Journal manuscript No.
(will be inserted by the editor)

An Empirical Evaluation of White-box and Black-box Test Case Prioritization Techniques in CPSs Modeled in Simulink

Aitor Arrieta

Received: date / Accepted: date

Abstract MATLAB/Simulink is the leading tool for simulating complex Cyber-Physical Systems (CPSs). The simulation models of complex CPSs are typically compute intensive, and the execution of test cases is long. Furthermore, the execution of test cases is typically triggered several times at different "in-the-Loop" test levels (i.e., Model, Software and Hardware-in-the-Loop). Therefore, test optimization techniques, such as test case prioritization, are paramount when testing these systems. In this paper, we present what to the best of our knowledge is the first empirical study on test case prioritization techniques for Simulink models by comparing the performance of white-box and black-box test case prioritization techniques. We assess traditional test case prioritization techniques, and we also propose new approaches for use in the context of Simulink models. We empirically compared 11 test case prioritization techniques using six Simulink models of different sizes and complexities. When comparing white-box against black-box test case prioritization techniques, we found that in general, white-box techniques were slightly better than black-box ones. In the context of white-box test case prioritization, the total greedy approach outperformed the additional greedy strategy in models with higher block interactions (>10) and connections (>200), metrics that better capture system coupling than raw block count. As for the test case prioritization time, black-box techniques were faster, although total greedy techniques were fast enough to be used in practice.

Keywords Cyber-Physical Systems · Simulink models · Test Prioritization

A. Arrieta
Mondragon University
Mondragón, Guipuzcoa, Spain
E-mail: aarrieta@mondragon.edu

1 Introduction

Simulation-based testing is a commonly used method for testing and verifying complex and untestable systems such as Cyber-Physical Systems (CPSs) [14]. In CPSs and other complex systems where the software interacts with parallel physical processes, simulation models are employed in early stages before implementing the system [65]. MATLAB/Simulink is one of the leading tools for modeling and simulating complex dynamic systems in many sectors and domains, including automotive [65,89,93], aerospace [20,68], railway [92,85] and system of elevators [81,80].

While simulation-based testing has several advantages, the execution of test suites in this context can be lengthy in real industrial settings, with single simulations requiring up to an hour to execute [68]. To a large extent, this is caused by the complex mathematical models that are required to model the physical parts of such systems. Other reasons include the need for co-simulation to enable high fidelity simulations that rely on realistic set-ups. For instance, Gladisch et al. [32] reported that for two autonomous driving case studies (adaptive cruise controller and lane keeping case studies), complex vehicle simulation models were required. Their simulations reduced real-time factors down to 10%, meaning that 1 minute of simulation time requires 10 minutes to simulate [32]. In another recent study [34], it was reported that around 8.2 hours were necessary to execute 20 realistic test cases. Besides, such test cases are not executed only once, but several times at different steps and levels, while moving from the simple Model-in-the-Loop (MiL) level to the more complex and realistic simulations that occur at the Hardware-in-the-Loop (HiL) level (see Section 2.2).

To address the high cost of testing CPSs, test optimization methods have relied on several approaches, including test case selection and test case prioritization techniques. Test case selection techniques attempt to select a subset of test cases from a full test suite that are relevant to a specific testing objective; test case prioritization techniques, in contrast, attempt to place test cases in an order that allows them to achieve their objectives (i.e., fault detection) earlier than might otherwise be possible. Arrieta et al. [6,7] studied how different black-box metrics combined with Pareto-based search algorithms performed when selecting a subset of test cases from a test suite. This problem aimed at reducing as much as possible the test execution time while not compromising test quality. However, when using Simulink models it is also important to detect faults as soon as possible. This way, it is possible the debugging process to start earlier [69]. When many test cases exist, many test executions can be performed without finding any faults; thus, effective test case prioritization techniques can be useful. A large corpus of studies have proposed and compared different test case prioritization techniques [78,77,79,25,24,26,38, 43,44,48,47,59,60,82,53] – although these have been conducted outside of the context of CPSs. Henard et al. [41] compared ten white-box and ten black-box test case prioritization techniques, and concluded that there was little difference between the two classes of techniques in terms of rate of fault detection.

However, Henard et al. [41] focused on unit testing of C programs, which is quite different from testing Simulink models. One major difference is related to semantics. While Simulink models can rely on both discrete and continuous-time semantics, programs written in C often rely only on discrete semantics. The presence of continuous-time semantics in Simulink enables the creation of closed-loop models in which the outputs of the system are continuously fed back to its inputs, producing dynamic feedback behavior over time. On the other hand, coverage-based white-box test case prioritization techniques might not be applicable/effective on some Simulink models due to the difficulty of analyzing coverage for continuous operations [66]. Furthermore, in the context of Simulink models, test cases are signals that stimulate the inputs of the system over the time. This provides opportunities to provide novel test metrics based on anti-patterns related to signals, which are among those used in this work.

In the past, test case prioritization techniques that rely on historical data have performed well, for general purpose software [61, 90, 73] and for simulation-based testing models [12, 11]. However, in order for these techniques to be effective, test cases need to be executed several times [46]. To address this problem, test quality metrics that focus either on black-box techniques (i.e., metrics that measure test quality focusing solely on the inputs and outputs of the simulation models) or on white-box techniques (i.e., metrics that measure which parts of the simulation models have been exercised) can be employed. To the best of our knowledge, this is the first paper in which different state-of-the-art white-box and black-box test case prioritization techniques are thoroughly empirically evaluated in the context of MATLAB/Simulink models.

Specifically, this paper makes the following key contributions.

- We present and revisit test case prioritization techniques for the context of MATLAB/Simulink models.
- We conduct an empirical evaluation considering six subject models of different sizes and complexities and 11 test case prioritization techniques. To the best of our knowledge, this is the largest test prioritization empirical evaluation performed to date in the context of Simulink models.
- We provide implementations of all of the techniques that we evaluated in a public repository that can be accessed by other researchers and practitioners. We further provide all of the evaluation materials, including scripts, models, mutants, test cases and sources for statistical analysis in a public repository to support replicability and reproducibility.[1]

The main findings of our study can be summarized as follows. First, we found that for Simulink models, unlike in the context of multi-objective test case selection [7], white-box metrics are generally as competitive as black-box metrics in the context of test case prioritization for Simulink models. Second, we found that there is no single technique that stands out over the rest of the techniques considered. Third, in the context of white-box test case prioritization, the total greedy based techniques perform better than the additional

[1] Replication package can be found in `https://doi.org/10.6084/m9.figshare.28777829`

greedy-based techniques used in 50% of the used subject models, specifically those with higher block interactions (>10) and connections (>200). Total and greedy-based test case prioritization algorithms are a class of techniques that utilize the coverage obtained by test cases following different strategies. Fourth, executing Mathworks' API for measuring white-box coverage in MATLAB for Simulink models is time consuming, and significantly increases the running time of white-box test case prioritization techniques.

2 Background

2.1 Simulation models and basic notation

Simulation is the process of executing a model to approximate the dynamic behavior of the real system it represents, allowing engineers to observe and analyze its responses under different conditions. CPS developers rely on simulation tools and Model-Based Design (MBD) work-flows, where graphical models of their systems are employed to do rapid prototyping [19]. Simulation tools oriented to the design and development of CPSs (e.g., Simulink) are often dataflow models, where each model contains a set of blocks [19]. Each block from these models accepts data through its inputs and may pass output through its output ports after a set of operations (e.g., mathematical, logical) are performed. Figure 1 provides an example of a simulation model, which includes six inputs (enable, brake, set, speed, inc and dec) and two outputs (throt and target).

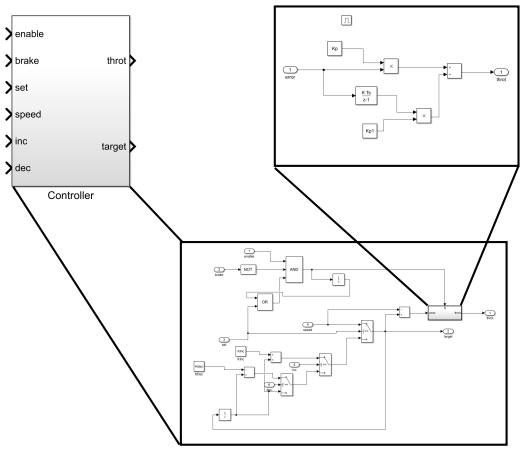


Fig. 1: Example of a simulation model of a Cruise Controller of a car [7]

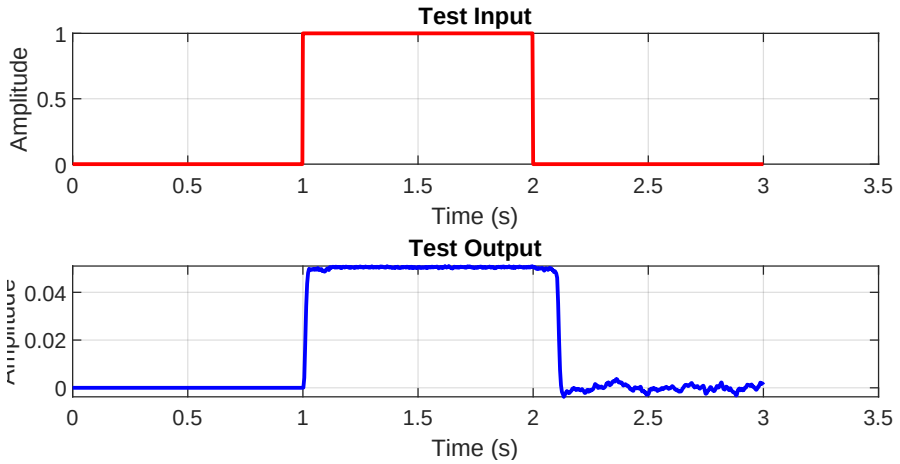


Fig. 2: An example of the execution of a test case in the EMB case study system

The remainder of this paper makes use of the following definitions and notation. We note that, similar to previous studies [65,66,5,12], we assume that the employed simulation employs a fixed sample step. Let $SM = (I, O)$ be a simulation model, where $I = \{i_1, i_2, ..., i_N\}$ is a subset of inputs and $O = \{o_1, o_2, ..., o_M\}$ is a subset of outputs [65]. Each input and output of the simulation model is a signal (i.e., a function of time), which is stored as a vector where elements are indexed by time [65]. Figure 2 shows the input and corresponding output of a test case for one of the case study systems used in our evaluation. The simulation time (T) is divided into a set of equal sample time steps (ΔT) [65]. A signal (sig) is a function in time of a set of k observed simulation steps (i.e., $sig : \{0, \Delta T, 2 \times \Delta T, ..., k \times \Delta T\}$). For instance, a simulation of 10 seconds (i.e., T=10), with a sample time of 0.05 seconds (i.e., $\Delta T = 0.05$) would have a total of 10/0.05+1 (i.e., k=201) simulation steps. The lower the sample time, the higher the precision of the simulation. However, the time required to simulate the system will also be lengthier. In our study, for each model being tested, we consider a fixed simulation sample time (i.e., the same for all the test cases). This ensured comparability across runs, which is required for subsequent signal similarity computations described later in Section 3.1.2. To our knowledge and via discussion with domain experts, having this time step fixed is the default practice in industry.

2.2 Simulation-based testing

Cyber-Physical Systems are typically validated through a sequence of simulation- and execution-based test levels. Development commonly begins with Model-in-the-Loop (MiL) testing, where engineers assess the behavior of the controller

model together with a plant representation. At this stage, both elements execute using floating-point arithmetic, and the resulting signals serve as reference values for later stages [84]. Once the MiL results are satisfactory, the controller model is translated into executable code (for example, a `*.dll`), giving rise to Software-in-the-Loop (SiL) testing, in which the generated software replaces the controller model while the plant remains simulated [84].

The next level, Processor-in-the-Loop (PiL), deploys this generated code onto the actual target processor. The processor interacts with a simulation of the physical system running on the host machine, and this stage is primarily intended to identify faults introduced during compilation or cross-compilation [84]. In many development workflows, PiL is bypassed and teams move directly to Hardware-in-the-Loop (HiL) testing [13]. HiL integrates the controller software with the Electronic Control Unit (ECU) and the complete real-time execution stack (e.g., drivers, communication layers, and the real-time operating system) [84]. The physical environment is emulated on dedicated hardware such as FPGAs, enabling real-time execution and allowing both functional and non-functional requirements to be assessed [84]. Numerous approaches have been proposed to support testing across these levels, from MiL and SiL techniques [62,65,66,8,4,81,80] to HiL-focused methods [23,45].

In our experience with industrial partners, testing is rarely performed only once per level. Instead, test suites are executed multiple times as the system evolves. Early MiL models typically begin as simplified, low-fidelity representations, and are gradually refined as subsystems are replaced with more complex or realistic versions. A comparable iterative refinement occurs at the HiL stage: ECUs are extended with additional tasks and functionalities, and configuration changes—such as task-priority adjustments, integration of new features not previously tested, modifications to communication protocols, or updates to operating-system parameters—are frequent. Each such change typically necessitates rerunning a substantial portion of the test suite. This iterative evolution across test levels highlights why regression-oriented test optimization techniques are particularly relevant in CPS development.

