Search-Based Test Case Prioritization for Simulation-Based Testing of Cyber-Physical System Product Lines

Aitor Arrieta[a], Shuai Wang[b], Goiuria Sagardui[a], Leire Etxeberria[a]

[a]*Mondragon Unibertsitatea, Mondragon, Spain*
[b]*Certus Software V&V Center, Simula Research Laboratory, Oslo, Norway*

Abstract

Cyber-Physical Systems (CPSs) integrate computation with physical processes. These systems are usually highly configurable to address different customer needs and are evolving to be CPS product lines. The variability of CPS product lines is large, which implies that they can be set into millions of configurations. As a result, different cost-effective methods are needed to optimize the test process of these systems. We propose a search-based approach that aims to cost-effectively optimize the test process of CPS product lines by prioritizing the test cases that are executed in specific products at different test levels. The prioritized test suite aims at reducing the fault detection time, the simulation time and the time required to cover functional and non-functional requirements.

We compared our approach by integrating five search algorithms as well as Random Search (RS) using four case studies. As compared with RS, the search algorithms managed to reduce fault detection time by 47%, the simulation time by 23%, the functional requirements covering time by 22% and the non-functional requirements covering time by 47%. Moreover, we observed that the performance of search algorithms varied for different case studies but the local search algorithms were more effective than the global search algorithms.

Keywords: Test Case Prioritization; Variability; Product Lines; Search Algorithms; Cyber-Physical System Product Lines

1. Introduction

Cyber-Physical Systems (CPSs) are an integration of computation with physical processes [1]. These systems integrate digital computations with physical processes, where embedded computers and networks monitor and control different physical units via sensors and actuators [2]. The two main layers of a CPS are the physical and the cyber. The physical layer is composed of a set of many parallel processes running in continuous time according to laws of physics [2]. The cyber layer is composed of a set of computational platforms and networks that have as a main objective to monitor and control the physical processes [2]. The computational platforms of CPSs consists of several sensors, actuators and embedded systems. These systems communicate with a network that provides a communication mechanism.

CPSs are concurrent, and thus, several tasks are performed simultaneously [3, 2]. Physical processes are parallel processes and their integration with computing requires a simultaneous composition of computing and physical tasks [3]. This requires the different parallel tasks to be observed from the validation perspectives. Furthermore, the physical world is not predictable and thus, CPSs will not be operating in a controlled environment [3]. This means that CPSs must be robust enough to withstand unexpected conditions [3] and have high adaptive capabilities to reconfigure themselves dynamically [4]. This unpredictability has to be efficiently managed during the validation stages by employing, for instance, reactive test cases that monitor the physical properties of the system. Consider the cruise controller of a car as an example, where the speed is manually set by the driver and the speed controller automatically sets the car within the specified speed. However, the time required by the car to achieve the set speed can be influenced by external factors, such as the slope of the terrain. The steeper the slope, the more time the car will need to achieve the set speed. Thus, when testing CPSs, test cases must observe the physical properties of the system.

As the use of CPSs increases, they become customizable, which provide a great possibility to give solutions to the needs that different users demand [5]. The consequence of this issue is that CPSs are evolving to be configurable, which means that particular options can be chosen for each customer. For instance, based on the functionalities, the manufacturer can configure a specific CPS configuration with different features (such as specific sensors, actuators, hardware, etc.) and configure the software parameters with the chosen features. As a result, product line engineering methods are been acquired by industrial companies to develop CPSs, which are evolving to be CPS product lines.

An example of a CPS product line involves the i-Robot Roomba.[1] In this case, a product is derived based on the users needs. Several options can be chosen by the customers based on

Email addresses: aarrieta@mondragon.edu (Aitor Arrieta), shuai@simula.no (Shuai Wang), gsagardui@mondragon.edu (Goiuria Sagardui), letxeberria@mondragon.edu (Leire Etxeberria)

[1] http://shop.irobot.co.uk/

their needs (e.g., the robot can have longer duration batteries or it can automatically turn on with a programmable clock). The variability of such systems is large, and as a result, they can be configured into millions of configurations. Although there exist many approaches to reduce the number of products to test product lines (e.g., employing Combinatorial Interaction Testing (CIT) techniques [6]), few studies aimed at proposing techniques to reduce the time for testing configurations of product lines.

In addition, testing configurations of CPS product lines faces other important challenges. As there are many products to test, real prototypes are too expensive to build, and thus, the system must be tested under simulation [5]. Simulation models permit raising the level of abstraction at which testing is performed [7]. Moreover, the use of simulation models permits software engineers to (1) execute more test cases, (2) develop test methods to select scenarios that should also be executed on the deployed system based on the risk level (e.g., fault revealing capability) and (3) specify test oracles for the automatic fault detection [7].

However, CPSs involve different engineering domains (such as mechanical or electrical); each engineer might use a different modeling and simulation tool. Consequently, co-simulation is often mandatory in the simulation process, which increments the overall simulation time. Moreover, the physical layer of these systems is often modeled with complex mathematical models that need high computations, and as a result, the time for simulating CPSs can be lengthy [5]. These limitations are even more important in CPS product lines, where many products have to be tested. Furthermore, simulation-based testing of CPSs involves different test levels (i.e., Model-, Software-, and Hardware-in-the-Loop (MiL, SiL and HiL)), where each level has different testing objectives [8].

The goal of this work is to optimize the testing process of CPS product lines by prioritizing the order in which the test cases are triggered for each test level. When testing CPSs at system level, the execution time of a test case is long, unlike unit testing, where the execution time of a test case is in the order of milliseconds [9]. Furthermore, the test suites for testing product lines are often composed of many test cases, and thus, the search space for test prioritization is huge. As a consequence, exploring the whole solution space could be impracticable, and thus a search process is mandatory to efficiently prioritize the test cases. By means of our test case prioritization approach, the overall test execution time, the time to detect faults and the time to test requirements is reduced in the context of CPS product lines.

The proposed approach has been empirically evaluated using four case studies with different configurations each. We compared five search algorithms (three local search algorithms and two global search algorithms) taking Random Search (RS) as a baseline. Results indicate that the selected search algorithms are significantly better than RS, which implies that the problem to solve is non-trivial. In fact, when compared to RS, on average, the selected algorithms can improve the fault detection time by 47%, the simulation time by 23%, the functional requirements covering time by 22% and the non-functional requirements covering time by 47%. Moreover, local search algorithms significantly outperformed global search algorithms, being in general a local search algorithm variant of the Alternating Variable Method (AVM) (coined as Permutation-based Local Search Algorithm (PLSA)) better at reducing the faults detection time and the simulation time, and the Additional Greedy algorithm better at reducing the requirements covering time.

This article is an extension of our previous paper that appeared at GECCO 2016 [10]. The key differences of this paper and the conference version paper are the following:

- We have extended our approach by integrating the tool FeatureIDE to support the variability modeling.
- We have defined three additional objectives including simulation time, functional requirements covering time and non-functional requirements covering time (for the Hardware-in-the-Loop level). The aim is to let our approach be capable of supporting multi-level testing of the CPS product line.
- We have extended our empirical evaluation by (1) adding two new case studies (i.e., Unmanned Aerial Vehicle and Direct Current engine) and in total 450 additional artificial problems and (2) reporting the evaluation results in a more comprehensive way (e.g., by adding new evaluation metrics).
- We have enriched the background and related work by adding more details and comparisons with the state-of-the-art.

The paper is structured as follows. Section 2 introduces the background related to this study. Section 3 presents our test case prioritization approach for testing CPS product lines. The experimental evaluation that was performed to evaluate the approach is presented in Section 4. Results for the performed experiments and their discussion are presented in Sections 5 and 6. The main threats to validity are discussed in Section 7. Section 8 positions our work with similar techniques in the field of product line engineering and test case prioritization. Section 9 summarizes the conclusions of our study and future work.

2. Background and Motivation

2.1. Cyber-Physical System Product Line Engineering

Product Line Engineering (PLE) methods can be effective to increase quality, productivity, and speed up development and test processes of CPS product lines [11]. Variability of CPS product lines can be found in different parts of the system. In our previous work, we identified the main variability points of a CPS simulation model [12], which can appear in both layers, the physical and the cyber. As for the physical layer, the mechanical elements are highly exposed to variability. This variability must be considered since a change in a mechanical element might change the dynamics of the system [12]. Similarly, the energy system of a CPS is a critical part and it is typically exposed to variability. For instance, long duration batteries might be bigger and heavier than short duration batteries

and thus, it might have a direct influence on the dynamics of the system. What is more, these variation points of the physical layer might directly influence the cyber layer, often requiring changes in parameters or even different control algorithms. Variability of the cyber layer can also be high. It can appear in different functionalities, in sensors (e.g., sensitivity, ranges they can measure, response time, etc.), in actuators (communication, ranges they can work in, response time, robustness properties, etc.), or in the communication system [12].

As in the case of SPLs, to apply CPS PLE in practice a variability modeling technique is required [13]. To this end, different variability modeling techniques have been proposed for this purpose, such as Clafer [14]. Due to the large variability these systems are exposed to, a key challenge of CPS product lines involves their configuration due to the high number of constraints that exist among the variabilities and commonalities. To this end, different product configurators have recently been proposed [15].

The test case prioritization of CPSs is different to a traditional application of test case prioritization in several aspects. The main differences include:

- When testing CPSs, reactive test cases provide a great possibility to observe the state of the system and react on them; this is important as CPSs might be influenced by the physical environment and thus, the test case must monitor the system state and react accordingly;
- In traditional test case prioritization, test cases are typically independent whereas in the context of CPSs when using reactive test cases, the test execution time of each test case varies depending on the previous test case;
- The time it takes to execute a reactive test case is much higher than the time it takes to execute test case for testing software systems. This is because CPSs are tested at system level in a time-continuous mode. Furthermore, the physical layer of the CPS is typically composed of complex mathematical models that are computationally heavy;
- When performing test case prioritization for testing CPSs, different test levels (i.e., MiL, SiL and HiL) should be considered, which is not case for the traditional software test prioritization.

2.2. Feature Modeling and Product Line Engineering

Feature modeling is the de-facto standard to hierarchically capture variabilities and commonalities in product lines [16, 17]. Structurally, a feature model is a tree-like structure in which nodes represent features and edges represent constraints among the features. A feature represents an increment in product functionality [18], and is releated to a set of assets that implements the feature's functionality (i.e., code, test cases, etc.). A product or configuration is a set of features satisfying the constraints of the feature model.

Figure 1 depicts the feature model of the i-Robot Roomba, which is an example of a CPS product line. Child features can be classified into mandatory and optional features. On the one hand, *mandatory* features (i.e., those features including a black circle at the top of the feature (e.g., Navigation)) must be included in all products that its parent feature appears. On the other hand, *optional* features (i.e., those features including a white circle at the top of the feature (e.g., Remote Control)) can be optionally included in those products containing its parent feature. In addition, a set of features can have an *alternative* relationship with their parent feature when only one of them can be selected when its parent feature is part of the product (e.g., only one kind of navigation system can be chosen, iAdapt or iAdapt2.0). Finally, in *or* relations, at least one of the child features must be selected in the products containing its parent feature (e.g., products supporting the "Remote Control" feature must contain the features "App" or "I-Robot" or both of them).

Apart from the parental relationships among features, feature models can include cross-tree constraints to model dependencies between constraints. These dependencies can include *requires* or *excludes* constraints. When a feature A requires feature B, this means that when the feature A is included in a product, the feature B must also be included. On the contrary, when a feature A excludes a feature B, this means that when the feature A is included in a product the feature B cannot be included, and vice versa.

2.3. Simulation-Based Testing of Cyber-Physical Systems

2.3.1. Testing Levels in Cyber-Physical Systems

Three test levels are typically employed to test CPSs using simulation-based testing: MiL, SiL and HiL. Different examples of testing CPSs employing different levels are presented in the current literature (e.g., MiL and SiL level [12, 19] or HiL level [20, 21]).

- MiL is the basic simulation that engineers use to analyse the embedded software's model with the physical layer of the CPS (also called plant model) [8]. This test configuration is performed using high precision, as the computations use floating-point arithmetic to obtain the simulation results as a reference for the following testing stages [8]. Functional requirements along with floating-point arithmetic computations are tested at this level.
- When the test results are considered satisfactory at the MiL level, the embedded software's model is replaced with an executable object code (e.g., a *.dll) [8]. Unlike MiL configuration, at this test level fixed-point computations are employed, which represent the computations done in the final target system [8]. Functional requirements along with fixed-point arithmetic computations are tested at this level.
- The third test level is named PiL and it uses real object code, which is cross-compiled and executed on the real target processor, which communicates with the simulation tool in the computer to obtain data from the physical layer [8]. In the PiL test configuration, the main objective is to detect potential inconsistencies introduced by the code compilation tool [8].

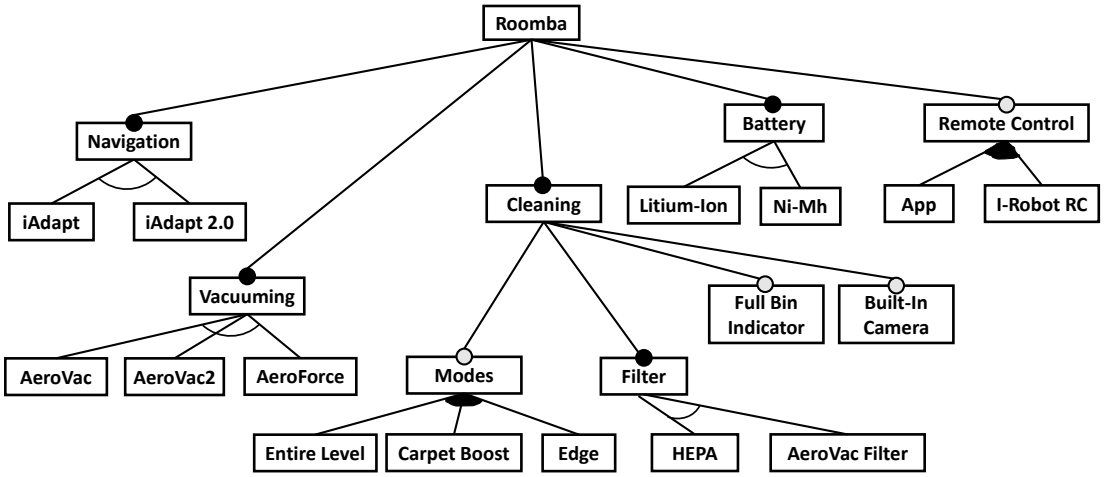

Figure 1: Feature model of the i-Robot Roomba

- The last test level refers to HiL. At this level, the embedded software is integrated with the Electronic Control Unit, as well as the real-time infrastructure (e.g., drivers) [8]. The physical layer, which was previously executed in the computer, is encapsulated in an embedded device (e.g., FPGA) with the objective of accomplishing a real-time simulation. HiL simulations are typically performed to validate non-functional requirements, including performance requirements and timing constraints [8]. Temporal constraints are critical when testing CPS, and thus, HiL test levels are being adopted to test CPSs in real-time. As an example, Eidson et al. developed a programming and simulation model called PTIDES that allows CPSs co-simulation facilitating HiL simulations [20]. Moreover, in the automotive domain, Kane et al. used test oracle mechanisms to monitor critical temporal properties at the HiL level [21]. At this test level, functional as well as non-functional requirements along with the ECU and the real-time infrastructure are tested.

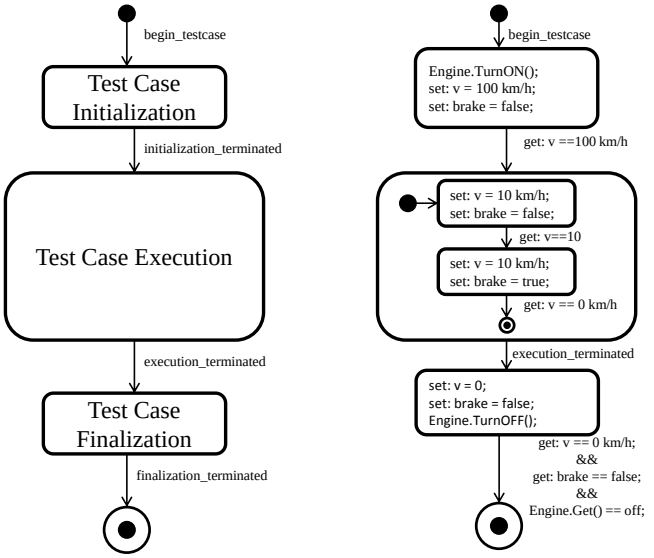

Figure 2: Typical structure of a reactive test case for a cruise control of a car example

2.3.2. Reactive Test Cases

As explained in Section1, CPSs are exposed to the unpredictability of the physical world [7, 22]. To deal with the unpredictability of the physical world, reactive test cases can be employed. Test reactivity refers to the capacity of the test cases to immediately react on the outputs of the System Under Test (SUT). Reactive test cases have the ability to observe the different states (e.g., physical processes) of the CPS and take decisions accordingly. These test cases are a set of signals that stimulate the SUT's inputs and observe a set of properties to react to them and change the values of the signals. Typically, reactive test cases can be modeled in a state chart similar to the one depicted in Figure 2, consisting of three main steps: (1) test case initialization, (2) execution and (3) finalization.

Figure 2 depicts a reactive test case for testing the cruise control of a car, where deceleration functionalities are tested. The states of the test case sets the car into a specific system state. When the system achieves the expected system state, the test case triggers another state. These test cases can also act as test oracles, since they monitor the output values of the CPS. For instance, consider that in the case of the test case shown in Figure 2, the initial state of the test case is triggered, but, for an unknown reason, the car does not achieve the expected system state (i.e., the speed of 100 km/h). By means of a timeout mechanism, it is possible to assume that the system does not achieve that state and that a fault has been detected [23].

The initialization time of a reactive test case varies based on the previously executed test case. By prioritizing the test cases efficiently, the initialization part of the test case can be skipped. Figure 3 illustrates an example of when the initialization phase can be skipped and when not. TC1 sets the system with the engine turned on and a speed of 90 km/h before triggering TC3. The initialization of TC3 consists of turning the engine on and setting the speed to 90 km/h. Thus, if TC3 was executed after

TC1, its initialization phase would be skipped. On the contrary, TC2 sets the system with the engine turned off and a speed of 0 km/h. If TC3 is executed after TC2 the initialization phase cannot be skipped: the engine would need to be turned on and speed set to 90 km/h.

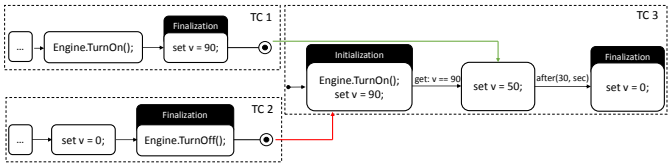

Figure 3: Example of how to reduce initialization time of reactive test cases with test case prioritization

2.4. Search-Based Testing of Product Lines

Figure 4 depicts an overview of the different steps and levels for testing product lines, and which algorithms have been previously employed at each level. The figure is structured into two parts: (1) domain testing and (2) application testing. The former refers to the test process that takes into account the variability that affects the test process of the product line (i.e., all the functionalities that can be addressed, different kinds of sensors and actuators, different features, variability in requirements, etc.). The latter refers to the testing phase that takes into account specific product configurations of the product line affecting the test process; in this layer, the variability is bound and specific functionalities, specific sensors and actuators, specific features, requirements, etc. are selected and integrated.

A product line can be configured into thousands or even millions of products. Nevertheless, it is infeasible to test each product due to time restrictions. The first step of the domain testing phase consists of the efficient generation of relevant products. The idea of this part is to generate products that would warrant the correctness of other products without testing them. In this step, different techniques (e.g., combinatorial testing [24]) can be used to select relevant products. Once the products to be tested are generated, it is important to prioritize them [25].

When a product is selected, the application testing phase provides two steps to optimize the test process. The first step consists of the test suite minimization. The objective of test suite minimization is to reduce as much as possible the number of test cases to execute, while maintaining the overall test quality (e.g., requirements coverage). This is performed by excluding redundant test cases or those that are not applicable in a specific product. Once the test suite has been minimized, the selected test cases are prioritized.

Figure 4 also classifies search algorithms used in the current state of the art for both phases (i.e., domain testing and application testing).[2] For the first layer, we have taken as a baseline the systematic mapping performed by Lopez-Herrejon et al. [26]. At this level, the most used algorithm is the Greedy, followed by the Genetic Algorithm (GA), but other algorithms such as Simulated Annealing (SA) or (1+1) Evalutionary Algorithm (EA) have also been used. For the application testing level, the classification was performed using the following papers, which to the best of our knowledge are the only ones employing search algorithms in the application layer: [5, 27, 28, 29]. In this phase, the GA and the (1+1)EA are the most used algorithms, although several are used especially in [28].

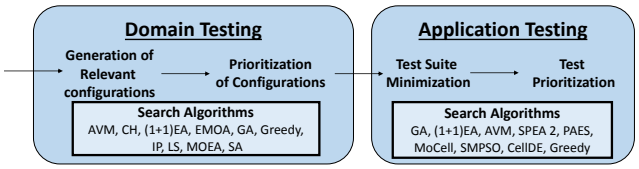

Figure 4: Test process of configurable systems and classification of different search algorithms for each level

2.5. Search Algorithms

Search-Based Software Engineering (SBSE) aims at converting a software engineering problem in a mathematical optimization problem. SBSE has been applied into many software engineering problems (e.g., requirements engineering [30]). Testing is the first software engineering activity where search algorithms were applied [31]. Search-Based Software Testing (SBST) aims at reformulating a software testing problem (e.g., test case prioritization) as an optimization problem, which is later solved by search algorithms [27, 29, 32, 33, 34]. Search algorithms aim at searching for optimal solutions by mimicking natural phenomenon such as natural evolution process [35].

To guide the search, a fitness function needs to be defined. A fitness function is the objective function that is used to assess the solutions (also known as individuals or chromosomes). Each solution (i.e., individuals) is composed of genes, which represent units for the solution [36].

2.5.1. Local Search Algorithms

Local search algorithms aim at optimizing solutions by performing local changes in an initial solution [37]. These changes produce some improvements each time according to a fitness function until a local optimum is found [37].

One of the most well-known local search algorithms is greedy. A greedy algorithm makes the locally optimal choice at each time, which allows for finding a local optimum in a fast way. Greedy algorithms have shown to be effective at solving several software engineering problems, such as Combinatorial Interaction Testing (CIT) [38]. Different greedy algorithms exist, and in SBSE the total greedy and the additional greedy are the most widely used ones. Total greedy follows the "next best"

[2]The algorithms acronyms in Figure 4 are the following: Alternating Variable Method (AVM), (CH), (1+1)Evolutionary Algorithm ((1+1)EA), Exact Multi-Objective Algorithm(EMOA), Genetic Algorithm (GA), Integer Programming(IP), Local Search (LS), Multi-Objective Evolutionary Algorithm (MOEA), Simualted Annealing (SA), Strength Pareto. Evolutionary Algorithm (SPEA2), Pareto Archived Evolution Strategy (PAES), Multi-Objective Cellular genetic (MoCell), Speed-constrained Multi-Objective Particle Swarm Optimization(SMPSO), Cellular Genetic Algorithm + Differential Evolution (CellDE)

search philosophy, which builds solutions by sorting the elements in a descending order (or ascending, if the objective is to minimize the fitness function) [39]. We show the pseudocode of the total greedy algorithm for prioritization in Appendinx A section, in Algorithm 2. On the contrary, additional greedy combines feedback from previous selections [39]. It iteratively builds solutions, selecting the best element at each iteration. The pseudocode of the additional greedy algorithm for prioritization is presented in Appendinx A section, in Algorithm 3.

