Single and Multi-objective Test Cases Prioritization for Self-driving Cars in Virtual Environments

Birchler, Christian ; Khatiri, Sajad ; Derakhshanfar, P.; Panichella, Sebastiano; Panichella, A.

DOI
10.1145/3533818

Publication date
2023

Document Version
Final published version

Published in
ACM Transactions on Software Engineering and Methodology

Citation (APA)
Birchler, C., Khatiri, S., Derakhshanfar, P., Panichella, S., & Panichella, A. (2023). Single and Multi-objective Test Cases Prioritization for Self-driving Cars in Virtual Environments. *ACM Transactions on Software Engineering and Methodology*, *32*(2), 1-30. Article 28. https://doi.org/10.1145/3533818

Single and Multi-objective Test Cases Prioritization for Self-driving Cars in Virtual Environments

CHRISTIAN BIRCHLER, Zurich University of Applied Science
SAJAD KHATIRI, Zurich University of Applied Science & Software Institute - USI, Lugano
POURIA DERAKHSHANFAR, Delft University of Technology
SEBASTIANO PANICHELLA, Zurich University of Applied Science
ANNIBALE PANICHELLA, Delft University of Technology

Testing with simulation environments helps to identify critical failing scenarios for self-driving cars (SDCs). Simulation-based tests are safer than in-field operational tests and allow detecting software defects before deployment. However, these tests are very expensive and are too many to be run frequently within limited time constraints.

In this article, we investigate test case prioritization techniques to increase the ability to detect SDC regression faults with virtual tests earlier. Our approach, called *SDC-Prioritizer*, prioritizes virtual tests for SDCs according to static features of the roads we designed to be used within the driving scenarios. These features can be collected without running the tests, which means that they do not require past execution results. We introduce two evolutionary approaches to prioritize the test cases using diversity metrics (black-box heuristics) computed on these static features. These two approaches, called *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*, use single-objective and multi-objective **genetic algorithms** (**GA**), respectively, to find trade-offs between executing the less expensive tests and the most diverse test cases earlier.

Our empirical study conducted in the SDC domain shows that *MO-SDC-Prioritizer* significantly (*P*-value $<= 0.1e-10$) improves the ability to detect safety-critical failures at the same level of execution time compared to baselines: random and greedy-based test case orderings. Besides, our study indicates that multi-objective meta-heuristics outperform single-objective approaches when prioritizing simulation-based tests for SDCs.

MO-SDC-Prioritizer prioritizes test cases with a large improvement in fault detection while its overhead (up to 0.45% of the test execution cost) is negligible.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; **Software testing and debugging;**

Additional Key Words and Phrases: Autonomous systems, software simulation, test case prioritization

We gratefully acknowledge the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS.

Authors' addresses: C. Birchler and S. Panichella, Zurich University of Applied Science, Switzerland; emails: {birc, panc}@zhaw.ch; S. Khatiri, Zurich University of Applied Science, Gertrudstrasse 15, 8401 Winterthur, Switzerland and Software Institute - USI Lugano, Via la Santa 1, 6962 Viganello, Switzerland; email: mazr@zhaw.ch; P. Derakhshanfar and A. Panichella, Delft University of Technology, 2628 CD Delft, Netherlands; emails: {p.derakhshanfar, a.panichella}@tudelft.nl.

1049-331X/2023/03-ART28
https://doi.org/10.1145/3533818

ACM Reference format:
Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. 2023. Single and Multi-objective Test Cases Prioritization for Self-driving Cars in Virtual Environments. *ACM Trans. Softw. Eng. Methodol.* 32, 2, Article 28 (March 2023), 30 pages.
https://doi.org/10.1145/3533818

1 INTRODUCTION

Self-driving cars (**SDCs**) are autonomous systems that collect, analyze, and leverage sensor data from the surrounding environment to control physical actuators at run-time [3, 13]. Testing automation for SDCs is vital to ensure their safety and reliability [49, 50], but it presents several limitations and drawbacks: (i) the limited ability to repeat tests under the same conditions due to ever-changing environmental factors [50]; (ii) the difficulty to test the systems in safety-critical scenarios (to avoid irreversible damages caused by dreadful outcomes) [43, 47, 80]; (iii) not being able to guarantee the system's reliability in its operational design domain due to a lack of testing under a wide range of execution conditions [49].

The usage of virtual simulation environments addresses several of the challenges above for SDCs testing practices [1, 14, 16, 30]. Hence, simulation environments are used in industry in multiple development stages of **Cyber-physical Systems** (**CPSs**) [76], including model (MiL), software (SiL), and hardware in the loop (HiL). As a consequence, multiple open-source and commercial simulation environments have been developed for SDCs, which can be *more effective and safer than traditional in-field testing methods* [4].

Adequate testing for SDCs requires writing (either manually or assisted by generation tools [2, 37]) a very large number of driving scenarios (test cases) to assess that the system behaves correctly in many possible critical and corner cases. The large running time of simulation-based tests and the large size of the test suites make regression testing particularly challenging for SDCs [35, 84]. In particular, regression testing requires running the test suite before new software releases to assess that the applied software changes do not impact the behavior of the unchanged parts [64, 86].

The ***goal*** of this article is to investigate and propose black-box **test case prioritization** (**TCP**) techniques for SDCs. TCP methods sort (prioritize) the test cases with the aim to run the fault-revealing tests as early as possible [86]. While various black-box heuristics have been proposed for traditional systems and CPSs, they cannot be applied to SDCs *as is*. Black-box approaches for "traditional" systems sort the tests based on their diversity, computed on the values of the input parameters [52] and the sequence of method calls [19]. However, SDC simulation scenarios (e.g., with road shape, weather conditions) do not consist of sequences of method calls as in traditional tests [2, 37]. Approaches targeting CPSs measure test distance based on signal [9], and fault-detection capability [12]. However, this data is unknown up-front without running the SDC tests.

The main *challenges* to address when designing black-box TCP methods for SDCs concern (i) the definition of features that can characterize SDC safety-critical scenarios in virtual tests; and (ii) design optimization algorithms that successfully prioritize the test cases based on the selected features. Therefore, to address these challenges, we formulated the following *research questions*:

— **RQ**$_1$: *To what extent is it possible to prioritize safety-critical tests in SDCs in virtual environments prior to their execution?*

We designed and computed 16 static features for driving scenarios in SDCs virtual tests, such as the length of the road, the number of left and right turns, and so on. These features are

extracted from the test scenarios prior to their execution, and for them, we investigated which ones are non-collinear (see Section 4.2.1) according to **Principal Component Analysis** (**PCA**). Hence, we introduce *SDC-Prioritizer*, a TCP approach based on GA that prioritizes test cases of SDCs by leveraging these features. This article introduces two variants of the *SDC-Prioritizer*, namely *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*. The former variant utilizes a single-objective GA for test prioritization. The latter variant leverages a well-known and commonly used multi-objective GA, called NSGA-II [27], to achieve this goal. Any search-based technique needs to balance between *exploitation* and *exploration* [25]. Exploitation refers to the ability of the search process to visit regions of the search space within the neighborhood of previously generated solutions (here, test execution orders). Exploration refers to the ability to generate entirely new solutions that are different from the current solutions. Poor exploration ability of the search process leads to low diversity between the generated solution, and thereby the search process may easily be trapped in local optima [25]. The rationale behind introducing *MO-SDC-Prioritizer* beside the *SO-SDC-Prioritizer* is to avoid the lack of exploration ability in *SDC-Prioritizer*. The NSGA-II algorithm, utilized in *MO-SDC-Prioritizer*, provides well-distributed Pareto fronts and thereby brings sufficient diversity into the generated solutions.

— **RQ**$_2$: *What is the cost-effectiveness of SDC-Prioritizer compared to baseline approaches?*

To answer RQ$_2$, we conducted an empirical study with three different datasets and composed of test scenarios that target the *lane-keeping* features of SDCs. In this context, fault-revealing tests are virtual test scenarios in which a self-driving car would not respect the *lane tracking safety requirement* [38]. We targeted `BeamNG` by BeamNG.research [14] (detailed in Section 2) as a reference simulation environment, which has been recently used in the **Search-Based Software Testing** (**SBST**) tool competition[1] [66]. The test scenarios for this environment have been produced with by SDC-Scissor [15] (which integrates also AsFault [37]), an open-source project that generates test cases to assess SDCs behavior (detailed in Section 2).

By comparing *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* with two baselines—namely random search, and the greedy algorithm—on these three benchmarks, we analyze the performance of our techniques in terms of its ability to detect more faults while incurring a lower test execution cost.

Finally, we assess whether *SDC-Prioritizer* techniques can be used in practical settings, i.e., it does not add a too large computational overhead to the regression testing process:

— **RQ**$_3$: *What is the overhead introduced by SDC-Prioritizer?*

The results of our empirical study show that *MO-SDC-Prioritizer* is the best performing technique in terms of identifying more safety-critical scenarios in less time. On average, this technique reduces the time required to identify more safety-critical scenarios by 6%, 25.5%, and 3% compared to *SO-SDC-Prioritizer*, random test case orders ("default" baselines for search-based approaches [76, 86]), and the greedy algorithm for TCP, respectively. It also shows that *MO-SDC-Prioritizer* leads to an increase of detected faults (about 63 more) in the first 20% of the test execution time compared to the greedy test prioritization (i.e., second best technique according to our assessments). Furthermore, *SDC-Prioritizer* approaches do not introduce significant computational overhead in the SDCs simulation process, which is of critical importance to SDC development in industrial settings.

The contributions of this article are summarized as follows:

(1) We designed static features that can be used to characterize safe and unsafe test scenarios prior to their execution in the SDC domain.

[1] https://sbst21.github.io/tools/.

(2) We introduce *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*, two black-box TCP approaches that leverage single and multi-objective GA, respectively, to achieve cost-effective regression testing with SDC tests in virtual environments.
(3) A comprehensive and publicly available replication package available on Zenodo [21], including all data used to run the experiments as well as the prototype of *SDC-Prioritizer*, to help other researchers reproduce the study results.

Article Structure. In Section 2, we summarize the related work, while in Section 3, we outline the approach we have designed and implemented to answer our research questions. In Section 4, we present our methodology and empirical studies performed to answer our research questions. In Section 5, we report the study results, while in Section 6, we detail the threats to validity of our study. Finally, Section 7 concludes our study, outlining directions for future work.

2 BACKGROUND AND RELATED WORK

This section discusses the literature concerning (i) test prioritization approaches in traditional systems; and (ii) studies closely related to test prioritization practices in the context of CPSs . Finally, the section describes the background on the SDC virtual environment adopted in this study.

2.1 Test Prioritization

Approaches aiming at reducing the cost of regression testing can be classified into three main categories [87]: *test suite minimization* [70], *test case selection* [20], and *TCP* [71]. Test case minimization approaches tackle the regression problem by removing test cases that are redundant according to selected testing criteria (e.g., branch coverage). Test case selection aims to select a subset of the test suite according to the software changes, coverage criteria, and execution cost. TCP, which is the main focus of our article, sorts the test cases to maximize some desired properties (e.g., code coverage, requirement coverage) that lead to detecting regression faults as early as possible. A complete overview of regression testing approaches can be found in the survey by Yoo and Harman [87].

2.1.1 Prioritization Heuristics. Approaches proposed in the literature to guide the prioritization of the test cases can be grouped into white-box and black-box heuristics [87]. White-box TCP uses past coverage data (e.g., branch, line, and function coverage) and iteratively selects the test cases that contribute to maximizing the chosen code coverage metrics.

Black-box prioritization techniques rely on diversity metrics and prioritize the most diverse test cases within the test suites (e.g., [5, 34, 52]). Widely-used diversity metrics include *input and output set diameter* [34], or the *Levenstein distance* computed on the input data [52] and method sequence [19]. Further heuristics include topic modeling [81], or models of the system [44]. Miranda et al. [62] proposed fast methods to speed up the pair-wise distance computation, namely *shingling* and *locality-sensitive hashing*. Recently, Henard et al. [45] empirically compared many white-box and black-box prioritization techniques. Their results showed a large overlap between the regression faults that can be detected by the two categories of techniques and that black-box techniques are highly recommended when the source code is not available [45], e.g., in the case of third-party components. CPSs (including SDCs) are typical instances of systems with many third-party components [76].

Prioritization heuristics for CPSs differ from those used for traditional software [8]. We elaborate more in detail on the related work on TCP for CPSs in Section 2.2.

2.1.2 Optimization Algorithms. Given a set of heuristics (either white-box or black-box), optimization algorithms are applied to find a test case order that optimizes the chosen heuristics. As shown by Yoo et al. [87] TCP (and regression testing in general) is inherently a multi-objective problem because test quality (e.g., code coverage, input diversity) and execution resources are conflicting in nature. The challenge is choosing balanced trade-offs that favor lower execution cost over higher code coverage or test diversity depending on the time constraints and resource availability (e.g., in continuous delivery or integration servers).

Cost-cognizant greedy algorithms are well-known deterministic algorithms introduced for the set-cover problem and adapted to regression testing [20]. The greedy algorithm first selects the test case with the most code coverage (white-box) or the most diverse one (black-box). Then, the algorithm iteratively selects the test case that increases coverage the most or that is the most diverse w.r.t. previously selected test cases [87].

Meta-heuristics have been shown to be very competitive, sometimes outperforming greedy algorithms [54, 57, 64, 81]. Marchetto et al. [57] used multi-objective GA to optimize trade-offs between *cumulative code coverage*, *cumulative requirement coverage*, and *execution cost*. Besides, GA have been widely used to optimize test case diversity [81] for black-box TCP.

