Empirical Software Engineering (2025) 30:112
https://doi.org/10.1007/s10664-025-10670-w

An industrial experience report on applying search-based boundary input generation to cyber-physical systems

Pablo Valle[1] · Vincenzo Riccio[2] · Aitor Arrieta[1] · Paolo Tonella[3] · Maite Arratibel[4]

Accepted: 5 May 2025

Abstract
Testing Cyber Physical Systems (CPS) is crucial, as they play a central role in modern society. In the complex input space of these systems, boundary test inputs provide a valuable asset for test engineers as they identify slight input modifications that dramatically impact Quality of Service. In this experience paper, we propose LIFTJANUS, the first search-based test generator for CPS that integrates test input minimization, boundary value detection, and automated system repair. We performed an empirical study involving two real-world elevator systems provided by our industrial collaborator, ORONA. Our results proved that LIFTJANUS generated boundary inputs twice as effective as the baselines, with the repair algorithm successfully enhancing the system's configuration in 76.25% of the cases. Interviews with domain experts confirmed that LIFTJANUS is a comprehensive solution for enhancing the quality of elevator systems.

Keywords Cyber-physical systems · Search-based test input generation · Simulation-based testing

Communicated by: Lei Ma

✉ Pablo Valle
pvalle@mondragon.edu

Vincenzo Riccio
vincenzo.riccio@uniud.it

Aitor Arrieta
aarrieta@mondragon.edu

Paolo Tonella
paolo.tonella@usi.ch

Maite Arratibel
marratibel@orona-group.com

1 Mondragon University, Mondragón, Guipuzcoa, Spain

2 Università di Udine, Udine, Italy

3 Università della Svizzera italiana, Lugano, Switzerland

4 Oorna, Hernani, Guipuzcoa, Spain

Published online: 17 May 2025

1 Introduction

Cyber-Physical Systems (CPSs) integrate digital cyber technologies with physical processes (Derler et al. 2011). Examples of CPSs include Unmanned Aerial Vehicles (UAVs) and Autonomous Driving Systems (ADSs). The case study system of this paper encompasses a system of elevators, particularly relevant due to their ubiquitous use in our daily lives. Like other CPSs, elevators operate in highly evolving and uncertain environments, where minor uncertainty factors can produce substantial deviations in the behavior of the system, often leading to failures (Han et al. 2023, 2022). For instance, a recent study demonstrated that variations in the weights of passengers can lead to a significant degradation in the quality of service (QoS) provided by the system (Han et al. 2023). Similar effects are also common in other CPSs from other domains. For instance, in Lane Keeping Assist Systems, minor variations in road shape, e.g., small changes in the radius of curvature or the number of turns, can result in the vehicle going out of the right lane (Gambi et al. 2019; Zohdinasab et al. 2023). Therefore, finding unforeseen test inputs for CPSs is paramount to ensure high system dependability prior to deploying a software version in operation. However, in the context of CPSs, this becomes an extremely complex task due to the following challenges.

Challenge 1 – Long test execution time Executing CPS test cases is extremely time consuming (Abdessalem et al. 2020; Nejati et al. 2019; Menghi et al. 2020; Haq et al. 2022; Humeniuk et al. 2022; Nejati et al. 2023; Zhang et al. 2023). On the one hand, CPSs need to be tested at system level (e.g., by using simulation techniques), as they work in a closed-loop fashion (Stocco et al. 2023). This implies that decisions taken by the control software affect the physical counterpart of the system, and vice-versa. The physical layer of the CPS is usually modeled through complex mathematical models that require high computational resources for simulation. Moreover, test inputs for CPSs are usually long streams of data over time (e.g., signals stimulating the inputs of the system, extended road segments) (Menghi et al. 2021; Valle et al. 2023). In our industrial case study system, practitioners employ full-day traffic data of passengers traveling throughout a building installation during an entire day.