Another local search algorithm is the Alternating Variable Method (AVM). AVM has been shown to be an effective local search algorithm in some SBSE problems, e.g., test data generation [40, 41]. The main idea of the AVM algorithm is to randomly generate a solution $S = (s_1, s_2, ..., s_N)$, and later, each variable in S, (i.e., s_i (where $1 \leq i \leq N$)) is individually optimized with a local search algorithm and according to a specific fitness function [40]. This process is iteratively repeated until either the search budget is consumed or there is no further local search improvements [40]. In this paper we adapt the idea of the AVM algorithm for prioritization, and we coin this algorithm as Permutation-based Local Search Algorithm (PLSA). In the Appendix A section, Algorithms 4 and 5 show the pseudocode for the PLSA algorithm.

2.5.2. Global Search Algorithms

Global search algorithms are those algorithms capable of dealing with the global optimization of a fitness function. Genetic Algorithms (GAs) are the most well known global search algorithms. After defining the corresponding objective function (i.e., fitness function), an initial population is randomly generated. Later, until the search budget is consumed, GA applies three operators (1) selection, (2) crossover and (3) mutation. The selection operator selects individuals to be involved in the reproduction. The selection is typically guided by the fitness function, so that individuals with good fitness values can have higher chances to survive [42]. Different selection strategies can be employed for this operator, such as rank-based, elitism or tournament selection. The crossover operator recombines selected individuals, as shown in Figure 5, where the crossover between two individuals with a single point crossover is performed for a permutation problem (e.g., test case prioritization). The mutation operator changes the genes in each individual with certain probability, typically equal to 1/N, being N the number of genes in the chromosome [42]. Figure 6 shows an example of a mutation for a permutation problem.

2.5.3. Stochastic Algorithms

Random Search (RS) is a stochastic algorithm commonly used as a baseline algorithm [28]. This algorithm randomly generates solutions from the search space and the best one is chosen.

3. Search-Based Test Case Prioritization Approach

The overall overview of our approach for test case prioritization is depicted in Figure 7. A feature model and the configuration file of the product to be tested is first processed by

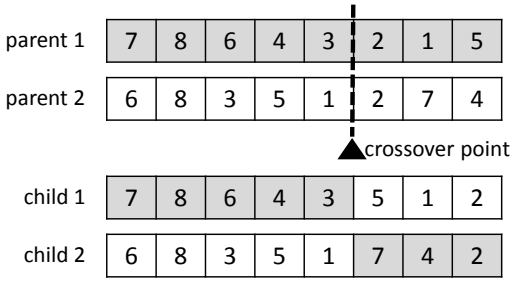

Figure 5: Crossover operator for permutation-based representation

Figure 6: Mutation operator for permutation-based representation

the requirements parser. This parser obtains the requirements assigned to the product to be tested. This information is employed by the search algorithm to prioritize the test cases in such way to reduce the amount of time required to test these requirements. In addition, the search algorithm employs information related to user preferences, which is highly valued in testing in order to give preference to some objectives rather than to others. The search algorithm also uses information related to previously executed test cases in other products, which permits a more precise test prioritization. When the search algorithm returns a prioritized test suite, this is executed using simulation, and when the execution of test cases is finished, the test results are updated in the test history. This process is iteratively repeated every time a product of the CPS product line is required to be tested.

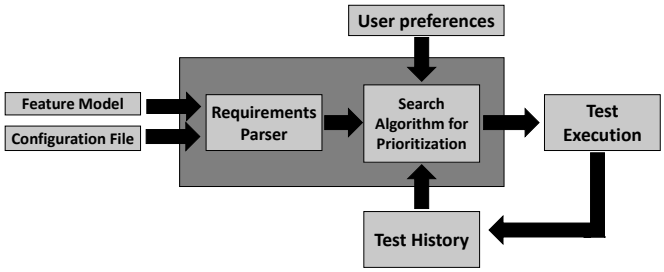

Figure 7: Overall overview of the proposed approach for test case prioritization

3.1. Basic Concepts

Let $CPSPL$ be a CPS Product Line that can be configured into N configurations cps, i.e., $CPSPL = \{cps_1, cps_2, ..., cps_N\}$.

Let FM be a feature model that captures the variability of $CPSPL$ with Nf number of features (f) and Nc number of constraints c, i.e., $FM_f = \{f_1, f_2, ..., f_{Nf}\}$, $FM_c = \{c_1, c_2, ..., c_{Nc}\}$.

Let FR be a set of Nfr functional requirements (fr) of $CPSPL$, and NFR are a set of $Nnfr$ non-functional requirements (nfr) of $CPSPL$, i.e., $FR = \{fr_1, fr_2, ..., fr_{Nfr}\}$ and $NFR = \{nfr_1, nfr_2, ..., nfr_{Nnfr}\}$.

Let TS_{CPSPL} be a test suite of Ntc test cases (tc) that tests $CPSPL$, i.e., $TS_{CPSPL} = \{tc_1, tc_2, ..., tc_{Ntc}\}$.

Let $F_{cps_i} = \{f_1, f_2, ..., f_{Nf_{cps_i}}\}$ be the features corresponding to a specific configuration i, where F_{cps_i} is a subset of FM, f_j can be any feature in FM (i.e., $f_j \in FM$). Nf_{cps_i} is the number of features representing Nf_{cps_i}, where $Nf_{cps_i} \leq NF$ [16]. Moreover, cps_i satisfies those constraints specified in FM_c.

Let $FR_{cps_i} = \{fr_1, fr_2, ..., fr_{Nfr_{cps_i}}\}$ be a set of Nfr_{cps_i} functional requirements for the configuration i, fr_j can be any functional requirement in FR (i.e., $fr_j \in FR$) and Nfr_{cps_i} are the number of functional requirements in cps_i, where $Nfr_{cps_i} \leq Nfr$. Accordingly, let $NFR_{cps_i} = \{nfr_1, nfr_2, ..., nfr_{Nnfr_{cps_i}}\}$ be a set of $Nnfr_{cps_i}$ non-functional requirements corresponding to the configuration i, nfr_j can be any non functional requirement in NFR (i.e., $nfr_j \in NFR$) and $Nnfr_{cps_i}$ are the number of non functional requirements in cps_i, where $Nnfr_{cps_i} \leq Nnfr$.

$TS_{cps_i} = \{tc_1, tc_2, ..., tc_{Ntc_{cps_i}}\}$ is a test suite of Ntc_{cps_i} test cases that tests configuration cps_i (i.e., FR_{cps_i}). Any test case j corresponding to TS_{cps_i} can be any test case from TS_{CPSPL} (i.e., $tc_j \in TS_{CPSPL}$) and $Ntc_{cps_i} \leq Ntc$. Given a test suite TS_{cps_i}, the goal of our approach is to find a test order that fulfills the objectives of the different test levels (i.e., MiL, SiL and HiL). The goal of our approach is to find an optimal test order for each test level that fulfills the objectives of the test level where the test suite needs to be executed (i.e., MiL, SiL, and HiL). Note that we define four cost-effectiveness objectives (i.e., test execution time, fault detection capability, simulation time and functional requirements coverage) for the test levels MiL and SiL. As for the test level HiL, we define five objectives (i.e., test execution time, fault detection capability, simulation time, functional requirements coverage and non-functional requirements coverage) since non-functional requirements coverage of CPSs (e.g., memory and CPU usage) cannot be tested at the levels of MiL and SiL. To this end, we have defined three fitness functions for each of the test levels (Section 3.5).

3.2. Variability Management for Test Optimization

We combine our test case prioritization algorithms with a variability modeling tool. We have employed the tool FeatureIDE [43] to manage the different features of our systems. The feature model is divided into two main parts. In the first part the variability of the system (e.g., the variability in sensors, actuators, software system, etc.) is modeled. In the second part, the variability related to functional and non-functional requirements is included. Later, the features of both parts are integrated by means of cross-tree constraints. This allows for the automatic selection of the features related to the test system when the features of the system are selected. This permits all these features to be stored into a configuration file, which is later parsed by our prioritization algorithms to perform an efficient test prioritization. Figure 8 depicts an example of a test feature model and how a configuration is selected in FeatureIDE.

Feature modeling is the most used notation in industry to manage variability according to a survey [17]. Industrial practitioners are not typically familiar with modeling notation [44]. However, it has been shown that feature models can be appropriate for industrial test engineers [44]. Simulink is the de-facto simulation tool for testing CPSs [19]. The variability considered in this paper is limited to the system model level, and thus, for the cases we have considered, feature models have supported our needs. In addition, a variability modeling tool named pure::variants [45] has shown feature modeling to be an effective notation to deal with variability of Simulink models.

3.3. Weight-based Search Algorithms

Weight-based search algorithms convert a multi-objective problem into a single objective function by assigning a particular weight to each optimization objective [28]. Given that there are N objectives (Obj), each Obj should range between 0 and 1 (i.e., $0 \leq Obj \leq 1$) [27]. If the magnitude of the objectives does not permit a range between 0 and 1, these values must be normalized. The sum of all weights (w) must be 1 (i.e., $\sum_{i=1}^{N} w_i = 1$)[27]. The general fitness function for Weight-based search algorithms is given in Equation 1.

$$FitnessFunction = \sum_{i=1}^{N} w_i * Obj_i \quad (1)$$

Weight-based search algorithms were used in this study for different reasons. The first reasons is that they allow optimization of a multi-objective problem into single objective algorithms (such as Greedy or PLSA). Moreover, they let test engineers give preference to a set of predefined properties by assigning higher or lower weights to each objective and are computationally faster as compared to pareto-based algorithms in addition to easier to implement [47]. A higher weight means that the objective to which the weight is assigned has greater importance. The weights of each objective can be assigned either statically (i.e., a predefined set of weights are assigned to each objective), or dynamically (i.e., the weights are assigned randomly in each generation) [27].

3.4. Cost-Effectiveness Measures

3.4.1. Test Case Execution Time

The execution time of a reactive test case varies depending on the number of states of the test case, i.e., time for switching from a state to another, as well as the time needed to initialize and finalize the test case. Based on the model presented in Figure 2, given a Prioritized Test Suite (PTS), of N Test Cases (TCs), (i.e., $PTS = \{TC_1, TC_2, ..., TC_N\}$), ET_{TC_i} is the estimated execution time of a test case corresponding to the position i of PTS. To estimate the execution time of a reactive test case, three phases have to be taken into account (1) initialization time, (2) test execution time and (3) test finalization time.

The initialization time (IT) of a test case in the position i depends on the finalization state of the previously executed test case (TC_{i-1}), as well as the initialization state of the test case i.

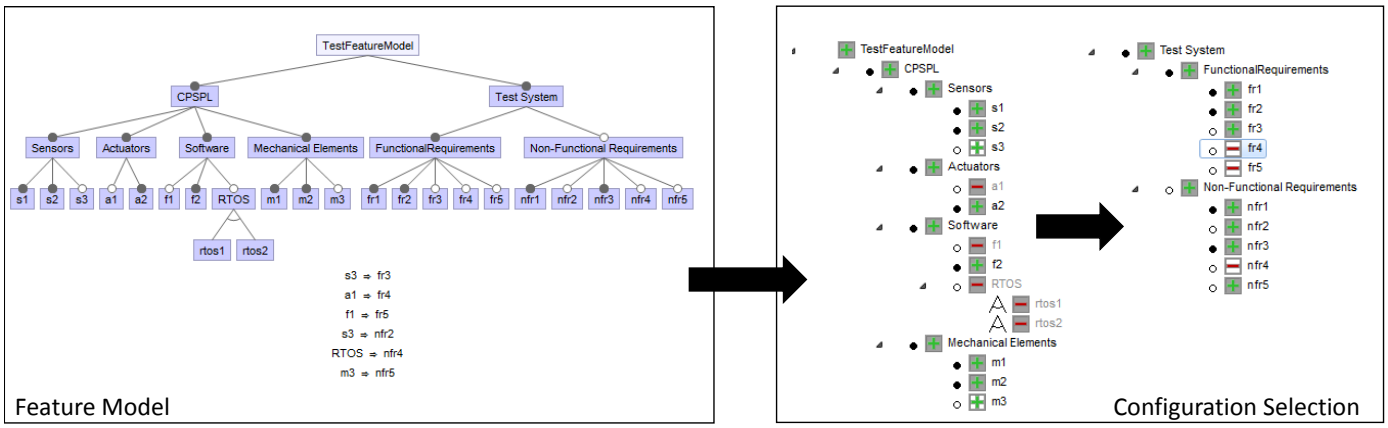

Figure 8: On the left, example of a test feature model. On the right, how is a configuration selected and requirements automatically assigned [46].

Considering the reactive test cases of the cruise control example, if the state of the system is 0 km/h, the initialization time for executing a test case will be shorter if the initialization state is 5 km/h rather than 100 km/h. Thus, the initialization time of TC_i is given by Equation 2, where $S_{TC_{i-1_{fin}}}$ is the final state of the test case in position $i-1$, and $S_{TC_{i_0}}$ is the initialization state of the test case in position i. Note that the function "time" in Equation 2 is system-specific; this means that the test engineer must specify how the function time is computed based on the system.

$$IT_{TC_i} = time(S_{TC_{i-1_{fin}}}, S_{TC_{i_0}}) \quad (2)$$

The test execution time (TET) of a test case in position i of PTS depends on the time needed to execute all the states. Thus, given that a reactive test case has NStates number of states in the test case execution phase, the TET is estimated as shown in Equation 3. $S_{TC_{i_j}}$ is the state j of the test case in position i. In the context of this study, reactive test cases do not contain loops.

$$TET_{TC_i} = \sum_{j=1}^{NStates} time(S_{TC_{i_{j-1}}}, S_{TC_{i_j}}) \quad (3)$$

The finalization time (FT) of a reactive test case i is calculated taking into account the last state of the test execution phase (i.e., $S_{TC_{i_{NStates}}}$) and the finalization state (i.e., $S_{TC_{i_{fin}}}$), as expressed in Equation 4.

$$FT_{TC_i} = time(S_{TC_{i_{NStates}}}, S_{TC_{i_{fin}}}) \quad (4)$$

Thus, the estimation time for a test case in position i of PTS is performed adding the three elements on Equations 2, 3 and 4, as represented in Equation 5.

$$ET_{TC_i} = IT_{TC_i} + TET_{TC_i} + FT_{TC_i} \quad (5)$$

3.4.2. Simulation time

In test case prioritization, the whole test suite is typically executed. Thus, it is also important to reduce as much as possible the overall test execution time. As the execution time of the reactive test cases varies based on the prioritization, we propose to reduce the overall simulation time that it takes to execute a specific test case. This is performed taking the initialization time of each test case into account, which is given in Equation 2. Given that PTS has N test cases, the simulation time is tried to be reduced following the Equation 6. Notice that the rest of the test case execution time provided in Section 3.4.1 is not calculated since the TET_{TC_i} and FT_{TC_i} would remain constant and it would not have an influence in the final simulation time.

$$SimTime = \sum_{i=1}^{N} IT_{TC_i} \quad (6)$$

3.4.3. Fault Detection Capability

To measure the effectiveness of a test case, we use the Fault Detection Capability (FDC), which is the success rate of the test case [27]. Another test quality metric for effectiveness could be the percentage of faults detected, which requires the different faults to be diagnosed. However, the faults might be difficult to diagnose in CPSs. For instance, there are cases in which two different faults provoke similar symptoms, whereas in other cases, one fault can provoke many different symptoms. In addition, another effectiveness metric could be test coverage, which is widely used in software systems to prioritize test cases [48]. Measuring different test coverage metrics in software systems might be easy, but in CPSs, apart from software a physical layer is also involved, encompassing sensors, actuators, mechanical elements, etc. Measuring test coverage in the physical layer of a CPS is not easy as their models are not discrete models, unlike in software. To overcome this fault localization problem, we chose the FDC metric, which has also been widely applied in the context of product line engineering [27, 28, 46, 49, 50, 51]. FDC identifies whether executing a test case can detect faults rather than distinguishing specific faults detected by the test case.

The FDC of a test case (TC_i) is given by the number of times it has been successful when testing other configurations ($NumSuc_{TC_i}$) with respect to the number of times the test case

has been executed, i.e., the number of times it has been successful and the number of times it has failed ($NumFail_{TC_i}$). Equation 7 reflects how the FDC is computed for TC_i. In this study, we consider a test case successful if during its execution it was capable of detecting faults. On the contrary, we consider a test case has failed if it has not detected any faults.

$$FDC_{TC_i} = \frac{NumSuc_{TC_i}}{NumSuc_{TC_i} + NumFail_{TC_i}} \quad (7)$$

It is noteworthy that when a test case is capable of finding faults in few configurations, the FDC of this test case will be low, whereas test cases that find faults in more configurations will lead to a higher FDC. For any configuration, those test cases with higher values of FDC have higher chances to detect faults and thus, we aim at prioritizing them for execution before test cases with low FDC values. Furthermore, our approach is a multi-objective search-based approach, which is also guided by other objectives, e.g., test case execution time, simulation time, requirements coverage and non-functional requirements coverage (only for HiL) (Section 3.4.5). By considering all the defined objectives, our approach aims at balancing different objectives for cost-effectively prioritizing test cases. In addition, the FDC metric has already been applied in the state-of-the-art for testing [49, 50, 51], e.g., product line testing [27, 28, 46].

3.4.4. Functional requirements covering time

When testing the functionality of any system, functional requirements are taken into account. An example of a functional requirement states as follows: "By pressing the gas pedal beyond 90% the speed limit is temporarily deactivated". This functional requirement is supposed to be tested (covered) by those test cases that first activate the speed limit, and once that the car obtains the speed limit, the state of the reactive test case would press the gas pedal beyond 90%. Our approach is designed to reduce the time to test functional requirements. To achieve this objective, we propose to analyze the time required by a specific PTS to test all the functional requirements. The time required by a PTS to test all the requirements is given in Equation 8. Given a set of N_{fr} functional requirements, $N_{tc_{fr}}$ is the position of the last test case that covers the last functional requirement of the configuration. Equation 8 measures the time required by PTS to test the N_{fr} functional requirements in cps_i.

$$FRCT = \sum_{i=1}^{N_{tc_{fr}}} ET_{TC_i} \quad (8)$$

3.4.5. Non-Functional requirements covering time

Non-functional requirements are specified to test non-functional properties of the system, such as the execution time of certain tasks, or the quality of service of a communication system [52]. An example of a non-functional requirement for the Adaptive Cruise Control of a car states as is: "The jitter of the packets related to the radar information shall not exceed 10 ms". Such non-functional requirement would be tested (covered) by those test cases that are capable of stressing the subsystem related to the ACC radar. In the CPS context, non-functional requirements are critical. For instance, the execution time of certain tasks or their deadline might lead to a completely new behavior of the CPS, unlike in general purpose software, in which these properties are related to performance [2, 46]. Moreover, some parts such as the network environment needs to be throughly tested [46]. All these properties are challenging to be tested at the MiL and SiL test levels, and thus, they are tested at the HiL test level [46]. The time required by a PTS to test all the non-functional requirements is given in Equation 9. Given a set of N_{nfr} non-functional requirements, $N_{tc_{nfr}}$ is the position of the last test case that covers the last functional requirement in the product. Equation 8 measures the time required by PTS to test the N_{nfr} functional requirements in cps_i.

$$NFRCT = \sum_{i=1}^{N_{tc_{nfr}}} ET_{TC_i} \quad (9)$$

3.5. Fitness Functions

We have defined a fitness function for each of the three test levels (i.e., MiL, SiL and HiL). A lower fitness value indicates a better solution with higher effectiveness (i.e., FDC) and lower cost (i.e., test execution time). Each objective has its own weight. The higher the weight, the higher the preference of the objective. Notice that the weights might be assigned differently at each test level. The use of three independent fitness functions for each test levels permits re-using information from the test execution of one test level before prioritizing test cases for another one. For instance, the fault detection capability of each test case can be updated when executing test cases at the MiL test level and later, this information can be used when prioritizing test cases at the SiL test level.

The MiL test level aims to perform simulation using a model as a representative of the cyber layer. This enables engineers perform simulations employing floating point arithmetic, which can help to obtain representative values. It is noteworthy that at this level, non-functional requirements of CPSs (e.g., CPS usage) cannot be tested. Thus, we derived the fitness function for the MiL test level (Equation 10) considering four objectives: (1) FDC, (2) TET, (3) SimTime and (4) FRCT.

$$F_{MiL} = w_{fdc} \times (1 - nor(\sum_{i=1}^{\#TC} \frac{FDC_{TC_i}}{i})) + w_{et} \times nor(\sum_{i=1}^{\#TC} \frac{ET_{TC_i}}{i}) + w_{simtime} \times nor(SimTime) + w_{frct} \times nor(FRCT) \quad (10)$$

The main difference between the MiL and the SiL test level is that the cyber layer of the CPS is replaced by executable code. This permits testing the system by using fixed point arithmetic operations, which might be critical in the context of CPSs. Similar to the MiL test level, non-functional requirements of CPSs cannot be tested at the SiL test level. Thus, we derived the fitness function for the SiL test level (Equation 11) considering four objectives: (1) FDC, (2) TET, (3) SimTime and (4) FRCT.

$$F_{SiL} = w_{fdc} \times (1 - nor(\sum_{i=1}^{\#TC} \frac{FDC_{TC_i}}{i})) + w_{et} \times nor(\sum_{i=1}^{\#TC} \frac{ET_{TC_i}}{i}) + w_{simtime} \times nor(SimTime) + w_{frct} \times nor(FRCT) \quad (11)$$

The HiL test level is the most expensive test level for testing CPSs. In this case, the software of the system is integrated with the real-time infrastructure (e.g., drivers and the Real-Time Operating System (RTOS)) and embedded in the target processor. Furthermore, the physical layer of the system is embedded in a real-time unit (e.g., an FPGA). However, despite this level being expensive, it enables a realistic simulation and testing non-functional properties of the CPS, including CPU and memory usage, delays in networks, etc. Subsequently, in this case we derived the fitness function by including a new objective (i.e., NFRCT) compared with the MiL and SiL test levels. Thus, we considered five objective function in the fitness function for the HiL test level (Equation 12): (1) FDC, (2) TET, (3) SimTime, (4) FRCT and (5) NFRCT.

$$F_{HiL} = w_{fdc} \times (1 - nor(\sum_{i=1}^{\#TC} \frac{FDC_{TC_i}}{i})) + w_{et} \times nor(\sum_{i=1}^{\#TC} \frac{ET_{TC_i}}{i}) + w_{simtime} \times nor(SimTime) + w_{frct} \times nor(FRCT) + w_{nfrct} \times nor(NFRCT) \quad (12)$$

Given a PTS, of N TCs, (i.e., $PTS = \{TC_1, TC_2, ..., TC_N\}$), the fitness function must give preference to the test cases located in the initial positions of the PTS. This means that the first scheduled test case (i.e., $i = 1$) is more important than the last one (i.e., $i = N$) when trying to find faults as quickly as possible. Ranking is given by dividing the cost-effectiveness measures by the position of the test cases (i.e., i) of a certain solution PTS. Subsequently, F is more sensitive to the cost-effectiveness measures of the test cases located in the initial positions of PTS. It is noteworthy to mention that this is not applied for simulation time, FRCT and NFRCT because these measures already consider the prioritization. In addition, the objectives of the fitness function must range between 0 and 1. Thus, we use Equation 13 to normalize the cost-effectiveness measures [30].

$$nor(x) = \frac{x}{x+1} \quad (13)$$

3.5.1. Adaption of the fitness function

In the experimental evaluation, two greedy algorithms were employed, Additional and Total Greedy (See Section 4.2). For both cases, the fitness function was required to be adapted. In the case of Additional Greedy, it makes the locally optimal choice at each time. This requires adding into the solution of PTS the best test case that is still not included in PTS at each iteration (i.e., the PTS size increases by 1 in each iteration), as shown in Algorithm 3 in Appendix A. Thus, in each iteration of the algorithm, the fitness is re-calculated. A problem with this algorithm is that when the PTS has few test cases the FRCT and the NFRCT objectives might not be calculated, as the test cases in the test suite might not cover all the requirements. To solve this problem, when this happens in the Additional Greedy algorithm, we included a penalty to the FRCT and NFRCT objectives (i.e., if the test cases in PTS do not cover FRCT and NFRCT, these objectives are given a high value); we selected this strategy to change as less as possible the original fitness function. This does not happen in the case of PLSA or the GAs as these algorithms provide the fitness function with the entire solution, and there are always in the solution test cases that cover all requirements.

