domain, the Dynamic seed was faster in 14.6% of the experimental scenarios. Conversely, in the second application domain, the Dynamic seeding strategy outperformed the non-seeded approach in terms of running times in more experimental scenarios; specifically, for 57.3% of the experimental scenarios, the Dynamic seed was faster with statistical significance, whereas the non-seeded approach outperformed this strategy in 33.4% of the scenarios. In the case of the DynamicARPG seeding strategy, results were worse when compared against the non-seeded approach. In both application scenarios, the non-seeded approach was faster. Specifically, in the first application context, the non-seeded approach was faster with statistical significance in 78.5% of the scenarios, whereas in the second application context in 63.5% of them.

However, while there is statistical significance, when considering the actual running times between Dynamic and DynamicARPG against the non-seeded approach, it can be seen that the differences are not large. The largest difference in the mean time of the 50 runs for all the scenarios were 6.49 seconds between the DynamicARPG and the non-seeded approach (Google case study, SPEA2 algorithm, C4 fitness function combination). In addition, when considering the Dynamic seed, the maximum average running time difference with regard to the non-seeded approach was 2.65 seconds (AC Engine case study, NSGA-II algorithm, C2 fitness function combination). The average running time difference of the mean running times between the Dynamic and the non-seeded algorithm was 0.38 seconds. In the case of the DynamicARPG, the average running time difference of the mean running time (for the 50 runs) against the non-seeded algorithm was 0.8 seconds. Therefore, the overhead that these algorithms might provoke is not a reason that should prevent their usage in practice.[6]

Furthermore, from RQ1, we have seen that the proposed seeding strategies help the search algorithms converge faster. This means that if the execution time of algorithms is a critical factor when deployed a test selection algorithm in practice, the seeding strategies can be used together with a lower number of generations, requiring this way a lower running time of the algorithms. For instance, in Figure 6, it can be seen that the algorithms obtain the maximum HV value by the 500th generation, or even before. This means that by using 500 generations, the results will be similar to using the 2500. However, around an 80% of time could be saved when compared to the non-seeded approach. Therefore, this RQ can be answered as follows:

The proposed seeding strategies do not produce significant overhead in the running times of the algorithms. While there is statistical significance when comparing the running times (both, in favour and against the seeding algorithms), when considering the overall running times of the algorithms showing best performance in RQ1 and RQ2, the difference on average was not longer than 2.65 seconds between the Dynamic and the non-seeded algorithm, which is a time that can be assumed in practice. Furthermore, both the Dynamic and DynamicARPG seeds help the search algorithms converge faster, which could be combined with having a lower number of generations should the algorithms' running times is critical.

Summary of the results and practical implications

In our evaluation, we have aimed at assessing whether the proposed techniques are appropriate for the test case selection problem. To this end, we have carried out an empirical evaluation by using two application domains, four search algorithms and different fitness function combinations. We now summarize some of the insights found that need to be considered when applying a multi-objective test case selection algorithm in practice:

- Two of the seeding strategies (i.e., Dynamic and DynamicARPG) stood out over the rest. More importantly, the performance of these two techniques was better than the traditionally employed non-seeded approach. Due to its simplicity and lower running time, we recommend the use of the Dynamic seeding strategy.
- In terms of the algorithms' running times, we could see that these two seeding strategies do not provoke significant overhead in the algorithms' running times when compared against the non-seeded approach. The small overhead provoked by the seeding strategies in some of the cases, does not prevent our techniques from being applicable in practice.
- These two seeding strategies converge fast towards optimal solutions. If the running times of the algorithms is an issue when applying test selection approaches in practice, the number of generations can be reduced when using the proposed seeding strategies.

[6]All this data is available in: https://sites.google.com/alumni.mondragon.edu/tosem-some-seeds-are-strong/appendix-b/running-time