This article uses greedy algorithm, single-objective GA, and multi-objective GA to prioritize simulation-based test cases for SDCs. This is because each type of algorithm has been shown to outperform its counterparts in different domains and programs [12, 54].

2.2 Regression Testing for CPSs

Regression testing is particularly critical for CPSs, which are characterized by interactions with simulation and hardware environments. Testing with simulation environments is a *de facto* standard for CPSs, and it is typically performed at three different levels [59]: MiL, SiL, and HiL. During *model in the loop* (MiL), the controller (cars) and the environments (e.g., roads) are both represented by models, and testing aims to assess the correctness of the control algorithms. During *software in the loop* (SiL), the controller model is replaced by its actual code (software), and its testing phase aims to assess the correctness of the software and its conformance to the model used in the MiL. Finally, during hardware in the loop (HiL), the controller is fully deployed while the simulation is performed with real-time computers that simulate the physical signals. The testing phase for the HiL aims to assess the integration of hardware and software in more realistic environments [59].

Regression testing for CPSs is more challenging as the execution time of the test cases is much longer due to the simulation [12]. Hence, researchers have proposed different regression testing techniques that are specific to CPSs. Shin et al. [77] proposed a bi-objective approach based on GA to prioritize acceptance tests for a satellite system. Their approach prioritizes the test cases according to the hardware damage risks it can expose (first objective) and maximizes the number of test cases that can be executed within a given time budget (second objective). Arrieta et al. [12] used both greedy algorithms and meta-heuristics to prioritize test cases for CPS product lines and with different test levels. In further studies, Arrieta et al. [10] focused on multiple objectives to optimize for both test case generation and TCP for CPSs. The objectives include requirement coverage, test case similarity, and test execution times. While test similarity for non-CPS systems is computed based on the lexicographic similarity for the method calls and test input, Arrieta et al. measured the similarity between the test cases based on the signal values for all the states in the simulation-based test case. Test case similarity computed at the signal-level has also been investigated in the context of test case selection for CPS [9, 11].

Our paper differs from the papers above w.r.t. the application domain and the optimization objectives. In particular, we focus on prioritized simulation-based test cases to assess the

lane-keeping features of SDCs. Instead, prior work focused on different domains, such as satellite [76], electric windows [9], industrial tanks [10, 12], and cruise controller [10]. In our context, test cases consist of driving test scenarios with virtual roads (e.g., see Figure 1) and aim at assessing whether the simulated cars violate the lane-keeping requirements.

Another important difference is related to the objectives (or heuristics) to optimize for regression testing. Prior works for CPS prioritize the test cases based on fault-detection capabilities [12], and diversity measured for simulation signals [9–11]. However, the fault-detection capability of the test cases is unknown a prior (i.e., without running the tests). Signal analysis requires knowing the states of the simulated objects in each simulated time step, which is also unknown before the actual simulation. Furthermore, a driving scenario (in our context) is not characterized by signals but only by the initial state of the car and the actual characteristics (e.g., shape) of the roads. Hence, we define features and diversity metrics that consider only the (static) characteristics of the roads that are used for the simulation. Unlike fault-detection capability and signals, our features can be derived from the driving scenario before the actual test execution.

2.3 Background on SDCs Simulation

2.3.1 Main Simulation Approaches. Simulation environments have been developed to support developers in various stages of design and validation. In the SDC domain, developers rely mainly on basic simulation models [41, 78], rigid-body [55, 88], and soft-body simulations [36, 69].

Basic simulation models, such as MATLAB/Simulink models [41, 78], implement fundamental signals but target mostly non-real-time executions and generally lack photo-realism. Consequently, while they are utilized for model-in-the-loop simulations and Hardware/Software co-design, they are rarely used for integration and system-level software testing.

Rigid-body simulations approximate the physics of static bodies (or entities), i.e., by modeling them as *undeformable* bodies. Basic simulation bodies consist of three-dimensional objects such as cylinders, boxes, and convex meshes [2].

Soft-body simulations can simulate deformable and breakable objects and fluids; hence, they can be used to model a wide range of simulation scenarios. Specifically, the **finite element method (FEM)** is the main approach for solid body simulations, while the **finite volume method** (**FVM**) and **finite difference method** (**FDM**) are the main strategies for simulating fluids [60].

Rigid-body vs. Soft-body simulations Both rigid- and soft-body simulations can be effectively combined with powerful rendering engines to implement photo-realistic simulations [14, 16, 30, 83]. However, soft-body simulations can simulate a wider variety of physical phenomena compared to rigid-body simulations. Soft-body simulations are a better fit for implementing safety-critical scenarios (e.g., car incidents [36]), in which a high simulation accuracy is of key importance. As follows, we describe the soft-body environment we used in our research investigation, i.e., BeamNG [14].

2.3.2 BeamNG and AsFault. Creating adequate test scenario suites for SDCs is a hard and laborious task. To tackle this issue, Gambi et al. [38] developed and proposed a tool called AsFault [37] to generate driving scenarios for testing SDCs automatically. From a high-level point of view, AsFault combines procedural content generation and search-based testing in order to automatically create virtual scenarios for testing the lane-keeping behavior in SDC software. Specifically, AsFault leverages a GA to iteratively refine virtual road networks towards those which cause the ***ego-car*** (the simulated car controlled by the SDC software under test) to move away from the center of the lane. The virtual roads are generated inside a driving simulator called BeamNG [14], which can generate photo-realistic, but synthetic, images of roads. Given such characteristics, BeamNG [14] has also been used as the main simulation platform in the 2021 edition of the SBST tool

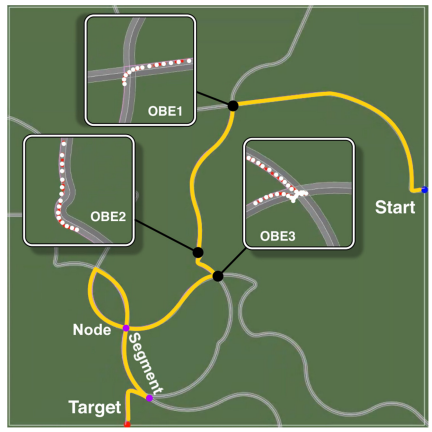


Fig. 1. Sample driving scenarios generated by SDC-Scissor [15] (which integrates also AsFault [38]).

competition [66]. Lane-keeping systems (described in the next sections) continuously track the striped and solid lane markings of the road ahead using advanced image processing, deep learning, or machine learning techniques and triggers needed control mechanisms (e.g., steering, braking, and speeding) to keep the car at the proper location regarding the road structure.

To evaluate the criticality of generated test cases, the road networks are instantiated in a driving simulation, during which the ego-car is instructed to reach a target location following a navigation path selected by AsFault. During the simulation, AsFault traces the position of the ego-car at regular intervals such that it can identify **Out of Bound Episodes** (**OBEs**), i.e., lane departures. An **out-of-bound incident** is defined as *"the case when the car went more than two meters out of the lane center"*. In our experiments, we use this information to label test scenarios as ***safe*** (causing no OBEs) or ***unsafe*** (causing at least one OBE).

Figure 1 illustrates a sample test scenario generated and executed by AsFault [38]. It includes start and target points for the ego-car on the map, the whole road network, the selected driving path (colored in yellow), and the detected OBE locations during the execution of the scenario by the ego-car. Hence, each generated test scenario by AsFault consists of a JSON file generated by AsFault, which reports multiple *nodes* and their connections, and form a *road network*, with the start and destination point and the driving path of the ego-car [38].

2.3.3 SDC Software Use-cases. AsFault supports two AI engines as test subjects while generating test cases, which we use to generate our test suites. These two test subjects allow to drive the ego-car by computing an ideal driving trajectory, which places the ego-car in the center of the lane while driving within a configurable speed limit:

— **BeamNG.AI.**[2] BeamNG.research ships with a driving AI that we refer to as *BeamNG.AI*. BeamNG.AI can be parameterized with an *"aggression"* factor which controls the amount of *risk* the driver takes in order to reach the destination faster. BeamNg.research developers say that low aggression factors (e.g., 0.7) result in a smooth driving whereas high aggression factors (e.g., 1.2 and above) lead the car to edgy driving and might cut corners [38].
— **Driver.AI.**[3] Driver.AI is a trajectory planner shipped with AsFault [38]. AsFault leverages an extension of Driver.AI, which monitors the quality of its predictions at run-time. Hence,

[2]https://wiki.beamng.com/Enabling_AI_Controlled_Vehicles#AI_Modes.
[3]https://github.com/alessiogambi/AsFault/blob/asfault-deap/src/asfault/drivers.py.

differently from BeamNG.AI, Driver.AI analyzes the road geometry and plans the trajectory of the car by computing, for each turn, the maximum safe driving speed (v) using the reference formula for centripetal force on flat roads with static friction (μ) [22]:

$$v = \sqrt{\mu \times r \times g}, \tag{1}$$

where r is the turn radius and g is the free-fall acceleration. It is important to note that, we use BeamNG since:

- BeamNG can be easily used by developers via Python APIs for creating scenarios
- BeamNG can access to sensor data, Camera, Lidar, IMU
- the BeamNG AI engine can simulate:
  - the aggressive driving style
  - Balanced driving style
  - Calm driving style

3 APPROACH

This section describes the investigated test scenario features and prioritization strategies introduced by *SDC-Prioritizer* and a greedy algorithm in the SDC domain.

3.1 SDC Road Features

In the context of SDC, we target the definition of features (or metrics) that characterize SDC tests in virtual environments according to the following requirements: the features (1) can be extracted before the actual execution of the virtual tests; and (2) these features can characterize (or identify) safe and unsafe scenarios without executing them. In the following, we describe how the SDC features have been designed and measured considering the BeamNG as the targeted SDC virtual environment.

In the context of BeamNG, it is possible to compute static features concerning the actual road characteristics of SDC virtual tests. Specifically, as illustrated in Figure 1, each virtual test scenario generated by AsFault (virtual roads), consists of multiple *nodes* and their connections (i.e., *road segments*) forming a so-called *road network*, along with the start and destination points and the driving path of the ego-car. This allows us to compute what we call *Road Features*, i.e., features or characteristics of the road that will be used during the simulation within the BeamNG virtual environment.

From the road data reported by AsFault, we extract various features for each test scenario (as described in the following paragraph), and we investigate ways to leverage these features to determine the criticality of the test scenarios (as described in Section 4).

Road Features extraction. To extract the features corresponding to each of the generated test scenarios, we leverage the JSON file generated as output by AsFault. These files, as explained before, consist of multiple *nodes* and their connections, and form a *road network*, with the start and destination point and the driving path of the ego-car. Hence, we extract two sets of road features, *the general road characteristics*, and *the road segment statistics*. The general road characteristics are attributes that refer to the road as a whole, e.g., direct distance and road length between the start and destination points, the total number of turns to left or right. For each road segment (see Figure 1), we can extract individual metrics such as road angle and pivot radius. For the segment statistics features, we apply aggregation functions (e.g., minimum, maximum, and average) on these individual segment metrics for all road segments in the scenario path. Table 1 reports the features extracted from the original fields in AsFault JSON (i.e., F1–16 features), specifying their

Table 1. Road Characteristics Features

ID	Feature	Description	Type	Range
F1	Direct Distance	Euclidean distance between start and finish	float	[0-490]
F2	Road Distance	Total length of the road	float	[56-3,318]
F3	Num. Left Turns	Number of left turns on the test track	int	[0-18]
F4	Num. Right Turns	Number of right turns on the test track	int	[0-17]
F5	Num. Straight	Number of straight segments on the test track	int	[0-11]
F6	Total Angle	Total angle turned in road segments on the test track	int	[105-6,420]
F7	Median Angle	Median of angle turned in road segment on the test track	float	[30-330]
F8	Std Angle	Standard deviation of angled turned in road segment on the test track	int	[0-150]
F9	Max Angle	The maximum angle turned in road segment on the test track	int	[60-345]
F10	Min Angle	The minimum angle turned in road segment on the test track	int	[15-285]
F11	Mean Angle	The average angle turned in road segment turned on the test track	float	[5-47]
F12	Median Pivot Off	Median of radius of road segment on the test track	float	[7-47]
F13	Std Pivot Off	Standard deviation of radius of turned in road segment on the test track	float	[0-23]
F14	Max Pivot Off	The maximum radius of road segment on the test track	int	[7-47]
F15	Min Pivot Off	The minimum radius of road segment on the test track	int	[2-47]
F16	Mean Pivot Off	The average radius of road segment turned on the test track	float	[7-47]

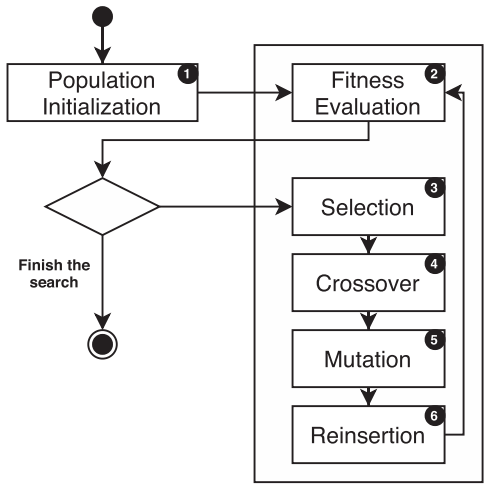


Fig. 2. An overview of GA.

description, type, and expected range of values for each feature. In the next sections, we described how the designed features are used as inputs to TCP strategies.