On the other hand, total greedy builds solutions by sorting the elements in a descending order [39], as illustrated in Algorithm 3. In this case, a fitness value is calculated for each test case. In this case we consider the above-mentioned objectives too. However, instead of considering FRCT and NFRCT, we consider the ratio of functional and non-functional requirements covered by each test case. This was considered this way similarly as proposed in other studies (e.g., [39, 53]). Nevertheless, in previous studies, instead of requirements coverage, other white-box coverage criteria are employed (e.g., Statements coverage). Thus, when employing this algorithm, a fitness value is assigned to each test case considering the objective of each test level (i.e., test execution time (without considering the initialization time, since this is not possible in total greedy), faults detection capability, simulation time of the test case (which is the test execution time), functional requirements ratio of the test case and non-functional requirements ratio (only at the HiL test level)). Later, the test cases are sorted based on the values of their assigned fitness function. It is noteworthy, that when estimating the TET of each test case, the initialization time needs to be considered. Notice however, that this is based on the final state of the prior test case, as explained in Section 3.4.1. With the total greedy algorithm this is not possible and thus, the initialization time is not considered. Furthermore, for the simulation time, the initialization time is considered, as explained in Section 3.4.2. In this case, since it cannot be calculated, for the simulation time, the TET of the test case is used.

4. Experimental Setup

The aim of the experiment is to evaluate the performance of the proposed search-based test case prioritization algorithms. Random Search (RS) has been used as a baseline to assess that the problem to be solved is not trivial (Research Question 1 (RQ1)). We wanted to identify the best search algorithm to solve test case prioritization in the context of configurable CPSs (RQ2). Finally, we wanted to assess the scalability of the search approach by incrementing the test suite size (RQ3). The RQs raised are the following:

RQ 1: Are the selected search algorithms cost-effective as compared to RS?

RQ 2: Which of the selected search algorithms fares best when solving test case prioritization problem?

RQ 3: How does the increment of test cases impact the performance of the selected search algorithms?

4.1. Case Studies

Four case studies were employed in the proposed empirical evaluation. All the case studies were modeled in Simulink. The first case study was the adaptation of an industrial case study from Daimler AG concerning the Adaptive Cruise Control of a car, which has been previously used in other of our evaluations [10, 23]. The second case study involved the model of a configurable Unmanned Aerial Vehicle (UAV) we used in other evaluations [46, 23, 12, 54]. The third case study, which was also used in prior evaluations [10, 5, 23], focused on controlling certain parameters (such as temperature, pressure, pH) of certain chemical products in industrial tanks. The last case study involved an open source system for the automatic control of a DC engine with safety-critical functionalities, also used in previous evaluations [23]. The characteristics of the selected case studies and products are shown in Tables 1 and 2. Notice that the requirements related to the DC Engine case study are artificial requirements. We employed a high amount of artificial requirements in this case study to assess the scalability of the approach. Further information of the selected case studies can be found in [55].

Table 1: Main characteristics of each case study. Column blocks is referred to the number of blocks of each Simulink model. FR refers to the total number of functional requirements. NFR refers to the total number of non-functional requirements. Feature and constraints refer to the number of feature and constraints related to the feature model of the case study.

	Blocks	FR	NFR	Features	Constraints
ACC	415	19	24	14	2
UAV	843	20	40	46	5
TANK	112	6	20	24	2
DC ENGINE	257	40	80	8	1

For each case study, 120 reactive test cases were generated automatically by employing our test case generation algorithms proposed in [23]. The reason for using this test generation algorithms was that this test prioritization approach has been integrated within our framework for testing CPS product lines. This framework includes a test generator [23], a test selection approach [46], test prioritization algorithms (this study) and a test system generation tool [12]. More specifically, a test case in our context is a reactive test case, which is formed with a set of states and each state is comprised of a set of stimuli signals. For the example in Figure 2, the test case includes in total four states (considering the test case initialization and the test case finalization). The initialization state includes the following three stimuli signals, i.e., Egnine.TurnOn() (turn on the system), set: v = 100 km/h (set the speed as 0 km/h) and set: brake=false (disable the brake). When a transition is triggered (e.g., the speed has been stabilized at 100 km/h), the state will be changed. For the tank case study, the reactive test cases included different system attributes, such as the level of the liquid. For the UAV case study, the states of the reactive test cases included coordination points as well as several communication variables with the ground station (e.g., different working modes, flying commands, etc.). For the ACC case study, reactive test cases included information related to the speed of the car, the position of the brake and acceleration pedal and information related to the outside environment (e.g., distance of the preceding car). Finally, the DC Engine case study included information related to the speed that the engine should obtain, the position of two emergency buttons and the temperature of the engine.

The mean test execution time for the test cases related to the ACC case study were around 30 seconds without considering the initialization time, with the longest execution time around 70 seconds and shortest execution time around 5 seconds. As for the UAV case study, each test case took on average around 1200 seconds for executing without considering the initialization time (the longest took 5500 seconds and the shortest 10 seconds). The mean execution time for each test case of the tank case study was around 1000 seconds (with the longest execution time of around 3700 seconds and shortest execution time of 300 seconds). Finally, for the DC Engine case study, the average test execution time without considering the initialization time took around 50 seconds, with the longest test case lasting around 200 seconds and the shortest one less than 1 second.

Figure 9 shows the distribution of the cost-effectiveness data for the 120 test cases, including, the test execution time, the FDC values of each test case, and the ratio of requirements covered by each test case, both for functional and non-functional. As for the TET, notice that the distribution is normalized in order the data to be more visual.

4.2. Selected Algorithms

Table 3 summarizes the characteristics of the selected search algorithms. Three local search algorithms were selected as representative for local search algorithms (i.e., the Additional Greedy algorithm, the Total Greedy Algorithm and the Permutation-based Local Search Algorithm (PLSA)). Regarding global search algorithms, other two algorithms were selected based on their good performance in similar studies (e.g., [27]) (i.e., the Weight Based Genetic Algorithm (WBGA) and the Random Weighted Genetic Algorithm (RWGA)). Random Search (RS) was taken as the baseline algorithm. The assigned weights for Additional and Total Greedy, PLSA and WBGA where the same for all objectives. On the other hand, notice that RWGA assigns weights randomly in each iteration (following the weight-based theory). In addition, notice that for the MiL and SiL test levels, the weight assigned to the non-functional requirements covering time objective (NFRCT) was 0, since at these levels, non-functional requirements are not tested.

4.3. Algorithms Configurations

We selected the same weight for each of the objectives, i.e., the weight for all the four objectives of the MiL and SiL test levels were set as 0.25 and the five weights for the five objectives of the HiL test level was set as 0.2. The population size for the genetic algorithms was 100 and the number of fitness evaluations was set as 50,000 (i.e., we obtain the optimal solutions after 50,000th fitness evaluations). We used a standard one-point crossover with a rate of 0.8 and the mutation of a variable was done with the standard probability 1/n, where n is the number of variables. We used this setup based on other studies and recommendations related to search-based software testing [56, 27].

Table 2: Main characteristics of selected product. NFR refers to the total number of non-functional requirements. The column features refers to the number of features that the selected product configuration has.

	Product 1			**Product 2**			**Product 3**			**Product 4**			**Product 5**		
	FR	NFR	Features	FR	NFR	Features	FR	NFR	Features	FR	NFR	Features	FR	NFR	Features
ACC	6	8	7	7	12	8	13	18	10	18	22	12	19	24	14
UAV	10	13	17	11	16	20	13	20	22	17	24	25	20	40	29
TANK	3	11	14	4	14	16	5	14	18	5	16	21	6	20	23
DC ENGINE	20	45	3	30	60	4	35	70	5	40	80	6			

Table 3: Summary of the selected search algorithms in the evaluation

Algorithm Type	**Algorithm**	**Acronym**	**Brief description**
Local	Alternating Variable Method	PLSA	It randomly generates a solution and each valriable is individually optimized based on the defined fitness function.
	Additional Greedy	A_GRE	Iterative greedy algorithm that makes the local optimal solution at each time. New test cases are included in the solution at each iteration
	Total Greedy	T_GRE	Greedy algorithm that builds solutions sorting the best elements. A fitness is assigned to each test case and later these are sorted based on the fitness value.
Global	Weight Based Genetic Algorithm	WBGA	Genetic algorithm that uses fixed weights to convert a multi-objective algorithm into a single objective algorithm.
	Random-Weighted Genetic Algorithm	RWGA	Genetic algorithm that randomly selects the weights of the fitness function in each generation to convert a multi-objective algorithm into a single objective algorithm.
Stochastic	Random Search	RS	It randomly generates certain numbers of solutions and the best one is selected

4.4. Mutation Testing

Mutation testing was employed to assess the effectiveness of our approach when detecting faults. The idea of mutation testing is to obtain an original program under test and create different versions of this program (i.e., mutants) by adding small syntactic variations (i.e., artificial faults) [57, 58]. We selected this technique because mutation testing has been demonstrated to be a valid substitute of real faults [58]. We manually generated 20 mutants for each of the case studies. Mutation testing in Simulink models is a time consuming activity since the physical layer of the CPS has to be considered in addition to the software. As a result, due to timing restrictions, we were unable to use a higher amount of mutants. However, in other studies where mutation testing was used with simulink models, a similar amount of mutants is used (e.g., 20 mutants [59], 13 to 44 mutants [19], 24 mutants [60]). The distribution of the faults was 50 % in the physical layer (i.e., sensors, actuators, mechanical elements or communications) and 50 % in the cyber layer (i.e, software system). Since one of the threats to the validity of mutation testing is the subsumed mutants [61], we employed different mutation operators in different parts of the system. As the case studies were modeled in Simulink, we used the mutation operators proposed by Hanh et al. [62], which are summarized in Table 4.

Table 4: Mutation operators for Simulink [62]

Operator	Description
VNO	Variable Negation Operator
VCO	Variable Change Operator
CCO	Constant Change Operator
CRO	Constant Replacement Operator
SCO	Statement Change Operator
SSO	Statement Swap Operator
ROR	Relational Operator Replacement Operator
AOR	Arithmetic Operator Replacement Operator
ASR	Arithmetic Sign Replacement Operator
LOR	Logic Operator Replacement Operator

4.5. Evaluation Metrics

We employed four different evaluation metrics to assess the performance of the selected algorithms. Two of them measured the fault revealing capability of the approach, one of them the simulation time to execute the entire test suite and an additional one (divided into two parts) the time required by a test suite to cover functional and non-functional requirements.

4.5.1. Faults Detection Time (FDT)

To evaluate the proposed approach, we defined a metric called Faults Detection Time (FDT), which measured the time required by a specific prioritized test suite (i.e., a solution given by the selected test case prioritization search algorithm) to detect the seeded faults. We selected the FDT because, in the context of this study, the APFD might show some limitations. The first reason is that we are interested in timing performance, whereas APFD measures the relative position of the ordered test cases and the number of faults detected. This means that for reactive test cases, where the time variance is high, the APFD metric could give a good result if the first test case to be executed detects all the injected faults. However, the first test case could employ too much time to detect all the faults. In a contrary example, the faults could be found using four short test cases, and the total amount of time could be shorter than a large test case that finds all the injected faults. Moreover, in large test suites the APFD measurement might not be meaningful to compare prioritization effectiveness and it cannot measure activities such as automatic fault localization or fault severity [63].

4.5.2. Average Percentage of Faults Detected (APFD)

Despite it having some disadvantages in the context of this study, to measure test case prioritization approaches and criteria, many approaches are evaluated using the Average Per-

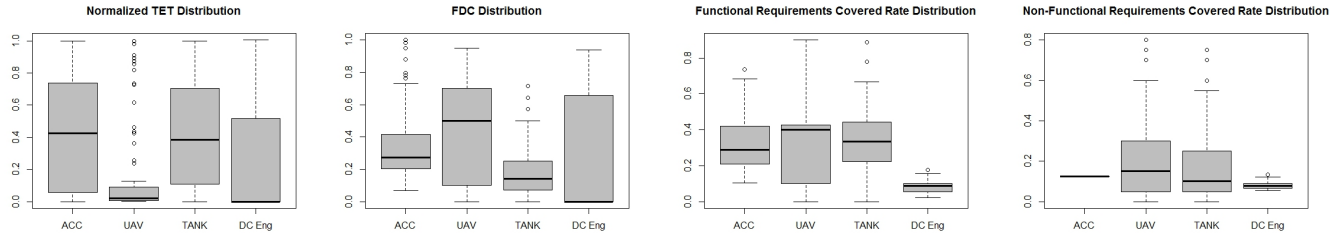

Figure 9: Distribution of the cost-effectiveness data provided as inputs

centage of Faults Detected (APFD) metric, which measures the weighted average of the percentage of faults detected over the life of the suite [48]. Since the APFD metrics is typically used to measure the performance of test case prioritization techniques, we decided to include it in our evaluation. To measure the APFD, let T be a test suite containing n test cases and F be a set of m faults detected by T. Notice that in our case, we consider mutants as faults. Let TF_i be the first test case in an ordering π of T that reveals fault i. Given an ordering π of the test suite T, the APFD metric is defined as expressed in Equation 14.

$$APFD(\pi) = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n} \tag{14}$$

4.5.3. Simulation Time

As explained before, the overall simulation time can be reduced by reducing as much as possible the initialization time of each test case. We defined the simulation time metric, which measured the time required by each of the solutions provided by the search algorithm to execute the whole test suite.

4.5.4. Requirements Covering Time (RCT)

The Requirements Covering Time (RCT) takes the time needed by a specific solution to cover all the requirements of the configuration being tested into account. We employed the RCT for both, functional requirements (at the MiL, SiL and HiL test levels) as well as for non-functional requirements (at the HiL test level). The metric was coined as FRCT for functional requirements coverage and as NFRCT for non-functional requirements coverage.

4.6. Artificial Problems and Experiment Runs

We divided the evaluation of the approach in different experimental scenarios, named as artificial problems. For each case study, each artificial problem involved (1) a test suite composed of a certain number of test cases, (2) a specific product configuration, (3) 50 algorithm runs to obtain 50 solutions and (4) evaluation of each of the solutions using the CPS case study model and 20 mutants. As recommended by Arcuri and Briand [56], we ran each algorithm 50 times to account for random variation produced by these algorithms. We employed five different product configurations for the first three case studies and four product configurations for the DC engine case study. We set the first test suite size to 30 test cases, and increment by 10 test cases when all the algorithm runs were performed, until a test suite of 120 test cases. In total, for the first three case studies, 50 independent artificial problems were employed, while for the last case study 40. The algorithm below shows the pseudocode of the developed script for executing the experiment.

```
for Each Case Study do
    for Each Search Algorithm do
        Test Suite Size = 30 test cases;
        for Each Configuration of Case Study do
            for Each test suite do
                Run search algorithm certain number of
                times
                for Each run do
                    Evaluate solution using the CPS model;
                    Calculate values of each evaluation
                    metric;
                end
                Test suite size = Test suite size + 10 test
                cases;
            end
        end
    end
end
```

Algorithm 1: Pseudocode of the performed experiment

On the one hand, the FDC of the test cases related to the ACC and Tank case studies was obtained based on historical data of previously executed configurations. For the ACC case study, the historical data was obtained from 47 previously tested configurations, that seeded a total amount of 12 faults. For the second case study, the historical data was obtained from 16 previously tested configurations, that in total seeded 8 faults. On the other hand, the FDC of the UAV and the DC engine case studies was estimated using the mutation testing technique. Specifically, twenty mutants were employed and we measured whether each test case was capable of detecting each mutant or not. This was performed this way because we did not have information from previous executions.

In addition to the statistical tests, the average improvement of the best algorithm with respect to the remaining algorithms was calculated and reported in Table 5.

4.7. Statistical Tests

We employed a rigorous statistical analysis to statistically assess the performance of the proposed algorithms. To answer RQ 1 and RQ 2 we compared the algorithms with RS and each

pair of them with the Mann-Whitney U test based on the result of the FDT for each given solution for each artificial problem. Notice that for the Tables involving RQ1 and RQ2 in Appendix B, for each of the test levels, the column indicating "+" means the number of artificial problems where the algorithm in the column A significantly outperformed the algorithm B (i.e., for all metrics except the APFD $\hat{A}_{12} < 0.5$ and p-value < 0.05, and for the APFD metric $\hat{A}_{12} > 0.5$ and p-value < 0.05). The column indicating "-" means the opposite. The column "=" indicates the number of artificial problems where there was no statistical significance between both algorithms in terms of the FDT (p-value > 0.05).

The RQ 3 analyzes the performance of the algorithms as the amount of test cases in the test suite increases. To answer RQ 3, Spearman's rank correlation (ρ) has been applied, which measures the correlation of the evaluation metric with respect to the number of test cases. The test returns a ρ value within the range [-1,1] and a p-value. For all the metrics with the exception of the APFD, a negative ρ value indicates that the performance increases as the test suite size increases, while a positive ρ indicates the opposite. For the APFD, a positive ρ value indicates that the performance increases, whereas a negative ρ indicates the opposite. Finally, the p-value indicates whether there is statistical significance in the correlation.

5. Results and Analysis

5.1. Fault Detection Time (FDT)

The FDT measures the time required by each of the prioritization techniques to kill the 20 mutants. While Figure 10 shows the distribution of the FDT results for all the artificial problems, Table B.6 (shown in the Appendix B) reports the statistical analysis performed for each case study in terms of the FDT for the three test levels (i.e., MiL, SiL and HiL) and RQs 1 and 2. RQ1 aims to answer whether the problem to solve was non-trivial by comparing each of the selected algorithms with RS. In general, all the algorithms outperformed RS for most of the artificial problems. Taking all the case studies as well as all the test levels into account, Additional Greedy outperformed RS with statistical significance in 306 out of 570 artificial problems, PLSA in 371 out of 570 artificial problems, WBGA in 318 out of 570 artificial problems, RWGA in 327 out of 570 artificial problems and Total Greedy in 312 out of 570 artificial problems. Conversely, RS outperformed the Additional Greedy algorithm with statistical significance in 108 out of 570 artificial problems, PLSA in 42 out of 570 artificial problems, WBGA in 71 out of 570 artificial problems, RWGA in 60 out of 570 artificial problems and Total Greedy in 108 out of 570 artificial problems. This means that all the selected algorithms performed in general better than RS, although there are some exceptions, such as the ones related to total and additional greedy for the ACC case study.

Regarding RQ2, where the performance among the rest of algorithms was compared, results were not consistent for each of the case studies. The PLSA algorithm was the best one for the ACC and the DC engine case study, whereas Additional Greedy performed best in the UAV and the Tank case studies. This suggests that in terms of the FDT, local search algorithms perform better than global search algorithms. However, while PLSA outperformed the rest of the algorithms for both the UAV and the tank case studies, both WBGA and RWGA outperformed Additional Greedy in the ACC and the DC engine case studies. Moreover, the difference between the PLSA and Additional Greedy for the UAV and the tank case studies were not very high (on average, as reported in Table 5, around 16.6% and 22.71%), whereas for the ACC and the DC engine case studies PLSA reduced the FDT as compared with Additional Greedy on average in 49.78% and 47.84%, as reported in Table 5.

RQ3 aimed at assessing the scalability of the approach. To this end, we applied the Spearman's rank correlation. The summary of the results for the Spearman's rank correlation with respect to each case study product is shown in Table B.7. All the selected algorithms with the exception of the Total Greedy algorithm showed improvement in terms of the FDT with the test suite size increase for the tank case study. For the rest of case studies, in the case of RS as well as the global search algorithms (i.e., WBGA and RWGA) in most of the products their performance decreased with the test suite increase. Conversely, the performance of the PLSA increased in most of the cases statistically significantly (i.e., $\rho < 0$ and p-value < 0.05), except for the DC engine case study. However, even if for the DC engine case study in most of the products the performance of the PLSA algorithm did not increase, its performance did not decrease with statistical significance (i.e., p-value > 0.05).

5.2. Average Percentage of Faults Detected (APFD)

The APFD measures the position of the test cases with respect to the detected faults, which are in this case mutants. The obtained APFD distributions are depicted in Figure 11. For each of the case studies we report the statistical tests summary in Table B.8, where the number of artificial problems in which each pair of algorithms outperform each other are reported. RQ1 evaluates whether the problem to solve is non-trivial by comparing each of the selected algorithms with RS. In general, the selected algorithms outperformed RS, although there are some exceptions. In this case, considering the statistical tests, RS outperformed both Greedy algorithms for most of the artificial problems (i.e., 420 out of 570 artificial problems for Additional Greedy and 292 out of 570 for Total Greedy). However, the remaining algorithms outperformed RS in terms of the APFD metric. Specifically, PLSA outperformed RS in 336 out of 570 artificial problems, WBGA outperformed RS in 280 out of 570 artificial problems and RWGA in 294 out of 570 artificial problems. Conversely, in terms of the APFD, RS outperformed PLSA with statistical significance in 55 out 570 artificial problems, WBGA in 96 out of 570 artificial problems and RWGA in 84 out of 570 artificial problems. This means that, except for both Greedy algorithms the selected algorithms outperformed RS.