- The combination of some algorithms along with some fitness functions are not reliable. For instance, the PESA-II algorithm performed wrongly when using the C4 fitness function combination in the first application context. IBEA algorithm did not also perform as expected in the Rails case study with the C3 fitness function combination or the ACEngine C4 fitness function combination. Before applying a search algorithm in practice, it is highly important to detect cases like these ones.
- Some fitness function combinations do not work correctly in specific systems. For instance, the C3 fitness function combination did not perform well in the Google case study. Before applying any test case selection algorithm in practice, it is important to detect which are the appropriate fitness function combinations.
- NSGA-II has been the algorithm traditionally used for the multi-objective test case selection problem [8, 81, 95, 98]. While the study was not exhaustive because we focused on the assessment of the seeding strategies, we found that NSGA-II was the algorithm that provided highest average HV values. Furthermore, unlike the cases of PESA-II or IBEA, the NSGA-II found to be a reliable algorithm. Moreover, this algorithm has been extensively studied in the test case selection context. Therefore, we recommend the use of this algorithm along with one of the two proposed seeding strategies when applying test case selection in practice.
- The Dynamic and DynamicARPG seeding strategies ensure a high fault and failure detection ratio. Conversely, the non-seeded approaches did not always provide a solution that detected all the failures in the second application scenario.
- In practice, decision makers are required. We found that for measuring the modified HV, the Dynamic and DynamicARPG seeding approaches provided more non-dominated solutions than the non-seeded approach. Therefore, we believe that decision makers have higher probabilities of selecting better solutions when using one of such seeding approaches.
- While it was not the main objective of the study, in the first application domain the C1 fitness function combination was the one who obtained the highest average HV values. This fitness function combines the discontinuity anti-pattern with the test execution time. This is in-line with the findings of our previous study [8]. In the second application domain, the C1 fitness function combination was the one who obtained the highest average HV. This combination combined the Fault Detection Capability (FDC) with the Test Execution Time, which would suggest that this metric is more effective when selecting test cases than the TSLE or the TSLF.

7 THREATS TO VALIDITY

Internal validity: An internal validity threat in our study could be related to the generated mutants, which relates to the first application context. In Simulink models, employing mutation testing is extremely expensive because the physical layer encompasses complex mathematical equations. Subsequently, it was not feasible to generate a large set of mutants. To reduce this threat, we employed the same mutants generated in our previous studies [7, 8]. 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 [11–13, 43, 51, 55–57, 64–66]. Additionally, we removed duplicated mutants as recommended by Papadakis et al. [76] 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 algorithms considering related guidelines and works that included Pareto-based search algorithms for test case selection [98]. 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 [25]. 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. Another threat of our study could be related to the selection of the search algorithms. As in the original version paper [6], we used the NSGA-II due to its wide usage in the context of test case selection [8, 75, 98]. Furthermore, we included three additional search algorithms (i.e., IBEA, SPEA2 and PESA-II) due to their wide usage by the search community. The use of other algorithms could have another implication in the results of the algorithms, which means that we cannot generalize our finding to all multi-objective search algorithms. Nevertheless, we believe that the selected algorithms are a good representation of multi-objective search algorithms, and their implementation is available in different programming languages, thus, practitioners can easily use them.

External validity: As in any search-based software engineering evaluation, an external validity threat exists related to the generalization of results. We tried to mitigate this threat by using several case studies applied within two different application contexts. For the first application context, six case studies involving Simulink models of different sizes and characteristics were used. 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 [28]. Four of the case studies we used had from 235 to 498 blocks, which means that most of our case studies were larger than most public subject models. With regards to the second application contexts, four different case studies where used, three of which were industrial case studies and one a real-world open-source case study. The characteristics of these case studies were also different (e.g., the number of test cases ranged from 89 to 5,555). Another external validity threat is related to how the faults and failures are distributed, which might have an influence in the performance of the algorithms, as discovered by Zhu et al. [103]. As discussed in Section 5.3, in our case studies, both the fault and failure distribution is quite different from the different case studies. In addition to the application context and the case studies mentioned above, the performance of the proposed seeding strategies were evaluated in four multi-objective search algorithms.

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 random variations, as recommended in guidelines [3]. Additionally, we carefully analyzed the results by applying appropriate statistical tests. As a pre-hoc analysis, we used the Kruskal-Wallis test to assess whether there were statistical differences. Only in one scenario there was no statistical significance (i.e., AC Engine, scenario 13 (p-value = 0.186)). In most of the scenarios the p-values were far from the specified threshold (i.e., p-value ≤ 0.05) for the Kruskal-Wallis statistical test. As a post-hoc analysis, we used the Holm's method to assess the statistical significance between the different seeding strategies. This permits correcting the p-values based on the alpha inflation. Furthermore, to complement the Holm's method, to assess the difference between the seeding strategies, we employed the Vargha and Delaney $\hat{A}_{12}$ values. To assess this difference, we further used different cut-off measures based on the classification performed by Romano et al. [85], to assess whether the difference existing between the seeding strategies were negligible, small, medium or large.

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).