3.2 Single-Objective Genetic Algorithm

Several prior studies have utilized evolutionary algorithms (particularly GA) for test prioritization to reduce regression testing costs in different types of systems [54]. A typical GA starts with generating a population of randomly generated individuals (box 1 in Figure 2). Each individual can be described as a sequence of parameters, called the chromosome, which encodes a potential solution to a given problem. This encoding can be performed in many forms (such as string, binary, etc.). After generating the first population, this algorithm determines the "fitness" of the individuals according to a fitness function (box 2 in Figure 2). Then, in the *Selection* phase (box 3 in Figure 2), a subset of individuals are selected according to their fitness values to be used as parents for mating.

Next, two genetic operators are applied to generate the next population using the selected parents: *Crossover* and *Mutation*. The former (box 4 in Figure 2) operator combines two parents to produce new individuals (called offspring). The latter (box 5 in Figure 2) operator alters one or more elements in the offspring to explore nearby solutions in the search space. Finally, the newly generated individuals are saved in a new population (box 6 in Figure 2). The process of generating a new population of individuals from the previous one will continue until either the search objective is fulfilled or when the algorithm reaches the maximal number of generations (iterations).

This section introduces a single-objective GA called *SO-SDC-Prioritizer* that prioritizes the most diverse tests (according to their corresponding feature vectors) per unit of cost in SDCs. The following subsections describe detailed information regarding the encoding, operators, and fitness function used in the *SDC-Prioritizer*.

3.2.1 Encoding. Since the solution for the test prioritization is an ordered sequence of tests, *SDC-Prioritizer* uses a *permutation encoding*. Assuming that, in our problem, we seek to order the execution of N tests, our approach encodes each chromosome as an N-sized array containing integers that denote the position of a test in the order. For example, let $\tau = \langle t_1, t_2, t_3 \rangle$ be a chromosome for a test suite with three test cases; then, test case t_1 will be executed first, followed by t_2 and t_3 during regression testing.

3.2.2 Partially-Mapped Crossover (PMX). In the crossover, an offspring o is formed from two selected parents p_1 and p_2 , with the size of N, as follows: (i) select a random position c in p_1 as the cut point; (ii) the first c elements of p_1 are selected as the first c elements of o; (iii) extract the $N - c$ elements in p_2 that are not in o yet and put them as the last $N - c$ elements of o.

3.2.3 Mutation Operators. A chromosome p can be mutated one or more times according to the given mutation probability. In each round of mutation, one of the three following mutation operators [75] is selected randomly with an equal chance of 0.33% to perform the mutation:

- **SWAP mutation:** This mutation operator randomly selects two positions in a chromosome p and swaps the index of two genes (test case indexes in the order) to generate a new offspring.
- **INVERT mutation:** This mutation operator randomly selects a segment (with a random size) of the given chromosome p. Then, it reverses the selected segment end to end and reattaches it to generate a new offspring.
- **INSERT mutation:** This mutation randomly selects a gene in the chromosome p and moves it to another index in the solution to generate a new offspring.

We consider the three operators above since prior studies [75] showed that using multiple mutation operators for permutation-based optimization problems increases the likelihood of escaping from solutions that are locally optimal under one mutation operator. This procedure used for the mutation is the same in both of the *SDC-Prioritizer* variants introduced in this article.

3.2.4 Fitness Function in SO-SDC-Prioritizer. Our goal is to promote (1) the diversity of the selected test cases and (2) minimize the execution cost. Hence, the ultimate goal is to run the most diverse test within a given time constraint. Hence, we define a fitness function that incorporates both test diversity and execution cost. This is in line with current practice in the literature, which combines surrogate metrics for test effectiveness (e.g., code coverage) with execution cost [53, 64, 85]. More specifically, let $\tau = \langle t_1, \ldots, t_n \rangle$ be a given test case ordering, its "fitness" (quality) is

measured using the following equation:

$$\begin{aligned} \max f(\tau) &= \sum_{i=2}^{n} \frac{diversity(t_i)}{cost(t_i) \times i} \\ &= \sum_{i=2}^{n} \frac{distance(t_i, t_{i-1})}{cost(t_i) \times i}, \end{aligned} \tag{2}$$

where n is the number of test cases; t_i is the ith test in the ordering τ; $cost(t_i)$ is the execution cost (simulation time) of the test case t_i; and $distance(t_i, t_{i-1})$ measures the Euclidean distance between the test cases t_i and t_{i-1}. In other words, each test case in position i positively contributes to the overall fitness (to be maximized) based on its distance to the prior test t_{i-1} in the order τ. Since we want to have as many diverse tests as possible in the same amount of time, the diversity score of each test t_i is divided by its execution cost (to be minimized) and its position i in τ. The factor i in the denominator of Equation (2) promotes solutions where test cases with the best diversity-cost ratio are prioritized early, i.e., they appear early within the order τ.

The distance between two tests t_i and t_j is measured using the Euclidean distance and computed on the feature vectors described in Table 1. It is important to highlight that the different features have different ranges and scales, as reported in Table 1. Hence, the distance values computed using the Euclidean distance might be biased toward the features with larger ranges. To remove this potential bias, we normalized the features using *z-score* normalization, which is a well-known method to address outliers and to re-scale a set of features with different ranges and scales [39]. The *z-score* normalization scale the features using the formula $\frac{x-\mu}{\sigma}$, where x is the feature to re-scale, μ is its arithmetic mean, and σ is the corresponding standard deviation [39].

The execution cost of each test case t_i is estimated based on the past execution cost gathered from previous test runs, as recommended in the literature [32, 86]. This estimation is accurate for SDC since the cost of running simulation-based tests is proportional to the length of the road and the cost of rendering the simulation, which are fixed simulation elements.

3.2.5 Selection in SO-SDC-Prioritizer. The fitness function defined in Section 3.2.4 allows GAs to determine the fittest individual (permutations in our case) that should have higher chances to be selected for mating. The selection is made using the *roulette wheel selection* [40], which assigns a selection probability to each of the individuals according to their fitness values (calculated by a fitness function). Assuming that our problem is a maximization problem, the selection probability of an individual p_i is calculated as follows:

$$P(p_i) = \frac{f(p_i)}{\sum_{j=1}^{N} f(p_j)}, \tag{3}$$

where N is the number of individuals in the population and f_i is the fitness value of p_i.

After allocating selection probability to individuals, the algorithm randomly selects some individuals according to their selection chance. Each individual with a lower fitness value has a lower allocated selection probability and thereby has a lower chance of transferring its genetic material to the next generation.

3.3 Multi-objective Genetic Algorithm

This article also proposes *MO-SDC-Prioritizer*, a multi-objective variant of *SDC-Prioritizer* that considers the execution cost and test case diversity as two different objectives to optimize simultaneously. Assume that $\tau = \langle t_1, \ldots, t_n \rangle$ is a solution (i.e., test execution order) generated by the

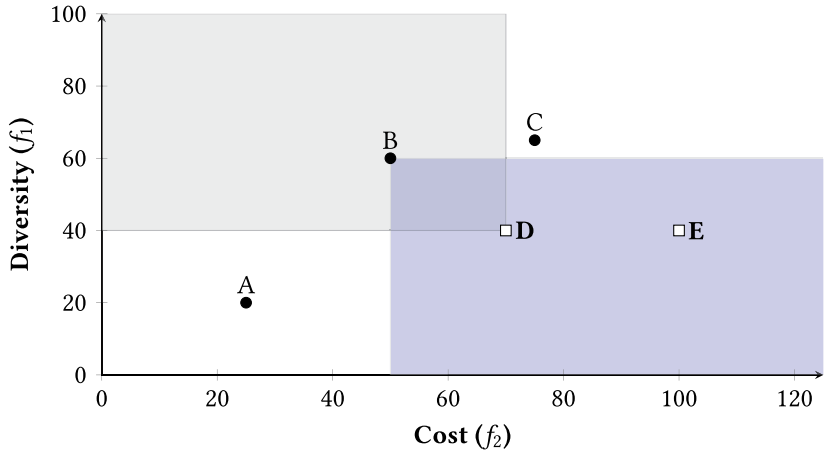


Fig. 3. Graphical representation of Pareto dominance for our two objectives, namely (1) test diversity (to maximize) and test cost (to minimize). In the example, points A, B, and C do not dominate one another, while point B dominates both D end E.

search process. The first goal to optimize is computed using the following equation:

$$\max f_1(\tau) = \sum_{i=2}^{n} \frac{distance(t_i, t_{i-1})}{i}, \tag{4}$$

where $distance(t_i, t_{i-1})$ denotes the distance between a test t_i and its predecessor $t(i-1)$ in the ordering. The contribution of each test case t_i to the cumulative diversity is divided by its position i in the ordering τ. In other words, this objective promotes solutions where the most diverse test cases are executed earlier.

The second objective in *MO-SDC-Prioritizer* measures how steadily the cumulative cost increases when executing the tests with a given order τ:

$$\min f_2(\tau) = \sum_{i=1}^{n} \frac{cost(t_i)}{i}, \tag{5}$$

where $cost(t_i)$ denotes the cost of executing the test case t_i in τ. The contribution of each test case t_i to the cumulative cost is divided by its position i in the ordering τ, with the goal of promoting solutions where the least expensive test cases are executed earlier. Notice that this objective should be minimized.

Different from *SO-SDC-Prioritizer*, finding optimal solutions for problems with multiple criteria requires trade-off analysis. Given the conflicting nature of our two objective,[4] it is not possible to obtain one single solution that optimizes both objectives at the same time [24]. Hence, we are interested in finding the set of solutions that are optimal compromises between the two objectives. For multi-objective problems, the concept of optimality is based on concepts of *Pareto dominance*, as explained in Figure 3, and Pareto optimality [24]. In particular, a solution τ_A *dominates* another solution τ_B ($\tau_A <_p \tau_B$) if and only if at the same level of diversity, τ_A has a lower cost than τ_B. Alternatively, τ_A *dominates* τ_B if and only if, at the same level of cost, τ_A has a larger diversity than τ_B. Among all possible solutions, we are interested in finding those that are not dominated by

[4]Diverse tests are not necessarily the least expensive to run.

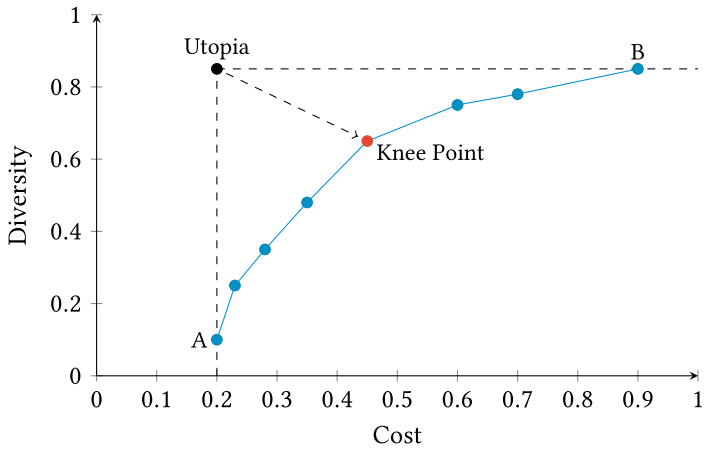


Fig. 4. Graphical representation of a Pareto front (in blue), the utopia (black point), and the knee point (red point).

any other possible solution (*Pareto optimality*). Pareto optimal solutions form the so-called *Pareto optimal set* while the corresponding objective values form the *Pareto front*.

Figure 4 provides a graphical example of Pareto optimality and non-dominance. All solutions in the grey rectangle (including B) dominate D since they achieve both lower cost and higher diversity. Instead, all solutions in the blue rectangle (including D and E) are dominated by B, since B achieves higher diversity with lower execution cost. Finally, A, B, and C do not dominate one another while D and E are dominated solutions.

3.3.1 NSGA-II. To find Pareto optimal solutions, *MO-SDC-Prioritizer* uses NSGA-II [27]. This GA provides well-distributed Pareto fronts and performs best when dealing with two or three search objectives [27]. NSGA-II shares the main loop of the GA depicted in Figure 2. Thus, it shares the same encoding schema as well as mutation and crossover operators discussed in Section 3.2. However, it differs on how parents are selected for reproduction and how the new population is formed for the next generation. Parents are selected using the *binary tournament selection*, which compares pairs of solutions in tournaments and selects the "fittest" solution from each pair for reproduction. Finally, the population for the next generation is obtained by selecting the "fittest" solutions among parent and offspring solutions (elitism).

In NGSA-II, the "fitness" of the solutions is determined using the *fast non-dominated sorting* algorithm and the concept of *crowding distance* [26]. The former ranks the solutions according to their dominance relations. All non-dominated solutions within a given population are inserted in the first front F_1 (rank $r = 1$); the subsequent front F_2 (rank $r = 2$) contains all solutions that are dominated only by the solutions in F_1; and so on. Hence, solutions in the fronts with lower rank are "fitter" according to the Pareto optimality.

Instead, the crowding distance aims at promoting more diverse (isolated) solutions within each dominance rank. The crowding distance for a given solution is computed as the sum of the distances between such an individual and all the other individuals with the same rank. This heuristics is put in place to avoid selecting individuals that are too similar to each other.

3.3.2 Choosing a Pareto Optimal Solution. As explained in Section 3.3.1, NSGA-II returns a set of non-dominated solutions at the end of the search process. Hence, the next step is to decide which Pareto optimal solution (best trade-off) among the many different alternatives. The necessity of this decision-making approach is also experienced in other optimization methods for various

engineering problems [58]. Researchers have suggested considering various points of interest in the Pareto front, such as the *knee points* [17], *mid points* [63], or the *extreme* of the Pareto front [65].