Regarding RQ2, in terms of the APFD metric, PLSA outperformed the rest of algorithms in the ACC case study. Total greedy performed best in the UAV and tank case studies in terms of the APFD. Last, unlike in terms of the FDT,

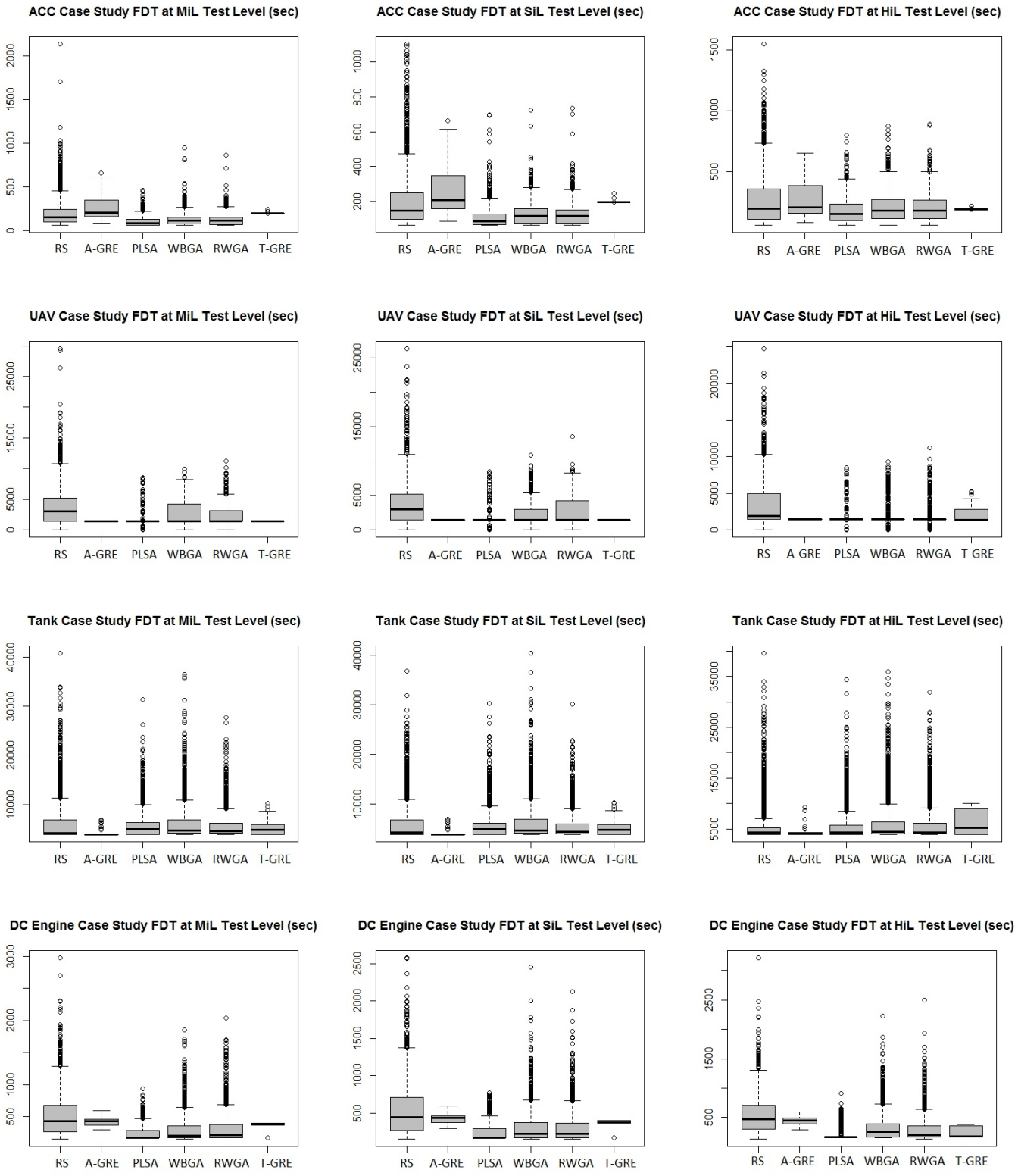

Figure 10: Distribution of the FDT for all the artificial problems of each case study

Table 5: Percentage of improvement by the best algorithm for each case study with respect to the remaining algorithms

		BEST ALG.	RS	A_GRE	PLSA	WBGA	RWGA	T_GRE	AVERAGE
FDT	ACC	PLSA	41.85	49.78	-	14.39	12.20	34.63	30.57
	UAV	A_GRE	60.66	-	16.60	39.54	38.56	14.39	33.95
	TANK	A_GRE	25.96	-	22.71	28.49	23.67	23.83	24.93
	DC_ENG	PLSA	58.04	47.84	-	28.53	27.03	29.45	38.18
APFD	ACC	PLSA	1.01	11.80	-	0.60	0.33	8.65	4.48
	UAV	T_GRE	2.20	1.92	1.85	1.77	1.79	-	1.90
	TANK	T_GRE	1.19	2.60	1.11	1.54	1.35	-	1.56
	DC_ENG	WBGA	2.40	53.63	0.04	-	0.02	51.69	21.56
SIM TIME	ACC	PLSA	25.51	5.97	-	8.51	10.95	26.05	15.4
	UAV	A_GRE	24.02	-	5.64	17.69	16.61	43.64	21.52
	TANK	A_GRE	37.74	-	0.25	15.04	15.37	40.89	21.86
	DC_ENG	PLSA	2.83	1.76	-	0.11	0.12	43.95	9.75
FRCT	ACC	RS	-	10.06	13.59	12.32	11.87	49.26	19.42
	UAV	RS	-	4.34	8.76	9.11	17.83	91.59	26.33
	TANK	A_GRE	17.92	-	20.12	16.34	16.28	35.65	21.25
	DC_ENG	A_GRE	84.32	-	91.07	87.90	87.90	39.58	88.15
NFRCT	ACC	A_GRE	58.32	-	2.73	4.31	6.41	75.79	29.51
	UAV	A_GRE	29.99	-	0.08	0.35	4.56	90.79	29.51
	TANK	A_GRE	14.99	-	17.21	17.91	14.71	65.54	26.07
	DC_ENG	T_GRE	94.25	58.00	96.48	95.16	95.26	-	83.32

both global search algorithms (i.e., WBGA and RWGA) outperformed local search algorithms for the APFD metric. This might mean that WBGA and RWGA prioritized those test cases with high FDC but giving lower importance to their test execution time. However, as shown in Figure 11 and the reported average improvement values in Table 5, the differences between PLSA and both global search algorithm is not high. It is noteworthy to mention that these algorithms outperformed Additional Greedy for the remaining three case studies in terms of the APFD.

As for RQ3, where the scalability of the approach is assessed, we applied the Spearman's rank correlation test. In this case, since we were aiming to increase the APFD, a positive ρ indicated an improvement of the algorithm with the increase of the test suite size. The obtained results for the Spearman's rank correlation test are reported in Table B.9. As it can be appreciated, for all the case studies and products, the performance of all the algorithms improved with the test suite size in terms of the APFD metric with statistical significance. This means that the algorithms are scalable in terms of the APFD.

5.3. Simulation Time

Figure 12 depicts the obtained normalized results for the simulation time metric. Notice that these values are normalized for representation because a larger test suite implies a larger simulation time. The values are normalized by dividing the obtained simulation time with the number of test cases in the test suite. In addition, Table B.10 summarizes the results of the statistical analysis performed for each case study for the simulation time metric. As for RQ1, where the selected algorithms are compared with RS, as it can be shown, all the algorithms outperformed RS for all the case studies at all the three test levels for every single artificial problem with the exception of Additional and Total Greedy. The Additional Greedy algorithm performed similarly to RS in terms of the simulation time for the DC engine case study. However, for the rest of case studies, Additional Greedy outperformed RS. Conversely, Total Greedy was outperformed by RS in 429 out of 570 artificial problems. On average, from Table 5, it can be seen that the selected algorithms outperformed RS up to 37.74 % in terms of the simulation time.

RQ2 aims at answering which of the algorithms showed better performance. For the case of simulation time, local search algorithms performed better than global search algorithms. As shown in Table 5, the average improvements between both local search algorithms is not very high (the higher average improvement was of 5.97% for the PLSA with respect to Additional Greedy for the ACC case study). Taking the statistical analysis into account, which is shown in Table B.10, the PLSA algorithm performed the best in the ACC, tank and the DC engine case studies, whereas Additional Greedy performed best in the UAV case study. In this last case study, PLSA also outperformed both global search algorithms (i.e., RWGA and WBGA). Moreover, for those cases where PLSA fared best, Additional Greedy also outperformed both global search algorithms for the ACC as well as the tank case studies, although during the DC engine case study both global search algorithms outperformed Additional Greedy.

Notice that for this metric we did not address RQ3, since a larger test suite always implies a larger simulation time.

5.4. Functional Requirements Covering Time (FRCT)

Figure 13 depicts the distribution of the FRCT for all the artificial problems of each case study, while Table B.11 reports the summary for the statistical results when compared each pair of the selected algorithms. RQ1 aims to answer whether the problem to solve was non-trivial by comparing the selected algorithms with RS. As for the statistical analysis reported in Table B.11, in general, except for Additional Greedy, the selected algorithms were outperformed by RS for most of the artificial problems, with the exception of the tank case study. This means that the problem of reducing the time for testing functional requirements is trivial in this context. However, it is positive that still Additional Greedy performs better than RS. The Additional Greedy algorithm outperformed RS with statistical significance in 334 out of 570 artificial problems. Conversely, RS outperformed the Additional Greedy algorithm in 156 out of 570 artificial problems.

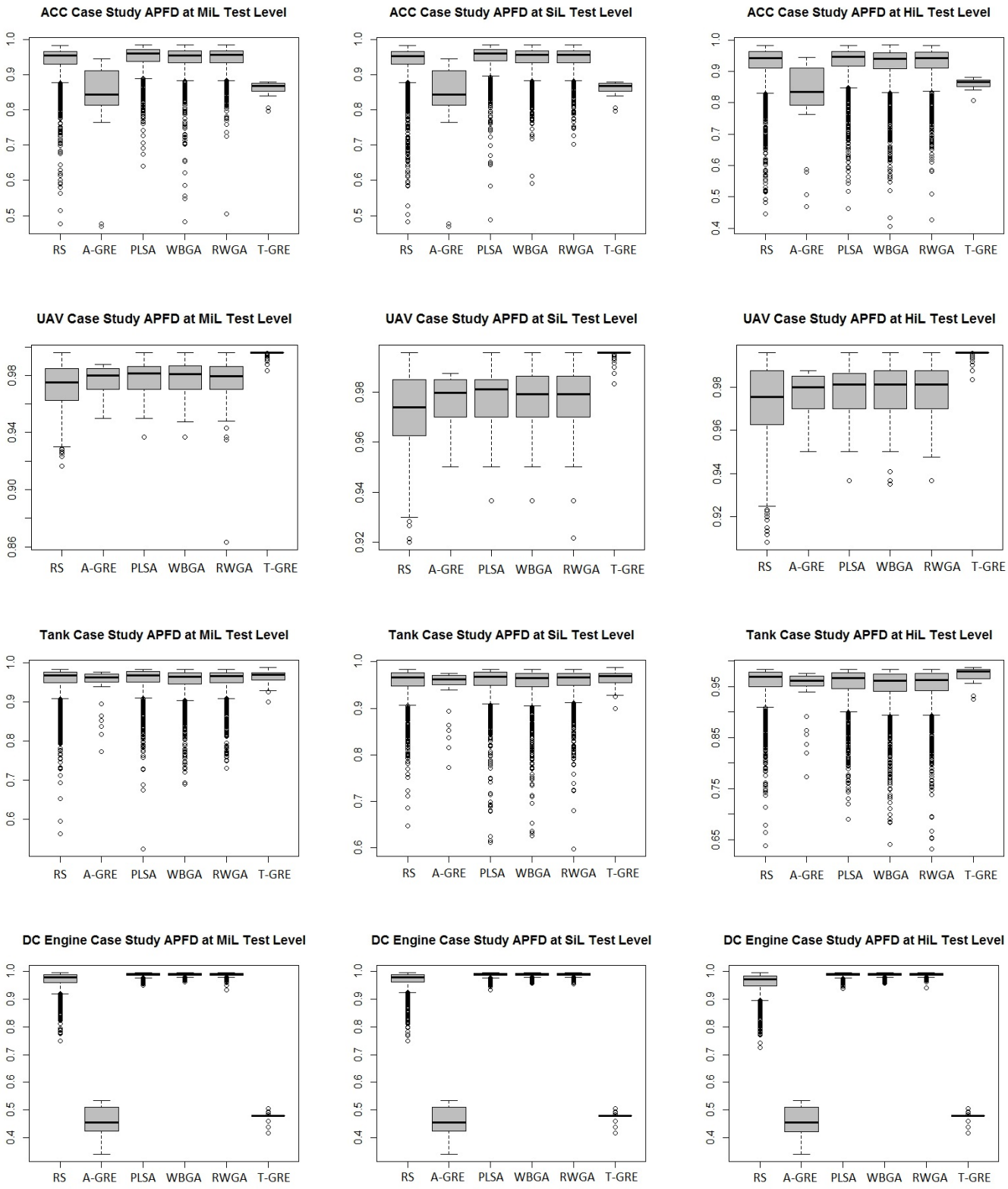

Figure 11: Distribution of the APFD for all the artificial problems of each case study

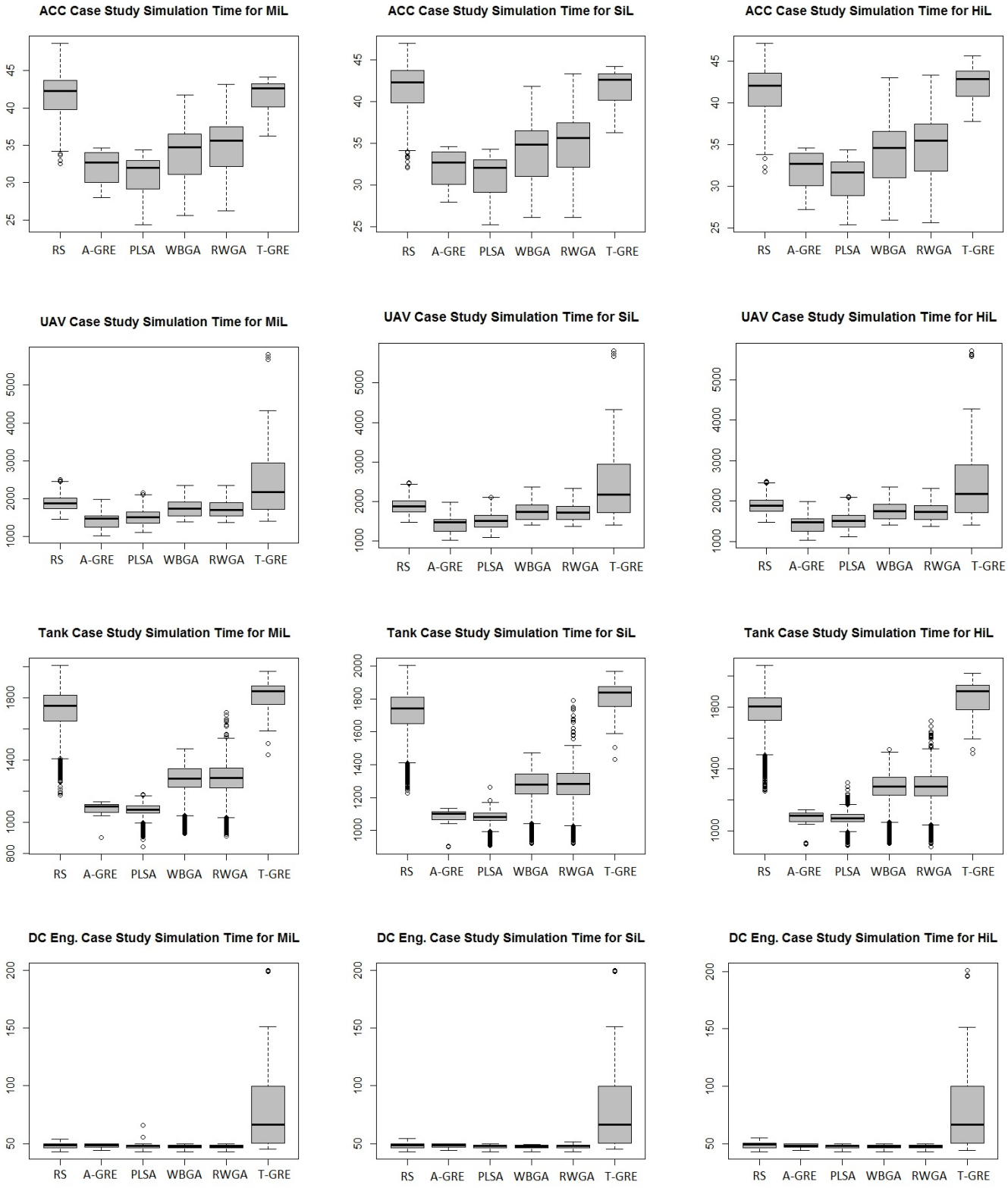

Figure 12: Distribution of the normalized simulation time (i.e., average simulation time per test case) for all the artificial problems of each of the case studies

As for RQ2, Additional Greedy performed best in most of the case studies for most of the test levels, except for the ACC case study and the MiL and SiL test levels. It is noteworthy the average improvement of the FRCT of Additional Greedy with respect to the rest of algorithms for the DC engine case study, which on average, it improved the rest of algorithms in around 80.76%, as reported in Table 5. In addition, unlike for previous evaluation metrics, PLSA performed worst when compared to the rest of algorithms. This might be because PLSA gives more importance to the rest of objectives. However, as shown in Figure 13 and reported in Table 5, the differences between PLSA and both global search algorithms in terms of the FRCT is not very high.

RQ3 evaluates the scalability of the selected algorithms. Table B.12 reports the Speaman's rank correlation for the FRCT metric with respect to the test suite size. Results vary for each algorithm depending on the case study as well as the product. For instance in the ACC case study, the spearman rank correlation ρ showed a negative correlation with statistical significance for all the algorithms in both p4 and p5 products. It is noteworthy that since these two products were more complex than the rest, their number of functional requirements was higher, as shown in Table 2. For the rest of case studies, in general Additional Greedy showed a negative ρ for most of the products, which means that its performance increases when the test suite size increases. In addition, except for the DC engine case study, the PLSA algorithm also showed negative correlation in most of the cases, except for non-complex products (typically p1). As for the global search algorithms, in general their performance decreased with the test suite size in most of the products and case studies, with the exception of the ACC case study.

5.5. Non-Functional Requirements Covering Time (NFRCT)

Figure 14 depicts the distribution of the results for the NFRCT and Table B.13 reports the summary for the statistical results when compared each pair of the selected algorithms. It is noteworthy that non-functional requirements are tested at the HiL test level. The first RQ compares the selected algorithms with RS. For the ACC and Tank case studies, the selected algorithms, with the exception of total greedy, outperformed RS. For the UAV case study, PLSA showed similar results as RS, whereas the rest of algorithms showed worst results than RS. Conversely, Additional Greedy and total greedy outperformed RS in the DC engine case study, while the rest of algorithms showed worse performance than RS. On average, Additional Greedy was the algorithm showing highest average improvement percentage when compared with RS, as shown in Table 5, improving the RS algorithm on average between 14.99% and 86.31%. As for the statistical tests, Additional Greedy outperformed RS with statistical significance in 157 out of 190 artificial problems. Conversely, RS only outperformed Greedy in 29 out of 190 artificial problems with statistical significance.

The second RQ evaluates which of the selected algorithms fared best. According to the statistical analysis, which is summarized in Table B.13, in this case, results vary depending on the case study. Additional Greedy showed the best performance for the ACC case study whereas the Total Greedy outperformed the remaining in the DC engine case study. PLSA outperformed the rest of algorithms in the UAV case study. Finally, RWGA performed better than the rest in the Tank case study. In addition, from Table 5, it can be appreciated that as for the average percentage of the improvement, Additional Greedy was the algorithm showing the best performance in terms of the NFRCT metric for the ACC, UAV and Tank case studies, whereas total greedy outperformed the remaining algorithms for the DC Engine case study.

RQ3 evaluates the scalability of the approach. Table B.14 reports the results of the Spearman's rank correlation for the NFRCT metric based on the test suite size. On the one hand, except for the tank case study, RS as well as both global search algorithms (i.e., WBGA and RWGA), showed a performance decrement as the test suite size increases. On the other hand, Additional Greedy improved with the test suite size for all products of the tank and DC Engine case studies, as well as in most of the products of the ACC and UAV case studies. The PLSA algorithm increased its performance with the test suite size in all products of the ACC and tank case studies, as well as in most of the UAV case study's products. However, in the DC engine case study the performance of the PLSA algorithm decreased as the test suite size increased. The performance of the total greedy algorithm improved with the test suite tize in terms of the NFRCT in all case studies and products with the exception of the ACC case study.

6. Discussion of the Results

6.1. Answer to RQ1

The first RQ refers to the comparison of the proposed algorithms with RS to assess that the problem to solve is not trivial. RS has been selected as the comparison baseline for several similar studies (e.g., [28, 27, 29, 46, 64]). Generally in RQ 1, for most of the metrics, the selected algorithms outperformed RS. The most striking exceptions were for the Functional Requirements Covering Time in the ACC and UAV case studies. In these cases, RS outperformed the rest of the algorithms. The reasons for this might be that either the rest of algorithms focused on other objectives, or for these case studies there were not a lot of functional requirements and thus (i.e., 6 and 10 functional requirements in the most simple products), the problem to solve was non-trivial. Nevertheless, for the remaining evaluation metrics in these case studies, the rest of the algorithms outperformed RS.

There are some exceptions were RS outperformed the selected algorithms. This happened in some cases, especially with the additional and total greedy algorithms. As for the additional greedy algorithm, when considering the ACC case study, RS slightly outperformed the additional greedy when considering the FDT. When having a close view to this case, the Additional Greedy prioritized test cases with quite low test execution time, rather than giving priority to those test cases with higher FDC. This is because the distribution of the TET of the test cases was quite broad, as it can be appreciated in Figure 9, and thus, the additional greedy algorithm prioritized those short test cases but

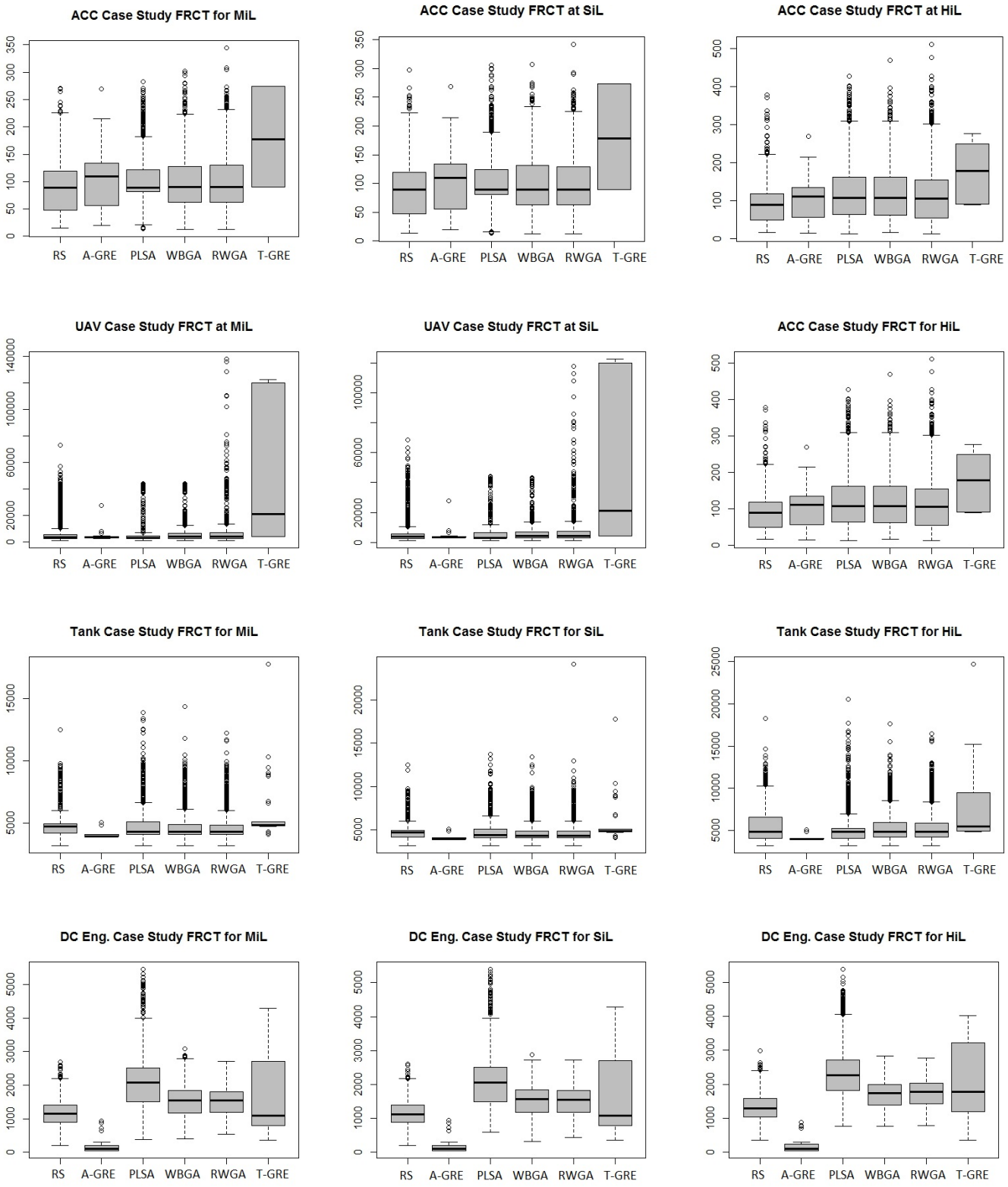

Figure 13: Distribution of the FRCT for all the artificial problems of each case study

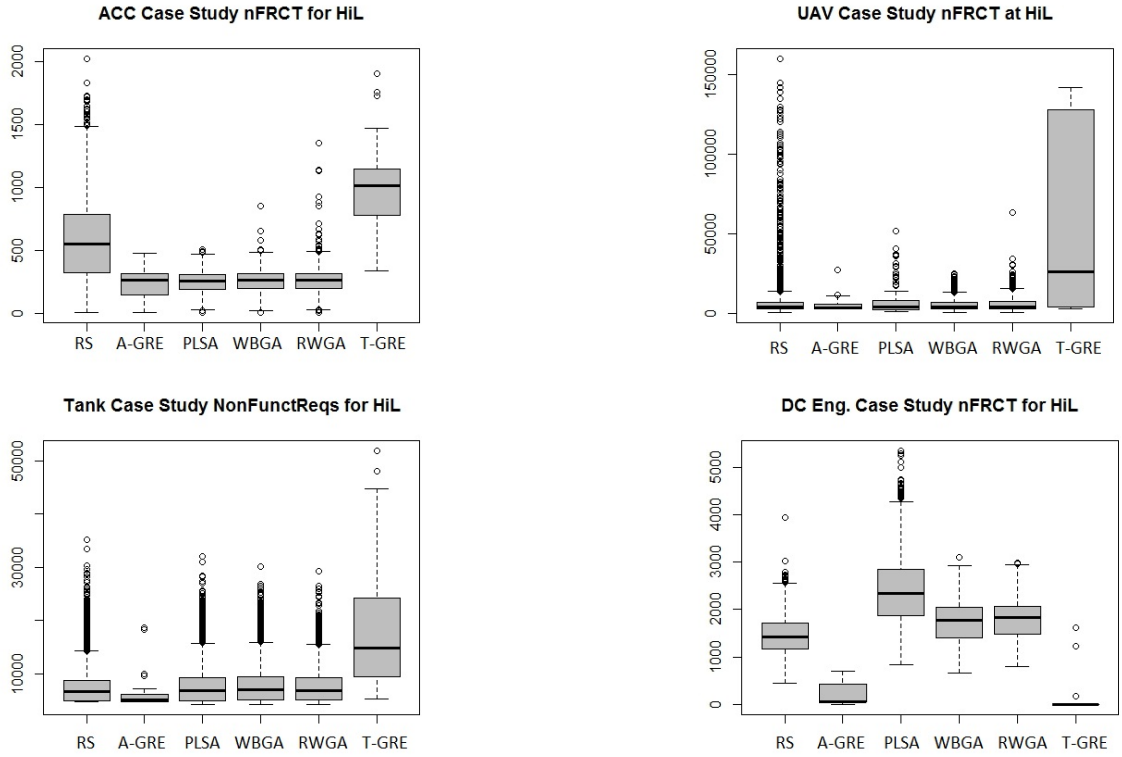

Figure 14: Distribution of the NFRCT for all the artificial problems of each case study

with not that good fault detection. This theory can also be appreciated in the APFD results. It can be seen that the APFD of the additional greedy is quite low when compared to the remaining algorithms. This is because in the prioritized test suite the test cases with relatively low TET were in the first positions. This is further exacerbated when considering the APFD of the DC engine case study. Notice that in this case, there are many test cases (more than half of the test suite) with very low TET (as it can be seen in Figure 9). Both additional and total greedy gave priority to these test cases with very low TET. These test cases later did not have a very high impact in the final FDT because their TET is very low when compared with the remaining test cases.