8 RELATED WORK

Test case selection. Test case selection has been widely studied in the current literature for different application areas (e.g., Java applications [44], software product lines [95], deep learning systems [59]). Yoo and Harman identified and analysed the positive and negative aspects of 12 different approaches based on an extensive analysis of the state-of-the-art [100]. Engströem et al. identified 28 techniques for regression test selection [35]. Besides evolutionary search-based approaches, other techniques have been proposed for test case selection, including multi-objective lineal programming techniques [97], greedy-based algorithms [33], or reinforcement learning [91]. Our approach is intended to support population-based search-based test case selection techniques. In addition, test case selection is performed by following certain criterion, such as, white-box criteria which aim at ensuring certain degree of code coverage [33, 70, 98], criteria based on historical data [34, 81, 83, 95] and black-box criteria (e.g., similarity-based test selection) [7, 8, 30, 46, 70]. These criteria are employed as fitness functions in the search-based test selection context, and typically combined with a cost function (e.g., estimated test execution time of the selected test cases). While in this paper we focused on two application domains, one using black-box test selection approaches (first application context) and the other one using historical data (second application context), the proposed seeding strategies do not consider such metrics and can therefore be used with any kind of technique.

Multi-objective test case selection. Multi-objective algorithms to solve the test case selection problem was first proposed by Yoo and Harman [98]. 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 [99]. 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 [41, 73], or compute-intensive CPSs [7, 8]. 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 [7, 8, 46, 50, 98]. 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 [81], or product lines [11, 94, 95]. 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 [75]. 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 [31]. 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 [31, 75].

Seeding strategies. 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 [39]. This study was further extended, confirming the positive impact of the seeding strategies in further subject programs [84]. Seeding was also applied on SAPIENZ [60], 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 [58], 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 [21, 22], software improvement [5] and software product line configuration [89]. 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.

9 CONCLUSION

In this paper we propose a set of seeding strategies for initializing the population of search-based test case selection algorithms. The proposed techniques are both domain and algorithm agnostic. In an empirical evaluation with four multi-objective search algorithms, two different application domains and several case studies in each of these application domains, we demonstrated that at least two of the seeding strategies (Dynamic and DynamicARPG) show significant improvement over the non-seeded approach. Specifically, these two seeding strategies provide a higher convergence towards optimal solutions, which might be beneficial in cases where the search algorithms take a long time to compute. Furthermore, the overall cost-effectiveness of two of the proposed seeding strategies was still higher than the non-seeded approach when using a common amount of fitness evaluations (i.e., those used in other search-based test case selection problems [7, 8, 98]). The proposed techniques do not pose any kind of disadvantages (e.g., higher computation time, development complexity), and subsequently, they are recommended to be used by practitioners that use multi-objective test case selection strategies for regression test optimization problems.

Replication package: For the sake of replicability, our replication package is available in the following link: https://doi.org/10.5281/zenodo.5795353

ACKNOWLEDGEMENTS

We would like to thank Jose Antonio Parejo, from University of Seville, for his help and feedback on the carried out statistical analysis. We would also like to thank the anonymous reviewers for their insightful comments and for helping us increase the quality of this paper. The authors are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1326-19), supported by the Department of Education, Universities and Research of the Basque Country.

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, Joseba Andoni Agirre, and Goiuria Sagardui. 2020. Seeding strategies for multi-objective test case selection: an application on simulation-based testing. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 1222–1231.

[7] 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

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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

[13] 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

[14] 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.

[15] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand. 2021. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering* (2021).

[16] 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.

[17] 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.

[18] 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

[19] J Brownlee. 2012. *Cleverl Algorithms: Nature-Inspired Programming Recipes*. lulu.com.

[20] 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*.

[21] 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.

[22] 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).

[23] 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.

[24] 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.

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

[26] Yiqun T Chen, Rahul Gopinath, Anita Tadakamalla, Michael D Ernst, Reid Holmes, Gordon Fraser, Paul Ammann, and René Just. 2020. Revisiting the Relationship Between Fault Detection, Test Adequacy Criteria, and Test Set Size. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 237–249.

[27] 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.

[28] 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.

[29] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates. 2001. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary

Computation. Morgan Kaufmann Publishers Inc., 283–290.

[30] Emilio Cruciani, Breno Miranda, Roberto Verdecchia, and Antonia Bertolino. 2019. Scalable approaches for test suite reduction. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 419–429.

[31] 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.

[32] 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.

[33] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, and Yvan Labiche. 2015. Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability* 25, 4 (2015), 371–396.

[34] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. ACM, 235–245.

[35] 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.

[36] 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.

[37] 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

[38] 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

[39] 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.

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

[41] 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.