One of the common techniques to select solutions from the Pareto front is to identify knee points [17, 61], which are the solutions that minimize the distance to a point in the vector of the objective function, called *Utopia Point* [58]. The utopia point is a (usually unreachable) point with the most-optimum observed value for each objective function. Assume that *MO-SDC-Prioritizer* returns a set of solutions $S = S_1, S_2, ..., S_i$ as the final answer. These solutions are non-dominated according to two search objective functions diversity ($f_1(\tau)$ in Equation (4)) and test execution cost ($f_2(\tau)$ in Equation (5)). In this case, the Utopia Point U is the following point in the two-dimensional objective functions vector:

$$U = (maximum(\{f_1(s)|s \in S\}), maximum(\{f_2(s)|s \in S\}), \tag{6}$$

Since the utopia point usually does not exist in the returned solutions, we select the *closest* non-dominated solution to this point as the trade-off to select for regression testing.

One common way to measure the distance between two points is using the *Euclidean distance* $N(x)$, which is defined as:

$$N(x) = \sqrt{\sum_{i=1}^{k}(f_i(x) - U_i)^2}, \tag{7}$$

where f_i is the value of the Pareto optimal solution x for each objective. Here, *MO-SDC-Prioritizer* has f_1 and f_2, as explained in Section 3.3. U_i is also the value of the utopia point for the ith objective fitness function.

It is notable that if the fitness functions have different units, the Euclidean norm becomes insufficient to represent the *closeness* [58]. This is the case in *MO-SDC-Prioritizer* as the execution cost and the test diversity have different units. To tackle this issue, we need to normalize the values to make them dimensionless. The most robust technique to perform this normalization is [51, 58, 68]:

$$norm(f_i(x)) = \frac{f_i(x) - U_i}{max(f_i) - U_i}, \tag{8}$$

where $f_i(x)$ is the fitness actual value of solution x according to search objective fitness function f_i, and $max(f_i)$ is the maximum fitness value of generated solutions for f_i.

3.4 Black-box Greedy Algorithm

Greedy algorithms are well-known deterministic algorithms that iteratively build a solution (tests ordering) based on greedy steps. Greedy algorithms have been widely used in regression testing for both white- and black-box TCP [81, 86]. Hence, we adapt the greedy algorithm to our context and use the set of features we have designed for SDCs (see Section 3.1).

The greedy algorithm first computes the pairwise distance among all test cases in the given test suite. Similarly to GAs, the distance between two test cases t_i and t_j is computed using the Euclidean distance between the corresponding feature vectors. These features are normalized up-front using the *z-score* normalization as done for GA as well. Then, the greedy algorithm computes the diversity per unit cost of each test t_i using the following equation:

$$score(t_i) = \frac{distance(t_i, \tau)}{cost(t_i))}, \tag{9}$$

where $distance(t_i, \tau)$ measures the distance between t_i and the tests $t_j \in \tau$ selected in the previous iterations of the algorithm. In this equation, a higher score for a test means that it has the highest dissimilarity to previously selected tests with the lowest execution cost.

The greedy algorithm initializes the test order τ by selecting the test with the largest ratio between (1) its average distance to all other tests in the suite and (2) its execution cost. Then, the algorithm iteratively finds the test case (among the non-selected ones) with the largest average (mean) score to the (already selected) test cases in τ. This selection step corresponds to the *greedy* heuristic. Suppose multiple tests have the same average score to τ. In that case, the tie is broken by randomly choosing one of the equally distant test cases. This process is repeated until all test cases are prioritized.

4 STUDY DESIGN

Study design overview. Our empirical study is steered by the following research questions:

- **RQ**$_1$: *To what extent is it possible to prioritize safety-critical tests in SDCs in virtual environments prior to their execution?*
- **RQ**$_2$: *What is the cost-effectiveness of SDC-Prioritizer compared to baseline approaches?*
- **RQ**$_3$: *What is the overhead introduced by SDC-Prioritizer?*

In Section 3.1, we have introduced multiple static features to virtual driving scenarios (see Table 1), some of which might be collinear or not useful for prioritizing test cases in a cost-effective way. Hence, our first research question (**RQ**$_1$) aims to determine which features to consider, by leveraging statistical methods based on collinearity analysis [29, 89]. Our second research question (**RQ**$_2$) aims to assess the extent to which test case orders produced by *SDC-Prioritizer* techniques (*SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*) can detect more faults (effectiveness) and with lower execution cost (efficiency) with respect to a naive random search. Specifically, as elaborated in detail later, a random search is a critical baseline for search-based solutions since it is a "sanity-check" to assess whether more "sophisticated" techniques are needed for a given domain [76]. In RQ$_2$, we compare the internal search algorithms discussed in Section 3, namely the greedy algorithm, single-objective and multi-objective GA. With our last research question (**RQ**$_3$), we want to measure the overhead required to prioritize SDC test cases in virtual environments with *SDC-Prioritizer* techniques. This is an important aspect to investigate since a critical constraint in regression testing is that the cost of prioritizing test cases should be smaller than the time needed to run the test suite [86]. Therefore, fast approaches are fundamental from a practical point of view to enable rapid and continuous test iterations during SDC development [62].

4.1 Benchmark Datasets

The benchmark used in our study consists of three experiments performed on corresponding datasets. For each experiment, virtual test scenarios are generated and labeled as safe or unsafe by SDC-Scissor [15] (which integrates also AsFault). As described in Table 2, the first experiment leverages a dataset (referred to as *BeamNG.AI.AF1*) that includes 1,178 virtual test scenarios generated with respect to BeamNG.AI with an aggression factor set to 1. Since this is a cautious driving setup for BeamNG.AI, this dataset includes mostly safe scenarios, with about 26% of the scenarios being unsafe (causing OBEs). For the second experiment, we created a new dataset (referred to as *BeamNG.AI.AF1.5*) where we configured BeamNG.AI to drive in a more aggressive driving style. This resulted in 5,638 test scenarios among which 45% are unsafe.

To increase the level of reliability and applicability of our results, we used another SDC driving AI, namely Driver.AI, to generate the dataset of our last experiment. This last experiment was needed because using test scenarios with Driver.AI allows drawing a direct comparison with BeamNG.AI and investigating if the features we investigate are limited to BeamNG.AI or can be applied to other driving AIs. Thus, we used SDC-Scissor [15] (which integrates also AsFault) to

Table 2. Datasets Composition

Dataset	Number of Test Scenarios			Running Time
	Unsafe	Safe	Total	
BeamNG.AI.AF1	312 (26%)	866 (74%)	1,178	16h
BeamNG.AI.AF1.5	2,543 (45%)	3,095 (55%)	5,638	28h
Driver.AI	1,045 (19%)	4,585 (81%)	5,630	106h

re-run the test scenarios in *BeamNG.AI.AF1.5* with Driver.AI, resulting in a more cautious driving with only 19% of the scenarios being unsafe.

4.2 Analysis Method

4.2.1 RQ_1: Feature Analysis. In a real scenario, we do not determine the tests' safety without executing all of them. Hence, we do not include the feature that indicates if a test is safe or unsafe in this research question. So, to answer our first research question, we analyze the orthogonality of the other 16 different features introduced in Section 3.1. In particular, we use the PCA to statistically assess whether all features are useful for TCP or whether certain features are *multi-collinear*. A group of features is said to be *collinear* if they are linearly related and implicit measures of the same phenomenon (road characteristics in our case). Addressing data collinearity is vital to avoid distance measurements being skewed toward the collinear features [29]. Besides, distance metrics (including the Euclidean distance) might not truly represent the extent to which the data points (test cases) are truly diverse when using a large number of features [31].

PCA is a well-founded, analytical, and established technique that allows to identify the orthogonal dimensions (principal components) in the data and measure the contributions of the different features to such components. Features that contribute to the same principal components are collinear and can be removed via dimensionality reduction. In particular, the PCA decomposes each dataset M (e.g., `BeamNG.AI.AF1`) in two matrices: $\underset{m\times n}{M} \approx \underset{m\times n}{S} * \underset{n\times k}{V}$ In this equation, m is the number of test cases; n is the number of original features; k is the number of principal components; S denotes the features-to-component score matrix. More specifically, S contains each feature's scores (contributions) to the latent components identified by the PCA. In an ideal dataset with zero collinearity, the features should exclusively contribute to different principal components.

PCA can be used not only to detect but also to alleviate collinearity via dimensionality reduction [31]. In particular, a lower-dimensional matrix can be obtained by choosing the top $h < k$ principal components and reconstructing the matrix as:

$$\underset{m\times h}{M'} \approx \underset{m\times n}{S} * \underset{n\times h}{V}. \tag{10}$$

Notice that M' will contain new (non-collinear) features that are built as a combination of the old ones. This process is widely known in machine learning as *feature extraction* [39].

To answer RQ_1, we use PCA to detect (eventual) multi-collinearity among the different road features. In the case multi-collinearity is detected, we use PCA for *dimensionality reduction* and *feature extraction* by selecting the top k principal components corresponding to 98% of the original data variance, as recommended in the literature [39]. The selected, relevant features in RQ_1 (discussed in Section 5.1) are then considered to investigate RQ_2 and RQ_3 and applied for all search algorithms, i.e., for both greedy and evolutionary algorithms.

4.2.2 RQ_2: Cost-effectiveness of SDC-Prioritizer Compared to Baseline Approaches. To assess the effectiveness of TCP techniques introduced in this study, we look at the rate of fault detection (i.e., how fast faults are detected during the test execution process). Hence, a better technique provides

a test execution order that detects more faults while executing fewer tests. To indicate the rate of fault detection in our evaluation, we use a well-known metric in TCP, called Cost cognizant Average Percentage of Fault Detection ($APFD_c$) [32, 33, 48, 56, 72]. In this metric, higher $APFD_c$ means a higher fault detection rate. Since there is no technique introduced for measuring the fault severity in the SDC domain, we consider the same severity for all of the faults. Hence, in our case, $APFD_c$ can be formally defined as follows:

$$APFD_c = \frac{\sum_{i=1}^{m} \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^{n} t_j \times m}, \tag{11}$$

where T is the list of tests that need to be sorted for execution; t_j is the execution time required to run the test positioned as the jth test; n and m are the number of tests and faults, respectively; and TF_i is the position in the given test permutation that detect fault i. We also assessed whether there is no significant variation in execution time (simulation time) of the simulation-based tests by executing them multiple times. In particular, we randomly selected 50 tests from our dataset and ran them ten times each. As a result, the average standard deviation of test execution time is 1.67s (less than 1% variation) and the average coefficient of variance is 0.01.

To draw a statistical comparison between *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, random search, and greedy algorithm, we use Vargha-Delaney $\hat{A}_{12}$ statistic [82] to assess the effect size of differences between the $APFD_c$ values achieved by these approaches. A value $\hat{A}_{12} > 0.5$ for a pair of factors (A, B) confirms that A has a higher fault detection rate and vice versa. Furthermore, to examine if the differences are statistically significant, we use the non-parametric Wilcoxon Rank Sum test, with $\alpha = 0.05$ for Type I error.

4.2.3 RQ_3: Overhead Introduced by SDC-Prioritizer. For RQ_3, we monitor the running time needed by *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, and the greedy algorithm to prioritize the test cases. This analysis aims to verify whether the extra overhead introduced by *SDC-Prioritizer* techniques, on average, leads to a disruption in the testing process or is negligible compared to the total time needed to run the entire test suite. To have a more reliable estimation of the running time, we run both *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* 30 times and using the parameter values discussed in Section 4.2.4. Then, we measure the overhead of the different algorithms as the average running time over the 30 runs.

4.2.4 Parameter Setting. We used the default parameter values of the GA as used in previous studies on TCP (e.g., [33, 64, 81]). In particular, we use the following parameter values:

— *Population size*: we used a pool of 100 test permutations.
— *Crossover operator*: we used the partially-mapped crossover (PMX) for permutation problems (see Section 3.2) with a crossover probability $p_c = 0.80$. This corresponds to the default value in Matlab and it is inline with the recommended range $0.45 \leq p_c \leq 0.95$ [18, 23].
— *Mutation operator*: we used the hybrid mutation operator, introduced in Section 3.2.3, with a mutation probability $p_m = 1/n$, where n is the number of the test cases to prioritize. This choice is in line with the recommendations from previous studies [18, 74] that showed how p_m values proportional to the chromosome length produce better results.
— *Stopping criterion*: the search ends after 4,000 generations (or equivalently 400K fitness evaluations). We opted for a larger number of generations compared to prior studies in TCP (e.g., [12, 28, 54]) since the test suites in our benchmark are much larger than those used in prior studies in TCP for traditional software (e.g., the programs in the SIR dataset [46]).