Other cases were RS outperformed the additional greedy algorithm is when considering the functional requirements covering time in the ACC and the UAV case studies. In these case studies, RS also outperformed the remaining algorithms. The main reason for this is the simplicity of the problem. For the most simple products (e.g., P1, P2 and P3), there are few functional requirements. Thus, RS managed to prioritize test cases that cover them in a short amount of time. Conversely, the remaining algorithms focus on other objectives, such as prioritizing test cases with higher FDC. Last, in some cases of the UAV with respect to the NFRCT metric, RS outperformed additional greedy in terms of the statistical significance. However, when considering the overall values, the difference is not high. In fact, RS in many cases spent too much time to cover all the non-functional requirements, as appreciated in Figure 14 for the UAV case study, and considering the average values, the additional greedy algorithm outperformed the UAV by 30%. RS could have outperformed the additional greedy in some of these cases because there are a few test cases that cover many non-functional requirements, as it can be seen in Figure 9, and thus, the problem of covering non-functional requirements might be very easy for RS in some of the artificial problems within this case study. These problems typically arise when optimal solutions appear in a global search space. The additional greedy algorithm might get stuck in some local optima whereas RS has some probabilities to explore the global search space.

In the case of the total greedy algorithm, RS also outperformed it in some artificial problems, which is different from existing test prioritization studies. This happens in different cases: when considering the FDT metric, RS performed better in the ACC and tank case studies. When considering the APFD, in ACC, tank and DC engine case studies. In most of the artificial problems related to simulation time and FRCT and in three out of four case studies when considering the NFRCT objective. Notice that all these metrics measure timing behavior of the prioritized test cases when finding faults, simulating the system or covering requirements. The reason of why RS performed better than total greedy in some cases is simple. Total greedy obtains a fitness function of each test case and later it sorts in descending order. However, the test cases are reactive, what mean that their initialization time depends on the last state of the previously executed test case. In this case, as the total Greedy measures the fitness function of each of the test cases individually, it is not possible for it to estimate the initialization time. That is why the simulation time metric for this algorithm

is much higher than the remaining algorithms. Since the algorithm does not consider the initialization time of test cases, the search might be somehow distorted. As for the APFD, in the ACC and DC Engine case study, similarly as in the case of additional greedy, total greedy prioritized those test cases with very low TET, and thus, the test cases that detect most faults are not prioritized in the initial positions of PTS.

We believe that these individual exceptions can be further improved in future versions by better tuning the weights of each objective depending on the algorithm that needs to be used. For instance, when using the PLSA algorithm it could be useful to add higher weights to requirements coverage objectives. On the contrary, the use of the Additional Greedy algorithm could assign a higher weight to the FDC objective. However, for most of the cases, there was at least one of the selected algorithms that outperformed RS, and thus, we can answer the first RQ as follows:

> *Based on the experimental results and the statistical tests of our study we can conclude that the test case prioritization for simulation-based testing of CPS product lines is a non-trivial problem and thus, search algorithms are recommended to be used.*

6.2. Answer to RQ2

RQ 2 aimed at answering which of the selected search algorithms was the most appropriate to solve the test case prioritization problem for the CPS product line engineering context. Generally in RQ 2, local search algorithms (i.e., PLSA and Additional Greedy) outperformed the selected global search algorithms (i.e., WBGA and RWGA). As for the fault revealing capability, which was measured by the FDT and APFD metrics, PLSA outperformed the rest of the algorithms in two case studies using the FDT and in three case studies using the APFD metric, whereas Additional Greedy outperformed the rest of algorithms in two out of four case studies using the FDT. For the APFD metric, WBGA outperformed the rest of metrics in one case study. As for the simulation time, the PLSA algorithm outperformed the rest of the algorithms in three out of four case studies, whereas Additional Greedy was the best algorithm for the simulation time in the UAV case study. As for the RCT metrics, for functional requirements Additional Greedy outperformed the rest for the DC engine as well as the tank case studies. Regarding non-functional case studies, Additional Greedy outperformed the rest of metrics for the ACC and the DC engine case studies, whereas the PLSA algorithm outperformed the rest for the UAV case study. Finally, RWGA outperformed the rest for the tank case study.

The reason that local search algorithms (i.e., PLSA and Additional Greedy) outperformed global search algorithms (e.g., WBGA) can be explained that optimal solutions of our problem may exist in a local search space and thereby local search algorithms are more efficient as compared with global search algorithms in terms of finding optimal solutions within a limited number of fitness evaluations (e.g., 50,000 in our case). However, when the number of evaluating fitness is increased, the performance of global search algorithms may be improved. However, it is worth mentioning that the time to run the global search algorithms will also be increased with the growth of fitness evaluations.

As for the FDT metric, Additional Greedy performed slightly better than PLSA in the UAV and tank case studies, having an average improvement of 16.60 and 22.71% respectively. Nevertheless, in the ACC and DC Engine case studies, PLSA outperformed Additional Greedy by 49.78 and 47.84%. This improvement is much higher than the improvement of the Additional Greedy over PLSA. A possible explanation of the Additional Greedy algorithm performing slightly better than PLSA in these two case studies might be due to the test execution time of their test cases, which are in the order of some thousands of seconds. Conversely, the test cases of the ACC and the DC Engine case studies take shorter test execution time. Notice that something similar happened with the simulation time objective, where when considering the average times, the Additional Greedy algorithm performed better in the UAV and Tank case studies whereas the PLSA performed better in the ACC and DC Engine case studies.

These results are consistent with the conference version of this article [10]. In the conference version, only two objectives were taken into account: (1) the test execution time and (2) the fault detection capability. We measured the performance of the algorithms with the FDT metric, which is also used in this paper. Results of our conference version paper showed that local search algorithms were better than global search ones. Specifically, Additional Greedy performed best with shorter test suites while PLSA performed best with longer test suites [10]. Thus, from the results of our experiment in this paper, and also supported by the results of the conference version paper, we can answer the second RQ as follows:

> *Results can vary depending on different case studies. However, local search algorithms performed better than global search algorithms. Generally, for faster fault detection and for simulation time reduction, PLSA performed better, whereas for faster requirements testing Additional Greedy outperformed the rest.*

6.3. Answer to RQ3

The last RQ aimed at assessing the scalability of the proposed search algorithms. To answer this RQ we applied the Spearman's rank correlation test, where the correlation between the selected evaluation metrics with respect to the number of test cases in the test suite was measured. Results varied depending on the case study and algorithms. From the previous RQs, we extracted the best algorithms for solving the test case prioritization problem for the CPS product line engineering context. On the one hand, for the FDT metric, the PLSA algorithm showed the best performance. For this metric, the performance of the algorithm increased with the test suite size for 44 out of 57 cases. Moreover, for the rest of the cases, the performance of the PLSA algorithm did not decrease with statistical signifi-

cance (i.e., p-value > 0.05). On the other hand, for the Requirements Covering Time, both functional and non-functional, Additional Greedy showed the best performance. As for the FRCT metric, Additional Greedy showed a performance increase in 43 out of 57 cases. Furthermore, regarding the NFRCT metric, the performance of the Additional Greedy algorithm increased in 16 out of 19 cases as the test suite size increased.

The performance of most of the algorithms decreased with the test suite size in the DC Engine case study for most of the evaluation metrics. The main reason for this could be the amount of functional and non-functional requirements of this case study. Notice that in order to assess the scalability of the approach, we included from 20 to 40 artificial functional requirements and from 45 to 80 artificial non-functional requirements in each product. The amount of requirements in the DC Engine case study is considerably higher than the number of requirements in the remaining three case studies. However, despite this, the algorithm showing best performance in detecting faults (i.e., PLSA) did not decrease its performance with statistical significance in terms of the FDT. Something similar happens for the Additional Greedy algorithm, which showed the best performance in terms of the FRCT and NFRCT; for the FRCT, the performance of the Additional Greedy increased with statistical significance for three out of four products whereas for the NFRCT, its performance increased with statistical significance in the four products. These results suggest that the selected algorithms are scalable for their purpose. However, in the future we foresee to investigate other techniques that would allow search algorithms to scale in these complex cases. From the results of our experiments, we can answer the RQ 3 as follows:

The increment of test cases showed a positive impact on the performance of the PLSA algorithm for solving the test case prioritization approach when finding faults as quick as possible (FDT and APFD metrics). In addition, the increment of test cases showed a positive impact on the performance of the Additional Greedy algorithm for covering Functional and Non-Functional Requirements (FRCT and NFRCT metrics). Thus, we can conclude that the selected search algorithms along with our defined fitness function are scalable to address the test case prioritization problems with increasing complexity.

7. Threats to Validity

7.1. Internal Validity

Internal validity threats exist when the results can be influenced by internal factors (e.g., parameters of the algorithms) [32]. An internal validity threat of our study could be that the configurations related to global search algorithms (e.g., population size, crossover rate, etc.) were not changed. However, the selected settings are in accordance with the common guidelines in the literature and other studies related to the application of search algorithms to testing [28, 56]. Another internal validity threat refers to the injected faults. In our study we have employed 20 mutants, which is a limited number, and thus, it might not be enough to represent all possible faults within the system. Notice that in our context, each mutant includes not only the software, but also the physical layer of the CPS, what makes the execution of the simulation time consuming. In addition, some studies have discovered that the subsumed mutants are one of the threats of this technique to assess the quality of test suites [61]. We have tried to mitigate these threats by distributing different types of faults (i.e., different mutation operators) distributed across the whole system (i.e., the physical and cyber layer of the CPSs).

7.2. Conclusion Validity

A *conclusion validity* threat in most of the evaluations where search algorithms are employed involves the random variations of the results [28]. To reduce this threat, we divided the experiment in different artificial problems and repeated the execution of each algorithm for each artificial problem 50 times to account for random variations. Moreover, we compared the results of the algorithms by applying statistical analysis.

7.3. Construct Validity

The *construct validity* threat is that the measures used are not comparable across the algorithms. However, we used the same stopping criterion for all the algorithms, i.e., the number of fitness evaluations (50,000 times), which is a comparable measure across all the algorithms.

7.4. External Validity

External validity threats appear when the outcome of results are influenced by external factors [32]. An *external validity* threat with any experiment is related to the generalization of results. We used four case studies, which might not be enough to ensure that our results can be extrapolated to all CPS product lines. However, to reduce this threat we used four case studies from different domains with different sizes and different characteristics to assure a sufficient degree of heterogeneity. In addition, one of the case studies was a real-world industrial case study.

8. Related Work

8.1. Product Line Engineering Testing

Testing product lines has gained important attention from the research community in the last few years. Several literature reviews and mapping studies reveal that most of the work on product line engineering testing focuses on the domain engineering level [26, 65, 66, 67, 68]. This is, to a large extent, caused by the problem of the huge number of possible products that the product lines can be set to. This leads to the problem of it not being possible to test every single product of the product line. As a result, most of the works in the product line engineering testing community focuses on the efficient derivation of products under test by employing CIT techniques [69]. To

this end, a small subset of products are generated by employing several criteria, such as the pairwise coverage (i.e., the interaction of two features at least once) [70, 71, 38, 6, 72, 73, 74, 75].

Similarly, at the domain engineering level, several studies aimed to optimize the testing process by prioritizing the execution of products under test. Sanchez et al. compared several functional test prioritization criteria (such as the product size or product complexity) to increase the fault detection rate [25]. The same authors extended their work to the drupal case study by including non-functional prioritization criteria (e.g., number of changes in the assets) [76]. Other works have focused on the prioritization of products employing search algortihms [64, 6], or by comparing the similarity of products [77, 78]. Other prioritization strategies include using domain knowledge to extract desirable features and prioritize products including these ones [79]. Devroy et al. employed statistical testing to prioritize configurations based on their probability to be executed [80, 81]. Compared to these studies, rather than prioritizing products of the product line, we prioritize the test cases that tests them (i.e., we perform the test case prioritization at the application engineering level).

Although most of the product line engineering testing studies focus on the domain engineering level, some other studies have proposed optimization of product lines at the application engineering level. As for search-based software engineering, Wang et al. proposed a test suite minimization approach for reducing test cases of SPLs [27, 28]. The same authors proposed a multi-objective approach for test case prioritization of SPLs, where three different weight-based search algorithms were compared [29]. Apart from search-based software engineering, other works have employed model-based techniques for the optimization of product line engineering testing at the application engineering level. Stricker et al. considered a model-based approach that employed data-flow dependencies to select test cases for customer-specific products with the objective of avoiding redundant test activities [82]. Other works have combined feature models with other elements such as "component family models" to systematically select test cases [44, 83]. Our approach builds upon these works by: (1) prioritizing test cases rather than minimizing and selecting, (2) applying it in the CPS product line engineering context, which faces several particularities such as the multi-level simulation-based testing (MiL, SiL and HiL test levels), reduction of simulation time by reducing the initialization time of test cases based on the test prioritization or testing functional and non-functional requirements, and (3) evaluating the performance of several algorithms using simulation and mutation testing.

8.2. Test Case Prioritization

Test case prioritization has been widely applied in the software engineering field [84, 85]. The problem was formally defined by Rothermel et al., where six techniques (four of them coverage-based and two of them estimated ability to reveal faults) were compared [86, 87]. Since then, several studies have been performed by the software engineering community in the test case prioritization context. According to a systematic mapping study performed by Catal and Mishra, most of the papers used only coverage-based prioritization methods [48]. In addition, many studies have used the Greedy algorithm for the test case prioritization problem; the use of metaheuristic and evolutionary algorithms for prioritizing test cases was first introduced by Li et al. in 2007 [39]. Since then, many techniques have been empirically evaluated for the test case prioritization problem, such as, comparison between different evolutionary multi-objective algorithms [88], comparison between white-box and black-box test prioritization techniques [89], or comparison between static and dynamic test prioritization techniques [90].

Although most of the studies on test case prioritization have proposed the use of coverage-based methods, in the last few years, importance has been given to test execution time. Malishevsky et al. combined test coverage information with test execution time for testing software systems [91]. A genetic algorithm to prioritize regression test suites that will always run within a given time budget and will have the highest possible potential for defect detection based on coverage information was proposed by Walcott et al. [63]. A similar study, but employing Integer Lineal Programming techniques was proposed by Zhang et al. [92]. Marijan et al. proposed the use of test execution time, along with historical failure data, to prioritize test cases for continuous regression testing [93]. Srikanth et al. proposed a prioritization scheme based on the setup time of different configurations of a system [94]. Knauss et al. employed a combination of test case failures and source code changes to prioritize system level test cases [95]. Elbaum et al. used the concept of time windows to track the execution time of test suites and combined with occurrence of failures to prioritize test cases in a continuous integration development environment [96]. Hemmati et al. found that the use of previously failed test cases were a good source for prioritizing test cases [97]. Notice that all these studies focused on testing software systems. On a much smaller scale, apart from purely software systems, test case prioritization has also been applied into other areas. For instance, Zhai et al. used test case prioritization for regression testing of location-based services (i.e., services with positional data) [98].

Recent approaches have also tackled the problem of test prioritization in the context of CPSs. In our previous work, we proposed a method to generate reactive test cases and prioritize it based on similarity measures [99], i.e., without considering historical data, unlike this paper. Shin et al., proposed a multi-objective test prioritization approach for testing CPS from the satellite domain from a functional perspective considering uncertainties [100]. Similar to this paper, both papers consider the problem of test execution time being dependent on the test order, as introduced in the conference version of this paper [10].

When compared to the current test case prioritization techniques, our paper faces some differences. Notice that we aim at prioritizing test cases for CPSs employing simulation, which requires multi-level testing (i.e., MiL, SiL and HiL test levels), and each of the test levels have their corresponding test objectives (e.g., the HiL test level includes non-functional requirements coverage). The test cases employed in this study are designed for testing CPS product line products at the system engineering level. These test cases take from seconds to

minutes to be executed, unlike unit testing, where the test execution time of each test case is in the order of some milliseconds [9]. In addition, the types of test cases we use are reactive test cases, which have some particularities, such as the varying test execution time depending on the previously prioritized test case. This is something that, to the best of our knowledge, is not considered in other test case prioritization methods; this allows for the reduction of the overall simulation time. Moreover, we evaluated the performance of the selected algorithms using simulation and mutation testing. Notice that the use of mutation testing in this context is especially challenging and expensive due to the fact that the physical layer, which is typically modeled with complex mathematical models, has to be simulated in each generated mutant, which makes the simulation time very long. Lastly, we integrated the test case prioritization approach with a variability modeling tool named FeatureIDE [43], so that when a specific product configuration is selected to test, its requirements are automatically selected. This allows for a systematic and fully automated test prioritization for the CPS product line engineering context.

9. Conclusion and Future Work

This paper proposes a search-based test case prioritization approach that optimizes the process of testing CPS product lines by reducing the fault detection time, simulation time and requirements covering time. To this end, we defined a fitness function that employed five objectives. Four different case studies were used to compare the performance of different algorithms. Based on the results of our empirical evaluation, we observed that the selected four search algorithms outperformed RS. Moreover, we discovered that the local search algorithms (i.e., PLSA and Additional Greedy) performed better than the global search algorithms (WBGA and RWGA). Specifically, the PLSA algorithm showed the best performance for fault detection and reduction of simulation time, whereas the Additional Greedy algorithm performed better to reduce the requirements covering time.

The proposed search-based test case prioritization approach has been integrated within our framework for testing CPS product lines employing simulation-based testing [5]. This framework includes (1) a test case generator [23], (2) a test system generator that automatically generates test systems instances to test products of CPS product lines in Simulink [12] and (3) a set of test case selection algorithms [46]. The proposed approach is the last step to optimally test CPS product lines. The framework relies on feature models for managing the variability. In the future we would like to adapt the different parsers of our test framework to be capable of employing other variability modeling notations more appropriate for the CPS context.

The empirical evaluation of Pareto-based multi-objective search algorithms remains as future work. We already have launched some preliminary experiments of two well known multi-objective search algorithms (NSGA-II and SPEA2). However, Pareto-based multi-objective search algorithms return a set of solutions and evaluating them with mutation testing is extremely expensive. In the future, we foresee to perform the evaluation of Pareto-based multi-objective algorithms with several clusters. Furthermore, we would like to empirically analyze the impact of assigning different weights on the objective functions of the defined three fitness functions.

Material

For the sake of replicability, all the experimental material can be found available in the following URL: https://tinyurl.com/JSSArrieta2018

Acknowledgments

Aitor Arrieta, Goiuria Sagardui and Leire Etxeberria are part of the embedded systems group of Mondragon Goi Eskola Politeknikoa, supported by the Department of Education, Universities and Research of the Basque Government. Shuai Wang is jointly supported by the Research Council of Norway (RCN) funded Certus SFI, RFF Hovedstaden funded MBE-CR project and COST Action CA15140 (ImAppNIO).

References

[1] E. A. Lee, S. A. Seshia, Introduction to Embedded Systems - A Cyber-Physical Systems Approach, 2nd Edition, Lee and Seshia, 2015.

[2] P. Derler, E. A. Lee, A. Sangiovanni-Vincentelli, Modeling cyber-physical systems, Proceedings of the IEEE (special issue on CPS) 100 (1) (2011) 13 – 28.

[3] E. A. Lee, Cyber physical systems: Design challenges, in: Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, IEEE, 2008, pp. 363–369.

[4] J. Shi, J. Wan, H. Yan, H. Suo, A survey of cyber-physical systems, in: Proc. of the Int. Conf. on Wireless Communications and Signal Processing, 2011, pp. 1–6.

[5] A. Arrieta, G. Sagardui, L. Etxeberria, Test control algorithms for the validation of cyber-physical systems product lines, in: Proceedings of the 19th International Conference on Software Product Line, SPLC '15, ACM, New York, NY, USA, 2015, pp. 273–282. doi:10.1145/2791060.2791095.

[6] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, Y. Le Traon, Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines, IEEE Trans. Software Eng. 40 (7) (2014) 650–670.

[7] L. Briand, S. Nejati, M. Sabetzadeh, D. Bianculli, Testing the untestable: Model testing of complex software-intensive systems, in: Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, ACM, 2016, pp. 789–792. doi:10.1145/2889160.2889212.

[8] H. Shokry, M. Hinchey, Model-based verification of embedded software, Computer 42 (4) (2009) 53 – 59.