3 Selected Test Case Prioritization Methods

In this section we explain the adapted test case prioritization techniques and how we used them in the context of Simulink models for CPSs testing. We first explain the adapted black-box techniques (Section 3.1) followed by the white-box techniques (Section 3.2). We deliberately focused on traditional single-objective techniques. These approaches are widely adopted in practice and serve as essential baselines. Studying each prioritization metric in isolation allows us to isolate the contribution of each metric individually and to obtain fundamental insights before moving towards multi-objective or hybrid approaches. We acknowledge that engineers often combine metrics in practice, and highlight the design of hybrid and multi-objective strategies as a promising direction for future work.

3.1 Black-Box Techniques

Black-box techniques are of special interest in the context of simulation-based testing, as white-box metrics are not always available. This might be due to several reasons. For instance, developers of CPSs typically involve several Original Equipment Manufacturers (OEMs), from different engineering domains (e.g., mechanical, electrical, computer science, etc.). Each of them provide their system models as a functional black-box model without access permission by other OEMs due to Intellectual Property (IP) issues. Subsequently, it is often infeasible to measure white-box coverage. Another reason is that complex software of Simulink models is typically developed in C, and later executable code is generated and incorporated as a subsection in Simulink. This executable code is also black-box and it is not possible to obtain coverage information. The proposed black-box techniques for test case prioritization can be catalogued into two main categories: (1) techniques based on anti-patterns and (2) techniques based on test similarity. These techniques were first proposed by Matinnejad et al. [65] for test case generation and later tailored by Arrieta et al. [7] for regression test optimization. We now explain in detail each of them.

3.1.1 Techniques based on anti-patterns

By stimulating the inputs of a simulation model, it is possible to obtain the output results of the simulation and provide some numerical values with respect to certain anti-patterns. In previous works [66,65,63] three different anti-patterns were defined for Simulink models, which we later adapted to the context of test case selection [6,7]. The anti-pattern metrics quantitatively measure certain patterns that are often related with incorrect/faulty behaviours of simulation models [66,65,63]. The hypothesis behind these measures is that the higher the anti-pattern degree a test case has, the higher the likelihood of detecting a fault. We thus sort the test cases in a descending order based on its degree of anti-pattern, which are explained in detailed below. The three anti-patterns we use (instability, discontinuity, and growth to infinity) are representative signal-level indicators adapted from prior work on CPS testing (e.g., [7,65]). They capture controller and signal-shape anomalies that often correlate with software faults in Simulink controllers. We acknowledge that broader system-level issues (e.g., integration faults, sensor/actuator anomalies, or solver-configuration pitfalls) may not be adequately reflected by these signal-level patterns. Our study deliberately isolates lightweight, model-agnostic metrics for software fault detection, excluding other faults related to hardware aspects (e.g., faults in sensors) as well as related to the simulator configuration (e.g., solver misconfigurations).

It is important to note that the anti-patterns considered in this work (instability, discontinuity, and growth-to-infinity) originate from prior studies primarily focused on hybrid controller testing (e.g., [7,63,65,66]). These metrics are representative of signal-level anomalies that often correlate with software

defects in Simulink control logic, but they do not constitute an exhaustive taxonomy of faults in CPSs. In particular, issues such as integration problems with hardware or actuator/sensor anomalies may not manifest as these signal-level patterns. Our study deliberately focuses on software-level behavioral faults and therefore adopts lightweight, model-agnostic anti-pattern metrics that can be computed solely from simulation signals, independently of specific architectural or hardware configurations. While our approach is appropriate for both regression testing contexts as well as for multi-level testing, our metrics are not so adequate to capture hardware-level faults. However, we trust that our metrics can be complemented with other test prioritization techniques that further focus on the HiL level (e.g., [74,83]. Extending our work and complementing it with such metrics constitutes an important avenue for future work and will require richer CPS-specific fault models and domain expertise.

Metrics for use in prioritization techniques based on anti-patterns:

For the context of Simulink models, three anti-patterns were identified in previous works [65,63,66], which are measured by considering the outputs signals of the models. These anti-patterns are named (1) instability, (2) discontinuity and (3) growth to infinity. These anti-patterns were used as black-box metrics to prioritize test cases in the context of Simulink models.

Instability: Instability measures the degree at which an output signals shows quick and frequent oscillations [63]. An example of when this anti-pattern appears is when a state-chart controller switches between states with a high frequency. This effect is (generally) undesirable in (most) physical processes of CPSs [63]. For instance, having this kind of patterns in the acceleration or the speed of an autonomous vehicle is not desirable. The instability of a signal output sig can be measured by applying Equation 1, where k is the number of simulation steps and ΔT is the simulation time step. The higher the value of $instability(sig)$, the higher the instability degree of the signal.[2]

$$instability(sig) = \sum_{i=1}^{k} |sig(i \cdot \Delta T) - sig((i-1) \cdot \Delta T)| \quad (1)$$

Simulink models usually have more than one output, and therefore, the instability degree of a test case needs to be normalized. In our approach, we assume that there is no domain knowledge, and thus, we give the same importance to all output signals. Given an $SM = \{I, O\}$ with M outputs (i.e., $O = \{o_1, o_2, ..., o_M\}$), the instability of a test case j in TS is measured with Equation 2, where $instability(Osig_{j_i})$ is the instability of the i-th signal for tc_j and $\max(instability(Osig_i))$ is the maximum instability degree obtained in the i-th signal when considering all test cases in TS.

$$TCInstability(tc_j) = \frac{\sum_{i=1}^{M} instability(Osig_{j_i})}{\sum_{i=1}^{M} \max(instability(Osig_i))} \quad (2)$$

[2] We consider the notation from Section 2.1 to explain the equations of this section

Discontinuity: Discontinuity refers to an anti-pattern where an output signal displays a short duration pulse [63]. This anti-pattern might appear, for instance, when an unforeseen situation from the environment arises and the system needs to self-regulate to adapt to the situation. This anti-pattern can be measured at a signal level by computing its derivative function [7], but as simulation time steps are not infinitesimal, Matinnejad et al. proposed a function that relies on discrete change rates [64]. Given a signal sig, $lc_i = |sig(i \cdot \Delta t) - sig((i - dt) \cdot \Delta t)|/\Delta t$ is the left change rate at step i, and $lc_i = |sig((i + dt) \cdot \Delta t - sig(i \cdot \Delta t)|/\Delta t$ is the right change rate at step i [64,63]. The discontinuity of a signal sig can be measured by applying Equation 3, i.e., the maximum of the minimum of the left and right change rates at each simulation step for a given set of simulation steps [64,63]. The goal of this metric is to capture short, pulse-like discontinuities in the output signals, as originally proposed by Matinnejad et al. [64,63]. First, for a fixed time index i and offset dt, the quantities $lc_i(dt)$ and $rc_i(dt)$ measure how sharply the signal changes when approaching and leaving i. We take $d_i(dt) = \min\{lc_i(dt), rc_i(dt)\}$ so that $d_i(dt)$ becomes large only when both sides exhibit a fast change. This suppresses one-sided ramps and ensures that the metric responds only to genuine two-sided, spike-like discontinuities. Using the minimum is therefore intentional: a point receives a high score only if it lies in the center of a short-duration pulse. Second, the outer maximum $\max_{i=dt,\ldots,k-dt} d_i(dt)$ aggregates these local two-sided change rates into a single scalar representing the strongest discontinuity in the signal at the given scale dt. Finally, the choice of $dt \in \{1, 2, 3\}$ reflects the intention to detect short pulses. In discrete-time simulations, a "short-duration discontinuity" spans only a small number of sampling steps. Following prior studies [64,63], we therefore evaluate the metric at $dt = 1, 2, 3$: $dt = 1$ captures single-step spikes, while $dt = 2$ and $dt = 3$ capture slightly wider pulses or pulses not perfectly aligned with the sampling grid. Using larger dt values would start detecting slower, legitimate trends rather than genuine discontinuities. The final metric $\max_{dt\in\{1,2,3\}} \max_i \min\{lc_i(dt), rc_i(dt)\}$ thus captures the strongest short-duration pulse across a small range of time scales.

$$discontinuity(sig) = \max_{dt=1}^{3}(\max_{i=dt}^{k-dt}(\min(lc_i, rc_i))) \tag{3}$$

To normalize the discontinuity degree of each test case for more than one output, the discontinuity of a test case j in TS is obtained with Equation 4. For a test case j, given an $SM = \{I, O\}$ with M outputs (i.e., $O = \{o_1, o_2, ..., o_M\}$), the discontinuity degree $discontinuity(Osig_{j_i})$ is the discontinuity of the i-th signal for tc_j and $max(discontinuity(Osig_i))$ is the maximum discontinuity value in the i-th signal when considering all test cases in TS.

$$TCDiscontinuity(tc_j) = \frac{\sum_{i=1}^{M} discontinuity(Osig_{j_i})}{\sum_{i=1}^{M} max(discontinuity(Osig_i))} \tag{4}$$

Growth to infinity: Another anti-pattern for simulation models can be the growth to infinity, which measures the maximum absolute value of a signal.

This metric can be effective as, for instance, the control algorithms of an hybrid system (e.g., a CPS) can be wrongly tuned. When this happens, the feedback control system can be saturated, leading the system to an uncontrolled state and resulting in signals of huge values. The anti-pattern "Growth to infinity" of a signal *sig* can be measured by obtaining its maximum absolute value across all sample steps (Equation 5).

$$inf(sig) = max|sig(i \cdot \Delta T)| \tag{5}$$

As for the previous anti-patterns, a normalization process is required to account for all the output signals of a simulation model. Given an $SM = \{I, O\}$ with M outputs (i.e., $O = \{o_1, o_2, ..., o_M\}$), the growth to infinity of a test case j in TS is obtained with Equation 6, being $inf(Osig_{j_i})$ the growth to infinity of the i-th signal for tc_j and $max(inf(Osig_i))$ the maximum infinity value in the i-th signal when considering all test cases in TS.

$$TCInfinity(tc_j) = \frac{\sum_{i=1}^{M} inf(Osig_{j_i})}{\sum_{i=1}^{M} max(inf(Osig_i))} \tag{6}$$

When normalizing the metrics across multiple output signals, we assigned equal weights to all signals. This choice reflects the absence of domain knowledge for the benchmark systems and to make sure that the results are not biased in favor of one (or several) techniques. In industrial contexts, however, weighting signals differently according to their criticality could improve effectiveness; we highlight this as a potential extension and recommendation for practitioners.

3.1.2 Techniques based on test similarity metrics

The hypothesis that dissimilar test cases have larger probabilities of finding faults has been widely investigated [27,18,16,17,40,39]. In the context of test case prioritization, information on how similar two test cases are can be used to prioritize test cases. Similarity between two test cases can be measured by a distance function, where a small distance means that two test cases are similar whereas a large distance means the opposite. With a distance metric, it is possible to build a distance matrix to measure similarity among all the test cases in the test suite. This information is later used by the test case prioritization algorithm to iteratively include a test case in the prioritized test suite. In each iteration, the algorithm includes the test case which is farthest from those test cases already included in the prioritized test suite. Several distance metrics have been proposed in the literature to measure similarity between test cases (e.g., Hamming, Yaccard) [40]. Based on previous studies [64,65], we selected the Euclidean distance to measure the similarity between two test cases. Since these metrics can be applied both on inputs as well as on outputs of the simulation models, we applied it on both, yielding two similarity-based test prioritization techniques, as explained below.

Metrics for test prioritization techniques based on similarity:

Two metrics were derived to prioritize test cases based on test similarity. The former measures the similarity of test cases by considering the stimulation signals, whereas the latter measures it by considering the outputs. As previously mentioned, the Euclidean distance is employed to measure the similarity of two test cases, as proposed in previous studies [64,65,6,7]. In the context of Simulink models, test cases can have different test execution times, and subsequently, a different number of simulation steps. In prior studies the Euclidean distance was adapted to consider this [6,7].