Challenge 2 – Large and multi-dimensional input space The potential combinations of test input values for a CPS can be extremely large, not only due to the number of inputs, but also due to the time dimension of the test inputs (e.g., signals evolving over time) (Nejati et al. 2019; Menghi et al. 2020; Matinnejad et al. 2016; Birchler et al. 2023). This makes the use of brute force impractical for identifying scenarios where the system misbehaves or degrades its QoS. As a result, solutions proposed in this field have mainly focused on the application of meta-heuristic search (Gambi et al. 2019; Abdessalem et al. 2020; Menghi et al. 2020; Matinnejad et al. 2016; Calò et al. 2020; Huai et al. 2023; Klikovits et al. 2023; Zohdinasab et al. 2023; Blattner et al. 2024; Zhang et al. 2016).

Challenge 3 – QoS Validation under Complex Operational Constraints Validating the QoS of CPSs is challenging, i.e., given a test input, it is not evident how to determine what the expected system output should be Ayerdi et al. (2024, 2022); Zhang et al. (2024); Yang et al. (2025); Laurent et al. (2024). While many CPSs requirements can be formalized (e.g., in signal temporal logic), there are often components and functionalities, especially those related to the optimality of the outputs, that are challenging to be formalized. Examples involve elevators' dispatching algorithms (Ayerdi et al. 2024, 2022) (i.e., the industrial case

study from this paper), schedulers to allocate delivery robots to the different orders (Laurent et al. 2024), or paths optimality algorithms for autonomous vehicles (Yang et al. 2025).

Challenge 4 – Risk of generating invalid test inputs The generated test scenarios might be physically or practically incompatible with the software system of the CPS, which is incapable of handling them by construction, not because of a fault. Specifically, within the framework of elevator systems, scenarios may emerge where it is physically unfeasible for the system to maintain an adequate QoS due to physical limitations. For example, should a test generator aim to maximize the waiting time for a passenger, it might create a scenario where the passenger presses the call button immediately after the elevator departs from their floor, while the remaining elevators are occupied. The only viable solutions to ensure high QoS in such a situation may be to either increase the speed of the elevators or integrate additional elevators into the system. Conversely, our interest lies in identifying scenarios that are physically feasible for the system, i.e., where its optimal operation is entirely dependent on the software responsible for controlling the CPS.

To address the aforementioned challenges, we propose and apply a novel combination of techniques to generate test inputs that degrade the QoS of a system of elevators, which can help the software developer in identifying potential issues. To deal with the *first and second challenges*, we apply a delta-debugging algorithm (Valle et al. 2023) that reduces the test input while preserving a set of pre-defined properties.

To deal with the *second and third challenges*, we develop a search algorithm, inspired by DeepJanus Riccio and Tonella (2020), which identifies the boundary of system behavior by generating a set of test input pairs, in which each member of the pair exposes a distinct behavior of the system. Our test generation approach aims at maximizing the difference of QoS for the input pair, while minimizing the distance within each pair, i.e., by applying minimal input changes.

Lastly, we address the *fourth challenge* by implementing an automated misconfiguration repair algorithm that adjusts the parameters of the software system (Valle et al. 2023) based on the boundary input pairs.

We apply and evaluate our approach in an industrial case study system provided by ORONA[1], one of the largest elevators company worldwide. We found that the combination of multiple state-of-the-art methods permits generating challenging, yet valid test inputs. The identification of boundary values provides valuable feedback to elevator practitioners to either improve their software systems or adjust configuration parameters to withstand situations that were previously unforeseen. We also conducted an open-ended interview with domain experts that helped us identify a set of key lessons learned, which are applicable beyond our case study and are relevant for other CPSs.