[42] D Greer and G Ruhe. 2004. Software release planning: an evolutionary and iterative approach. *Information and Software Technology* 46, 4 (2004), 243 – 253.

[43] 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.

[44] Mary Jean Harrold, James A Jones, Tongyu Li, Donglin Liang, Alessandro Orso, Maikel Pennings, Saurabh Sinha, S Alexander Spoon, and Ashish Gujarathi. 2001. Regression test selection for Java software. In *ACM Sigplan Notices*, Vol. 36. ACM, 312–326.

[45] 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.

[46] 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.

[47] Christopher Henard, Mike Papadakis, and Yves Le Traon. 2014. Mutation-based generation of software product line test configurations. In *International Symposium on Search Based Software Engineering*. Springer, 92–106.

[48] 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.

[49] Eric Knauss, Miroslaw Staron, Wilhelm Meding, Ola Söder, Agneta Nilsson, and Magnus Castell. 2015. Supporting continuous integration by code-churn based test selection. In *Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering (RCoSE'15)*. IEEE Press, 19–25.

[50] 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.

[51] 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.

[52] 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

[53] 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.

[54] Jingjing Liang, Sebastian Elbaum, and Gregg Rothermel. 2018. Redefining prioritization: continuous prioritization for continuous integration. In *Proceedings of the 40th International Conference on Software Engineering*. 688–698.

[55] 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.

[56] 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)*.

[57] 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.

[58] 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.

[59] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–22.

[60] 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.

[61] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. 2013. Test case prioritization for continuous regression testing: An industrial case study. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM'13)*. IEEE Computer Society, 540–543.

[62] 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.

[63] 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

[64] 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.

[65] 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.

[66] 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

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

[68] 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)*.

[69] 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

[70] Debajyoti Mondal, Hadi Hemmati, and Stephane Durocher. 2015. Exploring test suite diversification and code coverage in multi-objective test case selection. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10.

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

[72] 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

[73] 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).*

[74] 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).

[75] 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.

[76] 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.

[77] P Victer Paul, N Moganarangan, S Sampath Kumar, R Raju, T Vengattaraman, and P Dhavachelvan. 2015. Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems. *Applied soft computing* 32 (2015), 383–402.

[78] P Victer Paul, A Ramalingam, Ramachandran Baskaran, P Dhavachelvan, K Vivekanandan, and R Subramanian. 2014. A new population seeding technique for permutation-coded Genetic Algorithm: Service transfer approach. *Journal of Computational Science* 5, 2 (2014), 277–297.

[79] P Victer Paul, A Ramalingam, R Baskaran, P Dhavachelvan, K Vivekanandan, R Subramanian, and VSK Venkatachalapathy. 2013. Performance analyses on population seeding techniques for genetic algorithms. *International Journal of Engineering and Technology (IJET)* 5, 3 (2013), 2993–3000.

[80] 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.

[81] 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

[82] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. 2018. REMAP: Using rule mining and multi-objective search for dynamic test case prioritization. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST).* IEEE, 46–57.

[83] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. 2019. Employing rule mining and multi-objective search for dynamic test case prioritization. *Journal of Systems and Software* 153 (2019), 86–104.

[84] 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.

[85] Jeanine Romano, Jeffrey Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohens d indices the most appropriate choices. In *Annual meeting of the Souther Association for Institutional Research.*

[86] Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia. 2000. Regression test selection for C++ software. *Software Testing, Verification and Reliability* 10, 2 (2000), 77–109.

[87] 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.

[88] Federica Sarro, Alessio Petrozziello, and Mark Harman. 2016. Multi-objective software effort estimation. In *Proceedings of the 38th International Conference on Software Engineering.* ACM, 619–630.

[89] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 465–474.

[90] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2018. Test Case Prioritization for Acceptance Testing of Cyber Physical Systems: A Multi-Objective Search-Based Approach. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18).*

[91] Helge Spieker, Arnaud Gotlieb, Dusica Marijan, and Morten Mossige. 2017. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 12–22.

[92] 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.

[93] Ye Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin. 2017. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. *IEEE Computational Intelligence Magazine* 12, 4 (2017), 73–87.

[94] 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.

[95] 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.

[96] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. 2014. Multi-objective Test Prioritization in Software Product Line Testing: An Industrial Case Study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC '14)*. ACM, New York, NY, USA, 32–41. https://doi.org/10.1145/2648511.2648515

[97] Yinxing Xue and Yan-Fu Li. 2020. Multi-objective Integer Programming Approaches for Solving the Multi-criteria Test-suite Minimization Problem: Towards Sound and Complete Solutions of a Particular Search-based Software-engineering Problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 3 (2020), 1–50.