The Euclidean distance to measure the distance between two signals related to a specific input or output (i.e., sig and sig') in two different test cases can be measured by applying Equation 7. For both signals, $\min(k_{sig}, k_{sig'})$ is the number of steps of the signal whose test case has a lower Test Execution Time (TET), $max_{R_{sig}}$ is the maximum value that the signal in the simulation can obtain, $min_{R_{sig}}$ is the minimum value that the signal in the simulation can obtain and K is the number of steps that the test case with the highest TET in the test suite has [6,7]. As in prior work [6,7], K is considered as the number of steps for the longest test case for two main reasons. Firstly, it ensures that the distances between test cases are normalized. Secondly, because a longer test case might have a higher chance to detect faults. Therefore, the distance of those very short test cases is penalized. It is important to note that the simulation step (Δt) is considered the same for all test cases. A higher Euclidean distance means that two signals are less similar.

$$D(sig, sig') = \frac{\sqrt{\sum_{i=0}^{\min(k_{sig}, k_{sig'})} (sig(i \cdot \Delta t) - sig'(i \cdot \Delta t))^2}}{\sqrt{K+1} \times (max_{R_{sig}} - min_{R_{sig}})} \tag{7}$$

Input-based test similarity: Given two test cases (TC_a and TC_b), for a simulation model with N inputs, Equation 8 defines the input signal-based distance, which is the sum of the normalized Euclidean distance of each input signal between both test cases. A higher distance means that the test cases are less similar in terms of their input signals.

$$InTCD(TC_a, TC_b) = \sum_{i=1}^{N} D(Isig_{a_i}, Isig_{b_i}) \tag{8}$$

Output-based test similarity: Given two test cases (TC_a and TC_b), for a simulation model with N outputs, Equation 9 defines the output signal-based distance, which is the sum of the normalized Euclidean distance of each output signal between both test cases. A higher distance means that the test cases are less similar in terms of their output signals.

$$OutTCD(TC_a, TC_b) = \sum_{i=1}^{N} D(Osig_{a_i}, Osig_{b_i}) \tag{9}$$

We employed the Euclidean distance as the similarity metric between test cases. This choice was motivated by its widespread adoption in prior CPS testing works (e.g., [65, 7]), which makes our results directly comparable to existing studies. Moreover, the Euclidean distance is simple, computationally efficient, and interpretable in the context of signals. We acknowledge that alternative metrics, such as the Fréchet distance, Manhattan distance, or Fourier-based measures, may provide complementary advantages. However, to our knowledge, these techniques have not been explored for the context of testing of Simulink models yet, and therefore consider it out of the scope of our paper. Exploring these alternatives constitutes an interesting avenue for future work.

3.2 White-Box Techniques

Two variants of traditional "greedy" dynamic test case prioritization techniques are the most common test prioritization strategies when using white-box metrics: the total strategy and the additional strategy.[3] On the one hand, the total greedy technique prioritizes test cases based on their absolute code coverage [60]. Specifically, supposing that structural coverage is used as a proxy to measure the test quality (e.g., Decision Coverage (DC), Condition Coverage, Modified Condition/Decision Coverage (MC/DC), tests are sorted in descending order based on the number of test objectives covered by each test case. On the other hand, the additional greedy strategy focuses on test objectives (e.g., statements) not yet covered by test cases already prioritized. At each iteration, the additional greedy algorithm selects the test case that covers more statements that have not been covered yet. When all the test objectives have been covered or no additional test objective can be covered again, these objectives are reset and the test prioritization process begins again [52].

Both of the total and additional greedy strategies exhibit different strengths and weaknesses [36]. As discussed by Hao et al. [36], a limitation of the total strategy is that it may repeatedly prioritize test cases covering the same frequently executed objectives. If those objectives have already revealed all detectable faults, additional coverage of them yields diminishing returns. Conversely, the additional strategy attempts to diversify coverage by focusing on yet-uncovered objectives, but this may delay the re-execution of frequently covered but fault-prone areas. However, the covering of a statement does not ensure the detection of all faults. If the fault is not detected by the test case, but it is detectable by others, it is important to consider those statements. In summary, when using the total strategy, the detection of faults in statements that are covered infrequently may be delayed [36]. On the other hand, for the additional technique, as the strategy considers not covered objectives, it avoids the problem exhibited by the total greedy technique, where statements that already exhibited all the possible faults are considered again. However, if a fault is not detected in a statement covered by a test case, the detection of

[3] A greedy algorithm is a technique that makes next decisions based on local optima without considering subsequent backtracking

this may be delayed. In summary, the risk of using the additional technique is that the detection of faults may be delayed in statements that are covered frequently [36]. It is important to note that Coverage is a surrogate: a covered decision may still conceal a fault unless exercised under the right path or input interactions. This motivates total strategies (which can "re-cover" elements with diverse tests) and additional strategies (which expand to uncovered objectives).

Different coverage metrics have been investigated to use as a surrogate of test objectives and at different granularity levels (i.e., line, statement, method class, etc.). For this study, both techniques were used, i.e., total and additional. We instantiate both total and additional greedy prioritization using *Decision Coverage* (DC), *Condition Coverage* (CC), and *Modified Condition/Decision Coverage* (MC/DC) via the MATLAB/Simulink Model Coverage API.[4] Throughout this section, "test objectives" refer to the instances of these coverage goals. We do not evaluate path- or data-flow–oriented objectives here as these kind of objectives are not directly accessible through the coverage API provided by MATLAB, and therefore we do not think they are applicable in practice.

The selected coverage criteria are meaningful surrogates for control-logic adequacy in Simulink controllers for three reasons: (i) Core Simulink/Stateflow behavior is governed by Boolean guards and discrete decisions (e.g., If/Switch blocks, relational and logical operators, enabled/triggered subsystems, Stateflow transition predicates). DC, CC, and MC/DC directly target these points, ensuring that both decision outcomes and constituent conditions (and, with MC/DC, their independence) are exercised where control flow diverges. (ii) Defects in embedded controllers frequently manifest as incorrect mode changes, guard logic, or boundary decisions; exercising distinct decisions and condition combinations is widely believed to be a strong proxy for revealing such faults. MC/DC, in particular, is known to expose masking and interaction faults in compound guards that simpler criteria may miss. (iii) Unlike other existing white-box criteria, these objectives are natively supported by the MATLAB/Simulink Model Coverage API and are stable across simulation runs. They also align with long-standing industrial practice and safety guidance for logic-intensive software systems, making them both actionable and interpretable for engineers.

3.3 Test case prioritization techniques not selected in this study

In the literature, different techniques and metrics have been defined but do not fit into our study. In this section we explain why these techniques were discarded in our study.

Static Test Prioritization Techniques: Generally, the so-called static methods [60] are not applicable in the context of Simulink models. For instance,

[4] https://mathworks.com/help/slcoverage/

the call-graph-based static test case prioritization approaches make use of test code to extract which methods are invoked by each of them [60,96], either in a total way or in an accumulative way. Other static approaches include string-distance based, where similarities of test cases are measured based on string edit distances [51]. There are other static approaches (e.g., topic-based), which make use of test code information. In summary, static techniques rely on source code analysis to extract call graphs or textual features. Meanwhile, in Simulink models, behavior is defined by interconnected blocks and signal flows rather than executable code, which makes these static representations unavailable. Hence, such methods cannot be directly applied.

Dynamic Test Prioritization Techniques: There are some dynamic test case prioritization approaches that could not be selected for our evaluation. Some regression test optimization approaches that rely on **requirements covered by test cases** [49] have also been applied to test case prioritization of CPSs modeled in Simulink [12,10]. Another effective way for regression test case prioritization is the use of **historical data** (e.g., number of faults detected by test cases) in order to effectively prioritize test cases [11,12,91]. However, all this information is typically not available, specifically for those cases where Simulink Design Verifier or other Simulink test generation tools (e.g., [65,9]) are used to generate test cases. The idea of the selected techniques in this paper is that the test case prioritization techniques can be easily applied in any Simulink model. In the case of history-based test case prioritization, these have been found to require large start-up information in order to be effective [46].

Adaptive Random Testing based Test Prioritization Techniques: Adaptive Random Test Case Prioritization (ART) is a dynamic technique that it randomly selects a set of test cases iteratively to build a candidate set, and from this candidate set it selects the farthest test case (based on the distances) from the already prioritized test cases. It is similar to the selected test case prioritization techniques based on the similarity, but the main difference is that instead of picking the farthest test case from all the non-prioritized test cases, it picks the farthest test case from a subset of the non-prioritized test cases which are randomly selected. An advantage with respect to the techniques proposed in Section 3.1.2 is that it might be faster as it does not require to measure the distance between all the non-prioritized test cases and all the prioritized ones. Nevertheless, after implementing our distance-based test case prioritization techniques (Section 3.1.2), we figured out that it was fast enough to be used in practice (i.e., less than 0.02 seconds). Because the similarity-based techniques employed in our study consider the distance of all test cases, whereas ART only of certain test cases, we believe our selected techniques are stronger and more reliable. We therefore discarded ART techniques in our study.

Mutation Testing based Test Prioritization Techniques: Many approaches make use of mutation scores as a metric to assess the quality of test cases [41,42,75,58,82]. However, mutation testing is a very expensive technique in the context of CPSs [21,88], including Simulink models because, besides the control algorithms, they typically involve the physical layer of the

model [12,6,7]. This physical layer is usually developed with complex mathematical models, such as differential equations. Solving complex differential equations that represent the dynamics of physical processes require significant computational resources and time, as the differential equations must be numerically integrated over time to produce accurate results. Each mutant introduces a variation that requires a full re-simulation to observe its effects, significantly increasing the computational load and making this technique not scalable in this context. Although mutants were used as the oracle for evaluating fault detection effectiveness, we did not employ mutant-based metrics directly as prioritization criteria. Doing so would require generating and simulating multiple model variants, which is computationally prohibitive in the context of CPSs, where simulation runs are already costly. Our goal was to investigate lightweight prioritization metrics that are feasible in practice, while still validating their effectiveness against mutants. Thus, since the use of mutants to assess the adequacy of individual test cases is unfeasible in practice, we discarded this test prioritization technique in our study.

4 Empirical Evaluation

In this section we explain the carried out empirical evaluation of the proposed test case prioritization strategies in the previous section. Table 1 summarizes the employed black-box and white-box techniques and provides an acronym of each strategy.

Table 1: Summary table for evaluated black-box and white-box test prioritization techniques

Acronym	Strategy	Metric	Type
AP-Ins	Anti-patterns based	Instability	Black-Box
AP-Disc	Anti-patterns based	Discontinuity	
AP-GTI	Anti-patterns based	Growth to infinity	
SB-IS	Similarity-based	Input similarity	
SB-OS	Similarity-based	Output similarity	
Add-DC	Additional	Decision Coverage	White-Box
Add-CC	Additional	Condition Coverage	
Add-MCDC	Additional	Modified Condition/Decision coverage	
Tot-DC	Total	Decision Coverage	
Tot-CC	Total	Condition Coverage	
Tot-MCDC	Total	Modified Condition/Decision coverage	

4.1 Research Questions

To compare the selected test case prioritization methods explained in Section 3 in the context of Simulink models, we addressed the following Research Questions (RQs):

RQ1 – Black-box techniques: *How well do the studied black-box prioritization techniques perform in terms of fault detection rate?* We defined this

RQ with the aim of comparing the performance of the different black-box test case prioritization techniques. The aim was to determine whether there is a technique that stands out over the rest in order to be proposed to practitioners for the cases where there is no access to white-box data. RQ1 is especially important in the context of CPSs, as not having access to the code is a common scenario. Additionally, it is a common scenario in Simulink models that there are not many routing/flow control elements, such as switches or state-flow charts. Information of white-box coverage may not be sensitive enough when not having these types of elements available.

RQ2 – White-box techniques: *How well do the studied white-box prioritization techniques perform in terms of fault detection rate?* White-box test quality metrics have been traditionally employed for software testing. However, few studies have assessed their performance in the context of Simulink models. In some cases, it is possible to measure the white-box coverage of the models, and thus, it can be used as the metric for test case prioritization. Similar to RQ1, RQ2 aimed at comparing the performance of traditional white-box test case prioritization techniques. The goal is to determine whether there is a clear winner among the selected white-box test case prioritization techniques or whether this depends on the subject model.

RQ3 – Best technique: *How well do the black-box techniques compare with the white-box ones in terms of fault detection rate? How far are the techniques compared with the optimal test case prioritization technique in terms of fault detection rate?* From RQ1 and RQ2, for each of the used subject models, the best white-box and black-box techniques were obtained. The fourth RQ aimed at comparing the best black-box technique with the best white-box one for each subject model with the aim of determining the best overall approach. We also considered an optimal ordering of the test cases for each case study to provide an upper bound for the comparison [24]. This was obtained by considering the information of test cases exposing the faults. This is not feasible in practice, as the relation between faults and test cases is unknown, but it helps us gain insight into the success rate of the selected techniques [78]. To do this, we employ an additional greedy technique considering mutants as test objectives (i.e., instead of test coverage). The additional greedy strategy focuses on mutants not yet killed by test cases that have already been included in the prioritization queue. At each iteration, the additional greedy algorithm selects the test case that covers more mutants not killed yet by prioritized test cases. Notice that this comparison has been performed in other similar empirical studies (e.g., [24,78,95]).

RQ4 – Test case prioritization execution time: *How do black-box and white-box techniques compare in terms of the required execution time for the prioritization process?* In situations where very little time is available for the overall regression testing process (e.g., [33]), the time it takes each technique to return a prioritized test suite becomes an important factor [41,33]. The fifth RQ aims at studying which technique would be the most appropriate in situations where there is a limited test execution time budget.

4.2 Selected Benchmark

Besides investigating the performance of black-box and white-box test prioritization techniques in the context of Simulink models, we also wanted to contrast the findings of our work with those from the context of test case selection [7,6] by conducting a meta-analysis based on the general conclusions of both studies. To make our comparison as fair as possible, we used as much as possible the benchmark prepared by Arrieta et al. [7,12].