In summary, this paper makes the following contributions:

Technique We propose a novel combination of state-of-the-art techniques to efficiently generate a set of effective test input pairs that characterize the behavioral boundary of the system, i.e., minimal test input differences resulting in huge test output differences. Our technique is a novel combination of delta debugging and search based generation of boundary states. First, up to now, the delta debugging algorithm has only been employed to reduce the failure inducing test input with the goal of aiding debuggers localize the root cause of a failure (Valle et al. 2023). In our application, we employ the delta debugging with a new purpose: reduce the search space to make the search-based boundary input generation both

[1] https://www.orona-group.com/es-es/

efficient and effective. Secondly, the search-based test input generator has been previously applied in the context of DNN testing, also known as the DeepJanus algorithm (Riccio and Tonella 2020). In contrast, our approach applies this to a new industrial context, with some adaptions that involve custom fitness function and mutation operators (see Section 3.2). Lastly, in our previous work (Valle et al. 2023), the repair component had as exclusive goal to repair CPSs' misconfigurations. In contrast, the goal in this paper is to ensure that the generated test inputs are valid and physically feasible, gaining trust in the generated test inputs; we had to make some changes in this component to address this goal.

Application and evaluation We apply and evaluate the proposed techniques in a real industrial CPS quantitatively as well as qualitatively. We ensure this way that the approach scales to a real-world industry-level system and that the approach is useful for the developers of this industrial application.

Lessons learned By engaging open-ended discussions with practitioners, we distilled a set of lessons that can be beneficial to other practitioners in different domains.

The rest of the paper is structured as follows: In Section 2, we offer a basic overview of the industrial case study system. Section 3 details our approach. Section 4 covers our empirical evaluation setup, including the experimental setup. Section 5 presents the analysis and discussion of the results. , analysis, and discussion of the results. In Sections 6 and 7, we share the threats to validity and lessons learned from this evaluation. We compare our approach with the current state-of-the-art in testing elevator systems and search-based boundary testing for CPS in Section 8. Finally, we conclude and discuss future research directions in Section 9.

2 Industrial Case Study System

Figure 1 shows an overview of our industrial case study system. Our system under test (SUT) is the dispatching algorithm of a system of elevators. This algorithm is in charge of deciding, for a set of calls, which should be their assigned elevators. This is carried out by sensing environmental data (e.g., the weight each elevator is lifting, position of each elevator) and one or more optimization criteria (e.g., reduce passengers' waiting times, energy consumption). This algorithm is highly configurable to adapt to the demands imposed by the singularities of each building where the system of elevators is deployed. Our industrial partner develops several dispatching algorithms, which constantly evolve to deal with different needs (e.g., include a new functionality, fix a bug).

Every time a change is implemented, a set of carefully selected test cases is executed. A test input is composed by (1) the building data, and (2) the passenger data stored in a *passenger file*. The former characterizes a specific instance of a system of elevators, and includes aspects like the number of elevators, number of floors, which floor is attended by each elevator, the speed of each elevator. This is usually based on the expected population to travel through the installation in that building and on real installations or installations described in related standards (Barney 2010), which usually remain static (i.e., their parameters are not changed). The latter involves a set of passengers traveling through the building. Specifically, the passenger data includes a set of N passengers, each having a total of 15 attributes, from which we selected the 5 most important ones as specified by ORONA'S engineers: 1) Arrival time (at), 2) Arrival floor (af), 3) Destination floor (df), 4) Mass (m) and 5) Capacity factor