[9] A. Arcuri, M. Z. Iqbal, L. Briand, Black-box system testing of real-time embedded systems using random and search-based testing, in: Proceedings of the 22Nd IFIP WG 6.1 International Conference on Testing Software and Systems, ICTSS'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 95–110.

[10] A. Arrieta, S. Wang, G. Sagardui, L. Etxeberria, 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, 2016, pp. 1053–1060. doi:10.1145/2908812.2908871.

[11] T. Yue, S. Ali, B. Selic, Cyber-physical system product line engineering: comprehensive domain analysis and experience report, in: Proceedings of the 19th International Conference on Software Product Line, ACM, 2015, pp. 338–347.

[12] A. Arrieta, G. Sagardui, L. Etxeberria, J. Zander, Automatic generation of test system instances for configurable cyber-physical systems, Software Quality Journal 25 (3) (2017) 1041–1083.
[13] S. A. Safdar, T. Yue, S. Ali, H. Lu, Evaluating variability modeling techniques for supporting cyber-physical system product line engineering, in: International Conference on System Analysis and Modeling, Springer International Publishing, 2016, pp. 1–19.
[14] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, A. Wasowski, Clafer: unifying class and feature modeling, Software & Systems Modeling (2015) 1–35.
[15] H. Lu, T. Yue, S. Ali, L. Zhang, Model-based incremental conformance checking to enable interactive product configuration, Information and Software Technology 72 (2016) 68–89.
[16] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) (2010) 615 – 636.
[17] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Variability Modelling of Software-intensive Systems (VaMoS), 2013, pp. 7:1–7:8.
[18] D. Batory, Feature models, grammars, and propositional formulas, in: Proceedings of the 9th International Conference on Software Product Lines, SPLC'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 7–20.
[19] R. Matinnejad, S. Nejati, L. C. Briand, T. Bruckmann, 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, 2016, pp. 595–606.
[20] J. C. Eidson, E. A. Lee, S. Matic, Distributed real-time software for cyber-physical systems, Proceedings of the IEEE 100 (2011) 45–59.
[21] A. Kane, T. E. Fuhrman, P. Koopman, Monitor based oracles for cyber-physical system testing: Practical experience report, in: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014, 2014, pp. 148–155.
[22] M. Zhang, S. Ali, T. Yue, R. Norgre, Uncertainty-wise evolution of test ready models, Information and Software Technology 87 (2017) 140–159.
[23] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, L. Etxeberria, Search-based test case generation for cyber-physical systems, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, 2017, pp. 688–697.
[24] R. Kuhn, R. Kacker, Y. Lei, J. Hunter, Combinatorial software testing, Computer 42 (2009) 94–96.
[25] A. B. Sánchez, S. Segura, A. Ruiz-Cortés, A comparison of test case prioritization criteria for software product lines, in: IEEE International Conference on Software Testing, Verification, and Validation, 2014, pp. 41–50.
[26] R. E. Lopez-Herrejon, L. Linsbauer, A. Egyed, A systematic mapping study of search-based software engineering for software product lines, Information and Software Technology 61 (2015) 33 – 51.
[27] S. Wang, S. Ali, A. Gotlieb, Minimizing test suites in software product lines using weight-based genetic algorithms, in: Proceedings of the 2013 Genetic and Evolutionary Computation Conference, Amsterdam, Netherlands, 2013, pp. 1493 – 1500.
[28] S. Wang, S. Ali, A. Gotlieb, Cost-effective test suite minimization in product lines using search techniques, Journal of Systems and Software 103 (0) (2015) 370 – 391.
[29] S. Wang, D. Buchmann, S. Ali, A. Gotlieb, D. Pradhan, M. Liaaen, 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, 2014, pp. 32–41. doi:10.1145/2648511.2648515.
[30] D. Greer, G. Ruhe, Software release planning: an evolutionary and iterative approach, Information and Software Technology 46 (4) (2004) 243 – 253.
[31] W. Miller, D. L. Spooner, Automatic generation of floating-point test data, IEEE Transactions on Software Engineering (3) (1976) 223–226.
[32] S. Ali, L. C. Briand, H. Hemmati, R. K. Panesar-Walawege, 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.
[33] A. Arcuri, G. Fraser, On the effectiveness of whole test suite generation, in: International Symposium on Search Based Software Engineering, Springer, 2014, pp. 1–15.
[34] G. Fraser, A. Arcuri, Whole test suite generation, IEEE Transactions on Software Engineering 39 (2) (2013) 276–291.
[35] J. Brownlee, Cleverl Algorithms: Nature-Inspired Programming Recipes, lulu.com, 2012.
[36] S. Wang, Systematic product line testing: Methodologies, automation, and industrial application, Ph.D. thesis, University of Oslo (2015).
[37] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & operations research 24 (11) (1997) 1097–1100.
[38] M. B. Cohen, M. B. Dwyer, J. Shi, Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach, IEEE Trans. Softw. Eng. 34 (5) (2008) 633–650. doi:10.1109/TSE.2008.50.
URL http://dx.doi.org/10.1109/TSE.2008.50
[39] Z. Li, M. Harman, R. M. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on software Engineering 33 (4) (2007) 225–237.
[40] J. Kempka, P. McMinn, D. Sudholt, Design and analysis of different alternating variable searches for search-based software testing, Theoretical Computer Science 605 (2015) 1–20.
[41] P. McMinn, G. M. Kapfhammer, Avmf: An open-source framework and implementation of the alternating variable method, in: International Symposium on Search Based Software Engineering, Springer, 2016, pp. 259–266.
[42] J. Campos, Y. Ge, G. Fraser, M. Eler, A. Arcuri, An empirical evaluation of evolutionary algorithms for test suite generation, in: Symposium on Search-Based Software Engineering, 2017.
[43] T. Thuem, C. Kastner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, Featureide: An extensible framework for feature-oriented software development, Science of Computer Programming 79 (2014) 70 – 85.
[44] S. Wang, S. Ali, A. Gotlieb, M. Liaaen, A systematic test case selection methodology for product lines: results and insights from an industrial case study, Empirical Software Engineering (2016) 1–37.
[45] Pure-Systems, pure::variants, http://www.pure-systems.com (July 2014).
[46] A. Arrieta, S. Wang, G. Sagardui, L. Etxeberria, 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, 2016, pp. 297–306.
[47] A. Konak, D. W. Coit, A. E. Smith, Multi-objective optimization using genetic algorithms: A tutorial, Reliability Engineering & System Safety 91 (9) (2006) 992–1007.
[48] C. Catal, D. Mishra, Test case prioritization: A systematic mapping study, Software Quality Journal 21 (3) (2013) 445–478. doi:10.1007/s11219-012-9181-z.
[49] S. Wang, S. Ali, T. Yue, Y. Li, M. Liaaen, A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, 2016, pp. 631–642. doi:10.1145/2884781.2884880.
[50] S. Wang, S. Ali, T. Yue, Ø. Bakkeli, M. Liaaen, Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search, in: Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on, IEEE, 2016, pp. 182–191.
[51] S. Wang, S. Ali, T. Yue, M. Liaaen, Upmoa: An improved search algorithm to support user-preference multi-objective optimization, in: Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, IEEE, 2015, pp. 393–404.
[52] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (6) (2009) 957–976.
[53] S. Elbaum, A. G. Malishevsky, G. Rothermel, Test case prioritization: A family of empirical studies, IEEE transactions on software engineering 28 (2) (2002) 159–182.
[54] U. Markiegi, A. Arrieta, G. Sagardui, L. Etxeberria, Search-based product line fault detection allocating test cases iteratively, in: Proceedings of the 21st International Systems and Software Product Line Conference - Volume A, SPLC '17, ACM, New York, NY, USA, 2017, pp. 123–132. doi:10.1145/3106195.3106210.
URL http://doi.acm.org/10.1145/3106195.3106210

[55] A. Arrieta, Cyber-physical system product lines case studies for test experimentation, Tech. rep., University of Mondragon (2018). doi:https://tinyurl.com/CPSTechReport.
URL https://tinyurl.com/CPSTechReport

[56] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Software Engineering (ICSE), 2011 33rd International Conference on, IEEE, 2011, pp. 1–10.

[57] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, Software Engineering, IEEE Transactions on 37 (5) (2011) 649–678.

[58] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, G. Fraser, 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, 2014, pp. 654–665.

[59] B. Liu, L. Lucia, S. Nejati, L. Briand, 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), 2017.

[60] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development., in: ICSOFT, 2013, pp. 251–257.

[61] M. Papadakis, C. Henard, M. Harman, Y. Jia, Y. L. Traon, Threats to the validity of mutation-based test assessment, in: International Symposium on Software Testing and Analysis (ISSTA'16), 2016, pp. 354–365.

[62] L. T. M. Hanh, N. T. Binh, K. T. Tung, 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.

[63] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos, Time-aware test suite prioritization, in: Proceedings of the 2006 International Symposium on Software Testing and Analysis, ISSTA '06, ACM, New York, NY, USA, 2006, pp. 1–12. doi:10.1145/1146238.1146240.

[64] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. E. Lopez-Herrejon, A. Egyed, Multi-objective test case prioritization in highly configurable systems: A case study, Journal of Systems and Software (2016) –.

[65] P. A. d. M. S. Neto, I. do Carmo Machado, J. D. McGregor, E. S. De Almeida, S. R. de Lemos Meira, A systematic mapping study of software product lines testing, Information and Software Technology 53 (5) (2011) 407–423.

[66] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, E. S. De Almeida, On strategies for testing software product lines: A systematic literature review, Information and Software Technology 56 (10) (2014) 1183–1199.

[67] E. Engström, P. Runeson, Software product line testing–a systematic mapping study, Information and Software Technology 53 (1) (2011) 2–13.

[68] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, Y. Zhang, Search based software engineering for software product line engineering: a survey and directions for future work, in: Proceedings of the 18th International Software Product Line Conference-Volume 1, ACM, 2014, pp. 5–18.

[69] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, A. Egyed, A first systematic mapping study on combinatorial interaction testing for software product lines, in: Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on, IEEE, 2015, pp. 1–10.

[70] G. Perrouin, S. Sen, J. Klein, B. Baudry, Y. Le Traon, Automated and scalable t-wise test case generation strategies for software product lines, in: 2010 Third international conference on software testing, verification and validation, IEEE, 2010, pp. 459–468.

[71] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, Y. Le Traon, Pairwise testing for software product lines: Comparison of two approaches, Software Quality Journal 20 (3-4) (2012) 605–643.

[72] C. Henard, M. Papadakis, M. Harmany, Y. L. Traon, Combining multi-objective search and constraint solving for configuring large scale software product lines, in: 37th International Conference on Software Engineering (ICSE'15), 2015, pp. 517–528.

[73] R. M. Hierons, M. Li, X. Liu, S. Segura, W. Zheng, Sip: Optimal product selection from feature models using many-objective evolutionary optimization, ACM Transactions on Software Engineering and Methodology (TOSEM) 25 (2) (2016) 17.

[74] M. F. Johansen, O. Haugen, F. Fleurey, An algorithm for generating t-wise covering arrays from large feature models, in: Proceedings of the 16th International Software Product Line Conference - Volume 1, SPLC '12, ACM, New York, NY, USA, 2012, pp. 46–55.

[75] S. Oster, I. Zorcic, F. Markert, M. Lochau, Moso-polite - tool support for pairwise and model-based software product line testing, in: VaMoS, 2011, pp. 79–82.

[76] A. B. Sánchez, S. Segura, J. A. Parejo, A. Ruiz-Cortés, Variability testing in the wild: the drupal case study, Software & Systems Modeling (2015) 1–22.

[77] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, G. Saake, Similarity-based prioritization in software product-line testing, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 197–206. doi:10.1145/2648511.2648532.

[78] M. Al-Hajjaji, T. Thüm, M. Lochau, J. Meinicke, G. Saake, Effective product-line testing using similarity-based product prioritization, Software & Systems Modeling (2016) 1–23.

[79] A. Ensan, E. Bagheri, M. Asadi, D. Gasevic, Y. Biletskiy, Goal-oriented test case selection and prioritization for product line feature models, in: Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations, ITNG '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 291–298. doi:10.1109/ITNG.2011.58.
URL http://dx.doi.org/10.1109/ITNG.2011.58

[80] X. Devroey, G. Perrouin, M. Cordy, P.-Y. Schobbens, A. Legay, P. Heymans, Towards statistical prioritization for software product lines testing, in: Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems, ACM, 2014, p. 10.

[81] X. Devroey, G. Perrouin, M. Cordy, H. Samih, A. Legay, P.-Y. Schobbens, P. Heymans, Statistical prioritization for software product line testing: an experience report, Software & Systems Modeling (2015) 1–19.

[82] V. Stricker, A. Metzger, K. Pohl, Avoiding redundant testing in application engineering, in: Software Product Lines: Going Beyond, Springer, 2010, pp. 226–240.

[83] S. Wang, A. Gotlieb, S. Ali, M. Liaaen, Automated test case selection using feature model: An industrial case study, in: MoDELS, 2013, pp. 237–253.

[84] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11:1–11:61.

[85] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability 22 (2) (2012) 67–120.

[86] G. Rothermel, R. H. Untch, C. Chu, M. J. Harrold, Test case prioritization: An empirical study, in: Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, IEEE, 1999, pp. 179–188.

[87] G. Rothermel, R. H. Untch, C. Chu, M. J. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on software engineering 27 (10) (2001) 929–948.

[88] M. G. Epitropakis, S. Yoo, M. Harman, E. K. Burke, 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, 2015, pp. 234–245.

[89] C. Henard, M. Papadakis, M. Harman, Y. Jia, Y. Le Traon, Comparing white-box and black-box test prioritization, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 523–534.

[90] Q. Luo, K. Moran, D. Poshyvanyk, A large-scale empirical comparison of static and dynamic test case prioritization techniques, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2016, pp. 559–570.

[91] A. G. Malishevsky, G. Rothermel, S. Elbaum, Modeling the cost-benefits tradeoffs for regression testing techniques, in: In Proceedings of the International Conference on Software Maintenance, 2002, pp. 204–213.

[92] L. Zhang, S.-S. Hou, C. Guo, T. Xie, H. Mei, Time-aware test-case prioritization using integer linear programming, in: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA '09, ACM, New York, NY, USA, 2009, pp. 213–224. doi:10.1145/1572272.1572297.
URL http://doi.acm.org/10.1145/1572272.1572297

[93] D. Marijan, A. Gotlieb, S. Sen, 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, 2013, pp. 540–543.

[94] H. Srikanth, M. B. Cohen, X. Qu, Reducing field failures in system configurable software: Cost-based prioritization, in: Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE'09), IEEE Computer Society, 2009, pp. 61–70.

[95] E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, M. Castell, 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, 2015, pp. 19–25.

[96] S. Elbaum, G. Rothermel, J. Penix, 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, 2014, pp. 235–245.

[97] H. Hemmati, Z. Fang, M. V. Mäntylä, Prioritizing manual test cases in traditional and rapid release environments, in: Proceedings of the 8th International Conference on Software Testing, Verification and Validation (ICST'15), 2015, pp. 1–10.

[98] K. Zhai, B. Jiang, W. K. Chan, Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study, IEEE Trans. Serv. Comput. 7 (1) (2014) 54–67. doi:10.1109/TSC.2012.40.

[99] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, L. Etxeberria, 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.

[100] S. Y. Shin, S. Nejati, M. Sabetzadeh, L. Briand, F. Zimmer, 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), 2018.

Appendix A. Local search algorithms

PTS = {};
TS = TestSuite; //Original test suite array (from 1 to number of test cases)
for *i=1 to length(TS)* **do**
    F(i) = CalculateFitness(TS(i)); //calculates fitness value for the test case in position i
end
PTS = sort (TS,F); //Sorts TS in ascending order based on the fitness value;

Algorithm 2: Pseudocode of the total greedy algorithm

Appendix B. Summary of the performed statistical analysis in the empirical evaluation

PTS = {};
TS = TestSuite; //Original test suite array (from 1 to number of test cases)
for *i=1 to length(TS)* **do**
    F_Best = 1;
    for *j=1 to length(TS)* **do**
        F = CalculateFitness(PTS ∪ TS(j));
        if $F < F\_Best$ **then**
            F_Best = F;
            BestTestCase = TS(j);
        end
    end
    PTS = PTS ∪ BestTestCase;
    Remove(BestTestCase, TS); // Removes the test case from the original test suite
end

Algorithm 3: Pseudocode of the Additional greedy algorithm for permutation

x = random(TSsize); //randomly generate a solution of permutations
NonSelectedTestCases = generateVector(TSsize); //generatesVector from 1 to Number of test cases in TS
i = 1;
c = 0;
n = TSSize;
while $c < n$ && *numberOfFitnessEvaluations* < *budget* **do**
    f = fitnessFunction(x);
    x' = localSearch();
    if *fitnessFunction(x')*<*fitnessFunction(x)* **then**
        $x = x'$;
        c = 0;
        NonSelectedTestCases.RemoveElement($x'(i)$);
    end
    else
        c=c + 1;
        NonSelectedTestCases.RemoveElement($x(i)$);
    end
    $i = (i \bmod n) + 1$;
end

Algorithm 4: Pseudocode of the Permutation-based Local Search Algorithm (PLSA)

```
f_localsearch_best=1;
x' = x;
for j=1 to sizeOf(NonSelectedTestCases) do
    x'' = x;
    TestCaseIn_i = x''(i);
    x''(i) = NonSelectedTestCases(j);
    for k=i+1 to sizeOf(x'') do
        if x''(k)==x''(i) then
            x''(k) = TestCaseIn_i ;
        end
        break;
    end
    f_localsearch = fitnessFunction(x'');
    if f_localsearch < f_localsearch_best then
        x' = x'';
        f_localsearch_best = f_localsearch;
    end
end
```

Algorithm 5: Pseudocode of the local search function for the PLSA

Table B.6: Summary for the Mann-Whiteney U-Test Statistical Test results for the FDT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

				MiL			SiL			HiL		
Case Study	**RQ**	**A**	**B**	**+**	**=**	**-**	**+**	**=**	**-**	**+**	**=**	**-**
ACC	RQ1	RS	A_GRE	26	14	10	27	12	11	18	16	16
		RS	PLSA	0	7	43	0	5	45	1	27	22
		RS	WBGA	0	14	36	0	10	40	2	31	17
		RS	RWGA	0	9	41	0	12	38	1	32	17
		RS	T_GRE	12	29	1	20	29	1	5	26	18
	RQ2	A_GRE	PLSA	1	6	43	0	8	42	11	10	29
		A_GRE	WBGA	6	2	42	5	3	42	11	11	28
		A_GRE	RWGA	5	4	41	5	2	43	11	14	25
		A_GRE	T_GRE	23	0	27	23	0	27	19	0	31
		PLSA	WBGA	28	22	0	26	24	0	21	29	0
		PLSA	RWGA	23	26	1	26	24	0	18	32	0
		PLSA	T_GRE	50	0	0	50	0	0	19	29	2
		WBGA	RWGA	0	49	1	0	43	7	1	48	1
		WBGA	T_GRE	50	0	0	50	0	0	9	30	11
		RWGA	T_GRE	50	0	0	50	0	0	11	33	6
UAV	RQ1	RS	A_GRE	4	3	43	4	4	42	5	6	39
		RS	PLSA	3	5	42	1	7	42	3	9	38
		RS	WBGA	1	15	34	1	15	34	3	10	37
		RS	RWGA	1	15	34	1	17	32	0	10	40
		RS	T_GRE	0	2	48	0	2	48	10	6	34
	RQ2	A_GRE	PLSA	18	16	16	18	19	13	20	19	11
		A_GRE	WBGA	38	5	7	35	9	6	33	10	7
		A_GRE	RWGA	37	9	4	37	8	5	27	15	8
		A_GRE	T_GRE	0	0	50	0	0	50	19	0	31
		PLSA	WBGA	33	16	1	33	13	4	19	26	5
		PLSA	RWGA	34	14	2	31	19	0	15	27	8
		PLSA	T_GRE	5	4	41	4	6	40	14	2	34
		WBGA	RWGA	0	49	1	3	44	3	1	42	7
		WBGA	T_GRE	0	2	48	0	3	47	10	6	34
		RWGA	T_GRE	0	3	47	0	0	50	11	5	33
TANK	RQ1	RS	A_GRE	3	9	38	3	8	39	14	15	21
		RS	PLSA	12	31	7	6	39	5	16	27	7
		RS	WBGA	19	26	5	15	32	3	30	19	1
		RS	RWGA	18	27	5	12	30	8	27	21	2
		RS	T_GRE	17	9	24	17	8	25	24	5	21
	RQ2	A_GRE	PLSA	37	10	3	37	10	3	31	6	13
		A_GRE	WBGA	47	3	0	48	2	0	40	7	3
		A_GRE	RWGA	43	4	3	42	5	3	34	13	3
		A_GRE	T_GRE	29	0	21	29	0	21	30	0	20
		PLSA	WBGA	10	38	2	13	37	0	17	33	0
		PLSA	RWGA	8	40	2	8	37	5	9	41	0
		PLSA	T_GRE	17	10	23	18	10	22	26	5	19
		WBGA	RWGA	0	46	4	0	44	6	0	46	4
		WBGA	T_GRE	14	11	25	9	19	22	19	11	20
		RWGA	T_GRE	15	10	25	18	10	22	20	10	20
DCEng	RQ1	RS	A_GRE	1	26	13	2	22	16	1	21	18
		RS	PLSA	0	0	40	0	0	40	0	0	40
		RS	WBGA	0	2	38	0	3	37	0	2	38
		RS	RWGA	0	6	34	0	2	38	0	2	38
		RS	T_GRE	1	11	28	2	9	28	0	4	36
	RQ2	A_GRE	PLSA	0	0	40	0	0	40	0	0	40
		A_GRE	WBGA	0	3	37	0	2	38	0	1	39
		A_GRE	RWGA	0	5	35	0	3	37	0	5	35
		A_GRE	T_GRE	5	0	35	5	0	35	2	0	38
		PLSA	WBGA	22	12	6	20	14	6	23	15	2
		PLSA	RWGA	21	12	7	20	16	4	22	13	5
		PLSA	T_GRE	37	3	0	36	4	0	27	13	0
		WBGA	RWGA	0	40	0	1	39	0	1	37	2
		WBGA	T_GRE	20	18	2	24	14	2	14	8	18
		RWGA	T_GRE	21	17	2	22	16	2	20	2	18

Table B.7: Results for the Spearman's rank correlation, which measures the correlation of the FDT metric with respect to the test suite size. Notice that a negative ρ means an improve in the performance of the algorithm with a larger test suite.