4.2.1 Subject Models

Six Simulink models of different sizes and complexities were employed to compare the selected techniques. The key characteristics of the selected subject models are summarized in Table 2, which includes the number of inputs, outputs, blocks and other structural characteristics of the models. Connections refer to the total number of signal lines linking blocks within the Simulink model, representing structural coupling between components. Block interactions quantify the number of control-flow or data-flow interaction points (e.g., conditional routing blocks, merge points, and subsystem interfaces) that create dependencies among blocks beyond simple sequential flow. All these subjects have been previously used in similar studies for evaluating testing methods [7, 12,69,65,63,66,71].

Five out of the six subject models were the same as those employed by Arrieta et al. [7,12] in their benchmark. The Car Window (CW) subject involves an open-source system that models four car windows. The physical layer involves the electrical and mechanical models of the four windows. Each window is controlled by a subsystem that also involves a state machine each with 5 states and 21 transitions and modeled with Stateflow. The Electro-Mechanical Braking (EMB) system is an industrial open source model developed by Bosch [86]. This system combines physical and software models, with the software model controller including a discrete state machine and a continuous PID controller [63], and it has been used to evaluate the performance of a test generation algorithm [63]. The Cruise Controller (CC) model involves only the controller of a system and it does not contain any plant subsystem. This model was used in [65] to assess a test case generation approach for Simulink models. The AC Engine model involves an AC engine in combination with a controller that also includes some safety functionalities [12,9, 8]. The Two Tanks model involves a system where a controller regulates the incoming and outgoing flows of two tanks. This subject was used to evaluate a test oracle generation approach [69], a comparison between model testing and model checking approaches [71] or test case selection approach based on multi-objective search algorithms [12]. In the original benchmark, Arrieta et al. [7,12] employed an additional subject model involving a toy artificial simulink model named *"Tiny"*, which only involved 15 blocks. Because the size of this model is very small to draw any meaningful conclusions, similar to other studies (e.g., [54]), we decided to remove that subject model from

the benchmark. This subject was artificially generated, did not reflect realistic CPS complexity, and produced trivial prioritization outcomes. We therefore focused on more representative models where conclusions are generalizable. Instead, we decided to include an additional one, i.e., Swimairspeed. Swimairspeed is one model from "the ten lockheed martin cyber-physical challenges" that involves a safety algorithm for monitoring airspeed in the system wide integrity monitor suite [67]; the algorithm provides a warning to an operator when the vehicle speed is approaching a boundary where an evasive flyup maneuver cannot be achieved [67].

4.2.2 Test Cases

For the five models, we reused from the initial benchmark proposed by Arrieta et al. [7,12], we employed the exact same test cases. For the EMB, CC and Two Tanks models, Arrieta et al. [7,12] generated 150 test cases randomly. This was due to these models not being compatible with Simulink Design Verifier. For the CW case study, Arrieta et al. [7,12] used Simulink Design Verifier (SDV) to automatically generate test cases following an MC/DC white-box coverage criterion, yielding a total of 133 test cases given 100,000 seconds of test generation budget. For the Swimairspeed model, we had to generate test cases from scratch. After adapting the model configuration, we employed SDV. 3 of the test cases were generated by establishing the decision-condition coverage criterion. 52 test cases were generated following the MC/DC coverage criterion. To have a total of 150 test cases, we complemented these test cases by generating 95 additional test cases randomly. The size of the test suites is aligned with typical industrial CPSs (e.g., 50 to 150 test cases for a CPS for a satellite system [83]). For the AC Engine case study, Arrieta et al. [7,12] reused 120 test cases that were generated with a test case generation tool from a previous study [8]. For the Swimairspeed model, Simulink Design Verifier (SDV) was able to automatically generate a portion of the test suite, and we completed the remaining cases using random generation to reach the desired number of tests. For the other subject models (EMB, CC, and Two Tanks), SDV could not be used at all. In these models, the presence of unsupported blocks prevented SDV from initiating the analysis, resulting in no test objectives or partial test cases that could be reused. A hybrid approach similar to Swimairspeed was therefore not feasible. Unlike a partial failure, SDV does not return incomplete coverage or a subset of test cases when such incompatibilities exist; it simply cannot launch. We also considered isolating SDV-compatible subsystems, but doing so would have altered model semantics and broken key data dependencies, producing coverage that does not generalize to the full closed-loop system. For these reasons, random test generation was not a design choice but the only viable option for SDV-incompatible subjects, ensuring consistent and fair test case generation across models.

Table 3 reports the key characteristics of the employed test cases, including the obtained coverage as well as the execution time. It is important to highlight two key considerations. First, this execution was conducted at the

Model-in-the-Loop (MiL) testing level. Even at this stage, variations in execution may arise due to differences in the fidelity of the physical plant representation within the Cyber-Physical System (CPS). Second, these executions must be extended to higher testing levels, namely Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL). In particular, based on our experience with industrial partners, many tests at the HiL stage necessitate manual execution. Furthermore, testing at this level may involve risks of hardware damage [83]. Consequently, test prioritization is crucial not only from the test execution time context.

4.3 Evaluation metric

The Average Percentage of Faults Detected (APFD) was selected to evaluate the approach as it is the most used metric to evaluate test case prioritization approaches [15]. To measure the APFD, let T be a test suite containing n test cases and F be a set of m mutants detected by T. Let TF_i be the index of the first test case in an ordering π of T that detects mutant i. Given an ordering π of the test suite T, the APFD metric is defined as expressed in Equation 10.

$$APFD(\pi) = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n} \tag{10}$$

Similar to the case of test cases, we used the same exact mutants as prior studies [7,5,3] in order our findings to be comparable with those test case selection studies. To the best of our knowledge, the selected models are fault-free. However, employing mutation testing in the context of test prioritization has several advantages: it enables the systematic evaluation of test effectiveness, helps uncover subtle faults distributed across different parts of the code (or models, in this case), and provides a quantitative measure for comparing the fault detection capabilities of different test prioritization strategies. In our previous studies [7,5,3], mutants were generated following the operators proposed in [50], which use a total of 10 mutation operators (listed in Table 4). As in [7,5,3], to avoid artificially inflated fault counts, we removed equivalent or duplicated mutants by automatically checking for identical output traces, and manually verifying ambiguous cases. The number of discarded mutants per model is explicitly reported in Table 3. For the case of Swimairspeed, we generated and filtered out the mutants following the same criteria as Arrieta et al. [7,5,3].

4.4 Statistical tests and algorithm runs

For the selected techniques, it might be possible that at some points more than one option can be chosen by the test case prioritization algorithm because the selected test quality metric is the same in two or more test cases. As a result, tie-breaking strategies are required. As in previous studies [41], we opted to

Table 2: Structural characteristics of the subject models, including block count, I/O, and derived complexity metrics (CFC, Density, Connections, Coupling, Interactions) used to re-analyze prioritization performance.

Case Study	**Blocks**	**Inputs**	**Outputs**	**CFC**	**Density**	**Connections**	**Coupling**	**Non-Det**	**Interactions**
CW	235	15	4	102	0.0004	264	1.87	0	12
EMB	315	1	1	19	0.0005	353	1.73	1	27
CC	31	5	2	8	0.0008	42	1.97	0	6
Swimairspeed	130	7	5	18	0.0004	128	1.57	0	5
ACEngine	257	4	1	15	0.0005	344	2.06	0	47
Two Tanks	498	11	7	63	0.0004	468	1.95	0	42

Table 3: Characteristics of the generated mutants and test cases for each of the case study systems

Case Study	TCs	Test Generation	Initial Mut	Final Mut	DC Cov	CC Cov	MCDC Cov
CW	133	SDV (MC/DC)	250	96	86.41%	86.66%	73.21%
EMB	150	Random	40	18	82.69%	100%	66.67%
CC	150	Random	60	20	100%	100%	100%
Swimairspeed	150	SDV + Random	30	9	100%	100%	100%
ACEngine	120	[8, 9]	20	12	81.25%	89.29%	50%
Two Tanks	150	Random	34	6	89.13%	91.67%	46.67%

Table 4: Mutation Operators used in our study, obtained from [50]

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	Logical Operator Replacement Operator

randomly break these ties when they happen. To account for these random variations, we run each of the techniques 100 times and employ statistical tests to compare the different techniques, as proposed by Arcuri and Briand [2].

After obtaining the raw results by running the selected prioritization techniques, we employed the Shapiro-Wilk test to assess whether the data was normally distributed or not. Since the data was not normally distributed, we employed the Mann-Whitney U-test to obtain the statistical significance between two different techniques. The significance level was set to 5%, meaning that there was a statistical significance if the p-value was lower than 0.05. The Vargha and Delaney $\hat{A}_{12}$ value was also used to assess the difference existing between the techniques.

4.5 MATLAB Version and Hardware Setup

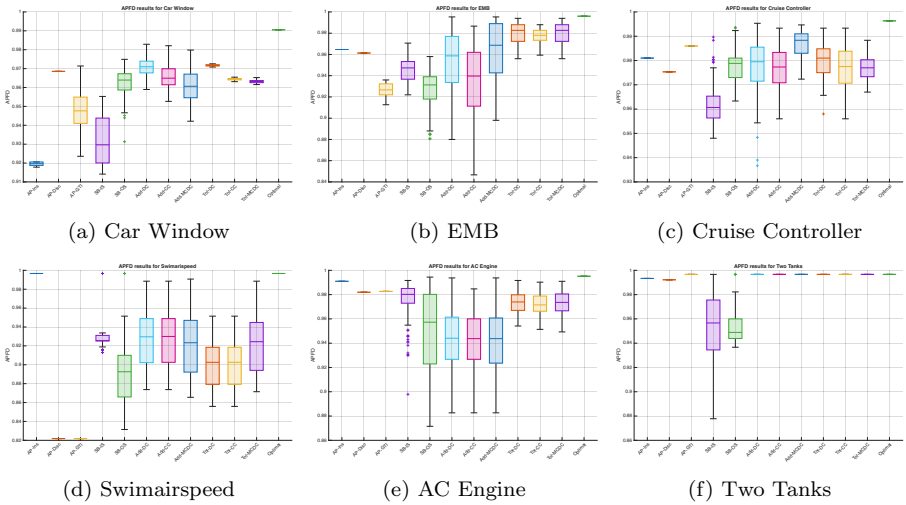
All experiments were conducted using MATLAB R2021b and Simulink with the Model Coverage toolbox enabled. The experiments were executed on a workstation equipped with an Intel i5 CPU, 16 GB RAM, running Windows 11.

5 Analysis of Results and Discussion

We now analyze the results and discuss their implications by answering the RQs. Table 5 reports the obtained statistical tests for the first three RQs. Specifically, the values reported in the table provide the Vargha and Delaney $\hat{A}_{12}$ values, used to assess the effect size between two techniques. The $\hat{A}_{12}$ measures, according to the obtained results, the probability that employing the technique in column *Technique 1* will be more effective (according to the APFD metric) than the technique in column *Technique 2*. A value higher than 0.5 means that the test case prioritization method in column *Technique 1* is more effective than the method in column *Technique 2*, whereas a value lower than 0.5 means the opposite.

Table 5: RQ1 and RQ2: Results of the Vargha and Delaney $\hat{A}_{12}$ values between two test prioritization techniques. Boldface values refer to statistical significance. i.e. p-value < 0.05 for the Mann-Whitney U-test

	Technique 1	**Technique 2**	**Car Window**	**EMB**	**Cruise Controller**	**Swimairspeed**	**AC Engine**	**Two Tanks**
RQ1	AP-Ins	AP-Disc	**0.00**	**1.00**	**1.00**	0.47	**1.00**	**1.00**
	AP-Ins	AP-GTI	**0.00**	**1.00**	**0.00**	0.47	**1.00**	**0.00**
	AP-Ins	SB-IS	**0.23**	**0.96**	**0.97**	**0.15**	**0.96**	**0.96**
	AP-Ins	SB-OS	**0.00**	**1.00**	**0.75**	**0.25**	**0.95**	**0.97**
	AP-Disc	AP-GTI	**0.98**	**1.00**	**0.00**	0.50	**0.00**	**0.00**
	AP-Disc	SB-IS	**1.00**	**0.93**	**0.92**	**0.00**	**0.63**	**0.96**
	AP-Disc	SB-OS	**0.88**	**1.00**	**0.32**	**0.00**	**0.80**	**0.97**
	AP-GTI	SB-IS	**0.83**	**0.11**	**0.98**	**0.00**	**0.68**	**0.98**
	AP-GTI	SB-OS	**0.12**	0.43	**0.97**	**0.01**	**0.82**	**0.99**
	SB-IS	SB-OS	**0.01**	**0.79**	**0.08**	**0.76**	**0.71**	**0.58**
RQ2	Add-DC	Add-CC	**0.75**	**0.67**	0.54	0.47	0.51	0.50
	Add-DC	Add-MCDC	**0.84**	**0.40**	**0.23**	**0.75**	0.52	0.50
	Add-DC	Tot-DC	0.42	**0.23**	0.45	**0.98**	**0.11**	0.50
	Add-DC	Tot-CC	**0.87**	**0.26**	0.54	**0.98**	**0.12**	0.50
	Add-DC	Tot-MCDC	**0.93**	**0.23**	0.57	**0.75**	**0.11**	0.50
	Add-CC	Add-MCDC	**0.66**	**0.25**	**0.17**	**0.76**	0.51	0.50
	Add-CC	Tot-DC	**0.17**	**0.08**	**0.39**	**0.98**	**0.09**	0.50
	Add-CC	Tot-CC	0.55	**0.09**	0.50	**0.98**	**0.10**	0.50
	Add-CC	Tot-MCDC	**0.63**	**0.08**	0.51	**0.76**	**0.10**	0.50
	Add-MCDC	Tot-DC	**0.10**	**0.36**	**0.76**	**0.92**	**0.11**	0.50
	Add-MCDC	Tot-CC	**0.34**	**0.39**	**0.83**	**0.92**	**0.12**	0.50
	Add-MCDC	Tot-MCDC	**0.39**	**0.36**	**0.90**	0.47	**0.11**	0.50
	Tot-DC	Tot-CC	**1.00**	**0.60**	**0.61**	0.50	0.53	0.50
	Tot-DC	Tot-MCDC	**1.00**	0.50	**0.65**	**0.06**	0.51	0.50
	Tot-CC	Tot-MCDC	**0.90**	**0.40**	0.51	**0.06**	0.48	0.50



(a) Car Window (b) EMB (c) Cruise Controller

(d) Swimairspeed (e) AC Engine (f) Two Tanks

Fig. 3: Distribution of the obtained APFD values for the six subject models and the selected techniques

5.1 RQ1 – Black-box techniques

When comparing the performance of black-box test prioritization techniques, it can be seen that for all subject models except for Swimairspeed, the test prioritization techniques that were based on anti-patterns stand out over the test prioritization techniques based on test similarity metrics. These results are consistent with those obtained by Arrieta et al., for the context of test selection approaches [7].