[98] 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.

[99] 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.

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

[101] 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.

[102] Tingting Yu and Ting Wang. 2018. A study of regression test selection in continuous integration environments. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 135–143.

[103] Yuecai Zhu, Emad Shihab, and Peter C Rigby. 2018. Test re-prioritization in continuous testing environments. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 69–79.

[104] Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*. Springer, 832–842.

[105] Eckart Zitzler, Marco Laumanns, Lothar Thiele, et al. 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. In *Eurogen*, Vol. 3242. 95–100.

A SUMMARY OF THE RESULTS OF THE VARGHA AND DELANEY $\hat{A}_{12}$ VALUES BASED ON THE CLASSIFICATION OF LARGE (A++, B++), MEDIUM (A+, B+), SMALL (A, B) AND NEGLIGIBLE (=)

Table 12. Summary of the performed statistical tests for the Hypervolume metric within the first application context

		CW	EMB	CC	Tiny	TwoTanks	ACEngine	Total
Seeding Strategy A	**Seeding Strategy B**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**
Non Seeded	Dynamic	0/1/0/1/0/1/21	0/0/0/0/0/1/23	0/0/0/2/0/0/22	0/0/0/0/0/1/23	0/0/0/1/0/1/22	0/0/2/15/0/4/3	0/1/2/19/0/8/114
	Static30	0/0/0/1/0/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	2/0/0/12/0/5/5	2/0/0/13/0/5/124
	Static70	2/8/13/0/0/1/0	0/0/24/0/0/0/0	0/4/20/0/0/0/0	0/0/24/0/0/0/0	0/1/23/0/0/0/0	16/1/1/6/0/0/0	18/14/105/6/0/1/0
	ARPG	0/0/0/21/3/0/0	1/0/0/23/0/0/0	1/0/0/22/1/0/0	0/0/0/23/1/0/0	0/0/0/22/2/0/0	0/0/0/24/0/0/0	2/0/0/135/7/0/0
	Orthogonal	5/0/0/18/1/0/0	14/2/0/8/0/0/0	5/0/0/19/0/0/0	13/0/0/11/0/0/0	11/2/0/11/0/0/0	0/0/0/24/0/0/0	48/4/0/91/1/0/0
	DynamicARPG	0/1/0/1/0/1/21	0/0/0/0/0/0/24	0/0/0/2/0/0/22	0/0/0/0/0/1/23	0/0/0/0/1/1/22	0/0/2/14/0/4/4	0/1/2/17/1/7/116
Dynamic	Static30	0/2/14/6/0/1/1	3/0/16/3/0/0/2	0/1/15/5/1/1/1	0/4/17/3/0/0/0	3/1/16/1/0/0/3	0/0/0/20/1/1/2	6/8/78/38/2/3/9
	Static70	0/0/23/0/0/0/1	0/0/24/0/0/0/0	1/0/23/0/0/0/0	0/0/24/0/0/0/0	0/0/24/0/0/0/0	6/3/9/4/0/0/2	7/3/127/4/0/0/3
	ARPG	1/3/18/1/0/0/1	2/1/21/0/0/0/0	1/1/20/2/0/0/0	0/2/22/0/0/0/0	0/0/22/2/0/0/0	2/1/3/16/0/0/2	6/8/106/21/0/0/3
	Orthogonal	1/0/21/1/0/0/1	0/3/21/0/0/0/0	0/1/21/2/0/0/0	0/0/24/0/0/0/0	0/1/22/1/0/0/0	3/2/1/15/0/1/2	4/7/110/19/0/1/3
	DynamicARPG	1/0/0/23/0/0/0	0/0/0/22/1/1/0	0/0/0/24/0/0/0	0/0/0/23/1/0/0	1/0/0/22/0/1/0	0/0/0/24/0/0/0	2/0/0/138/2/2/0
Static30	Static70	0/0/23/0/0/0/1	0/0/24/0/0/0/0	0/0/24/0/0/0/0	0/0/24/0/0/0/0	0/0/24/0/0/0/0	6/7/9/1/0/1/0	6/7/128/1/0/1/1
	ARPG	2/5/16/0/1/0/0	0/0/24/0/0/0/0	0/1/23/0/0/0/0	0/0/24/0/0/0/0	0/0/24/0/0/0/0	6/1/3/12/2/0/0	8/7/114/12/3/0/0
	Orthogonal	0/1/21/1/1/0/0	0/0/24/0/0/0/0	1/2/21/0/0/0/0	0/0/24/0/0/0/0	0/0/24/0/0/0/0	4/1/2/15/2/0/0	5/4/116/16/3/0/0
	DynamicARPG	0/1/0/7/0/0/16	1/0/0/5/1/1/16	2/1/0/4/1/0/16	2/0/0/0/0/1/21	2/1/0/1/0/0/20	1/2/0/21/0/0/0	8/5/0/38/2/2/89
Static70	ARPG	0/0/0/1/0/1/22	0/0/0/0/0/0/24	0/0/0/0/0/1/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/5/5/7/7	0/0/0/6/5/9/124
	Orthogonal	0/0/0/1/0/4/19	0/0/0/0/0/0/24	0/0/0/0/0/1/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/5/3/6/10	0/0/0/6/3/11/124
	DynamicARPG	0/0/1/0/0/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/2/4/2/2/14	0/0/3/4/2/2/133
ARPG	Orthogonal	10/0/0/14/0/0/0	16/1/0/7/0/0/0	4/0/0/20/0/0/0	17/0/0/7/0/0/0	9/2/0/13/0/0/0	0/0/0/24/0/0/0	56/3/0/85/0/0/0
	DynamicARPG	0/0/1/1/0/2/20	0/0/0/0/0/0/24	0/0/0/2/0/0/22	0/0/0/0/0/0/24	0/0/0/0/1/1/22	0/0/2/15/2/0/5	0/0/3/18/3/3/117
Orthogonal	DynamicARPG	0/0/1/1/0/1/21	0/0/0/0/0/0/24	0/0/0/2/0/0/22	0/0/0/0/0/0/24	0/0/0/0/1/0/23	1/0/2/15/0/1/5	1/0/3/18/1/2/119