			Product 1		Product 2		Product 3		Product 4		Product 5	
			ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value
ACC	**MiL**	RS	0.102	0.02189	0.098	0.02898	0.073	0.10468	-0.009	0.84038	0.091	0.04247
		A_GRE	0.413	<0.00001	0.255	<0.00001	-0.213	<0.00001	-0.213	<0.00001	-0.213	<0.00001
		PLSA	-0.209	<0.00001	-0.225	<0.00001	-0.084	0.05981	-0.118	0.00815	-0.108	0.01568
		WBGA	-0.026	0.55910	-0.032	0.48022	0.003	0.94846	0.032	0.47356	0.023	0.60212
		RWGA	0.024	0.59179	0.032	0.47333	0.107	0.01699	0.082	0.06584	-0.003	0.95185
		T_GRE	0.522	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.201	<0.00001
	SiL	RS	0.122	0.00629	0.092	0.03874	0.110	0.01371	0.040	0.37555	0.027	0.55174
		A_GRE	0.413	<0.00001	0.255	<0.00001	-0.213	<0.00001	-0.213	<0.00001	-0.213	<0.00001
		PLSA	-0.156	0.00045	-0.216	<0.00001	-0.059	0.19010	-0.035	0.43311	-0.021	0.64149
		WBGA	-0.061	0.17464	-0.017	0.70785	-0.059	0.19029	0.000	0.99940	0.041	0.35958
		RWGA	0.002	0.96014	0.042	0.35371	0.076	0.08921	0.134	0.00274	0.114	0.01084
		T_GRE	0.522	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.201	<0.00001
	HiL	RS	-0.092	0.03970	-0.186	0.00003	0.027	0.55312	-0.056	0.21305	0.119	0.00793
		A_GRE	0.596	<0.00001	0.255	<0.00001	-0.323	<0.00001	-0.213	<0.00001	-0.213	<0.00001
		PLSA	-0.151	0.00073	-0.055	0.22292	-0.078	0.08203	-0.046	0.30965	0.009	0.84489
		WBGA	-0.039	0.38964	-0.073	0.10231	0.017	0.69939	-0.011	0.81038	0.036	0.41748
		RWGA	0.080	0.07283	-0.027	0.54689	-0.105	0.01931	0.058	0.19921	0.133	0.00283
		T_GRE	0.704	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.522	<0.00001	-0.038	0.401
UAV	**MiL**	RS	-0.288	<0.00001	0.005	0.90918	0.204	<0.00001	-0.168	0.00016	-0.094	0.03475
		A_GRE	*	*	*	*	*	*	*	*	*	*
		PLSA	-0.331	<0.00001	0.092	0.03925	-0.204	<0.00001	-0.520	<0.00001	-0.531	<0.00001
		WBGA	-0.147	0.00095	-0.002	0.96768	0.232	<0.00001	0.142	0.00147	0.006	0.88990
		RWGA	0.004	0.92804	-0.173	0.00010	0.201	0.00001	0.064	0.15447	-0.027	0.54200
		T_GRE	*	*	*	*	*	*	*	*	*	*
	SiL	RS	-0.325	<0.00001	-0.090	0.04474	0.217	<0.00001	-0.069	0.12323	-0.127	0.00457
		A_GRE	*	*	*	*	*	*	*	*	*	*
		PLSA	-0.335	<0.00001	-0.002	0.96853	-0.180	0.00005	-0.533	<0.00001	-0.526	<0.00001
		WBGA	-0.098	0.02881	0.025	0.58394	0.252	<0.00001	0.083	0.06466	0.061	0.17393
		RWGA	-0.109	0.01431	-0.036	0.42789	0.275	<0.00001	0.071	0.11062	0.067	0.13747
		T_GRE	*	*	*	*	*	*	*	*	*	*
	HiL	RS	-0.294	<0.00001	-0.267	<0.00001	0.222	<0.00001	0.118	0.00847	0.140	0.00173
		A_GRE	*	*	*	*	*	*	*	*	*	*
		PLSA	-0.684	<0.00001	-0.182	0.00004	0.027	0.54447	-0.040	0.37196	-0.137	0.00218
		WBGA	-0.201	0.00001	0.187	0.00003	0.278	<0.00001	0.069	0.12342	-0.025	0.58357
		RWGA	-0.221	<0.00001	0.019	0.67659	0.215	<0.00001	0.136	0.00236	0.017	0.69947
		T_GRE	-0.798	<0.00001	-0.798	<0.00001	-0.798	<0.00001	-0.809	<0.00001	*	*
TANK	**MiL**	RS	-0.4293	<0.00001	-0.4107	<0.00001	-0.4057	<0.00001	-0.3495	<0.00001	-0.4421	<0.00001
		A_GRE	-0.7979	<0.00001	-0.7979	<0.00001	-0.7979	<0.00001	-0.7531	<0.00001	-0.7531	<0.00001
		PLSA	-0.4350	<0.00001	-0.2490	<0.00001	-0.1702	0.00013	-0.5323	<0.00001	-0.2376	<0.00001
		WBGA	-0.1969	0.00001	-0.2479	<0.00001	-0.2355	<0.00001	-0.2734	<0.00001	-0.3617	<0.00001
		RWGA	-0.2727	<0.00001	-0.2255	<0.00001	-0.1762	0.00007	-0.3064	<0.00001	-0.3320	<0.00001
		T_GRE	-0.701	<0.00001	0.323	0.013	-0.070	0.12	0.350	6.94E-02	0.804	<0.00001
	SiL	RS	-0.3833	<0.00001	-0.3804	<0.00001	-0.3828	<0.00001	-0.2770	<0.00001	-0.4584	<0.00001
		A_GRE	-0.7979	<0.00001	-0.7979	<0.00001	-0.7979	<0.00001	-0.7531	<0.00001	-0.7531	<0.00001
		PLSA	-0.4116	<0.00001	-0.2194	<0.00001	-0.1817	0.00004	-0.5445	<0.00001	-0.2865	<0.00001
		WBGA	-0.1909	0.00002	-0.1781	0.00006	-0.2926	<0.00001	-0.3027	<0.00001	-0.3137	<0.00001
		RWGA	-0.1519	0.00065	-0.1867	0.00003	-0.2403	<0.00001	-0.2162	<0.00001	-0.3413	<0.00001
		T_GRE	-0.701	<0.00001	0.323	0.013	-0.070	0.12	0.350	6.94E-02	0.804	<0.00001
	HiL	RS	-0.4609	<0.00001	-0.2646	<0.00001	-0.2296	<0.00001	-0.4880	<0.00001	-0.4255	<0.00001
		A_GRE	-0.7979	<0.00001	-0.1427	0.00138	-0.1427	0.00138	-0.7680	<0.00001	-0.7680	<0.00001
		PLSA	-0.4631	<0.00001	-0.3054	<0.00001	-0.1392	0.00180	-0.4626	<0.00001	-0.4165	<0.00001
		WBGA	-0.3027	<0.00001	-0.1901	0.00002	-0.2059	<0.00001	-0.2890	<0.00001	-0.3795	<0.00001
		RWGA	-0.2582	<0.00001	-0.2390	<0.00001	-0.1606	0.00031	-0.3640	<0.00001	-0.3782	<0.00001
		T_GRE	0.716	<0.00001	0.665	<0.00001	0.716	<0.00001	-0.809	<0.00001	*	*
DC Eng.	**MiL**	RS	0.1670	0.00018	0.1211	0.00669	0.2253	<0.00001	0.2176	<0.00001		
		A_GRE	0.8909	0.00000	0.6727	<0.00001	0.9273	<0.00001	0.8667	<0.00001		
		PLSA	0.0529	0.23758	0.0741	0.09783	0.0325	0.46908	0.0058	0.89646		
		WBGA	0.6120	<0.00001	0.5979	<0.00001	0.6396	<0.00001	0.5733	<0.00001		
		RWGA	0.6264	<0.00001	0.5892	<0.00001	0.6264	<0.00001	0.5974	<0.00001		
		T_GRE	0.903	<0.00001	0.696	<0.00001	0.522	<0.00001	*	*		
	SiL	RS	0.1686	0.00015	0.2008	<0.00001	0.2829	<0.00001	0.3179	<0.00001		
		A_GRE	0.8909	<0.00001	0.6727	<0.00001	0.9273	<0.00001	0.8667	<0.00001		
		PLSA	0.0438	0.32880	0.0741	0.09808	0.0148	0.74143	0.0464	0.30066		
		WBGA	0.6195	<0.00001	0.6074	<0.00001	0.5725	<0.00001	0.5948	<0.00001		
		RWGA	0.6280	<0.00001	0.6037	<0.00001	0.6153	<0.00001	0.6187	<0.00001		
		T_GRE	0.903	<0.00001	0.696	<0.00001	0.522	<0.00001	*	*		
	HiL	RS	0.2472	<0.00001	0.2078	<0.00001	0.2325	<0.00001	0.2006	<0.00001		
		A_GRE	0.8909	<0.00001	0.4061	<0.00001	0.5273	<0.00001	0.7576	<0.00001		
		PLSA	-0.0513	0.25186	0.0467	0.29755	-0.0112	0.80207	0.0324	0.47019		
		WBGA	0.6195	<0.00001	0.6074	<0.00001	0.5725	<0.00001	0.5973	<0.00001		
		RWGA	0.6325	<0.00001	0.6324	<0.00001	0.6597	<0.00001	0.6564	<0.00001		
		T_GRE	1	<0.00001	-0.703	<0.00001	-0.881	<0.00001	-0.904	<0.00001		

Table B.8: Summary for the Mann-Whiteney U-Test Statistical Test results for the APFD metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

				MiL			SiL			HiL		
Case Study	**RQ**	**A**	**B**	**+**	**=**	**-**	**+**	**=**	**-**	**+**	**=**	**-**
ACC	RQ1	RS	A_GRE	50	0	0	50	0	0	50	0	0
		RS	PLSA	0	16	34	0	16	34	4	34	12
		RS	WBGA	0	36	14	1	32	17	18	22	10
		RS	RWGA	0	29	21	0	28	22	9	30	11
		RS	T_GRE	49	1	0	47	3	0	45	3	2
	RQ2	A_GRE	PLSA	0	0	50	0	0	50	0	0	50
		A_GRE	WBGA	0	0	50	0	0	50	0	1	49
		A_GRE	RWGA	0	0	50	0	0	50	0	0	50
		A_GRE	T_GRE	24	0	26	24	0	26	19	0	31
		PLSA	WBGA	23	27	0	20	30	0	22	28	0
		PLSA	RWGA	25	22	3	21	28	1	16	34	0
		PLSA	T_GRE	50	0	0	50	0	0	46	4	0
		WBGA	RWGA	1	46	3	2	45	3	1	49	0
		WBGA	T_GRE	50	0	0	50	0	0	45	4	1
		RWGA	T_GRE	50	0	0	50	0	0	46	4	0
UAV	RQ1	RS	A_GRE	9	8	33	10	9	31	12	8	30
		RS	PLSA	5	11	34	5	14	31	6	13	31
		RS	WBGA	2	9	39	1	11	38	4	15	31
		RS	RWGA	1	17	32	4	11	35	4	15	31
		RS	T_GRE	0	0	50	0	0	50	0	0	50
	RQ2	A_GRE	PLSA	7	30	13	7	27	16	7	25	18
		A_GRE	WBGA	4	26	19	8	23	19	6	26	18
		A_GRE	RWGA	9	25	16	8	25	17	2	31	17
		A_GRE	T_GRE	0	0	50	0	0	50	0	0	50
		PLSA	WBGA	3	28	19	5	31	14	5	31	14
		PLSA	RWGA	5	35	10	4	34	12	3	35	12
		PLSA	T_GRE	0	0	50	0	0	50	0	0	50
		WBGA	RWGA	2	48	0	1	47	2	0	49	1
		WBGA	T_GRE	0	0	50	0	0	50	0	0	50
		RWGA	T_GRE	0	0	50	0	0	50	0	0	50
TANK	RQ1	RS	A_GRE	40	7	3	43	3	3	36	10	4
		RS	PLSA	9	24	17	9	22	19	17	29	4
		RS	WBGA	19	24	7	17	29	4	34	16	0
		RS	RWGA	16	23	11	15	24	11	35	15	0
		RS	T_GRE	16	10	24	15	12	23	0	1	49
	RQ2	A_GRE	PLSA	3	8	39	3	7	40	5	13	32
		A_GRE	WBGA	5	12	32	4	14	32	10	18	22
		A_GRE	RWGA	4	9	37	5	10	35	8	15	27
		A_GRE	T_GRE	10	0	40	10	0	40	0	0	50
		PLSA	WBGA	23	27	0	23	27	0	18	32	0
		PLSA	RWGA	15	35	0	13	36	1	19	30	1
		PLSA	T_GRE	22	7	21	24	6	20	0	1	49
		WBGA	RWGA	0	49	1	0	46	4	0	49	1
		WBGA	T_GRE	9	14	27	14	10	26	0	0	50
		RWGA	T_GRE	11	13	26	13	12	25	0	0	50
DCEng	RQ1	RS	A_GRE	40	0	0	40	0	0	40	0	0
		RS	PLSA	0	0	40	0	0	40	0	0	40
		RS	WBGA	0	0	40	0	0	40	0	0	40
		RS	RWGA	0	0	40	0	0	40	0	0	40
		RS	T_GRE	40	0	0	40	0	0	40	0	0
	RQ2	A_GRE	PLSA	0	0	40	0	0	40	0	0	40
		A_GRE	WBGA	0	0	40	0	0	40	0	0	40
		A_GRE	RWGA	0	0	40	0	0	40	0	0	40
		A_GRE	T_GRE	17	0	23	17	0	23	17	0	23
		PLSA	WBGA	15	15	10	17	14	9	18	15	7
		PLSA	RWGA	14	17	9	15	15	10	18	15	7
		PLSA	T_GRE	40	0	0	40	0	0	40	0	0
		WBGA	RWGA	0	40	0	0	37	3	0	39	1
		WBGA	T_GRE	40	0	0	40	0	0	40	0	0
		RWGA	T_GRE	40	0	0	40	0	0	40	0	0

Table B.9: Results for the Spearman's rank correlation test, which measures the correlation of the APFD metric with respect to the test suite size. Notice that a positive ρ means an improve in the performance of the algorithm with a larger test suite.

			Product 1		Product 2		Product 3		Product 4		Product 5	
			ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value
ACC	MiL	RS	0.892	<0.00001	0.888	<0.00001	0.796	<0.00001	0.841	<0.00001	0.823	<0.00001
		A_GRE	0.976	<0.00001	0.891	<0.00001	0.939	<0.00001	0.939	<0.00001	0.939	<0.00001
		PLSA	0.956	<0.00001	0.964	<0.00001	0.889	<0.00001	0.863	<0.00001	0.862	<0.00001
		WBGA	0.923	<0.00001	0.941	<0.00001	0.876	<0.00001	0.854	<0.00001	0.862	<0.00001
		RWGA	0.922	<0.00001	0.936	<0.00001	0.855	<0.00001	0.834	<0.00001	0.853	<0.00001
		T_GRE	1	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001
	SiL	RS	0.883	<0.00001	0.903	<0.00001	0.742	<0.00001	0.865	<0.00001	0.836	<0.00001
		A_GRE	0.976	<0.00001	0.891	<0.00001	0.939	<0.00001	0.939	<0.00001	0.939	<0.00001
		PLSA	0.947	<0.00001	0.961	<0.00001	0.878	<0.00001	0.870	<0.00001	0.857	<0.00001
		WBGA	0.927	<0.00001	0.934	<0.00001	0.888	<0.00001	0.866	<0.00001	0.841	<0.00001
		RWGA	0.916	<0.00001	0.926	<0.00001	0.872	<0.00001	0.843	<0.00001	0.848	<0.00001
		T_GRE	1	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001
	HiL	RS	0.748	<0.00001	0.775	<0.00001	0.782	<0.00001	0.824	<0.00001	0.825	<0.00001
		A_GRE	0.818	<0.00001	0.758	<0.00001	0.915	<0.00001	0.891	<0.00001	0.818	<0.00001
		PLSA	0.895	<0.00001	0.789	<0.00001	0.795	<0.00001	0.813	<0.00001	0.812	<0.00001
		WBGA	0.856	<0.00001	0.762	<0.00001	0.739	<0.00001	0.767	<0.00001	0.747	<0.00001
		RWGA	0.844	<0.00001	0.772	<0.00001	0.754	<0.00001	0.768	<0.00001	0.768	<0.00001
		T_GRE	0.924	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001	1	<0.00001
UAV	MiL	RS	0.892	<0.00001	0.888	<0.00001	0.796	<0.00001	0.841	<0.00001	0.823	<0.00001
		A_GRE	0.976	<0.00001	0.891	<0.00001	0.939	<0.00001	0.939	<0.00001	0.939	<0.00001
		PLSA	0.956	<0.00001	0.964	<0.00001	0.889	<0.00001	0.863	<0.00001	0.862	<0.00001
		WBGA	0.923	<0.00001	0.941	<0.00001	0.876	<0.00001	0.854	<0.00001	0.862	<0.00001
		RWGA	0.922	<0.00001	0.936	<0.00001	0.855	<0.00001	0.834	<0.00001	0.853	<0.00001
		T_GRE	1	<0.00001	*	*	*	*	*	*	*	*
	SiL	RS	0.883	<0.00001	0.903	<0.00001	0.742	<0.00001	0.865	<0.00001	0.836	<0.00001
		A_GRE	0.976	<0.00001	0.891	<0.00001	0.939	<0.00001	0.939	<0.00001	0.939	<0.00001
		PLSA	0.947	<0.00001	0.961	<0.00001	0.878	<0.00001	0.870	<0.00001	0.857	<0.00001
		WBGA	0.927	<0.00001	0.934	<0.00001	0.888	<0.00001	0.866	<0.00001	0.841	<0.00001
		RWGA	0.916	<0.00001	0.926	<0.00001	0.872	<0.00001	0.843	<0.00001	0.848	<0.00001
		T_GRE	1	<0.00001	*	*	*	*	*	*	*	*
	HiL	RS	0.748	<0.00001	0.775	<0.00001	0.782	<0.00001	0.824	<0.00001	0.825	<0.00001
		A_GRE	0.818	<0.00001	0.758	<0.00001	0.915	<0.00001	0.891	<0.00001	0.818	<0.00001
		PLSA	0.895	<0.00001	0.789	<0.00001	0.795	<0.00001	0.813	<0.00001	0.812	<0.00001
		WBGA	0.856	<0.00001	0.762	<0.00001	0.739	<0.00001	0.767	<0.00001	0.747	<0.00001
		RWGA	0.844	<0.00001	0.772	<0.00001	0.754	<0.00001	0.768	<0.00001	0.768	<0.00001
		T_GRE	1.000	<0.00001	*	*	*	*	*	*	*	*
TANK	MiL	RS	0.936	<0.00001	0.903	<0.00001	0.900	<0.00001	0.916	<0.00001	0.888	<0.00001
		A_GRE	0.964	<0.00001	0.964	<0.00001	0.964	<0.00001	1.000	<0.00001	1.000	<0.00001
		PLSA	0.941	<0.00001	0.921	<0.00001	0.935	<0.00001	0.914	<0.00001	0.850	<0.00001
		WBGA	0.896	<0.00001	0.896	<0.00001	0.879	<0.00001	0.877	<0.00001	0.850	<0.00001
		RWGA	0.918	<0.00001	0.884	<0.00001	0.892	<0.00001	0.886	<0.00001	0.858	<0.00001
		T_GRE	1	<0.00001	0.903	<0.00001	0.927	<0.00001	0.927	<0.00001	0.709	<0.00001
	SiL	RS	0.923	<0.00001	0.895	<0.00001	0.895	<0.00001	0.904	<0.00001	0.890	<0.00001
		A_GRE	0.964	<0.00001	0.964	<0.00001	0.964	<0.00001	1.000	<0.00001	1.000	<0.00001
		PLSA	0.938	<0.00001	0.945	<0.00001	0.947	<0.00001	0.915	<0.00001	0.858	<0.00001
		WBGA	0.904	<0.00001	0.886	<0.00001	0.885	<0.00001	0.880	<0.00001	0.845	<0.00001
		RWGA	0.914	<0.00001	0.891	<0.00001	0.895	<0.00001	0.874	<0.00001	0.858	<0.00001
		T_GRE	1	<0.00001	0.903	<0.00001	0.927	<0.00001	0.927	<0.00001	0.709	<0.00001
	HiL	RS	0.959	<0.00001	0.925	<0.00001	0.923	<0.00001	0.914	<0.00001	0.838	<0.00001
		A_GRE	0.964	<0.00001	0.964	<0.00001	0.964	<0.00001	1.000	<0.00001	1.000	<0.00001
		PLSA	0.860	<0.00001	0.870	<0.00001	0.945	<0.00001	0.846	<0.00001	0.850	<0.00001
		WBGA	0.881	<0.00001	0.824	<0.00001	0.839	<0.00001	0.817	<0.00001	0.800	<0.00001
		RWGA	0.886	<0.00001	0.863	<0.00001	0.836	<0.00001	0.839	<0.00001	0.827	<0.00001
		T_GRE	01	<0.00001	0.964	<0.00001	1	<0.00001	1	<0.00001	0.964	<0.00001
DC Eng.	MiL	RS	0.502	<0.00001	0.492	<0.00001	0.387	<0.00001	0.362	<0.00001		
		A_GRE	0.770	<0.00001	0.891	<0.00001	0.939	<0.00001	0.624	<0.00001		
		PLSA	0.806	<0.00001	0.779	<0.00001	0.828	<0.00001	0.822	<0.00001		
		WBGA	0.646	<0.00001	0.557	<0.00001	0.615	<0.00001	0.588	<0.00001		
		RWGA	0.620	<0.00001	0.622	<0.00001	0.653	<0.00001	0.591	<0.00001		
		T_GRE	0.624	<0.00001	*	*	*	*	*	*		
	SiL	RS	0.485	<0.00001	0.441	<0.00001	0.391	<0.00001	0.329	<0.00001		
		A_GRE	0.770	<0.00001	0.891	<0.00001	0.939	<0.00001	0.624	<0.00001		
		PLSA	0.800	<0.00001	0.796	<0.00001	0.851	<0.00001	0.793	<0.00001		
		WBGA	0.560	<0.00001	0.578	<0.00001	0.579	<0.00001	0.603	<0.00001		
		RWGA	0.666	<0.00001	0.650	<0.00001	0.595	<0.00001	0.611	<0.00001		
		T_GRE	0.624	<0.00001	*	*	*	*	*	*		
	HiL	RS	0.295	<0.00001	0.283	<0.00001	0.246	<0.00001	0.247	<0.00001		
		A_GRE	0.770	<0.00001	0.891	<0.00001	0.939	<0.00001	0.612	<0.00001		
		PLSA	0.865	<0.00001	0.829	<0.00001	0.859	<0.00001	0.797	<0.00001		
		WBGA	0.560	<0.00001	0.578	<0.00001	0.579	<0.00001	0.591	<0.00001		
		RWGA	0.640	<0.00001	0.606	<0.00001	0.596	<0.00001	0.524	<0.00001		
		T_GRE	0.624	<0.00001	0.853	<0.00001	0.853	<0.00001	0.852	<0.00001		

Table B.10: Summary for the Mann-Whiteney U-Test Statistical Test results for the Simulation Time. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