Table 3. Results of the PCA for BeamNG.AI.AF1

Features	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
Direct Distance	−0.331	0.249	**0.874**	−0.068	0.218	−0.068	−0.074	−0.032	0.037	−0.003	−0.004	−0.001	0.003	0.017	0.007	−0.001
Road Distance	0.223	−0.129	0.187	0.315	0.013	−0.049	**0.810**	−0.148	0.275	−0.008	0.106	−0.019	0.000	0.179	0.004	0.002
Num. Left Turns	**0.421**	−0.033	0.189	0.127	−0.098	−0.213	−0.137	0.014	−0.233	0.040	0.063	−0.224	0.002	−0.076	−0.329	−0.687
Num. Right Turns	0.242	−0.333	0.199	0.103	−0.113	−0.193	−0.131	−0.003	−0.171	−0.113	−0.138	**0.580**	0.072	0.008	**0.552**	−0.075
Num. Straight	0.196	−0.144	−0.121	0.059	**0.956**	−0.012	−0.059	0.002	−0.063	−0.002	−0.005	0.011	0.013	−0.031	0.004	0.002
Total Angle	0.388	**0.797**	−0.089	−0.016	0.034	0.016	0.035	0.041	0.084	−0.318	−0.080	0.293	0.028	0.028	−0.005	0.006
Median Angle	**0.437**	−0.041	0.200	0.134	−0.107	−0.224	−0.147	0.018	−0.243	0.063	0.067	−0.217	0.013	−0.086	−0.169	**0.718**
Std Angle	0.151	0.299	−0.034	−0.021	0.012	0.020	0.008	−0.059	−0.046	**0.534**	0.092	−0.384	−0.034	0.006	**0.656**	−0.081
Max Angle	0.119	−0.073	0.025	0.199	−0.020	0.090	−0.448	0.015	**0.419**	−0.036	0.235	−0.058	0.056	**0.702**	0.005	0.002
Min Angle	0.147	−0.007	0.055	0.122	−0.021	0.126	−0.170	−0.119	**0.484**	**0.556**	−0.089	0.367	−0.155	−0.372	−0.230	0.015
Mean Angle	−0.065	0.107	−0.056	−0.163	0.026	−0.069	0.157	−0.041	−0.425	**0.512**	−0.085	0.344	0.150	0.508	−0.273	0.024
Median Pivot Off	−0.273	0.159	−0.121	**0.656**	0.007	−0.076	−0.115	−0.414	−0.246	−0.053	−0.170	0.004	−0.412	0.060	0.002	0.004
Std Pivot Off	−0.234	0.127	−0.089	**0.486**	0.005	−0.091	−0.035	0.168	−0.013	0.084	0.225	0.057	**0.736**	−0.212	−0.003	−0.009
Max Pivot Off	0.061	−0.009	0.111	0.096	−0.017	**0.633**	0.028	0.085	−0.328	−0.023	**0.610**	0.196	−0.198	−0.085	0.002	−0.002
Min Pivot Off	0.029	−0.018	0.096	0.300	−0.009	0.322	0.060	**0.698**	−0.052	0.078	−0.516	−0.104	−0.108	0.094	0.006	−0.001
Mean Pivot Off	−0.165	0.072	−0.109	0.033	0.036	−0.559	0.054	0.512	0.081	0.079	0.399	0.151	−0.426	−0.005	0.013	−0.003
Importance	31.35%	25.72%	13.76%	9.00%	6.14%	3.96%	3.32%	2.14%	1.71%	1.10%	0.54%	0.48%	0.41%	0.22%	0.12%	0.01%

Values in Boldface Indicate the Features that Contribute the Most to the Main Components (Cs) Extracted by PCA.

Table 4. Results of the PCA for BeamNG.AI.AF1.5

Features	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
Direct Distance	−0.3013	0.1697	**0.9106**	−0.0966	0.1742	−0.0920	−0.0470	−0.0146	0.0191	−0.0007	−0.0079	0.0009	−0.0012	0.0171	−0.0009	0.0005
Road Distance	0.2283	−0.0926	0.1635	0.3187	−0.0731	−0.0344	**0.7700**	−0.1680	0.3680	0.0025	0.0910	0.0113	0.0069	0.1987	0.0034	0.0055
Num. Left Turns	**0.4457**	0.0213	0.1532	0.0835	−0.1044	−0.2147	−0.0823	0.0475	−0.2461	0.0502	0.0436	−0.2968	0.0148	−0.0031	−0.3152	−0.6722
Num. Right Turns	0.3084	−0.3444	0.1755	0.0794	−0.1146	−0.2337	−0.0905	0.0292	−0.2168	−0.1006	−0.0110	0.6109	0.0538	−0.1288	**0.4669**	−0.0667
Num. Straight	0.1994	−0.0987	−0.0950	0.0410	0.9622	−0.0499	0.0662	0.0004	−0.0797	−0.0093	−0.0120	−0.0006	0.0081	−0.0209	0.0028	−0.0021
Total Angle	0.3007	**0.8304**	−0.0566	−0.0104	0.0210	0.0450	0.0081	0.0032	0.0601	−0.3113	−0.0037	0.3348	0.0019	−0.0334	−0.0194	0.0092
Median Angle	**0.4437**	0.0118	0.1536	0.0829	−0.1033	−0.2171	−0.0836	0.0501	−0.2489	0.0641	0.0356	−0.2888	0.0202	−0.0315	−0.1391	**0.7306**
Std Angle	0.1117	0.2956	−0.0191	−0.0040	0.0119	0.0229	−0.0105	−0.0115	0.0170	0.4787	−0.0104	−0.3483	−0.0090	0.0480	**0.7329**	−0.0916
Max Angle	0.1514	−0.0713	0.0231	0.1558	0.0425	−0.0111	−0.5247	−0.0747	**0.4051**	0.0470	0.1824	0.0978	0.1140	**0.6640**	−0.0328	0.0174
Min Angle	0.1366	0.0091	0.0453	0.1147	0.0176	0.0329	−0.1839	−0.1109	**0.4188**	**0.5555**	0.0046	0.2402	−0.1757	−0.5312	−0.2520	0.0159
Mean Angle	−0.0962	0.1302	−0.0554	−0.1244	−0.0040	−0.0033	0.2201	0.0609	−0.4133	**0.5867**	−0.0663	0.3854	0.1786	0.3766	−0.2459	0.0294
Median Pivot Off	−0.2875	0.1305	−0.0771	**0.6761**	0.0135	−0.1822	−0.1126	−0.3885	−0.2939	−0.0136	−0.1116	0.0198	−0.3706	0.0589	0.0060	0.0017
Std Pivot Off	−0.2112	0.0925	−0.0489	**0.4256**	0.0189	−0.1034	−0.0374	0.1325	0.0158	0.0081	0.1811	−0.0445	**0.7980**	−0.2478	0.0013	−0.0070
Max Pivot Off	0.0913	−0.0493	0.1334	0.1354	0.0043	**0.6999**	−0.0222	−0.0839	−0.2883	0.0017	**0.6017**	0.0295	−0.0778	−0.0576	0.0177	0.0008
Min Pivot Off	0.0551	−0.0323	0.1066	0.3956	−0.0009	0.3609	−0.0222	**0.6912**	0.0343	0.0069	−0.4418	0.0097	−0.1319	0.0746	−0.0047	−0.0006
Mean Pivot Off	−0.1854	0.0752	−0.1024	0.0329	0.0233	−0.4151	0.0758	**0.5394**	0.0562	0.0419	**0.5899**	0.0507	−0.3506	0.0126	0.0164	0.0010
Importance	30.96%	24.23%	13.80%	9.35%	5.74%	5.18%	3.38%	2.41%	1.81%	1.21%	0.70%	0.52%	0.34%	0.23%	0.10%	0.01%

Values in boldface indicate the features that contribute the most to the main components (Cs) extracted by PCA.

Notice that we did not fine-tune the parameters but opted for the default values. This choice is motivated by recent studies that showed that default values are a reasonable choice in search-based software engineering [7, 73]. Indeed, parameter tuning is a quite laborious and expensive process that does not assure better performances when using meta-heuristics. Our initial experiments confirm this finding as default values already provide good results in our case.

5 RESULTS

This section reports, for each research question, the obtained results and main findings.

5.1 RQ_1: SDC Features Analysis

Tables 3, 4, and 5 show the results of the PCA for datasets `BeamNG.AI.AF1`, `BeamNG.AI.AF1.5`, and `Driver.AI`, respectively. As we can observe, for each dataset, PCA identifies 16 (independent) principal components, whose relative importance is reported on the last (bottom) row of the corresponding Table. As these rows indicate, the importance of components in all of the tables (i.e., datasets) are similar: the first component (C1) covers about 30% of the variance in the data (importance), followed by the second components (C2) with about 25%, and so on. Moreover, in all of the datasets, the last six principal components are negligible as they contribute to less than 1% of the total variance.

Looking at the scores achieved for the different features, we can observe that they contribute to different (orthogonal) latent components. Hence, the features capture different characteristics of the road segments in the test scenarios. Individual features exclusively capture certain components. For example, in Table 3, C2 (which corresponds to 26% of the proportion) is fully captured by the

Table 5. Results of the PCA for Driver.AI

Features	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
Direct Distance	−0.1570	0.2939	**0.8788**	−0.0933	0.3088	−0.0926	−0.0499	−0.0153	−0.0294	0.0084	−0.0092	0.0064	0.0011	0.0173	−0.0012	−0.0003
Road Distance	0.1532	−0.1709	0.1659	0.3213	−0.0573	−0.0128	**0.7938**	−0.0402	−0.3669	0.0512	0.0751	0.0055	0.0020	0.1961	−0.0046	0.0027
Num. Left Turns	0.3838	−0.1879	0.1669	0.1319	−0.1174	−0.2410	−0.0898	−0.0012	0.2592	−0.0025	0.0384	−0.2417	0.0110	−0.0283	−0.3206	−0.6793
Num. Right Turns	0.0943	−0.3614	0.1681	0.1273	−0.1083	−0.2190	−0.0707	−0.0137	0.1842	−0.1474	−0.0088	**0.5874**	0.0922	−0.1185	**0.5677**	−0.0751
Num. Straight	0.1414	−0.2220	−0.2185	0.1112	**0.9269**	−0.0774	0.0262	0.0017	0.0606	−0.0221	−0.0078	−0.0069	0.0142	−0.0191	0.0005	−0.0009
Total Angle	**0.6742**	**0.5909**	−0.0893	−0.0497	0.0323	0.0720	0.0043	0.0011	−0.1245	−0.2761	−0.0155	0.2941	0.0234	−0.0214	−0.0070	0.0069
Median Angle	0.3858	−0.1979	0.1719	0.1352	−0.1190	−0.2465	−0.0922	−0.0011	0.2679	0.0111	0.0378	−0.2360	0.0154	−0.0419	−0.1552	**0.7250**
Std Angle	0.2334	0.2013	−0.0285	−0.0177	0.0139	0.0190	0.0112	0.0044	0.0838	**0.4907**	0.0118	−0.4344	−0.0506	0.0610	**0.6736**	−0.0807
Max Angle	0.0801	−0.1131	0.0146	0.1994	−0.0055	−0.0443	−0.5054	−0.0193	−0.4072	0.1097	0.1743	0.0810	0.1024	**0.6764**	−0.0111	0.0062
Min Angle	0.1119	−0.0437	0.0299	0.1272	0.0011	0.0117	−0.1629	−0.0436	−0.3222	**0.6446**	0.0343	0.3028	−0.1618	−0.5037	−0.2181	0.0148
Mean Angle	−0.0180	0.1361	−0.0565	−0.1477	0.0191	0.0074	0.2188	0.0212	**0.5247**	0.4732	−0.0530	0.3907	0.1923	0.4074	−0.2220	0.0220
Median Pivot Off	−0.2307	0.3422	−0.1052	**0.6554**	−0.0080	−0.1726	−0.0473	−0.4003	0.2179	−0.0509	−0.0995	0.0373	−0.3643	0.0513	0.0027	0.0011
Std Pivot Off	−0.1630	0.2366	−0.0680	0.3908	−0.0040	−0.0959	−0.0200	0.1901	−0.0124	0.0239	0.1577	−0.0874	**0.7919**	−0.2322	0.0001	−0.0037
Max Pivot Off	0.0615	−0.0875	0.1406	0.2016	0.0269	**0.7212**	−0.0486	−0.0572	0.2615	−0.0544	**0.5681**	0.0246	−0.0673	−0.0476	0.0085	−0.0011
Min Pivot Off	0.0280	−0.0266	0.0891	0.3459	−0.0095	0.3006	−0.0711	**0.6949**	0.0570	−0.0001	−0.5096	0.0150	−0.1592	0.0626	−0.0064	0.0009
Mean Pivot Off	−0.1191	0.1603	−0.1018	−0.0265	0.0076	−0.4015	0.0712	**0.5591**	0.0349	−0.0167	**0.5824**	0.0687	−0.3527	0.0128	0.0091	0.0003
Importance	29.97%	25.43%	12.33%	9.34%	7.09%	5.25%	3.60%	2.32%	1.67%	1.18%	0.77%	0.49%	0.38%	0.24%	0.10%	0.01%

Values in boldface indicate the features that contribute the most to the main components (Cs) extracted by PCA.

feature F6 (i.e., number of turns) with a score greater than 79%. Similar observations can be made for other components: C3 (14% of importance) is captured by F1 (direct end-to-end distance) with an 87% score; C5 (6% of importance) is exclusively related to F5 (number of straight segments) with 96% score; and so on. Similar results can also be observed in Tables 4 and 5.

Closely looking at C1, C9, and C10, in Table 3, (or C1, C4 in Table 4 and C8 and C10 in Table 5) we can observe that there are at least two features that equally contribute to them. In other words, some road features show some degree of collinearity. Finally, Features F3 (number of left turns) and F7 (median angle of turns in the road) both contributed about 40% to the first components (C1), which is the most important component according to PCA.

Therefore, we can conclude that the designed road features show some level of multi-collinearity, which is limited to a few features and for a few latent components. Hence, we use PCA for dimensionality reduction and feature extraction as described in Section 4.2.1. In particular, we select the top $h = 10$ principal components as they correspond to (cumulatively) 98% of the original data variance. According to the PCA Tables, the last six components are negligible as together account for less than 2% of the data variance in all of the datasets.