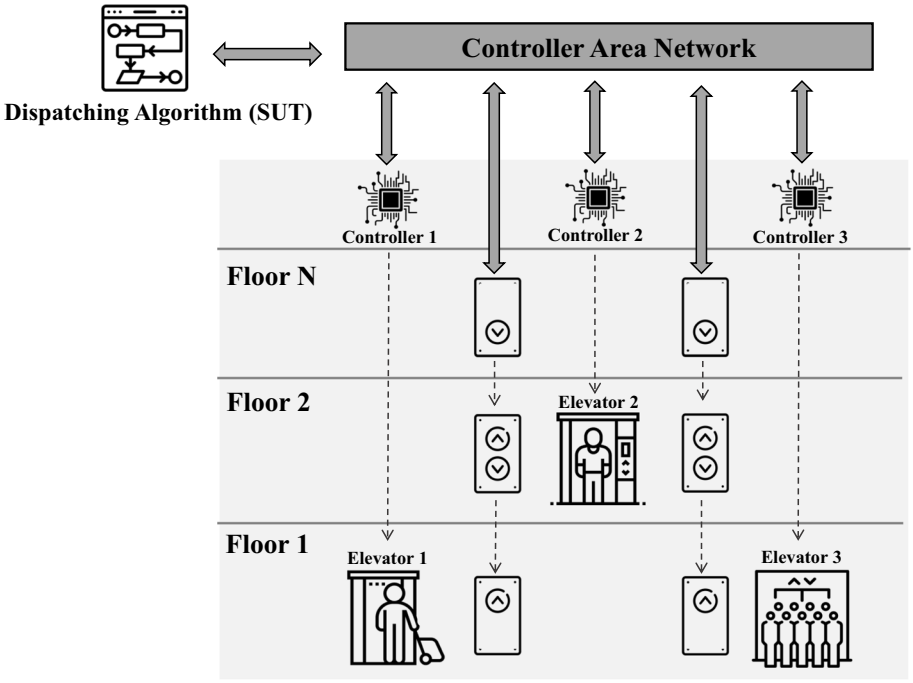


Fig. 1 Overview of our industrial case study

(cf). Table 1 shows an excerpt of a passenger file from a real building, including the values of these attributes.

For instance, the attribute *mass* reflects the weight of a passenger. The default value (recommended by related standards (Barney 2010)) employed for testing dispatching algorithms is 75 Kg.

Among the possible choices for test attribute values, our industrial partner prioritizes those encompassing real installations and extracted from operation, as they better represent real-world scenarios. These test inputs are executed by employing a domain-specific simulator, named Elevate[2]. When the simulation finishes, Elevate returns a CSV file with different data, used to obtain domain-specific QoS metrics that allow developers to assess whether a dispatching algorithm is giving adequate service to the passengers in the building. A typical metric, employed in this study, concerns the waiting time of passengers. This metric serves as an indicator of human's perception on whether a system of elevators is working properly or not. Other metrics can also be considered, such as the consumed energy, the queues in a floor or the number of times each elevator starts and stops its engines.

Previous studies have found that changing the attributes of a passenger file may significantly diminish the QoS of the system (Han et al. 2022, 2023). However, in those studies, all attributes from passenger profiles were changed. In contrast, our industrial partner is interested in finding cases in which a minimal change in the inputs provokes a significant downgrade in the QoS of the system, as this helps improving the testing and validation process of the dispatching algorithms.

[2] https://elevate.helpdocsonline.com/home

Table 1 Example of a test input with 15 passengers

ID	at	af	df	m	cf
p_1	51226 (14:13:46)	1	2	75	80
p_2	51230 (14:13:50)	2	3	85	66
p_3	51232 (14:13:52)	5	4	75	80
p_4	51233 (14:13:53)	7	4	75	80
p_5	51237 (14:13:57)	6	4	90	40
p_6	51249 (14:14:09)	8	7	75	80
p_7	51254 (14:14:14)	9	9	75	50
p_8	51260 (14:14:20)	2	12	60	80
p_9	51265 (14:14:25)	1	9	75	80
p_{10}	51271 (14:14:31)	1	12	75	80
p_{11}	51276 (14:14:36)	2	9	75	80
p_{12}	51283 (14:14:43)	3	7	66	20
p_{13}	51286 (14:14:46)	6	9	75	80
p_{14}	51294 (14:14:54)	5	6	87	66
p_{15}	51297 (14:14:57)	6	2	75	80

For instance, instead of changing the capacity factor of all passengers in a uniformly random way, as carried out in related studies (Han et al. 2022, 2023), it is more interesting to to find a single passenger whose capacity factor differs by a small amount in two passenger files, which is found to be enough to have a drastic impact on the system's performance. Given this need, the problem we address in this paper can be formulated as follows:

Problem Given an initial test input f_0, i.e., a passenger file, find a set of input pairs $IP = \{ip_1, ip_2, ..., ip_n\}$ exposing boundary values for the system under test: each input pair $ip_i = \langle f_{i_1}, f_{i_2} \rangle$ consists of two passenger files f_{i_1}, f_{i_2} with the same number of passengers that are close to each other and that, upon system execution, produce diverging test outputs, i.e., test outputs associated with a diverging QoS metric M_{QoS}. More formally, the goal is to maximize the difference between the QoS metrics corresponding to the two members of an input pair, i.e., $max(|M_{QoS_{i_1}} - M_{QoS_{i_2}}|)$, while minimizing the changes within input pairs (i.e., $min(dist(f_{i_1}, f_{i_2}))$). At the same time, we aim to achieve a diverse set of results by maximizing the distance among all input pairs in IP.