Similar to a study conducted in the context of test selection [7,6,5], anti-patterns seem to be, in general, more appropriate test quality metrics than similarity-based metrics. However, similarity-based techniques outperformed them in the Swimairspeed case study system. A potential reason could be the types of outputs the Swimairspeed system provides, which are related to discrete (software) values providing warning flags. In contrast, the rest of the case study systems provide outputs that correspond to time-continuous physical phenomena (e.g., position of the 4 windows in a car in CW, or the engine speed (in rpm-s) in the ACEngine system). Anti-pattern metrics (e.g., instability, discontinuity) are shape-sensitive and become weakly discriminative when signal variance is low, as is the case of discrete outputs in the Swimairspeed models. In contrast, similarity-based ordering can still separate tests that trigger different activation patterns and timings of flags, thereby improving early fault exposure despite low-amplitude outputs.

By analysing the characteristics of each case study from Table 2, we could not find one specific pattern by which we could recommend a test quality metric over the others for test prioritization. We conclude that the best anti-pattern metric for each case may depend on the system and type of test cases

that are used, and thus, it is difficult to recommend one such metric. All anti-patterns based metrics showed strong APFD values, all having a median above 0.9 in all case study systems. Specifically, we see that AP-Disc performs consistently well, showing median APFD values above 0.95 in 5 out of the 6 case study systems, and not having any single APFD value for all runs below 0.90. Notably, the variances of the APFD results with these subject models for the anti-patterns-based test prioritization techniques were minimal. This is because the anti-pattern degree for each test case is unique in most cases and, therefore, ties usually do not exist. As for the similarity-based metrics, similar to previous works in the context of simulation-based testing [7,6], results were not as good as the anti-patterns. Only in one case study system (i.e., Swimairspeed), similarity-based techniques performed better than anti-pattern ones. We further discuss the key issues and potential solutions for similarity-based techniques in Section 5.5.2.

Having discussed this, we can answer the RQ1 as follows:

> **RQ1:** *Anti-pattern–based techniques appear more effective than similarity-based ones when model outputs correspond to physical signals. In such cases, the shape and continuity of the output traces make anti-pattern metrics (e.g., instability, discontinuity, or growth to infinity) highly discriminative once faults are triggered. Conversely, similarity-based techniques performed better for the Swimairspeed model, whose outputs are primarily discrete control variables (e.g., flags). In these settings, where output variance is low, shape-based anti-patterns lose discriminative power, whereas similarity metrics can still distinguish tests based on differences in activation timing and control-flow behavior.*

5.2 RQ2 – White-box techniques

The third RQ aimed at assessing the performance of six white-box test prioritization techniques. In the Two Tanks subject model, all techniques performed equally in terms of the APFD. When considering statistical tests, in three simulation models (Car Window, EMB and ACEngine), Tot-DC outperformed the remaining techniques, whereas in the remaining two (i.e., Cruise Controller and Swimairspeed), the Add-MCDC and Add-CC technique performed best. When exploring the internal structure of the models, the total greedy approach outperformed additional greedy in models with higher block interactions and connections (CW, EMB, ACEngine). This pattern held across all coverage criteria (DC, CC, MCDC). Notably, the two smallest models by block count (CC, Swimairspeed) favored additional greedy, yet both had low interactions (≤ 6). This suggests that total greedy benefits from rich inter-block dependencies. The good performance of the total techniques might be because a single test case covering a test objective might not be enough to detect a fault.

This finding also explains the poor performance of white-box techniques in the context of multi-objective test case selection of simulation models [7]. In this setting, the goal of test case selection is to minimize execution time while maximizing coverage. However, even when 100% coverage is achieved with a small set of test cases, high coverage alone does not guarantee the detection of all faults. Furthermore, the subject models where Tot-DC performed best were the largest in terms of number of blocks. In larger models, more test cases are required to achieve a larger additional coverage, whereas total techniques (e.g., Tot-DC) consistently prioritize test cases that cover large portions of objectives. In the presence of complex faults, some test cases may exercise the fault without triggering a failure. In such cases, the Tot-DC may have an advantage over others because it prioritizes test cases that maximize the overall decision coverage rather than just incremental gains. This ensures that test cases selected early in the prioritization process exercise a wide variety of decision points, potentially exposing faults that only manifest under interactions across multiple decisions. In contrast, additional greedy techniques might focus on test cases that cover only a single new test objective. Another explanation could be that some faults may require the interaction of multiple decisions, which might not be covered by additional greedy techniques that solely adds minimal new coverage. Instead, a test case focusing on large total decision coverage objectives will simultaneously exercise various parts of the model, increasing the chances of revealing faults that are dependent on various the interaction of multiple decision points of the model. Moreover, similar to what happened in the previous RQ with anti-pattern metrics, the variance of the APFD results with the total test prioritization approaches seems to be lower than the other techniques. Low variance in results is a positive aspect for using this technique in practice. Thus, we can answer the second RQ as follows:

> **RQ2:** *Overall, when considering median APFD values, all white-box techniques show high competitiveness at prioritizing test cases in the context of Simulink models. However when considering statistical significance, we found that Tot-DC performed best in three out of six subject models, and obtained the same results as the rest in a fourth subject models; all these models exhibited higher structural interdependence, particularly in terms of block interactions (>10) and connections (>200).* In contrast, Add-MCDC and Add-CC performed best in the two models with lower interactions (i.e., Cruise Controller and Swimairspeed).

5.3 RQ3 – Overall best technique

RQ3 aimed at selecting the best overall metric. To this end, we compared the best techniques obtained from RQ1 and RQ2 for each subject model. As reported in Table 6, white-box techniques outperformed black-box in four models (CW, EMB, CC, Swimairspeed), all with moderate interactions (5–27). Conversely, black-box (AP-Ins) dominated in ACEngine (Interactions = 47), and

Table 6: RQ3: Results of the Vargha and Delaney $\hat{A}_{12}$ values between the best two test prioritization techniques obtained from RQ1 and RQ2

Subject	Best BB Technique	Best WB Technique	$\hat{A}_{12}$	p-value	Overall Best
Car Window	AP-Disc	Tot-DC	0	<0.0001	Tot-DC
EMB	AP-Ins	Tot-DC	0.11	<0.0001	Tot-DC
Cruise Controller	AP-GTI	Add-MCDC	0.36	<0.0001	Add-MCDC
Swimairspeed	SB-IS	Add-CC	0.0125	<0.0001	Add-CC
ACEngine	AP-Ins	Tot-DC	0.99	<0.0001	AP-Ins
TwoTanks	AP-GTI	ALL	0.5	-	same

tied in Two Tanks (Interactions = 42). This threshold effect (black-box gains advantage above around 40 interactions) suggests that anti-pattern metrics on output signals become more informative as system coupling increases, likely due to richer dynamic behaviors.

Unlike in the context of multi-objective test case selection [7], where black-box approaches significantly outperformed white-box ones, for the context of test prioritization, white-box techniques are as competitive as black-box ones, performing better in most of the studied case study systems. A reason why white-box metrics performed well in the context of test prioritization and not that well in the context of test selection could be that, in the latter case, the objective was to reduce the test execution time as much as possible while increasing test coverage. Because of this, such techniques will tend to execute specific test objectives only once in order to maximize the coverage while minimizing execution times, which might not be a good approach for detecting (all the) faults. Regardless, it is still possible for coverage-based test prioritization techniques to execute (the same) test objectives multiple times, especially for total approaches. Although when considering statistical results, in general, white-box techniques were better than the black-box ones, when considering median values, the black-box techniques were not far from the white-box ones. Moreover, in one of the models (i.e., ACEngine) black-box techniques showed better performance, and obtained the same APFD scores as white-box ones in another one. Furthermore, for those subject models where white-box techniques performed better, the median values of the APFD for the black-box techniques were quite close. Besides, the variances of white-box techniques were larger than the variances of the black-box anti-patterns based test prioritization techniques, which means that tie-breaking is required more often for the former.

We further compared the overall best technique extracted with the optimal one, which was obtained by considering the information of test cases exposing the seeded faults. As can be seen in Table 7, the best techniques are quite close to the optimal one. In fact, for one of the subject simulation models (i.e., Two Tanks), the best techniques yielded optimal results. For the remaining subject models, the optimal test order outperformed the best technique by a margin of 0.4% to 1.9% in terms of the APFD. We consider these values fairly good, which means that the proposed test prioritization algorithms can be useful in practice and close to optimality. The gap to optimal ordering was

Table 7: Improvement extent by the optimal test prioritization with respect to the best technique for each subject model

Subject model	**Best technique**	**Improvement Extent**
CW	Tot-DC	1.9%
EMB	Tot-DC	1.63%
CC	Add-MCDC	0.92%
Swimairspeed	Add-CC	0.53%
ACEngine	AP-Inst	0.42%
TwoTanks	same	0%

highest in CW (1.9%) and EMB (1.63%), both with high CFC (102, 19) and moderate-to-high connections. Lower gaps in Two Tanks (0%) and ACEngine (0.42%), despite large size, may correspond to more uniform coverage distribution, stronger test cases in the sample, or simpler fault exposure patterns. CFC and Connections better predict optimization potential than block count alone. In conclusion, the third RQ can be answered as follows:

> **RQ3:** *In four of the models, the best white-box techniques performed better than the black-box ones. The best black-box technique only performed better than the white-box one in one of the models. However, when considering average values, the differences were quite small. The best test prioritization techniques are quite close to the optimal test ordering in all case studies, which means that they can be recommended for practitioners.*

5.4 RQ4 – Test prioritization execution time

RQ4 aimed at evaluating the test prioritization time required by each technique. The overall results are reported in Table 8, where the average running time for each technique and its standard deviation is shown. From these results, it can be seen that black-box metrics were, in the cases considered, significantly better than white-box techniques in this aspect; this was corroborated by statistical tests, where black-box techniques were shown to outperform all white-box ones with statistical significance for the six subject models (in all cases). When taking a closer look at black-box test prioritization techniques, it can be seen that the techniques based on anti-patterns, which require less than a millisecond on average, are faster than the techniques based on similarity. Nevertheless, both similarity-based metrics are around 0.01 seconds on average, which is something that is, by far, affordable by practitioners.