Given the results above, we used the lower-dimensional M' matrix produced by the PCA with $h = 10$ to compute the Euclidean distances and the fitness function used by *SDC-Prioritizer* and greedy-based test prioritization in RQ_2 and RQ_3. In particular, we use the new set of (non-collinear) features obtained with Equation (10).

Finding 1. The designed road features show some level of multi-collinearity. The first ten principal components produced by PCA allowed the identification of the ten meta-features, representing 98% of the original datasets' variance, to consider for experimenting with prioritization strategies (i.e., RQ_2).

5.2 RQ_2: Cost-effectiveness of *SDC-Prioritizer* Compared to Baseline Approaches

This section compares *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, random and greedy-based test prioritizations in terms of $APFD_c$. For both *SDC-Prioritizer* and the greedy-based approach, we use the first 10 principal components produced by PCA (detailed in Section 5.1). This allows us to perform an unbiased evaluation. We do not use these features nor the PCA for random search since (unlike *SDC-Prioritizer* and greedy) it does not require features to measure the distance between two tests. Figure 5 depicts the $APFD_c$ values achieved by *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, greedy-based, and random test prioritization approaches. As we can see in this figure, the best performing test prioritization in all of the datasets is *MO-SDC-Prioritizer*. In each dataset, the minimum $APFD_c$ achieved by *MO-SDC-Prioritizer* is higher than the maximum $APFD_c$ achieved by other test

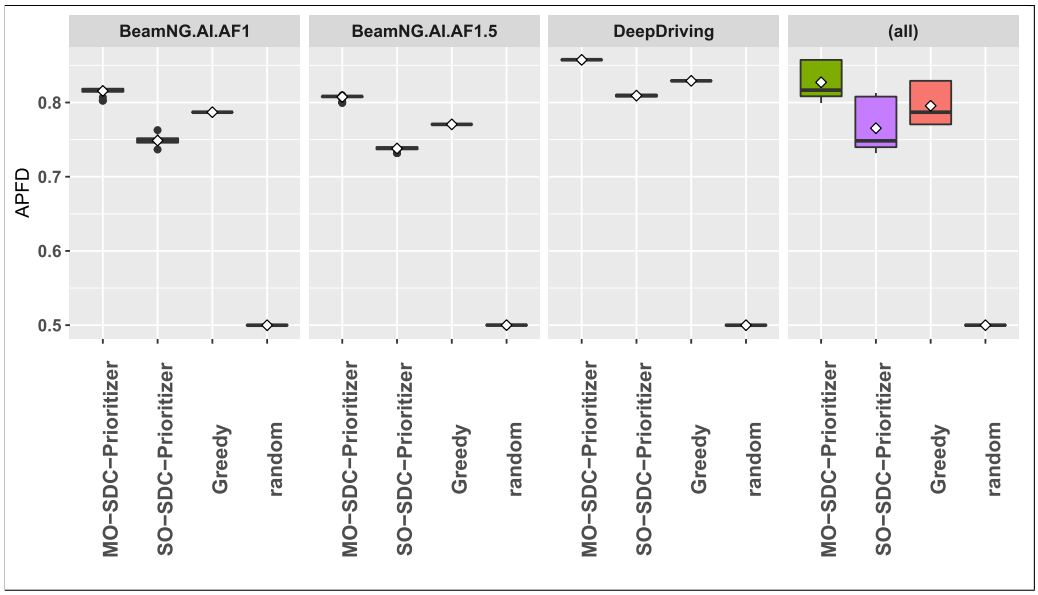


Fig. 5. APFD$_c$ achieved by *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, and greedy approach with ten features and random test prioritization. The diamond (⋄) denotes the arithmetic mean, and the bold line (—) is the median.

prioritization configurations. In all three datasets, the minimum $APFD_c$ achieved by *MO-SDC-Prioritizer* is at least 2%, 4%, 30% is higher than the highest $APFD_c$ produced by greedy, *SO-SDC-Prioritizer*, and random test prioritization, respectively. On average, *MO-SDC-Prioritizer* reaches about 3%, 6%, and 25.5% higher $APFD_c$ than Greedy, *SO-SDC-Prioritizer*, and random test prioritization, respectively. The second-best test prioritization technique is the greedy search (achieving an average $APFD_c$ of 79.5%), followed by *SO-SDC-Prioritizer* (with an average $APFD_c$ of 76.5%) and random test prioritization (with an average $APFD_c$ of 49.9%).

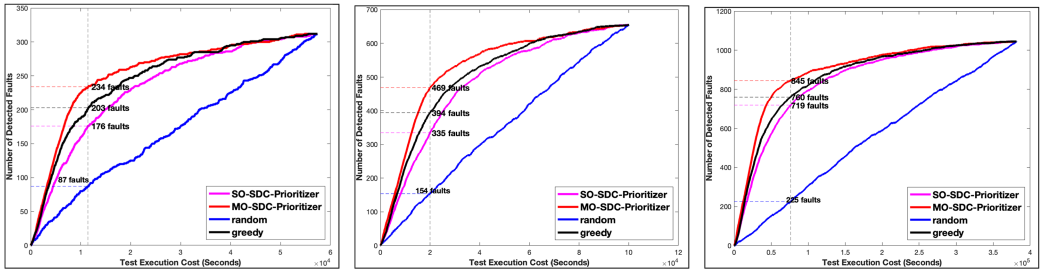
Moreover, as reported in Table 6, *MO-SDC-Prioritizer* significantly (P-values $< 1.0e-10$) outperforms (as all $\hat{A}_{12}$ values are all higher than 0.5) both random and greedy test prioritization in terms of $APFD_c$ score. The magnitude of the difference (effect size) is *large* in all datasets. Same as *MO-SDC-Prioritizer*, *SO-SDC-Prioritizer* significantly outperforms random test prioritization. However, this test prioritization technique achieves significantly lower $APFD_c$ values in comparison with greedy-based test prioritization in all datasets. Similar to the pairwise comparison of *SDC-Prioritizer* variants with baselines, *MO-SDC-Prioritizer* significantly achieves higher $APFD_c$ than *SO-SDC-Prioritizer* in all datasets (P-values $< 1.0e-10, \hat{A}_{12} = 1$, and large magnitude of effect sizes).

To provide more insights into these results, we graphically compare the cumulative number of faults detected by the different approaches when running the test cases incrementally according to the test prioritizations they produced. For each dataset, we took a more detailed look at the permutations generated by each *SDC-Prioritizer* variant that achieve an $APFD_c$ value equal to the median of the $APFD_c$ values delivered by all applications of that *SDC-Prioritizer* variant on a specific dataset. Specifically, for each of the *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*, we sampled three permutations generated by these techniques for each of the datasets. For each dataset, we compare the sampled permutations against the best output of random (i.e., the permutation generated by random that gains the best $APFD_c$) and greedy strategies. For this comparison, we

Table 6. Comparison of $APFD_c$ Score Achieved by *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* Against the Baselines, for each of the Datasets Used in this Study

GA Config.	Dataset	Vs. Random			Vs. Greedy		
		$\hat{A}_{12}$	p	Magnitude	$\hat{A}_{12}$	p	Magnitude
MO-SDC-Prioritizer	BeamNG.AI.AF1	1.0	3.016e-11	large	1.0	1.21e-12	large
	BeamNG.AI.AF1.5	1.0	3.016e-11	large	1.0	1.211e-12	large
	DeepDriving	1.0	2.113e-11	large	1.0	7.602e-13	large
SO-SDC-Prioritizer	BeamNG.AI.AF1	1.0	3.018e-11	large	0.0	1.211e-12	large
	BeamNG.AI.AF1.5	1.0	3.018e-11	large	0.0	1.212e-12	large
	DeepDriving	1.0	3.018e-11	large	0.0	1.211e-12	large

p-values for Wilcoxon tests, Vargha Delaney's estimates ($\hat{A}_{12}$), and magnitudes are reported.



(a) BeamNG.AI.AF1 (b) BeamNG.AI.AF1.5 (c) Driver.AI

Fig. 6. Cost-effectiveness curves produced by the different TPC methods. Each curve depicts the cumulative number of detected faults the cumulative test execution costs yielded by the TCPs.

analyze the rate of fault occurrences during the execution of tests, according to the generated permutations.

Figure 6 depicts this comparison for each dataset. As we can see from the figure, in all of the benchmarks, running the tests using the test case orders generated by *MO-SDC-Prioritizer* leads to a higher rate of fault occurrence in a shorter time. As a concrete example, in this figure, we highlighted the number of faults that occurred with the first 20% of the test execution. In the dataset BeamNG.AI.AF1 (Figure 6(a)), the permutation generated by *MO-SDC-Prioritizer* leads to the detection of 234 faults in the first 20% of test execution time. This value reduces for greedy (203), *SO-SDC-Prioritizer* (176), and random (87) test prioritization approaches. Similarly, in the second dataset (Figure 6(b)), *MO-SDC-Prioritizer* generates a permutation, which is able to detect 469 faults in the first 20% of the test execution. Also, in this case, this number is lower for the other approaches: 394, 335, and 154 faults detected by the greedy, *SO-SDC-Prioritizer*, and random approaches, respectively. The same trend is observed in the dataset of Driver.AI (Figure 6(c)), in which, the sampled permutation from *MO-SDC-Prioritizer* can detect 845 faults, i.e., +85, +126, and +620 more faults compared to greedy, *SO-SDC-Prioritizer*, and random algorithms, respectively.

Finding 2. *MO-SDC-Prioritizer* increases the $APFD_c$ score on average compared with random and greedy approaches. The improvement achieved by *SDC-Prioritizer*, in terms of fault detection rate, is statistically significant. Unlike *MO-SDC-Prioritizer*, which is the best performing test prioritization technique in terms of fault detection capability, *SO-SDC-Prioritizer* only achieves higher $APFD_c$ than random approach. This observation stems from the lack of exploration ability in this single-objective meta-heuristic, which drives the search process to trap local optima.

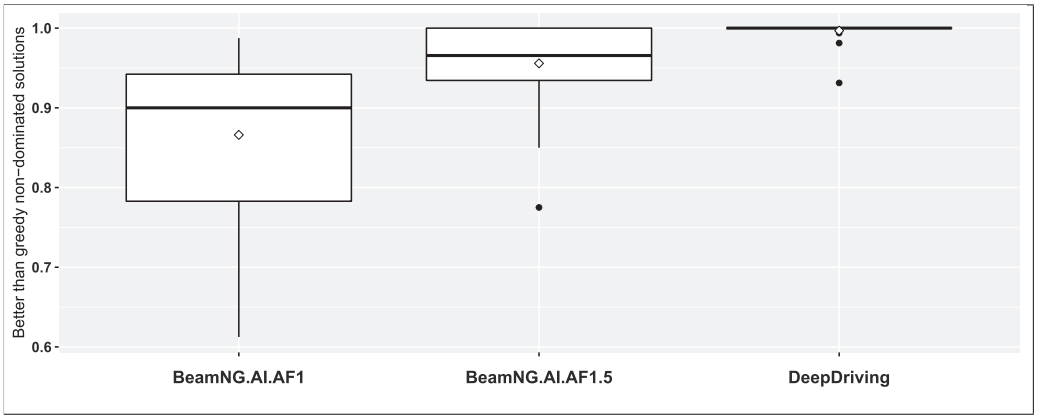


Fig. 7. Percentage of non-dominated solutions generated by *MO-SDC-Prioritizer* that achieve a higher $APFD_c$ compared to greedy-based test prioritization.

5.2.1 Pareto Fronts in MO-SDC-Prioritizer. As explained in Section 3, same as any other multi-objective approaches, *MO-SDC-Prioritizer* returns a set of non-dominated solutions in output. To answer RQ_2, we selected the closest non-dominated solution to the utopia point (explained in Section 3.3.2). Results presented by this section indicated that this solution has higher $APFD_c$ compared to the test execution orders generated by other techniques. However, we perform a more in-depth analysis to understand whether other non-dominated solutions could be selected from the Pareto front. To this aim, we compare the Pareto fronts (i.e., non-dominated test orders) generated by each *MO-SDC-Prioritizer*'s run with the $APFD_c$ achieved by the second-best technique (i.e., greedy-based test prioritization) in terms of fault detection capability. Figure 7 presents the percentage of non-dominated solutions generated by *MO-SDC-Prioritizer* that achieves a higher $APFD_c$ compared to the Greedy approach. On average, about 94% of non-dominated solutions generated by *MO-SDC-Prioritizer* can detect more unsafe tests than Greedy and in shorter times (i.e., they have higher $APFD_c$). Even in the worst scenario (17th execution of *MO-SDC-Prioritizer* on BeamNG.AI.AF1 dataset), more than 61% of generated solutions in the final Pareto front produced by *MO-SDC-Prioritizer* has higher $APFD_c$ compared to Greedy. The highest performance of *MO-SDC-Prioritizer* can be observed when this test prioritization technique is utilized to prioritize tests for the DeerDriving dataset in which, on average, 99.7% of solutions have higher $APFD_c$ than the ones generated by Greedy test prioritization.

To better understand the impacting factors that lead the generated non-dominated solutions to achieve a high $APFD_c$, we manually analyzed the $APFD_c$ values of Pareto fronts generated by *MO-SDC-Prioritizer* in each dataset. In all of the cases, we observed the same trend as the sample, presented in Figure 8. This figure is a two-dimensional vector in which each dimension indicates one of the *MO-SDC-Prioritizer*'s search objectives (diversity and execution cost). As we can see, all solutions with the lowest $APFD_c$ (red points in the Pareto front) are the extreme points with the maximum diversity and maximum test execution costs. In addition, the solution with the highest $APFD_c$ (the orange diamond point) is not in the extreme parts of the Pareto front (i.e., it has a good balance between the diversity and execution cost). As we can observe, the knee point selected by *MO-SDC-Prioritizer* is among the middle points in the front with the largest $APFD_c$. Besides, it is very close to the best point (in terms of $APFD_c$) within the Pareto front. This observation empirically supports the technique we used for selecting the final test order (the yellow diamond point).