Table 13. Summary of the performed statistical tests for the Average Convergence metric within the first application context

		CW	EMB	CC	Tiny	TwoTanks	ACEngine	Total
Seeding Strategy A	**Seeding Strategy B**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**	**A++/A+/A/=/B/B+/B++**
Non Seeded	Dynamic	1/0/0/0/0/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	1/0/1/0/0/1/21	2/0/1/0/0/1/140
	Static30	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/144
	Static70	0/0/1/23/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/1/4/19/0/0/0	0/1/5/138/0/0/0
	ARPG	7/0/0/13/4/0/0	12/0/0/11/0/1/0	14/0/0/7/3/0/0	10/0/0/9/5/0/0	14/0/0/8/2/0/0	17/0/0/5/2/0/0	74/0/0/53/16/1/0
	Orthogonal	14/10/0/0/0/0/0	12/9/3/0/0/0/0	22/2/0/0/0/0/0	13/10/1/0/0/0/0	13/8/2/1/0/0/0	4/0/0/14/5/1/0	78/39/6/15/5/1/0
	DynamicARPG	1/0/0/0/0/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	1/1/0/0/0/1/21	2/1/0/0/0/1/140
Dynamic	Static30	1/2/0/19/1/0/1	2/1/0/19/2/0/0	1/1/2/20/0/0/0	0/0/1/23/0/0/0	2/0/0/22/0/0/0	7/4/7/3/0/1/2	13/8/10/106/3/1/3
	Static70	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/1/0/23/0/0/0	0/1/0/143/0/0/0
	ARPG	0/0/1/22/0/1/0	0/0/0/24/0/0/0	0/1/0/23/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/2/2/18/0/1/1	0/3/3/135/0/2/1
	Orthogonal	1/0/1/22/0/0/0	0/0/0/24/0/0/0	0/0/1/23/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	1/2/2/17/0/0/2	2/2/4/134/0/0/2
	DynamicARPG	14/0/0/9/1/0/0	14/0/0/7/2/1/0	19/0/0/5/0/0/0	17/0/0/6/1/0/0	16/0/0/6/1/1/0	10/0/0/13/1/0/0	90/0/0/46/6/2/0
Static30	Static70	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/144/0/0/0
	ARPG	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/1/5/18/0/0/0	0/1/5/138/0/0/0
	Orthogonal	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	0/2/5/17/0/0/0	0/2/5/137/0/0/0
	DynamicARPG	1/0/0/1/1/0/21	1/0/0/1/0/1/21	2/0/0/0/1/0/21	2/0/0/0/0/0/22	2/0/0/0/1/1/20	3/0/0/4/4/1/12	11/0/0/6/7/3/117
Static70	ARPG	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/144
	Orthogonal	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/144
	DynamicARPG	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/144
ARPG	Orthogonal	13/9/2/0/0/0/0	15/8/1/0/0/0/0	24/0/0/0/0/0/0	15/8/1/0/0/0/0	14/7/2/1/0/0/0	4/0/0/10/8/2/0	85/32/6/11/8/2/0
	DynamicARPG	1/0/0/0/0/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	1/0/1/0/0/1/21	2/0/1/0/0/1/140
Orthogonal	DynamicARPG	0/0/0/0/1/0/23	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/1/1/0/1/0/21	0/1/1/0/2/0/140