As for white-box metrics, it can be observed that additional techniques were much slower than total techniques. On average, total techniques required below 0.4 seconds, whereas additional techniques took more than 135 seconds for all subject models, and more than 180 seconds for two of the subject models. In order to get higher insights regarding the execution time of these algorithms, we launched a profiler that measured the time it takes for each

line of code in the algorithm to execute.[5] When we saw the results, we found out that the bottleneck was on the API calls for measuring the coverage of Simulink models. In fact, for the selected technique, five lines of code from the algorithm took 99.8% of the running time. These five lines of code are all related with the measurement of the white-box code coverage with the API, i.e., the lines of code required to determine, for a newly executed test case, the additional coverage objectives it satisfies. Figure 4 shows a screenshot of this analysis, where these five lines of code are shown. In the case of the total greedy approach, the time taken is lower because the API is called only a few times (i.e., the number of test cases it has). On the contrary, the additional techniques are iterative, and they require many more API calls, thus, considerably increasing the total running time. Our profiler indicates that the overhead is dominated by coverage-API calls (99.8% of time in five lines using the API). This suggests a tooling bottleneck rather than an inherent limitation of white-box strategies. In practice, caching or batching coverage queries, or precomputing coverage matrices, could mitigate this overhead, as it is well-known that MATLAB has quick optimization for matrix calculus; this, however, would require significant engineering work, a cost that rarely would be assumed by most industrial companies. Instead, our evaluation uses the off-the-shelf API as typically experienced by practitioners. With all this discussion in mind, can answer the last RQ as follows:

> **RQ4:** *Black-box test prioritization techniques are significantly faster than white-box ones. On the other hand, total techniques are much faster than additional techniques. The overhead of the additional techniques is mainly caused by the high number of calls to the coverage API, which significantly increases the test prioritization time.*

5.5 Discussion

5.5.1 Concluding Remark and Recommendations

The empirical evaluation showed that there is no single best technique for prioritizing test cases for Simulink models. Nevertheless, there are some insights that are worth mentioning as a summary:

- White-box techniques are overall slightly stronger than black-box ones, which contrasts with previous studies [7].
- Among the black-box techniques, overall, anti-patterns based test prioritization techniques performed better than distance-based techniques.
- Among the white-box techniques, the total technique along with the DC metric (Tot-DC) performed best in three out of six subject models, and it

[5] We launched this with the Add-DC technique along with the two tanks subject model, but it is expectable that similar results will yield with other additional greedy-based techniques and other subject models too

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
49	`covdata = covdataWithNewTC + d...`	11026	241.047 s	84.7%	
59	`decisiondata = decisioninfo(co...`	11175	36.222 s	12.7%	
71	`covdataWithNewTC = covdata + d...`	150	3.280 s	1.2%	
25	`covdata = covdataWithNewTC + d...`	149	3.081 s	1.1%	
35	`decisiondata = decisioninfo(co...`	150	0.538 s	0.2%	
All other lines			0.554 s	0.2%	
Totals			284.722 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
cvdata.plus	function	11325	247.058 s	86.8%	
decisioninfo	function	11325	36.633 s	12.9%	
Self time (built-ins, overhead, etc.)			1.031 s	0.4%	
Totals			284.722 s	100%	

Fig. 4: Screenshot of the profiler for the time taken by a single instance run of the Add-DC technique

performed equally in the Two Tanks subject model. This technique was the best for models with higher structural interdependence, particularly in terms of block interactions (>10) and connections (>200), whereas Add-MCDC and Add-CC were the best for models with lower interactions ($\leq$6; i.e., Cruise Controller and Swimairspeed). This suggests that total techniques are more appropriate in models with higher structural interdependence, whereas additional techniques in models with lower structural interdependence.
– Black-box techniques were faster than white-box ones. Nevertheless, total greedy based test prioritization techniques are fast enough to be applied in practice.
– Similarity-based test prioritization techniques were overall less competitive than the remaining techniques. In Section 5.5.2 we give some insights of why these metrics may fail to perform well in the context of Simulink models and guide future researchers on potential solutions.
– Results may differ depending on the Simulink model at which testing is being applied. However, our study shows evidence that either AP-Ins or Tot-DC could perform competitively, and thus, we recommend either of both when test prioritization methods need to be applied.

Based on our experiments and observations, we offer the following recommendations for practitioners using test prioritization techniques with Simulink models:

– **Recommendation 1:** Within our six subject models, no single technique consistently dominated. Provided that Simulink models take long test execution times, experiments with large corpus of case study systems is not usually possible, therefore we caution against universal conclusions. Outcomes

Table 8: RQ5: Average running time (in seconds) and standard deviation for each test prioritization technique

		AP-ins	AP-Disc	AP-GTI	SB-IS	SB-OS	AR-DC	AR-CC	AR-MCDC	Tot-DC	Tot-CC	Tot-MCDC
Car Window	Avg	0.00071	0.00053	0.00040	0.01474	0.01295	152.39616	152.09648	152.26632	0.34866	0.34939	0.34914
	stdDev	0.00425	0.00279	0.00187	0.01398	0.00458	4.08801	3.79466	3.56654	0.01348	0.01592	0.01365
EMB	Avg	0.00032	0.00024	0.00023	0.01508	0.01465	156.87667	151.42402	156.90839	0.38903	0.38674	0.38819
	stdDev	0.00029	0.00016	0.00014	0.00187	0.00137	3.79300	3.56347	4.03221	0.02409	0.02171	0.02260
Cruise Controller	Avg	0.00087	0.00028	0.00025	0.01501	0.01340	146.45570	51.91876	107.30989	0.28479	0.28276	0.28205
	stdDev	0.00569	0.00063	0.00024	0.00732	0.00036	3.42112	0.91751	3.55128	0.00557	0.00617	0.00421
Swimairspeed	Avg	0.00084	0.00041	0.00037	0.01805	0.01593	197.84829	194.58293	187.34234	0.30118	0.3102	0.31038
	stdDev	0.000314	0.00026	0.00033	0.00956	0.00079	10.53843	10.3582	10.16534	0.00476	0.00493	0.00475
AC Engine	Avg	0.00615	0.00032	0.00028	0.01407	0.01220	136.40267	135.96562	139.81619	0.42410	0.42207	0.42517
	stdDev	0.04902	0.00060	0.00028	0.00718	0.00212	16.41926	16.00144	39.72185	0.05656	0.05402	0.05721
Two Tanks	Avg	0.00039	0.00024	0.00022	0.01589	0.01367	184.91517	184.94793	180.45511	0.30852	0.30580	0.30632
	stdDev	0.00084	0.00018	0.00017	0.00956	0.00041	9.76272	9.79499	9.73668	0.00476	0.00342	0.00405

likely depend on system type, available test suites, and controller structure, including metrics such as Control Flow Complexity (CFC), Density, Connections, Coupling, and Interactions. In practice, we suggest monitoring how anti-patterns and coverage objectives correlate with detected faults in a given project, where such data exist, and otherwise treating the following recommendations as provisional guidance rather than general rules.

- **Recommendation 2:** In general, we do not recommend the use of similarity-based test prioritization techniques. We found that the Euclidean distance may fail to capture some important signal features in the context of Simulink models testing (further discussed in Section 5.5.2).
- **Recommendation 3:** In our sample, white-box techniques (total and additional) were broadly competitive for Simulink-based prioritization. We observed that total strategies, and particularly Tot-DC, often fared well on models with higher structural interdependence, as measured by block interactions (>10) and connections (>200). However, we cannot ascribe this to size alone. Additional strategies, such as Add-MCDC or Add-CC, were more competitive in models with lower interactions ($\leq$ 6; i.e., Cruise Controller and Swimairspeed). We hypothesize that other structural factors (e.g., CFC, Density, Coupling) may further mediate their effectiveness and scalability. Accordingly, for models with high interactions or connections, Tot-DC is a reasonable default to try first, whereas on models with low interactions, Add-MCDC or Add-CC can be competitive (bearing in mind that additional strategies may incur higher runtime). These recommendations are tentative and should be validated against project-specific structure and cost constraints.
- **Recommendation 4:** White-box techniques are not always available (e.g., if the subsystem is from an external company that does not provide the model, but only the executable). In those cases, we recommend practitioners to resort to anti-patterns based test prioritization techniques, but only when the outputs of the models relate to signals of physical phenomena (e.g., speed of a car). In contrast, if the outputs are related to discrete software-related values (e.g., flags, communication signals), although not ideal, we recommend the use SB-IS.

5.5.2 Qualitative Analysis and Future Research Avenues

This paper conducts the first attempt to assess traditional white-box and black-box techniques in the context of Simulink models test prioritization, with interesting insights discussed through 5 different RQs and summarized in Section 5.5.1. By extracting such insights, we now conduct a qualitative analysis and extract future research avenues that can help guide researchers in their future research efforts:

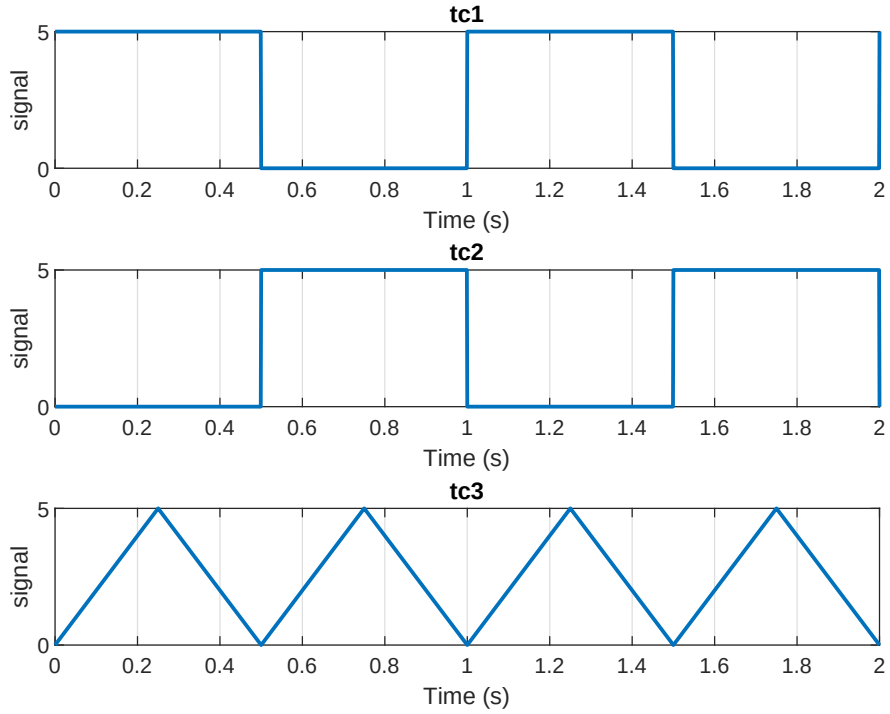
Research novel metrics for similarity-based test prioritization: Our results reveal that traditional similarity-based test prioritization methods are less competitive than other techniques. We believe that future research avenues could focus on researching novel distance metrics that consider signal

shapes and other features that the Euclidean distance (i.e., the one employed in this and other studies [64,65,6,7]) fails to capture. As an illustration of this problem, consider Figure 5, which includes 3 different test cases. Test cases tc_1 and tc_2 have very similar shapes (square signal with the same frequency), but the Euclidean distance between both of these test cases is the maximum (assuming no other signal exists in the test case). In many Simulink models, both of these test cases will test similar functionalities. In contrast, tc_3 provides a triangle wave signal. The Euclidean distance between this signal with tc_1 is lower than that with tc_3, whereas there is a high likelihood that this test case tests other properties not covered by tc_1 and tc_2. This example highlights a key limitation of relying solely on Euclidean distance as a similarity metric: it fails to account for the structural differences in signal shapes that can be critical for assessing the diversity and functional coverage of test cases. Future work could explore the development of more sophisticated distance metrics or embedding techniques that capture features such as signal shape, frequency content, and temporal patterns, enabling more effective prioritization strategies for simulation-based testing of CPSs in Simulink and other domains. Other distance metrics, such as the Fréchet distance and the Manhattan distance, as well as more sophisticated distances (e.g., the Fourier coefficient distance), represent a future research avenue.

Criticality of bugs: The criticality of bugs in CPSs varies based on the violated requirements. For example, in autonomous vehicles, drivability requirements [30] related to comfort are less critical than safety requirements, such as collision avoidance. Bugs violating safety requirements are more crucial to detect early. Since our test prioritization techniques make no assumptions about which system properties are critical, comparing the types of bugs detected would be unfair. Future work could explore prioritization techniques that consider system criticality. For instance, a Domain Specific Language could be proposed to specify the criticality of Simulink subsystems or test outputs affecting safety. This information could later be processed by the prioritization technique. This would also require adapting the APFD metric to account for bug criticality.

Severity of failures: Likewise, severity of bugs is also an aspect that could be measured in the future. Although related to criticality, these concepts capture different aspects of impact. Criticality refers to how essential the affected function is to system safety or mission success (i.e., the potential consequences if it fails), whereas severity describes the magnitude of the system degradation or performance loss when the failure actually occurs. Following with an example from the automotive domain, crashing a vehicle at *20 km/h* is less critical than doing it at *120 km/h*. In the future, it would be interesting to adapt the APFD metric to consider such aspect.

Test prioritization for flaky tests: While the models in our benchmark are deterministic, some CPS simulators exhibit flakiness [1], meaning that tests may pass or fail inconsistently even when the system and simulator configurations remain unchanged. Moreover, in later stages (e.g., HiL), where hardware devices are involved, stochastic simulations are a common challenge.



ht

Fig. 5: An example showing the problems of the Euclidean distance at capturing signal shapes

Adapting test prioritization techniques to address this issue presents a promising direction for future research.

Adequacy techniques in AI-enabled CPSs: AI models (e.g., neural networks) are increasingly prevalent in the control of CPSs. However, traditional white-box techniques cannot be directly applied to assess the quality of test cases for these AI components. Exploring the integration of adequacy techniques from this domain (e.g., neuron coverage) with conventional white-box test prioritization techniques and evaluating their impact on fault detection rates represents another promising and unexplored research avenue.

Tie-breaking strategies: As shown in Figure 3, many techniques have a large distribution. This distribution could eventually be reduced, and the APFD probably increased, by integrating tie-breaking strategies in the test prioritization techniques.