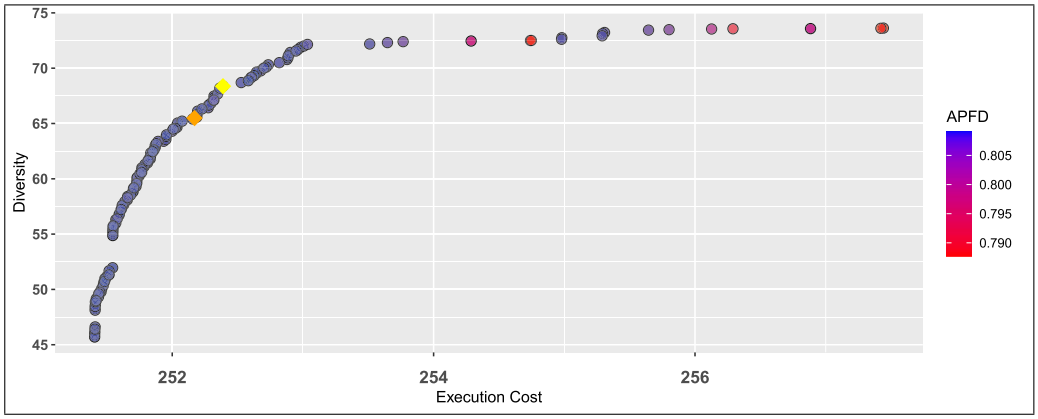


Fig. 8. A sample of Pareto front generated by *MO-SDC-Prioritizer* in BeamNG.AI.AF1.5 dataset. Each circle point represents one of the non-dominated solutions in the Pareto front. The blue points are the solutions with an $APFD_c$ score larger than the one produced by the greedy algorithm. The orange and yellow diamond points indicate the solution with the highest APFD and the closest solution to the utopia point, respectively.

Finding 3. On average, the majority (94%) of the solutions generated by *MO-SDC-Prioritizer* has higher $APFD_c$ than Greedy (the second-best test prioritization technique for detecting faults in a shorter time). By taking a deeper look at non-dominated solutions generated by *MO-SDC-Prioritizer*, we can see that the few solutions with lower $APFD_c$ are at the extremes of the Pareto front. Moreover, the solutions with the highest $APFD_c$ values are the ones that have a balance between tests diversity and test execution cost.

5.3 RQ_3: Overhead of *SDC-Prioritizer*

Figure 9 illustrates the distribution of the time consumed by *SO-SDC-Prioritizer*, *MO-SDC-Prioritizer*, and greedy test prioritization. As this figure shows, on average, *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* require about 12.5 and 11.5 minutes to finish the search process with 4,000 generations, respectively. Practically, this amount of time is negligible if we consider the total 16 to 106 hours needed to run the entire set of tests, and that both variants of *SDC-Prioritizer* do not negatively impact the performance (e.g., on fault detection) of testing practices. In fact, the overall overhead accounts for 0.38% (for Driver.AI) and a maximum of 0.45% (for BeamNG.AI.AF1.5) of the cost needed to run the entire test suites.

Finding 4. The overhead introduced by each *SDC-Prioritizer* variants is less than 13 minutes and is imperceptible for an SDC simulation pipeline used by developers to test the SDCs behavior in critical scenarios.

Figure 9 shows that (right side of the Figure) the average time required by the greedy approach is about five times shorter than what *SO-SDC-Prioritizer* or *MO-SDC-Prioritizer* needs. Even though *MO-SDC-Prioritizer* is slower than greedy (i.e., it needs about 10 minutes more time), it performs better in terms of $APFD_c$ score (as shown by Section 5.2).

Finding 5. On average, *MO-SDC-Prioritizer* needs about 10 minutes more than the greedy test prioritization. However, this negligible extra overhead significantly increases the $APFD_c$ values achieved by the subsequently generated test prioritization.

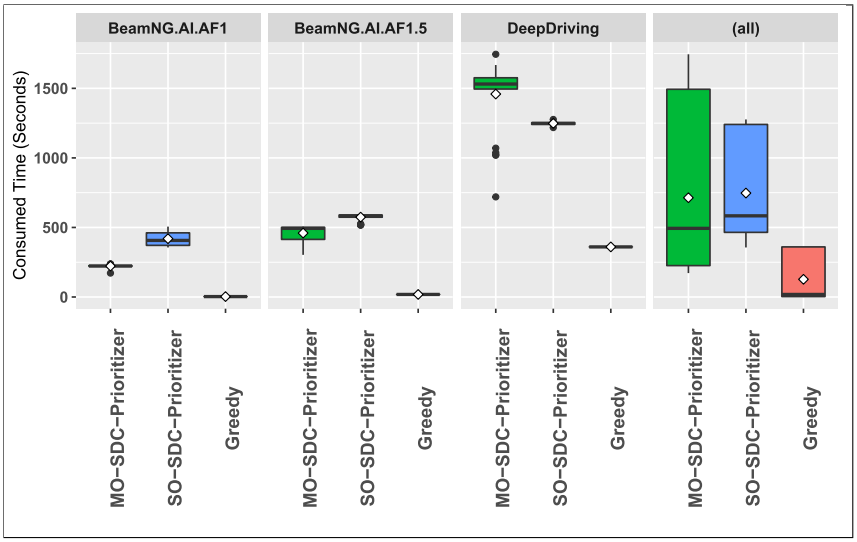


Fig. 9. Running time of the different TCP approaches.

Finally, it is worth mentioning that *SDC-Prioritizer* techniques include two main parts: (i) pairwise comparison of distances between every two tests (using Euclidean distance), and (ii) running the GA. The former is a one-time task (i.e., by one execution, we can run the GA multiple times) with the time complexity of $O(n^2)$, where n is the number of tests. Since the latter part uses the values calculated in pairwise distance calculation for fitness function evaluation, the complexity of this task is $O(n)$ (this complexity is due to the search for the most diverse test). Also, the time complexities of mutation and crossover operators are $O(n)$. Hence, *SDC-Prioritizer* has $O(n^2)$ one-time cost (for calculating the distances) and $O(n \times m)$ for the whole search process, where n is the number of tests, and m is the number of fitness evaluations. According to this information, we can confirm that *SDC-Prioritizer* scales for a large-size test set. Similarly, the test suites used in our study are much larger than the other ones reported in prior studies on regression testing [71]. Our largest test suite (Driver.AI) contains 5,630 tests. On average, *SDC-Prioritizer* approaches performed the test prioritization for this test suite in less than 25 minutes.

6 THREATS TO VALIDITY

Threats to *construct validity* concern the relationship between theory and observation. In this case, threats can be mainly due to the imprecision in simulation realism as well as the automated classification of safe and unsafe scenarios. We mitigated both threats by leveraging BeamNG (used in this year's SBST tool competition [66]) as a soft-body simulation environment (which ensures a high simulation accuracy in safety-critical scenarios) and SDC-Scissor [15] (which integrates also AsFault) as a technological reference solution to generate and execute test cases, as detailed in Section 4. Furthermore, to address the potential threat to have high variability in execution time of the executed tests, we selected a sample of 50 test cases (using a stratified random sampling, equal distribution of safe and unsafe tests) and executed them 10 times each. As mentioned in Section 4.2.2, the standard deviation of the execution time is negligible.

Threats to *internal validity* may concern, as for previous work [38], the relationships between the technologies used to generate the scenarios and the realism of simulation results. Specifically, we did not recreate all the elements that can be found on real roads (e.g., weather conditions

and light conditions). However, to increase our internal validity, we focused on the usage of both BeamNG.AI and Driver.AI as test subjects. This allows us to assess the cost-effectiveness of our approach by experimenting with different driving styles and driving risk levels. Both BeamNG.AI and Driver.AI leverage a good knowledge of the roads, which means that they do not suffer from limitations of vision-based lane-keeping systems. However, since with BeamNG.AI it is possible to adjust the driving risk level, a higher amount of unsafe test scenarios can be observed. Hence, an AI implemented in physical SDC might be much more conservative in its driving style, which is something we plan to investigate for future work.

Finally, threats to *external validity* concern the generalization of our findings. The number of experimented test case scenarios in our study is larger than in previous studies [38] and we experimented with different AI engines. However, our results could not generalize with the universe of general open-source CPS simulation environments used in other domains. Therefore, further studies considering more SDC data, other CPS domains, and different safety requirements are expected. To minimize potential external validity in our evaluation setting, we followed the guidelines by Arcuri et al. [6]: we compared the results of *SDC-Prioritizer* with randomized test generation algorithms (the baseline approaches described in Section 4) presented and repeated the experiment 30 times. Finally, we applied sound non-parametric statistical tests and statistics to analyze the achieved results.

7 CONCLUSIONS & FUTURE WORK

Regression testing for SDCs is particularly expensive due to the cost of running many test driving scenarios (test cases) that interact with simulation engines. To improve the cost-effectiveness of regression testing, we introduced two black-box TCP approaches, called *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer*. These approaches rely on a set of static road features and are suitably designed for SDCs. These features can be extracted from the driving scenarios prior to running the tests. Both of these techniques utilize GAs to prioritize the test cases based on their distances (diversity) computed using the proposed road features and test execution costs. *SO-SDC-Prioritizer* performs a single-objective optimization to fulfill this task (i.e., both test diversity and execution costs are included in a single fitness function), while *MO-SDC-Prioritizer* leverages one of the common multi-objective GA (NSGA-II) to prioritize tests according to two search objectives (one for differences of tests and the other one for test execution costs).

We empirically investigated the performances of *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* and compared it with two baselines: random search and greedy algorithms. Finally, we assessed whether these proposed techniques do not introduce a too large computational overhead to the regression testing process. Our results show that *MO-SDC-Prioritizer* is more cost-effective than the baseline approaches. Specifically, the single solution provided by *MO-SDC-Prioritizer* dominates the solutions provided by *SO-SDC-Prioritizer* and the baselines in terms of test execution time and fault detection capability. Moreover, both *SDC-Prioritizer* techniques successfully prioritize the test cases independently of which AI engine is used (i.e., Driver.AI and BeamNG.AI) or different risk levels (i.e., different driving styles). Interestingly, looking at the running time, we can observe that the overhead required by *SO-SDC-Prioritizer* and *MO-SDC-Prioritizer* in prioritizing the test scenarios is negligible with regards to the overall test execution cost.

We plan to replicate our study on further SDC AIs and additional SDC features as future work. Moreover, we plan to perform new empirical studies on further CPS domains to investigate additional safety criteria concerning new types of faults different from those investigated in this work. Specifically, important for this is to investigate approaches that are more human-oriented or are able to integrate humans into-the-loop [42, 67, 79, 80]. Moreover, we want to investigate different meta-heuristics in addition to the GA used in this article. Complementary, we aim to investigate

different distance functions to measure the diversity of the test cases (e.g., graph-based distances over feature-vector-based distances). Finally, we plan to integrate the proposed solution based on the experimented simulation environments to prioritize devise signals into industrial context such as AICAS context,[5] involved in the COSMOS H2020 project.[6]

REFERENCES

[1] 2020. NVIDIA DRIVE Constellation. Retrieved 16 June 2022 from https://developer.nvidia.com/drive/drive-constellation.

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 143–154.

[3] Academies of Sciences. 2017. *A 21st Century Cyber-Physical Systems Education*. National Academies Press.

[4] Afsoon Afzal, Deborah S. Katz, Claire Le Goues, and Christopher S. Timperley. 2020. A study on the challenges of using robotics simulators for testing. *CoRR*, arXiv:2004.07368. Retrieved from https://arxiv.org/abs/2004.07368.

[5] Mustafa Al-Hajjaji, Thomas Thüm, Jens Meinicke, Malte Lochau, and Gunter Saake. 2014. Similarity-based prioritization in software product-line testing. In *Proceedings of the 18th International Software Product Line Conference,* Stefania Gnesi, Alessandro Fantechi, Patrick Heymans, Julia Rubin, Krzysztof Czarnecki, and Deepak Dhungana (Eds.). ACM, 197–206. DOI: https://doi.org/10.1145/2648511.2648532

[6] Andrea Arcuri and Lionel C. Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing Verification and Reliability* 24, 3 (2014), 219–250. DOI: https://doi.org/10.1002/stvr.1486

[7] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18, 3 (2013), 594–623.

[8] Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria, and Justyna Zander. 2016. Automatic generation of test system instances for configurable cyber-physical systems. *Software Quality Journal* 25, 3 (Sept. 2016), 1041–1083. DOI: https://doi.org/10.1007/s11219-016-9341-7

[9] Aitor Arrieta, Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2018. Multi-objective black-box test case selection for cost-effectively testing simulation models. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1411–1418.

[10] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2018. Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics* 14, 3 (2018), 1055–1066. DOI: https://doi.org/10.1109/TII.2017.2788019

[11] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. 2016. Search-based test case selection of cyber-physical system product lines for simulation-based validation. In *Proceedings of the 20th International Systems and Software Product Line Conference*, Hong Mei (Ed.). ACM, 297–306. DOI: https://doi.org/10.1145/2934466.2946046

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

[13] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The Impact of Control Technology* 12, 1 (2011), 161–166.