Table 14. Summary of the performed statistical tests for the Hypervolume metric within the second application context

		Google	IOF	Paint	Rails	Total
Seeding Strategy A	**Seeding Strategy B**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**
Non Seeded	Dynamic	4/0/0/0/0/0/20	0/0/0/0/0/0/24	1/0/0/1/0/2/20	1/1/0/1/2/5/14	6/1/0/2/2/7/78
	Static30	18/0/0/1/5/0/0	23/1/0/0/0/0/0	5/0/0/6/5/5/3	8/1/0/3/8/2/2	54/2/0/10/18/7/5
	Static70	10/3/6/1/0/3/1	23/0/0/1/0/0/0	18/3/2/1/0/0/0	9/4/5/15/1/0/0	60/10/13/8/1/3/1
	ARPG	10/0/0/13/1/0/0	12/0/0/10/2/0/0	11/0/0/11/2/0/0	16/0/0/5/3/0/0	49/0/0/39/8/0/0
	Orthogonal	13/0/0/3/3/3/2	11/0/0/8/4/1/0	21/0/0/3/0/0/0	13/8/0/3/0/0/0	58/8/0/17/7/4/2
	DynamicARPG	3/1/0/0/0/0/20	0/0/0/0/0/0/24	1/0/0/1/0/4/18	1/1/0/1/4/4/13	5/2/0/2/4/8/75
Dynamic	Static30	5/2/7/10/0/0/0	0/0/0/24/0/0/0	11/4/6/3/0/0/0	10/7/1/6/0/0/0	26/13/14/43/0/0/0
	Static70	0/0/0/20/0/0/4	0/0/0/24/0/0/0	3/1/4/16/0/0/0	2/3/7/11/0/0/1	5/4/11/71/0/0/5
	ARPG	0/2/6/12/1/2/1	0/0/0/24/0/0/0	4/4/7/9/0/0/0	9/3/1/9/1/1/0	13/9/14/54/2/3/1
	Orthogonal	1/2/5/15/1/0/0	0/0/0/24/0/0/0	2/4/9/9/0/0/0	2/3/7/10/0/1/1	5/9/21/58/1/1/1
	DynamicARPG	14/0/0/9/0/0/1	12/0/0/9/3/0/0	16/0/0/8/0/0/0	17/0/0/4/3/0/0	59/0/0/30/7/0/0
Static30	Static70	13/2/3/2/0/0/4	6/0/0/9/4/3/2	7/8/6/1/2/0/0	8/0/0/5/5/3/3	34/10/9/17/11/6/9
	ARPG	6/0/0/5/9/3/1	0/0/0/4/9/10/1	17/1/0/5/0/1/0	14/1/0/5/2/2/0	37/2/0/19/20/16/2
	Orthogonal	7/0/0/4/7/4/2	1/0/0/4/2/8/9	19/3/1/0/0/1/0	14/2/0/3/2/2/1	41/5/1/11/18/15/34
	DynamicARPG	0/0/0/3/1/0/20	0/0/0/0/0/0/24	1/0/0/1/1/4/17	1/0/0/2/6/3/12	2/0/0/6/8/7/73
Static70	ARPG	4/0/0/1/4/3/12	2/0/0/4/11/5/2	1/0/0/3/4/9/7	4/0/0/0/5/3/12	11/0/0/8/24/20/33
	Orthogonal	0/1/3/1/1/3/15	5/0/0/2/6/7/4	4/0/0/2/8/5/5	9/0/0/3/2/0/10	18/1/3/8/17/15/34
	DynamicARPG	0/0/2/2/0/0/20	0/0/0/0/0/0/24	1/0/0/0/0/1/22	1/0/0/1/0/2/20	2/0/2/3/0/3/86
ARPG	Orthogonal	11/0/0/7/3/1/2	13/0/0/5/5/1/0	19/0/0/5/0/0/0	14/3/2/4/1/0/0	57/3/2/21/9/2/2
	DynamicARPG	3/0/1/0/0/0/20	0/0/0/0/0/0/24	1/0/0/1/0/1/21	2/0/0/0/4/6/12	6/0/1/1/4/7/77
Orthogonal	DynamicARPG	2/0/0/2/0/0/20	0/0/0/0/0/0/24	0/0/0/1/1/0/22	1/1/0/0/0/2/20	3/1/0/3/1/2/86