Use of multiple metrics as test prioritization method: In this study, we deliberately focused on traditional single-objective prioritization techniques. These approaches are widely adopted in practice and provide fundamental baselines that allow isolating the contribution of each metric in a controlled and interpretable manner. Nonetheless, practical testing often involves multiple (and often competing) goals (e.g., balancing white-box and

black-box metrics). Extending our findings to hybrid or multi-objective prioritization strategies therefore represents a promising research avenue. Future work could investigate search-based formulations that combine metrics such as coverage, similarity, and instability into Pareto-efficient solutions, leveraging insights from both the white- and black-box perspectives identified in this study.

Weighting of signals based on signal importance: In our experiments, all output signals were assigned equal weights when normalizing metric values, as domain knowledge about their relative importance was unavailable for the benchmark systems. In industrial contexts, however, certain signals (e.g., safety-critical control variables) may have greater diagnostic relevance than others. Assigning differentiated weights could therefore make prioritization metrics more representative of actual engineering priorities and potentially improve their effectiveness. Future work could explore adaptive or expert-informed weighting schemes to better align test prioritization with system-criticality considerations.

Study with large-scale industrial models: Although our benchmark includes models ranging from 31 to 498 blocks, we note that this scale is representative of the subsystem-level Simulink models typically used in CPS development. Prior work surveying hundreds of open-source Simulink projects shows that most models fall below this size range, and industrial test activities are commonly performed at the subsystem level rather than on monolithic plant–controller assemblies. The inclusion of the EMB system, an industrial model used in previous studies, further grounds our evaluation in realistic practice. While production models can indeed be larger or more heterogeneous, the comparative nature of our study mitigates this limitation: the relative behaviors of the different prioritization techniques depend primarily on structural properties of the models and the metrics rather than on absolute model size. Nevertheless, we see substantial value in extending this work to larger proprietary systems, and we have released all our artifacts to support such industrial replications.

6 Threats to Validity

We now discuss the threats to validity of our study and how we tried to mitigate them.

A potential **internal validity** threat of our empirical evaluation is related to the generated mutants. In the context of Simulink, models often have a physical layer that is typically modeled through complex mathematical equations. As a result, the execution of test cases is time consuming, making the use of a large set of mutants impractical. However, the amount of mutants used in this study is similar to other studies where MATLAB/Simulink models were used [65, 11, 10, 12, 64, 56, 57, 55, 50, 35, 66]. Besides, we have tried to mitigate this threat by removing duplicated mutants, as recommended by Papadakis et al. [76]. As for subsumed mutants, our goal was to compare the

relative ordering effectiveness of prioritization techniques rather than absolute fault detection. Even when subsumption exists, some test cases detect more mutants (both subsuming and subsumed) earlier than others, allowing meaningful comparisons of fault detection rate in the field of test case prioritization.

An **external validity** threat in all software engineering empirical evaluations is related to the generalization of results. We used six subject models, which might not be enough to generalize our results. Nevertheless, we employed simulation models of different characteristics and sizes to reduce this threat. Additionally, one of the subject models, the EMB, was an industrial case study developed by Bosch. When referring to the size of the subject models, according to a previous paper where 391 public Simulink models were analyzed, more than half of the analyzed models had less than 100 blocks and around 75% of the models had less than 300 blocks [19]. In our empirical study, five out of six subject models had from 130 to 498 blocks, meaning that their complexities in terms of model size are higher than most of the public subject models. In addition, four of the blocks from the Car Window subject model were state machines with 5 states and 21 transitions each (implemented as Stateflow charts). Although our benchmark spans models ranging from 31 to 498 blocks, including widely studied subjects such as EMB and Two Tanks, we acknowledge that some industrial CPS models can be larger or contain more heterogeneous subsystems. However, based on our experience with industrial partners (e.g., Orona), subsystem-level Simulink models frequently fall within or below this size range, as Simulink is commonly used to model individual control loops or plant components rather than entire systems. Thus, the selected models reflect the granularity typically targeted in CPS testing research. Still, replications on larger industrial models would further strengthen external validity, and we provide all artifacts to support such efforts.

A **conclusion validity** threat in our study might be related to the stochastic nature of our experiments. This is caused by our strategy of randomly breaking ties between multiple test cases with the same adequacy score, which has been proposed in other studies (e.g., [41]). To reduce the threat produced by the randomization of our algorithms, we ran each algorithm 100 times and applied statistical tests to analyze the results, as recommended by Arcuri and Briand [2].

7 Related Work

Testing has been widely applied to MATLAB/Simulink, the defacto tool for modeling and simulating CPSs. A number of studies have proposed automated test generation approaches, including mutation-based test generation [35], generation based on formal methods [37] and model checking [70], search-based test case generation [65,63,66,29,28], falsification-based test generation [68, 72,31] and fuzzing [87]. Besides test generation, other approaches have proposed other testing activities, including automated fault localization [57,55, 56], test oracle generation [69], mutation testing [94] and test case selection [6,

7,10,5,3]. Unlike all these studies, this paper focuses on test case prioritization for regression testing of Simulink models. The use of anti-patterns for Simulink models was first proposed by Matinnejad et al. [65,63]. In this paper, we adapted the proposed anti-patterns measures to the context of test case prioritization. However, it is important to note that the definition of some measures for test suite generation has not prevented successfully reusing the same measures in test case prioritization in other contexts [41].

As for test case prioritization in the context of Simulink models, Arrieta et al. [12,11] considered information related to a historical database to prioritize test cases in a cost-effective manner by using weighted search algorithms. Matinnejad et al. [66], along with their test generation approach, proposed a complementary test case prioritization algorithm that uses a combination of model coverage information (white-box) and the output diversity of the test suites (black-box) as surrogate criteria for a greedy algorithm to estimate the probability of revealing new faults with each test case. In their mutation-based experiments with two industrial Simulink models, they conclude that this approach significantly outperforms random prioritization, as well as total and additional coverage-based algorithms (white box). The study we present differs from those ones by (1) focusing exclusively on either white-box or black-box techniques, (2) using traditionally employed test case prioritization approaches in the context of Simulink models and (3) providing an empirical evaluation for a total of 11 test case prioritization techniques.

Empirical evaluations of test case prioritization techniques have been widely proposed in the past [78,77,79,25,24,26,38,43,44,48,47,59,60,41]. One of the first empirical studies in test case prioritization techniques was proposed by Rothermel et al. [78]. Specifically, they compared several total and additional test case prioritization techniques, in addition to random and unordered test case prioritization for applications programmed in C. Do et al. proposed an evaluation of additional and total strategies to test Java programs tested under the JUnit framework [22]. Luo et al. compared the performance of static and dynamic test case prioritization techniques applied in Java programs [59,60]. A major finding was that static techniques performed best at test-class level whereas dynamic ones performed better at test-method level when considering APFD values. Shin et al. [82] empirically evaluated how mutation testing criteria (traditional and diversity-aware) could be employed as test quality metric to prioritize test cases. Their findings included that there was no single dominant technique. Similar to this work, Henard et al. proposed a comparison between black-box and white-box test case prioritization techniques [41]. They found little difference between white-box and black-box techniques. The main difference between prior empirical evaluations and the study we propose is related to the type of systems being targeted. While most of the empirical works for test case prioritization focus on testing languages like C or Java, in this paper we propose and evaluate techniques for test case prioritization designed for Simulink models, the main models used to simulate CPSs. The main difference between programming languages like Java or C with respect to Simulink is the continuous-time semantics of Simulink with respect to dis-

crete semantics of such languages. This leverages opportunities to propose new test quality metrics, such as those proposed at the signal-level and considered as "anti-patterns". While there are other modeling/simulation languages with continuous-time semantics (e.g., Labview or BCVTB), to the best of our knowledge, this is the first work that performs an empirical evaluation of classical test case prioritization techniques in the context of modeling/simulation languages.

8 Conclusion

This paper conducted, what, to the best of our knowledge is, the first empirical study of test case prioritization techniques for the context of Simulink models. Specifically, we analyzed a total of 11 black-box and white-box techniques for test case prioritization adapted to Simulink models. Our key finding were interesting. On the one hand, we found that, unlike for the context of test case selection [7], for the context of test case prioritization white-box techniques are slightly more competitive than black-box techniques. Another finding was that, in general, total greedy-based test case prioritization techniques were more effective than additional greedy-based test case prioritization techniques, particularly in models exhibiting higher structural interdependence (e.g., block interactions >10 and connections >200). On the other hand, as expected and similar to the context of test case selection [7], we found that black-box techniques that rely on anti-patterns to measure the quality of test cases are more effective than similarity-based black-box test case prioritization techniques. Overall, results may differ from a system to another, although most techniques provide consistently high APFD values, suggesting that they are appropriate for test case prioritization. Our recommendations in Section 5.5.1 provides more insights and specific recommendations based on the setup of practitioners (e.g., availability of access to white-box metrics, models structural characteristics).

Data Availability: We make all the scripts, model and results available: `https://doi.org/10.6084/m9.figshare.28777829`

Acknowledgments

The author is part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country. This work has been partially funded by the Spanish Ministry of Science, Innovation and Universities (project PID2023-152979OA-I00), funded by MCIU /AEI /10.13039/501100011033 / FEDER, UE). The author would like to thank Prof. Gregg Rothermel for comments and feedback in a prior version of the manuscript.

Compliance With Ethical Standards

Conflict of Interest. The author declared that they have no conflict of interest that might have affected the outcome of the empirical study with developers.

Funding. The author is part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country.
Ethical Approval. According to the regulations of Mondragon University, where the study was carried out, there was no need for ethical approval.

Informed Consent. Informed consent was obtained from all individual participants involved.

Author Contributions. The single author conducted all the work.

Data Avaliability. We make all the scripts, model and results available: `https://doi.org/10.6084/m9.figshare.28777829`

References

1. Mohammad Hossein Amini, Shervin Naseri, and Shiva Nejati. Evaluating the impact of flaky simulators on testing autonomous driving systems. Empirical Software Engineering, 29(2):47, 2024.
2. Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In Software Engineering (ICSE), 2011 33rd International Conference on, pages 1–10. IEEE, 2011.
3. Aitor Arrieta and Miren Illarramendi. A novel mutation operator for search-based test case selection. In International Symposium on Search Based Software Engineering, pages 84–98. Springer, 2023.
4. Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria, and Justyna Zander. Automatic generation of test system instances for configurable cyber-physical systems. Software Quality Journal, 25(3):1041–1083, 2017.
5. Aitor Arrieta, Pablo Valle, Joseba A Agirre, and Goiuria Sagardui. Some seeds are strong: Seeding strategies for search-based test case selection. ACM Transactions on Software Engineering and Methodology, 32(1):1–47, 2023.
6. Aitor Arrieta, Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Multi-objective black-box test case selection for cost-effectively testing simulation models. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, pages 1411–1418, New York, NY, USA, 2018. ACM.
7. Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. Pareto efficient multi-objective black-box test case selection for simulation-based testing. Information & Software Technology, 114:137–154, 2019.
8. Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Search-based test case generation for cyber-physical systems. In Evolutionary Computation (CEC), 2017 IEEE Congress on, pages 688–697, 2017.
9. Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire 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):1055–1066, 2018.

10. Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire 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, pages 297–306, 2016.
11. Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire 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, pages 1053–1060, New York, NY, USA, 2016. ACM.
12. Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. Journal of Systems and Software, 149:1 – 34, 2019.
13. Jon Ayerdi, Aitor Garciandia, Aitor Arrieta, Wasif Afzal, Eduard Enoiu, Aitor Agirre, Goiuria Sagardui, Maite Arratibel, and Ola Sellin. Towards a taxonomy for eliciting design-operation continuum requirements of cyber-physical systems. In 2020 IEEE 28th International Requirements Engineering Conference (RE), pages 280–290. IEEE, 2020.
14. Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. Testing the untestable: Model testing of complex software-intensive systems. In Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, pages 789–792. ACM, 2016.
15. Cagatay Catal and Deepti Mishra. Test case prioritization: A systematic mapping study. Software Quality Journal, 21(3):445–478, September 2013.
16. Tsong Yueh Chen, F-C Kuo, Robert G Merkel, and Sebastian P Ng. Mirror adaptive random testing. Information and Software Technology, 46(15):1001–1010, 2004.
17. Tsong Yueh Chen, Fei-Ching Kuo, Robert G Merkel, and TH Tse. Adaptive random testing: The art of test case diversity. Journal of Systems and Software, 83(1):60–66, 2010.
18. Tsong Yueh Chen, Hing Leung, and IK Mak. Adaptive random testing. In Annual Asian Computing Science Conference, pages 320–329. Springer, 2004.
19. Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T Johnson, and Christoph Csallner. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In Proceedings of the 40th International Conference on Software Engineering, pages 981–992. ACM, 2018.
20. David Christhilf and Barton Bacon. Simulink-based simulation architecture for evaluating controls for aerospace vehicles (sarec-asv). In AIAA Modeling and Simulation Technologies Conference and Exhibit, page 6726, 2006.
21. Oscar Cornejo, Fabrizio Pastore, and Lionel C Briand. Mutation analysis for cyber-physical systems: Scalable solutions and results in the space domain. IEEE Transactions on Software Engineering, 48(10):3913–3939, 2021.
22. Hyunsook Do, Gregg Rothermel, and Alex Kinneer. Empirical studies of test case prioritization in a junit testing environment. In 15th international symposium on software reliability engineering, pages 113–124. IEEE, 2004.
23. John C. Eidson, Edward A. Lee, and Slobodan Matic. Distributed real-time software for cyber-physical systems. Proceedings of the IEEE, 100:45–59, 2011.
24. Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. IEEE transactions on software engineering, 28(2):159–182, 2002.
25. Sebastian Elbaum, Gregg Rothermel, and John 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), pages 235–245. ACM, 2014.
26. Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund 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, pages 234–245, New York, NY, USA, 2015. ACM.
27. Robert Feldt, Simon M. Poulding, David Clark, and Shin Yoo. Test set diameter: Quantifying the diversity of sets of test cases. In 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016, pages 223–233, 2016.