				MiL			SiL			HiL		
Case Study	**RQ**	**A**	**B**	**+**	**=**	**-**	**+**	**=**	**-**	**+**	**=**	**-**
ACC	RQ1	RS	A_GRE	0	0	50	0	0	50	0	0	50
		RS	PLSA	0	0	50	0	0	50	0	0	50
		RS	WBGA	0	0	50	0	0	50	0	0	50
		RS	RWGA	0	0	50	0	0	50	0	0	50
		RS	T_GRE	14	19	17	14	18	18	32	10	8
	RQ2	A_GRE	PLSA	0	1	49	0	1	49	1	1	48
		A_GRE	WBGA	45	1	4	45	0	5	44	2	4
		A_GRE	RWGA	46	1	3	46	1	3	46	0	4
		A_GRE	T_GRE	50	0	0	50	0	0	50	0	0
		PLSA	WBGA	49	1	0	49	1	0	50	0	0
		PLSA	RWGA	50	0	0	50	0	0	50	0	0
		PLSA	T_GRE	50	0	0	50	0	0	50	0	0
		WBGA	RWGA	47	3	0	47	3	0	42	8	0
		WBGA	T_GRE	50	0	0	50	0	0	50	0	0
		RWGA	T_GRE	50	0	0	50	0	0	50	0	0
UAV	RQ1	RS	A_GRE	0	0	50	0	0	50	0	0	50
		RS	PLSA	0	0	50	0	0	50	0	0	50
		RS	WBGA	0	0	50	0	0	50	0	0	50
		RS	RWGA	0	0	50	0	0	50	0	0	50
		RS	T_GRE	43	6	1	43	6	1	40	3	7
	RQ2	A_GRE	PLSA	50	0	0	50	0	0	50	0	0
		A_GRE	WBGA	50	0	0	50	0	0	50	0	0
		A_GRE	RWGA	50	0	0	50	0	0	50	0	0
		A_GRE	T_GRE	50	0	0	50	0	0	50	0	0
		PLSA	WBGA	45	5	0	46	4	0	47	3	0
		PLSA	RWGA	48	2	0	48	2	0	47	3	0
		PLSA	T_GRE	50	0	0	50	0	0	50	0	0
		WBGA	RWGA	4	14	32	3	17	30	0	15	35
		WBGA	T_GRE	50	0	0	50	0	0	50	0	0
		RWGA	T_GRE	50	0	0	50	0	0	50	0	0
TANK	RQ1	RS	A_GRE	0	0	50	0	50	0	0	0	50
		RS	PLSA	0	0	50	0	50	0	0	0	50
		RS	WBGA	0	0	50	0	50	0	0	0	50
		RS	RWGA	0	0	50	0	50	0	0	0	50
		RS	T_GRE	43	6	1	43	6	1	40	3	7
	RQ2	A_GRE	PLSA	10	16	24	11	23	16	18	12	20
		A_GRE	WBGA	50	0	0	50	0	0	50	0	0
		A_GRE	RWGA	50	0	0	50	0	0	50	0	0
		A_GRE	T_GRE	50	0	0	50	0	0	50	0	0
		PLSA	WBGA	50	0	0	50	0	0	49	1	0
		PLSA	RWGA	50	0	0	50	0	0	48	2	0
		PLSA	T_GRE	50	0	0	50	0	0	50	0	0
		WBGA	RWGA	4	45	1	6	2	42	0	48	2
		WBGA	T_GRE	50	0	0	50	0	0	50	0	0
		RWGA	T_GRE	50	0	0	50	0	0	50	0	0
DCEng	RQ1	RS	A_GRE	18	5	17	19	4	17	16	0	24
		RS	PLSA	0	0	40	0	0	40	0	0	40
		RS	WBGA	0	0	40	0	0	40	0	0	40
		RS	RWGA	0	0	40	0	0	40	0	0	40
		RS	T_GRE	40	0	0	40	0	0	37	0	3
	RQ2	A_GRE	PLSA	0	0	40	0	0	40	0	0	40
		A_GRE	WBGA	0	0	40	0	0	40	0	0	40
		A_GRE	RWGA	0	0	40	0	0	40	0	0	40
		A_GRE	T_GRE	40	0	0	40	0	0	38	0	2
		PLSA	WBGA	31	9	0	30	10	0	21	19	0
		PLSA	RWGA	33	7	0	34	6	0	31	9	0
		PLSA	T_GRE	40	0	0	40	0	0	40	0	0
		WBGA	RWGA	7	33	0	6	34	0	31	9	0
		WBGA	T_GRE	40	0	0	40	0	0	40	0	0
		RWGA	T_GRE	40	0	0	40	0	0	40	0	0

Table B.11: Summary for the Mann-Whiteney U-Test Statistical Test results for the FRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

				MiL			SiL			HiL		
Case Study	**RQ**	**A**	**B**	**+**	**=**	**-**	**+**	**=**	**-**	**+**	**=**	**-**
ACC	RQ1	RS	A_GRE	26	15	9	23	18	9	23	14	13
		RS	PLSA	19	30	1	21	29	0	37	5	0
		RS	WBGA	14	36	0	8	42	0	31	7	1
		RS	RWGA	15	35	0	12	38	0	29	12	0
		RS	T_GRE	47	1	2	47	1	2	49	1	0
	RQ2	A_GRE	PLSA	16	11	23	19	8	23	28	16	7
		A_GRE	WBGA	13	19	18	13	19	18	32	15	8
		A_GRE	RWGA	13	21	16	11	22	17	29	18	8
		A_GRE	T_GRE	45	0	5	45	0	5	45	0	5
		PLSA	WBGA	2	37	11	4	35	11	1	29	7
		PLSA	RWGA	4	36	10	6	34	10	0	36	6
		PLSA	T_GRE	47	0	3	47	0	3	46	3	1
		WBGA	RWGA	2	46	2	1	46	3	0	35	2
		WBGA	T_GRE	47	1	2	47	1	2	47	2	1
		RWGA	T_GRE	47	2	1	47	2	1	49	0	1
UAV	RQ1	RS	A_GRE	21	7	21	22	4	23	29	5	16
		RS	PLSA	24	12	14	25	12	13	29	24	7
		RS	WBGA	32	10	8	30	11	9	29	16	5
		RS	RWGA	30	15	5	30	13	7	28	14	8
		RS	T_GRE	50	0	0	50	0	0	50	0	0
	RQ2	A_GRE	PLSA	18	4	28	18	4	28	12	8	30
		A_GRE	WBGA	21	6	23	21	6	23	20	7	23
		A_GRE	RWGA	21	9	20	21	7	22	19	6	25
		A_GRE	T_GRE	50	0	0	50	0	0	50	0	0
		PLSA	WBGA	14	31	5	14	28	8	17	27	6
		PLSA	RWGA	17	29	4	15	29	6	16	30	4
		PLSA	T_GRE	50	0	0	50	0	0	50	0	0
		WBGA	RWGA	1	49	0	0	50	0	5	42	3
		WBGA	T_GRE	50	0	0	48	2	0	48	2	0
		RWGA	T_GRE	50	0	0	49	1	0	50	0	0
TANK	RQ1	RS	A_GRE	5	4	41	7	2	41	0	7	43
		RS	PLSA	15	26	9	15	27	8	9	27	14
		RS	WBGA	2	29	19	2	31	17	17	22	11
		RS	RWGA	2	26	22	2	35	13	14	28	8
		RS	T_GRE	24	21	5	26	18	4	32	3	14
	RQ2	A_GRE	PLSA	41	6	3	39	7	4	48	1	1
		A_GRE	WBGA	39	4	7	42	3	5	41	7	1
		A_GRE	RWGA	39	6	5	40	5	5	44	5	1
		A_GRE	T_GRE	45	2	3	45	2	3	50	0	0
		PLSA	WBGA	1	31	18	3	24	23	17	23	10
		PLSA	RWGA	1	31	18	3	28	19	16	27	7
		PLSA	T_GRE	23	19	8	22	20	8	31	11	8
		WBGA	RWGA	0	49	1	1	47	2	0	50	0
		WBGA	T_GRE	39	9	2	38	11	1	32	9	9
		RWGA	T_GRE	40	7	3	39	9	2	31	10	9
DCEng	RQ1	RS	A_GRE	0	1	39	0	1	39	0	0	40
		RS	PLSA	40	0	0	40	0	0	40	0	0
		RS	WBGA	34	6	0	37	1	2	37	2	1
		RS	RWGA	35	5	0	36	3	1	36	3	1
		RS	T_GRE	14	1	25	14	3	23	23	3	14
	RQ2	A_GRE	PLSA	40	0	0	40	0	0	40	0	0
		A_GRE	WBGA	39	0	1	39	0	1	40	0	0
		A_GRE	RWGA	39	0	1	39	0	1	40	0	0
		A_GRE	T_GRE	40	0	0	40	0	0	40	0	0
		PLSA	WBGA	0	0	40	0	0	40	0	0	40
		PLSA	RWGA	0	1	39	0	0	40	0	0	40
		PLSA	T_GRE	12	0	28	12	0	28	15	1	24
		WBGA	RWGA	1	37	2	0	39	1	4	34	2
		WBGA	T_GRE	13	1	26	13	0	27	22	0	18
		RWGA	T_GRE	13	1	26	13	1	26	21	1	18

Table B.12: Results for the Spearman's rank correlation test, which measures the correlation of the FRCT metric with respect to the test suite size. Notice that a negative ρ means an improve in the performance of the algorithm with a larger test suite.

			Product 1		Product 2		Product 3		Product 4		Product 5	
			ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value	ρ	p-value
ACC	MiL	RS	0.052	0.24214	-0.020	0.65343	-0.285	<0.00001	-0.163	0.00026	-0.167	0.00017
		A_GRE	0.587	<0.00001	-0.222	<0.00001	0.128	0.00413	-0.116	0.00952	-0.116	0.00952
		PLSA	0.043	0.34097	0.191	0.00002	-0.248	<0.00001	-0.363	<0.00001	-0.387	<0.00001
		WBGA	-0.065	0.14734	-0.102	0.02296	-0.188	0.00002	-0.181	0.00004	-0.121	0.00675
		RWGA	-0.110	0.01364	-0.116	0.00953	-0.186	0.00003	-0.166	0.00020	-0.090	0.04352
		T_GRE	-0.683	<0.00001	-0.701	<0.00001	0.881	<0.00001	0.813	<0.00001	0.813	<0.00001
	SiL	RS	-0.057	0.20100	-0.006	0.88524	-0.221	<0.00001	-0.181	0.00005	-0.177	0.00007
		A_GRE	0.587	<0.00001	-0.222	<0.00001	0.128	0.00413	-0.116	0.00952	-0.116	0.00952
		PLSA	0.029	0.52398	0.171	0.00013	-0.323	<0.00001	-0.302	<0.00001	-0.284	<0.00001
		WBGA	-0.098	0.02852	-0.094	0.03530	-0.181	0.00005	-0.146	0.00102	-0.133	0.00284
		RWGA	-0.066	0.14320	-0.108	0.01541	-0.284	<0.00001	-0.133	0.00279	-0.113	0.01166
		T_GRE	-0.683	<0.00001	-0.701	<0.00001	0.881	<0.00001	0.813	<0.00001	0.813	<0.00001
	HiL	RS	0.088	0.04974	0.161	0.00031	-0.291	<0.00001	-0.132	0.00303	-0.113	0.01132
		A_GRE	-0.245	<0.00001	-0.222	<0.00001	0.128	0.00413	-0.116	0.00952	-0.116	0.00952
		PLSA	0.129	0.00385	0.029	0.51926	-0.118	0.00831	-0.139	0.00189	-0.077	0.08540
		WBGA	0.146	0.00105	0.047	0.29833	-0.055	0.22332	-0.031	0.48526	0.039	0.38198
		RWGA	0.125	0.00521	0.045	0.31242	-0.108	0.01601	-0.009	0.83594	0.039	0.37895
		T_GRE	-0.110	1.39E-02	-0.701	<0.00001	-0.395	<0.00001	-0.142	1.49E-03	0.276	<0.00001
UAV	MiL	RS	0.556	<0.00001	-0.060	0.18055	0.047	0.29204	-0.488	<0.00001	-0.430	<0.00001
		A_GRE	0.813	<0.00001	0.813	<0.00001	-0.287	<0.00001	-0.287	<0.00001	-0.287	<0.00001
		PLSA	0.194	0.00001	-0.086	0.05344	-0.146	0.00103	-0.115	0.01021	-0.098	0.02851
		WBGA	0.246	<0.00001	0.099	0.02735	0.328	<0.00001	-0.001	0.98137	-0.015	0.74082
		RWGA	0.212	<0.00001	0.017	0.70894	0.273	<0.00001	-0.040	0.36830	-0.012	0.78122
		T_GRE	-0.798	<0.00001	*	*	-0.813	<0.00001	-0.762	<0.00001	0.652	<0.00001
	SiL	RS	0.588	<0.00001	-0.121	0.00695	0.076	0.08866	-0.467	<0.00001	-0.446	<0.00001
		A_GRE	0.813	<0.00001	0.813	<0.00001	-0.287	<0.00001	-0.287	<0.00001	-0.287	<0.00001
		PLSA	0.202	0.00001	-0.138	0.00194	-0.123	0.00569	-0.163	0.00025	-0.134	0.00262
		WBGA	0.241	<0.00001	0.052	0.24453	0.313	<0.00001	0.022	0.62604	0.021	0.64036
		RWGA	0.257	<0.00001	0.046	0.30504	0.347	<0.00001	0.006	0.89444	0.027	0.54427
		T_GRE	-0.798	<0.00001	*	*	-0.813	<0.00001	-0.762	<0.00001	0.652	<0.00001
	HiL	RS	0.462	<0.00001	-0.454	<0.00001	0.103	0.02148	0.005	0.90770	-0.258	<0.00001
		A_GRE	0.798	<0.00001	0.798	<0.00001	-0.287	<0.00001	-0.287	<0.00001	-0.287	<0.00001
		PLSA	0.257	<0.00001	-0.423	<0.00001	-0.216	<0.00001	-0.057	0.20256	-0.054	0.22988
		WBGA	0.269	<0.00001	0.103	0.02100	0.256	<0.00001	0.263	<0.00001	0.141	0.00155
		RWGA	0.190	0.00002	-0.109	0.01504	0.300	<0.00001	0.265	<0.00001	0.147	0.00101
		T_GRE	0.813	<0.00001	-0.813	<0.00001	-0.813	<0.00001	-0.652	1.31E-04	-0.790	<0.00001
TANK	MiL	RS	0.302	<0.00001	0.257	<0.00001	0.195	0.00001	0.179	0.00005	0.163	0.00025
		A_GRE	-0.809	<0.00001	-0.809	<0.00001	-0.809	<0.00001	-0.798	<0.00001	-0.798	<0.00001
		PLSA	0.039	0.38525	-0.134	0.00273	-0.193	0.00001	0.276	<0.00001	-0.081	0.07003
		WBGA	0.129	0.00396	0.211	<0.00001	0.110	0.01418	0.073	0.10513	-0.003	0.95221
		RWGA	0.172	0.00011	0.205	<0.00001	0.174	0.00009	0.086	0.05578	0.026	0.55942
		T_GRE	0.683	<0.00001	0.057	0.203	0.158	3.78E-04	-0.467	<0.00001	-0.790	<0.00001
	SiL	RS	0.297	<0.00001	0.207	<0.00001	0.263	<0.00001	0.147	0.00101	0.121	0.00680
		A_GRE	-0.809	<0.00001	-0.809	<0.00001	-0.809	<0.00001	-0.798	<0.00001	-0.798	<0.00001
		PLSA	-0.001	0.97580	-0.141	0.00163	-0.185	0.00003	0.322	<0.00001	-0.161	0.00029
		WBGA	0.117	0.00880	0.227	<0.00001	0.233	<0.00001	0.022	0.62111	0.031	0.48279
		RWGA	0.117	0.00868	0.192	0.00002	0.151	0.00070	0.043	0.33262	0.051	0.25402
		T_GRE	0.683	<0.00001	0.057	0.203	0.158	3.78E-04	-0.467	<0.00001	-0.790	<0.00001
	HiL	RS	-0.107	0.01708	-0.179	0.00006	-0.202	0.00001	-0.004	0.92599	0.048	0.28582
		A_GRE	-0.809	<0.00001	-0.809	<0.00001	-0.809	<0.00001	-0.798	<0.00001	-0.798	<0.00001
		PLSA	-0.361	<0.00001	-0.285	<0.00001	-0.415	<0.00001	0.050	0.25974	-0.027	0.54164
		WBGA	0.204	<0.00001	0.315	<0.00001	0.226	<0.00001	0.171	0.00013	0.097	0.02989
		RWGA	0.152	0.00064	0.208	<0.00001	0.218	<0.00001	0.132	0.00321	0.089	0.04714
		T_GRE	-0.138	2.03E-03	0.213	<0.00001	0.640	<0.00001	-0.921	<0.00001	-0.480	<0.00001
DC Eng.	MiL	RS	0.087	0.05196	0.205	<0.00001	0.188	0.00002	0.184	0.00004		
		A_GRE	0.158	0.00039	-0.515	<0.00001	-0.693	<0.00001	-0.467	<0.00001		
		PLSA	0.345	<0.00001	0.571	<0.00001	0.667	<0.00001	0.781	<0.00001		
		WBGA	0.287	<0.00001	0.318	<0.00001	0.392	<0.00001	0.510	<0.00001		
		RWGA	0.214	<0.00001	0.321	<0.00001	0.447	<0.00001	0.486	<0.00001		
		T_GRE	1	<0.00001	-0.796	<0.0000	-0.304	0.0386	0.213	<0.00001		
	SiL	RS	0.102	0.02304	0.191	0.00002	0.131	0.00334	0.166	0.00020		
		A_GRE	0.158	0.00039	-0.515	<0.00001	-0.693	<0.00001	-0.467	<0.00001		
		PLSA	0.372	<0.00001	0.572	<0.00001	0.698	<0.00001	0.799	<0.00001		
		WBGA	0.264	<0.00001	0.325	<0.00001	0.388	<0.00001	0.510	<0.00001		
		RWGA	0.245	<0.00001	0.308	<0.00001	0.409	<0.00001	0.461	<0.00001		
		T_GRE	1	<0.00001	-0.796	<0.0000	-0.304	0.0386	0.213	<0.00001		
	HiL	RS	0.206	<0.00001	0.376	<0.00001	0.265	<0.00001	0.343	<0.00001		
		A_GRE	0.158	0.00039	-0.552	<0.00001	-0.644	<0.00001	-0.382	<0.00001		
		PLSA	0.629	<0.00001	0.720	<0.00001	0.764	<0.00001	0.841	<0.00001		
		WBGA	0.264	<0.00001	0.325	<0.00001	0.388	<0.00001	0.675	<0.00001		
		RWGA	0.369	<0.00001	0.451	<0.00001	0.482	<0.00001	0.582	<0.00001		
		T_GRE	1	<0.00001	-0.921	<0.00001	-0.149	<0.00001	-0.564	<0.00001		

Table B.13: Summary for the Mann-Whiteney U-Test Statistical Test results for the NFRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

				HiL		
Case Study	**RQ**	**A**	**B**	**+**	**=**	**-**
ACC	RQ1	RS	A_GRE	0	2	48
		RS	PLSA	1	6	43
		RS	WBGA	2	3	45
		RS	RWGA	2	4	44
		RS	T_GRE	50	0	0
	RQ2	A_GRE	PLSA	24	10	16
		A_GRE	WBGA	27	7	16
		A_GRE	RWGA	26	9	15
		A_GRE	T_GRE	50	0	0
		PLSA	WBGA	5	40	5
		PLSA	RWGA	10	37	3
		PLSA	T_GRE	50	0	0
		WBGA	RWGA	2	48	0
		WBGA	T_GRE	50	0	0
		RWGA	T_GRE	50	0	0
UAV	RQ1	RS	A_GRE	27	1	22
		RS	PLSA	13	24	13
		RS	WBGA	15	27	8
		RS	RWGA	15	24	11
		RS	T_GRE	50	0	0
	RQ2	A_GRE	PLSA	12	8	30
		A_GRE	WBGA	20	8	22
		A_GRE	RWGA	20	5	25
		A_GRE	T_GRE	43	0	7
		PLSA	WBGA	17	26	7
		PLSA	RWGA	14	32	4
		PLSA	T_GRE	49	0	1
		WBGA	RWGA	5	42	3
		WBGA	T_GRE	45	5	0
		RWGA	T_GRE	46	9	1
TANK	RQ1	RS	A_GRE	2	1	47
		RS	PLSA	1	2	47
		RS	WBGA	1	4	45
		RS	RWGA	1	4	45
		RS	T_GRE	49	0	1
	RQ2	A_GRE	PLSA	3	6	41
		A_GRE	WBGA	4	21	25
		A_GRE	RWGA	3	12	35
		A_GRE	T_GRE	49	0	1
		PLSA	WBGA	1	6	43
		PLSA	RWGA	2	5	43
		PLSA	T_GRE	49	0	1
		WBGA	RWGA	2	5	43
		WBGA	T_GRE	49	0	1
		RWGA	T_GRE	49	0	1
DCEng	RQ1	RS	A_GRE	0	0	40
		RS	PLSA	39	1	0
		RS	WBGA	31	5	4
		RS	RWGA	31	4	5
		RS	T_GRE	2	0	38
	RQ2	A_GRE	PLSA	40	0	0
		A_GRE	WBGA	40	0	0
		A_GRE	RWGA	40	0	0
		A_GRE	T_GRE	8	0	32
		PLSA	WBGA	0	0	40
		PLSA	RWGA	0	0	40
		PLSA	T_GRE	2	0	38
		WBGA	RWGA	6	33	1
		WBGA	T_GRE	2	0	38
		RWGA	T_GRE	2	0	38

Table B.14: Results for the Spearman's rank correlation test, which measures the correlation of the NFRCT metric with respect to the test suite size. Notice that a negative ρ means an improve in the performance of the algorithm with a larger test suite.

			Product 1		**Product 2**		**Product 3**		**Product 4**		**Product 5**	
			ρ	**p-value**	ρ	**p-value**	ρ	**p-value**	ρ	**p-value**	ρ	**p-value**
ACC	**HiL**	RS	-0.265	<0.00001	0.123	0.00608	0.262	<0.00001	0.226	<0.00001	0.155	0.00052
		A_GRE	-0.685	<0.00001	-0.165	0.00022	-0.182	0.00004	-0.195	0.00001	0.024	0.58751
		PLSA	-0.129	0.00395	-0.126	0.00491	-0.056	0.20819	-0.149	0.00082	-0.131	0.00327
		WBGA	0.026	0.56781	0.085	0.05695	0.079	0.07932	0.151	0.00071	0.086	0.05446
		RWGA	0.061	0.17604	0.178	0.00006	0.098	0.02778	0.003	0.94266	0.138	0.00204
		T_GRE	0.202	<0.00001	0.948	<0.00001	0.673	<0.00001	0.661	<0.00001	0.661	<0.00001
UAV	**HiL**	RS	0.446	<0.00001	-0.454	<0.00001	0.193	0.00001	0.200	0.00001	-0.252	<0.00001
		A_GRE	0.798	<0.00001	0.798	<0.00001	-0.287	<0.00001	-0.287	<0.00001	-0.287	<0.00001
		PLSA	0.257	<0.00001	-0.423	<0.00001	-0.216	<0.00001	-0.057	0.20256	-0.327	<0.00001
		WBGA	0.134	0.00267	0.103	0.02093	0.283	<0.00001	0.289	<0.00001	-0.412	<0.00001
		RWGA	0.012	0.78057	-0.109	0.01504	0.326	<0.00001	0.299	<0.00001	-0.375	<0.00001
		T_GRE	-0.798	<0.00001	-0.813	<0.00001	-0.813	<0.00001	-0.652	<0.00001	-0.790	<0.00001
TANK	**HiL**	RS	-0.237	<0.00001	-0.722	<0.00001	-0.785	<0.00001	-0.437	<0.00001	-0.502	<0.00001
		A_GRE	-0.798	<0.00001	-0.921	<0.00001	-0.921	<0.00001	-0.809	<0.00001	-0.931	<0.00001
		PLSA	-0.362	<0.00001	-0.605	<0.00001	-0.769	<0.00001	-0.288	<0.00001	-0.323	<0.00001
		WBGA	-0.266	<0.00001	-0.629	<0.00001	-0.647	<0.00001	-0.292	<0.00001	-0.389	<0.00001
		RWGA	-0.233	<0.00001	-0.575	<0.00001	-0.594	<0.00001	-0.259	<0.00001	-0.299	<0.00001
		T_GRE	-0.238	<0.00001	-0.916	<0.00001	-0.907	<0.00001	-0.201	<0.00001	-0.832	<0.00001
DC Eng.	**HiL**	RS	0.078	0.08260	0.224	<0.00001	0.250	<0.00001	0.284	<0.00001		
		A_GRE	-0.697	<0.00001	-0.697	<0.00001	-0.745	<0.00001	-0.733	<0.00001		
		PLSA	0.673	<0.00001	0.758	<0.00001	0.776	<0.00001	0.844	<0.00001		
		WBGA	0.369	<0.00001	0.470	<0.00001	0.513	<0.00001	0.641	<0.00001		
		RWGA	0.421	<0.00001	0.493	<0.00001	0.532	<0.00001	0.598	<0.00001		
		T_GRE	-0.697	<0.00001	-0.884	<0.00001	-0.891	<0.00001	-0.86	<0.00001		