Table 15. Summary of the performed statistical tests for the Average Convergence metric within the second application context

		Google	IOF	Paint	Rails	Total
Seeding Strategy A	**Seeding Strategy B**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**	**A++/A+/A=/B/B+/B++**
Non Seeded	Dynamic	4/0/0/0/0/20	0/0/0/0/0/24	0/0/0/0/0/24	1/0/0/1/0/22	5/0/0/1/0/90
	Static30	19/1/0/2/2/0/0	23/1/0/0/0/0/0	7/5/3/7/2/0/0	18/4/0/2/0/0/0	67/11/3/11/4/0/0
	Static70	4/7/7/2/1/3/0	18/5/0/1/0/0/0	3/4/14/3/0/0/0	15/2/0/3/3/1/0	40/18/21/9/4/4/0
	ARPG	10/0/0/11/3/0/0	12/0/0/9/3/0/0	17/0/0/6/1/0/0	15/0/0/6/3/0/0	54/0/0/32/10/0/0
	Orthogonal	12/3/1/2/5/1/0	16/0/0/6/0/0/0	8/9/6/1/0/0/0	18/4/0/1/1/0/0	54/16/7/10/8/1/0
	DynamicARPG	4/0/0/0/0/20	0/0/0/0/0/24	0/0/0/0/0/24	1/0/0/0/1/0/22	5/0/0/0/1/0/90
Dynamic	Static30	4/1/3/16/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	1/4/7/12/0/0/0	5/5/10/76/0/0/0
	Static70	0/0/20/0/0/0/4	0/0/0/24/0/0/0	0/0/0/24/0/0/0	1/0/6/16/0/1/0	1/0/6/84/0/1/4
	ARPG	0/0/2/18/2/1/1	0/0/0/24/0/0/0	0/0/0/24/0/0/0	2/3/6/13/0/0/0	2/3/8/79/2/1/1
	Orthogonal	3/0/21/0/0/0/0	0/0/0/24/0/0/0	0/0/0/24/0/0/0	2/0/2/20/0/0/0	5/0/2/89/0/0/0
	DynamicARPG	13/0/0/10/0/0/0	12/0/0/10/1/1/0	15/0/0/9/0/0/0	11/0/0/9/3/0/0	51/0/0/38/5/1/0
Static30	Static70	13/1/2/4/0/0/4	15/0/0/4/2/3/0	10/2/6/1/1/2/2	4/0/0/0/4/4/12	42/3/8/9/7/9/18
	ARPG	5/0/0/3/9/5/2	0/0/0/2/10/10/2	4/0/0/3/1/3/13	1/0/0/4/7/9/3	10/0/0/12/27/27/20
	Orthogonal	11/1/0/2/5/4/1	3/0/0/5/8/6/2	8/1/1/7/2/2/3	8/0/0/4/5/3/4	30/2/1/18/20/15/10
	DynamicARPG	0/0/0/0/3/1/20	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/0/1/23	0/0/0/0/3/2/91
Static70	ARPG	4/0/0/0/2/2/16	1/0/0/0/5/11/7	0/0/0/1/1/2/20	7/0/0/6/6/3/2	12/0/0/7/14/18/45
	Orthogonal	1/0/4/3/3/1/12	3/0/0/3/4/6/8	2/0/0/7/2/7/6	12/2/0/2/6/2/0	18/2/4/15/15/16/26
	DynamicARPG	0/1/2/1/0/0/20	0/0/0/0/0/0/24	0/0/0/0/0/0/24	1/0/0/0/0/1/22	1/1/2/1/0/1/90
ARPG	Orthogonal	13/6/0/2/1/2/0	12/0/0/8/3/1/0	9/7/8/0/0/0/0	17/2/1/2/2/0/0	51/15/9/12/6/3/0
	DynamicARPG	3/0/1/0/0/0/20	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/2/1/21	3/0/1/0/2/1/89
Orthogonal	DynamicARPG	1/0/0/0/2/1/20	0/0/0/0/0/0/24	0/0/0/0/0/0/24	0/0/0/0/2/0/22	1/0/0/0/4/1/90