28. Federico Formica, Tony Fan, and Claudio Menghi. Search-based software testing driven by automatically generated and manually defined fitness functions. ACM Transactions on Software Engineering and Methodology, 2022.
29. Federico Formica, Tony Fan, Akshay Rajhans, Vera Pantelic, Mark Lawford, and Claudio Menghi. Simulation-based testing of simulink models with test sequence and test assessment blocks. IEEE Transactions on Software Engineering, 2023.
30. Federico Formica, Nicholas Petrunti, Lucas Bruck, Vera Pantelic, Mark Lawford, and Claudio Menghi. Test case generation for drivability requirements of an automotive cruise controller: An experience with an industrial simulator. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1949–1960, 2023.
31. Federico Formica, Nicholas Petrunti, Lucas Bruck, Vera Pantelic, Mark Lawford, and Claudio Menghi. Test case generation for drivability requirements of an automotive cruise controller: An experience with an industrial simulator. In Satish Chandra, Kelly Blincoe, and Paolo Tonella, editors, Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023, pages 1949–1960. ACM, 2023.
32. Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. Experience paper: Search-based testing in automated driving control applications. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 26–37. IEEE, 2019.
33. Milos Gligoric, Stas Negara, Owolabi Legunsen, and Darko Marinov. An empirical evaluation and comparison of manual and automated test selection. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pages 361–372. ACM, 2014.
34. Carlos A. González, Mojtaba Varmazyar, Shiva Nejati, Lionel C. Briand, and Yago Isasi. Enabling model testing of cyber-physical systems. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18, pages 176–186, New York, NY, USA, 2018. ACM.
35. Le Thi My Hanh, Nguyen Thanh Binh, and Khuat Thanh 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):17–30, 2016.
36. Dan Hao, Lingming Zhang, Lu Zhang, Gregg Rothermel, and Hong Mei. A unified test case prioritization approach. ACM Transactions on Software Engineering and Methodology (TOSEM), 24(2):10, 2014.
37. Nannan He, Philipp Rümmer, and Daniel Kroening. Test-case generation for embedded simulink via formal concept analysis. In Proceedings of the 48th Design Automation Conference, pages 224–229, 2011.
38. H. Hemmati, Z. Fang, and 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), pages 1–10, 2015.
39. Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity. ACM Transactions on Software Engineering and Methodology, 22(1):6:1–6:42, 2013.
40. Hadi Hemmati and Lionel Briand. An industrial investigation of similarity measures for model-based test case selection. In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, pages 141–150. IEEE, 2010.
41. Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. Comparing white-box and black-box test prioritization. In Proceedings of the 38th International Conference on Software Engineering, pages 523–534. ACM, 2016.
42. Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on, pages 188–197. IEEE, 2013.
43. Bo Jiang and W.K. Chan. Input-based adaptive randomized test case prioritization: A local beam search approach. Journal of Systems and Software, 105(0):91 – 106, 2015.

44. James A Jones and Mary Jean Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on software Engineering, 29(3):195–209, 2003.
45. Aaron Kane, Thomas E. Fuhrman, and Philip 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, pages 148–155, 2014.
46. Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang NA Jawawi, and Rooster Tumeng. Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 2017.
47. Bogdan Korel, George Koutsogiannakis, and Luay H Tahat. Application of system models in regression test suite prioritization. In Software Maintenance, 2008. ICSM 2008. IEEE International Conference on, pages 247–256. IEEE, 2008.
48. Bogdan Korel, Luay Ho Tahat, and Mark Harman. Test prioritization using system models. In Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on, pages 559–568. IEEE, 2005.
49. Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. Multi-objective black-box test case selection for system testing. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 1311–1318. ACM, 2017.
50. Khuat Thanh Le Thi My Hanh and Nguyen Thanh Binh Tung. Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing. International Journal of Computer and Information Technology, 3(04):763–771, 2014.
51. Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. Prioritizing test cases with string distances. Automated Software Engineering, 19(1):65–95, 2012.
52. Feng Li, Jianyi Zhou, Yinzhu Li, Dan Hao, and Lu Zhang. Aga: An accelerated greedy additional algorithm for test case prioritization. IEEE Transactions on Software Engineering, 48(12):5102–5119, 2021.
53. Zheng Li, Mark Harman, and Robert M Hierons. Search algorithms for regression test case prioritization. IEEE Transactions on software Engineering, 33(4):225–237, 2007.
54. Xiao Ling and Tim Menzies. What not to test (for cyber-physical systems). IEEE Transactions on Software Engineering, 49(7):3811–3826, 2023.
55. Bing Liu, Lucia, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Simulink fault localization: an iterative statistical debugging approach. Software Testing, Verification and Reliability, 26(6):431–459, 2016.
56. Bing Liu, Lucia Lucia, Shiva Nejati, and Lionel 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.
57. Bing Liu, Shiva Nejati, Lionel C Briand, et al. Effective fault localization of automotive simulink models: achieving the trade-off between test oracle effort and fault localization accuracy. Empirical Software Engineering, pages 1–47, 2018.
58. Yiling Lou, Dan Hao, and Lu Zhang. Mutation-based test-case prioritization in software evolution. In 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pages 46–57. IEEE, 2015.
59. Qi Luo, Kevin Moran, and Denys 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, pages 559–570. ACM, 2016.
60. Qi Luo, Kevin Moran, Lingming Zhang, and Denys Poshyvanyk. How do static and dynamic test case prioritization techniques perform on modern software systems? an extensive study on github projects. IEEE Transactions on Software Engineering, pages 1054–1080, 2019.
61. Dusica Marijan, Arnaud Gotlieb, and Sagar 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), pages 540–543. IEEE Computer Society, 2013.

62. Reza Matinnejad, Shiva Nejati, Lionel Briand, Thomas Bruckmann, and Claude Poull. Search-based automated testing of continuous controllers: Framework, tool support, and case studies. Information and Software Technology, 57:705 – 722, 2015.
63. Reza Matinnejad, Shiva Nejati, and Lionel C. Briand. Automated testing of hybrid simulink/stateflow controllers: Industrial case studies. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, pages 938–943, New York, NY, USA, 2017. ACM.
64. Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Effective test suites for mixed discrete-continuous stateflow controllers. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pages 84–95. ACM, 2015.
65. Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Automated test suite generation for time-continuous simulink models. In Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pages 595–606, New York, NY, USA, 2016. ACM.
66. Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Test generation and test prioritization for simulink models with dynamic behavior. IEEE Trans. Software Eng., 45(9):919–944, 2019.
67. Anastasia Mavridou, Hamza Bourbouh, Dimitra Giannakopoulou, Thomas Pressburger, Mohammad Hejase, Pierre-Loic Garoche, and Johann Schumann. The ten lockheed martin cyber-physical challenges: formalized, analyzed, and explained. In 2020 IEEE 28th International Requirements Engineering Conference (RE), pages 300–310. IEEE, 2020.
68. Claudio Menghi, Shiva Nejati, Lionel C Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In International Conference on Software Engineering (ICSE), 2020.
69. Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C. Briand. Generating automated and online test oracles for simulink models with continuous and uncertain behaviors. In Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, pages 27–38, 2019.
70. Swarup Mohalik, Ambar A Gadkari, Anand Yeolekar, KC Shashidhar, and S Ramesh. Automatic test case generation from simulink/stateflow models using model checking. Software Testing, Verification and Reliability, 24(2):155–180, 2014.
71. Shiva Nejati, Khouloud Gaaloul, Claudio Menghi, Lionel C. Briand, Stephen Foster, and David Wolfe. Evaluating model testing and model checking for finding requirements violations in simulink models. In Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, pages 1015–1025, 2019.
72. Shiva Nejati, Lev Sorokin, Damir Safin, Federico Formica, Mohammad Mahdi Mahboob, and Claudio Menghi. Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial adas and simulink models. Information and Software Technology, page 107286, 2023.
73. Tanzeem Bin Noor and Hadi Hemmati. A similarity-based approach for test case prioritization using historical failure data. In 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pages 58–68. IEEE, 2015.
74. Raphaël Ollando, Seung Yeob Shin, Mario Minardi, and Nikolas Sidiropoulos. Test schedule generation for acceptance testing of mission-critical satellite systems. Empirical Software Engineering, 31(1):1–35, 2026.
75. Mike Papadakis, Christopher Henard, and Yves Le Traon. Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing. In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, pages 1–10. IEEE, 2014.
76. Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In Proceedings of the 37th International Conference on Software Engineering-Volume 1, pages 936–946. IEEE Press, 2015.

77. Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology (TOSEM), 6(2):173–210, 1997.
78. Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, pages 179–188. IEEE, 1999.
79. Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. IEEE Transactions on software engineering, 27(10):929–948, 2001.
80. Goiuria Sagardui, Joseba Agirre, Urtzi Markiegi, Aitor Arrieta, Carlos Fernando Nicolás, and Jose María Martín. Multiplex: A co-simulation architecture for elevators validation. In Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017 IEEE International Workshop of, pages 1–6. IEEE, 2017.
81. Goiuria Sagardui, Leire Etxeberria, Joseba A Agirre, Aitor Arrieta, Carlos Fernando Nicolas, and Jose Maria Martin. A configurable validation environment for refactored embedded software: An application to the vertical transport domain. In 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pages 16–19. IEEE, 2017.
82. Donghwan Shin, Shin Yoo, Mike Papadakis, and Doo-Hwan Bae. Empirical evaluation of mutation-based test case prioritization techniques. Software Testing, Verification and Reliability, 29(1-2):e1695, 2019.
83. Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel Briand, and Frank 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.
84. H. Shokry and M. Hinchey. Model-based verification of embedded software. Computer, 42(4):53 – 59, 2009.
85. Guangwei Shu, Reinhold Meisinger, and Gang Shen. Simulation of a maglev train with periodic guideway deflections. In 2008 Asia Simulation Conference-7th International Conference on System Simulation and Scientific Computing, pages 421–425. IEEE, 2008.
86. T. Strathmann and J. Oehlerking. Verifying properties of an electro-mechanical braking system. In In 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, pages 49–56, 2015.
87. Zhuo Su, Zehong Yu, Dongyan Wang, Wanli Chang, Bin Gu, and Yu Jiang. Test case generation for simulink models using model fuzzing and state solving. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, pages 117–128, 2024.
88. Enrico Viganò, Oscar Cornejo, Fabrizio Pastore, and Lionel C Briand. Data-driven mutation analysis for cyber-physical systems. IEEE Transactions on Software Engineering, 49(4):2182–2201, 2022.
89. Tanja E. J. Vos, Felix F. Lindlar, Benjamin Wilmes, Andreas Windisch, Arthur I. Baars, Peter M. Kruse, Hamilton Gross, and Joachim Wegener. Evolutionary functional black-box testing in an industrial setting. Software Quality Journal, 21(2):259–288, 2013.
90. Shuai Wang, Shaukat Ali, Tao Yue, Øyvind Bakkeli, and Marius 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, pages 182–191. IEEE, 2016.
91. Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius 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, pages 32–41, New York, NY, USA, 2014. ACM.
92. Di Yu, KL Lo, Xiaodong Wang, and Xiaobao Wang. Mrts traction power supply system simulation using matlab/simulink. In Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No. 02CH37367), volume 1, pages 308–312. IEEE, 2002.
93. Justyna Zander-Nowicka, Xuezheng Xiong, and Ina Schieferdecker. Systematic test data generation for embedded software. In Proceedings of the 2008 International Conference

on Software Engineering Research, volume vol.1, pages 164 – 70, Las Vegas, NV, USA, 2008.
94. Yuan Zhan and John A Clark. Search-based mutation testing for simulink models. In Proceedings of the 7th annual conference on Genetic and evolutionary computation, pages 1061–1068. ACM, 2005.
95. Lingming Zhang, Dan Hao, Lu Zhang, Gregg Rothermel, and Hong Mei. Bridging the gap between the total and additional test-case prioritization strategies. In Proceedings of the 2013 International Conference on Software Engineering, pages 192–201. IEEE Press, 2013.
96. Lingming Zhang, Ji Zhou, Dan Hao, Lu Zhang, and Hong Mei. Prioritizing junit test cases in absence of coverage information. In 2009 IEEE International Conference on Software Maintenance, pages 19–28. IEEE, 2009.