[14] BeamNG.research. [n.d.]. BeamNG GmbH, "BeamNG.research – BeamNG." Retrieved April 2021 from https://beamng.gmbh/research/.

[15] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2022. Cost-effective simulation-based test selection in self-driving cars software with SDC-scissor. 2022. In *Proceedings of the 29th IEEE International Conference on Software Analysis, Evolution, and Reengineering*. To appear.

[16] Elizabeth Bondi, Debadeepta Dey, Ashish Kapoor, Jim Piavis, Shital Shah, Fei Fang, Bistra Dilkina, Robert Hannaford, Arvind Iyer, Lucas Joppa, and Milind Tambe. 2018. AirSim-W: A simulation environment for wildlife conservation with UAVs. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, Ellen W. Zegura (Ed.). ACM, 40:1–40:12. DOI: https://doi.org/10.1145/3209811.3209880

[17] Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. 2004. Finding knees in multi-objective optimization. In *Proceedings of the 8th International Parallel Problem Solving from Nature* LNCS, Vol. 3242. Springer, Berlin, 722–731.

[5] https://www.aicas.com/wp/.

[6] https://www.cosmos-devops.org/.

[18] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. 2006. Using genetic algorithms for early schedulability analysis and stress testing in real-time systems. *Genetic Programming and Evolvable Machines* 7, 2 (2006), 145–170.

[19] Jinfu Chen, Lili Zhu, Tsong Yueh Chen, Dave Towey, Fei-Ching Kuo, Rubing Huang, and Yuchi Guo. 2018. Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. *Journal of Systems and Software* 135 (2018), 107–125.

[20] Tsong Yueh Chen and Man Fai Lau. 1996. Dividing strategies for the optimization of a test suite. *Information Processing Letters* 60, 3 (Nov. 1996), 135–141.

[21] Birchler Christian, Khatiri Sajad, Derakhshanfar Pouria, Panichella Sebastiano, and Panichella Annibale. 2021. *Dataset of the paper "Automated Test Cases Prioritization for Self-driving Cars in Virtual Environments"*. DOI: https://doi.org/10.5281/zenodo.5771017

[22] OpenStax CNX. 2021. OpenStax University Physics. Retrieved 16 June 2022 from http://cnx.org/contents/d50f6e32-0fda-46ef-a362-9bd36ca7c97d@10.16.

[23] Helen G. Cobb and John J. Grefenstette. 1993. *Genetic Algorithms for Tracking Changing Environments.* Technical Report. Naval Research Lab Washington DC.

[24] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. 2006. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[25] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys* 45, 3 (2013), 1–33.

[26] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2000. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2000), 182–197.

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

[28] Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2018. A test case prioritization genetic algorithm guided by the hypervolume indicator. *IEEE Transactions on Software Engineering* 46, 6 (2018), 674–696.

[29] Carsten F. Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carré, Jaime R. García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J. Leitao, and Munkemuller T. 2013. Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. *Ecography* 36, 1 (2013), 27–46.

[30] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, 1–16. Retrieved from http://proceedings.mlr.press/v78/dosovitskiy17a.html.

[31] Mingjing Du, Shifei Ding, and Hongjie Jia. 2016. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Systems* 99 (2016), 135–145.

[32] Sebastian Elbaum, Alexey Malishevsky, and Gregg Rothermel. 2001. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering.* IEEE, 329–338.

[33] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. 2015. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis.* ACM, 234–245.

[34] Robert Feldt, Simon Poulding, David Clark, and Shin Yoo. 2016. Test set diameter: Quantifying the diversity of sets of test cases. In *Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation.* IEEE, 223–233.

[35] Erik Flores-García, Goo-Young Kim, Jinho Yang, Magnus Wiktorsson, and Sang Do Noh. 2020. Analyzing the characteristics of digital twin and discrete event simulation in cyber physical systems. In *Proceedings of the Advances in Production Management Systems. Towards Smart and Digital Manufacturing*, Bojan Lalic, Vidosav D. Majstorovic, Ugljesa Marjanovic, Gregor von Cieminski, and David Romero (Eds.). Springer, 238–244. DOI: https://doi.org/10.1007/978-3-030-57997-5_28

[36] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 257–267. DOI: https://doi.org/10.1145/3338906.3338942

[37] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. AsFault: Testing self-driving car software using search-based procedural content generation. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings.* IEEE. DOI: https://doi.org/10.1109/icse-companion.2019.00030

[38] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis.* ACM, New York, NY, 318–328. DOI: https://doi.org/10.1145/3293882.3330566

[39] Aurélien Géron. 2019. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.
[40] David E. Goldberg. 2006. *Genetic Algorithms*. Pearson Education India.
[41] Carlos A. González, Mojtaba Varmazyar, Shiva Nejati, Lionel C. Briand, and Yago Isasi. 2018. Enabling model testing of cyber-physical systems. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, New York, NY, 176–186. DOI: https://doi.org/10.1145/3239372.3239409
[42] Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald Gall. 2018. Exploring the integration of user feedback in automated testing of Android applications. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*.
[43] The Guardian. 2018. Self-driving Uber kills Arizona woman in first fatal crash involving pedestrian. Retrieved 16 June 2022 from https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe.
[44] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology* 22, 1 (2013), 6.
[45] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering*. IEEE, 523–534.
[46] Sebastian G. Elbaum Hyunsook Do and Gregg Rothermel. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering* 10 (2005), 405–435.
[47] Félix Ingrand. 2019. Recent trends in formal validation and verification of autonomous robots software. In *Proceedings of the 3rd IEEE International Conference on Robotic Computing*. 321–328.
[48] Bo Jiang, Zhenyu Zhang, Wing Kwong Chan, and TH Tse. 2009. Adaptive random test case prioritization. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 233–244.
[49] Nidhi Kalra and Susan Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (Dec. 2016), 182–193. DOI: https://doi.org/10.1016/j.tra.2016.09.010
[50] Jiseob Kim, Sunil Chon, and Jihwan Park. 2019. Suggestion of testing method for industrial level cyber-physical system in complex environment. In *Proceedings of the 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops*. IEEE. DOI: https://doi.org/10.1109/icstw.2019.00043
[51] Juhani Koski and Risto Silvennoinen. 1987. Norm methods and partial weighting in multicriterion optimization of structures. *Internaternational Journal for Numerical Methods in Engineering* 24, 6 (1987), 1101–1121.
[52] Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. 2012. Prioritizing test cases with string distances. *Automated Software Engineering* 19, 1 (2012), 65–95.
[53] Hong Li, Yong-Chang Jiao, Li Zhang, and Ze-Wei Gu. 2006. Genetic algorithm based on the orthogonal design for multidimensional knapsack problems. *Advances in Natural Computation* 4221 (2006), 696–705.
[54] Zheng Li, Mark Harman, and Robert M. Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering* 33, 4 (2007), 225–237. DOI: https://doi.org/10.1109/TSE.2007.38
[55] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. 2020. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics* 36, 1 (2020), 1–14.
[56] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, and Sebastian Elbaum. 2006. *Cost-cognizant Test Case Prioritization*. Technical Report. Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln.
[57] Alessandro Marchetto, Md Mahfuzul Islam, Waseem Asghar, Angelo Susi, and Giuseppe Scanniello. 2015. A multi-objective technique to prioritize test cases. *IEEE Transactions on Software Engineering* 42, 10 (2015), 918–940.
[58] R. T. Marler and J. S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26 (2004), 369–395.
[59] Reza Matinnejad, Shiva Nejati, Lionel Briand, Thomas Bruckmann, and Claude Poull. 2013. Automated model-in-the-loop testing of continuous controllers using search. In *Proceedings of the International Symposium on Search Based Software Engineering*. Springer, 141–157.
[60] Jaruwan Mesit and Ratan K. Guha. 2011. A general model for soft body simulation in motion. In *Proceedings of the Winter Simulation Conference*, S. Jain, Roy R. Creasey Jr., Jan Himmelspach, K. Preston White, and Michael C. Fu 0001 (Eds.). IEEE, 2690–2702. Retrieved from http://dl.acm.org/citation.cfm?id=2431518.
[61] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*. 167–177.
[62] Breno Miranda, Emilio Cruciani, Roberto Verdecchia, and Antonia Bertolino. 2018. Fast approaches to scalable similarity-based test case prioritization. In *Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 222–232.

[63] Réka Nagy, Mihai A. Suciu, and Dumitru Dumitrescu. 2012. Lorenz equilibrium: Equitability in non-cooperative games. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York, NY, 489–496.

[64] Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. 2020. A test case prioritization genetic algorithm guided by the hypervolume indicator. *IEEE Transactions on Software Engineering* 46, 6 (2020), 674–696. DOI: https://doi.org/10.1109/TSE.2018.2868082

[65] Annibale Panichella, Fitsum M. Kifetew, and Paolo Tonella. 2015. Reformulating branch coverage as a many-objective optimization problem. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation*. IEEE, Piscataway, NJ, 1–10.

[66] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. 2021. SBST tool competition 2021. In *Proceedings of the International Conference on Software Engineering, Workshops, 2021*. ACM.

[67] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *Proceedings of the International Conference on Software Maintenance and Evolution*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 281–290. DOI: https://doi.org/10.1109/ICSM.2015.7332474

[68] Singiresu S. Rao and TI Freiheit. 1991. A modified game theory approach to multiobjective optimization. *Journal of Mechanical Design* 113, 3 (1991), 286–291.

[69] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 13 pages. DOI: https://doi.org/10.1145/3368089.3409730

[70] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. 1998. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proceedings of the International Conference on Software Maintenance*. IEEE CS Press, 34–44.

[71] G. Rothermel, R. H. Untch, Chengyun Chu, and M. J. Harrold. 1999. Test case prioritization: An empirical study. In *Proceedings of the IEEE International Conference on Software Maintenance*. 179–188. DOI: https://doi.org/10.1109/ICSM.1999.792604

[72] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 1999. Test case prioritization: An empirical study. In *Proceedings of the IEEE International Conference on Software Maintenance*. IEEE, 179–188.

[73] Abdel Salam Sayyad, Katerina Goseva-Popstojanova, Tim Menzies, and Hany Ammar. 2013. On parameter tuning in search based software engineering: A replicated empirical study. In *Proceedings of the 2013 3rd International Workshop on Replication in Empirical Software Engineering Research*. IEEE, 84–90.

[74] J. David Schaffer, Rich Caruana, Larry J. Eshelman, and Rajarshi Das. 1989. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*. 51–60.

[75] Martin Serpell and James E. Smith. 2010. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation* 18, 3 (Sep. 2010), 491–514. DOI: https://doi.org/10.1162/EVCO_a_00006

[76] Seung Yeob Shin, Karim Chaouch, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. 2018. HITECS: A UML profile and analysis framework for hardware-in-the-loop testing of cyber physical systems. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*. ACM, 357–367.

[77] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. 2018. Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Frank Tip and Eric Bodden (Eds.). ACM, 49–60. DOI: https://doi.org/10.1145/3213846.3213852

[78] Sebastian Sontges and Matthias Althoff. 2018. Computing the drivable area of autonomous road vehicles in dynamic road scenes. *IEEE Transactions on Intelligent Transportation Systems* 19, 6 (2018), 1855–1866. DOI: https://doi.org/10.1109/TITS.2017.2742141

[79] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the International Symposium on Foundations of Software Engineering*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 499–510. DOI: https://doi.org/10.1145/2950290.2950299

[80] The-Washington-Post. 2019. Uber's radar detected Elaine Herzberg nearly 6 seconds before she was fatally struck, but "the system design did not include a consideration for jaywalking pedestrians" so it didn't react as if she were a person. Retrieved 16 June 2022 from https://mobile.twitter.com/faizsays/status/1191885955088519168.

[81] Stephen W. Thomas, Hadi Hemmati, Ahmed E. Hassan, and Dorothea Blostein. 2014. Static test case prioritization using topic models. *Empirical Software Engineering* 19, 1 (2014), 182–212.

[82] András Vargha and Harold D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[83] Jiaxuan Xu, Qi Luo, Kecheng Xu, Xiangquan Xiao, Siyang Yu, Jiangtao Hu, Jinghao Miao, and Jingao Wang. 2019. An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform. In *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5049–5056. DOI: https://doi.org/10.1109/IROS40897.2019.8968102

[84] Rajaa Vikhram Yohanandhan, Rajvikram Madurai Elavarasan, Premkumar Manoharan, and Lucian Mihet-Popa. 2020. Cyber-physical power system (CPPS): A review on modeling, simulation, and analysis with cyber security applications. *IEEE Access* 8 (2020), 151019–151064. DOI: https://doi.org/10.1109/ACCESS.2020.3016826

[85] Shin Yoo. 2010. A novel mask-coding representation for set cover problems with applications in test suite minimisation. In *Proceedings of the 2nd International Symposium on Search-Based Software Engineering*. IEEE.

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

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

[88] Eleni Zapridou, Ezio Bartocci, and Panagiotis Katsaros. 2020. Runtime verification of autonomous driving systems in CARLA. In *Proceedings of the Runtime Verification*, Jyotirmoy Deshmukh and Dejan Ničković (Eds.). Springer International Publishing, Cham, 172–183.

[89] Béla Újházi, Rudolf Ferenc, Denys Poshyvanyk, and Tibor Gyimóthy. 2010. New conceptual coupling and cohesion metrics for object-oriented systems. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. 33–42. DOI: https://doi.org/10.1109/SCAM.2010.14

Received 13 July 2021; revised 21 December 2021; accepted 27 April 2022