3 LiftJanus

LiftJanus identifies boundary values of the SUT, i.e., slight changes in the test input which discriminate between notably different QoS. As shown in Fig. 2, LiftJanus is a novel combination of three state-of-the-art techniques, specifically adapted to the vertical transport domain. LiftJanus consists of three main components, i.e., test minimization, test generation and system repair, each corresponding to one of the considered techniques.

1) The **test input minimization** component focuses on reducing the size of the test input, to increase the efficiency of test generation, by reducing both the search space and the simulation time.

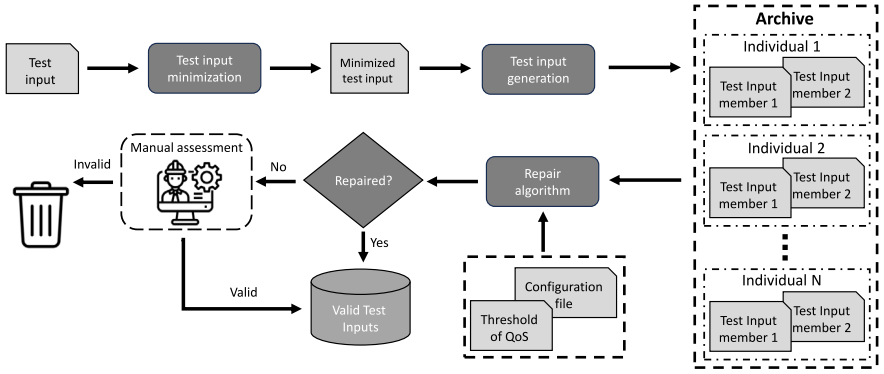


Fig. 2 Overview of LIFTJANUS

2) The **test generation** component aims to detect boundary values by generating multiple test input pairs, whose members are similar but produce very different performance metrics for the system under test. We aim at generating two passenger files within an individual that are as similar as possible, to ensure that any observed performance differences are likely due to the SUT rather than the test inputs. We address two key objectives: 1) ensuring high similarity within individuals (i.e., the test inputs in a pair) and 2) maximizing diversity between individuals (i.e., new test input pairs w.r.t. those in the archive). We measure the similarity between two members using the Hamming distance. To compute the distance between two individuals I_1 and I_2, we follow the approach from prior work (Riccio and Tonella 2020). Specifically, the distance is defined as the minimum of the two averaged Hamming distances $(Hamming(I_1.m_1,I_2.m_1) + Hamming(I_1.m_2,I_2.m_2))/2$ *and* $(Hamming(I_1.m_1,I_2.m_2) + Hamming(I_1.m_2,I_2.m_1))/2$.
3) The **system repair component** aims to generate new system configurations that enhance the QoS of the system for each test input pair when the system is exercised with such input pairs. The purpose of the repair component in this paper differs from that of our previous study (Valle et al. 2023), where we focused on finding a configuration that addresses QoS failures for individual test inputs while ensuring similar QoS outcomes across the rest of the search space. In contrast, this paper's repair component aims at finding a configuration that fixes the QoS results for each pair of generated test inputs. If the repair component is capable of finding such a configuration, both test inputs will perform according to the QoS requirements. With this configuration we aim at providing engineers with clear insights that assist them in identifying the root cause for the poor performance triggered by each boundary input pair. However, the use of our repair suggestions to find an overall configuration that solves all problematic cases, while ensuring similar QoS outcomes across the rest of the input space (i.e., ensuring no regression), remains instead a manual work delegated to developers.

3.1 Test Input Minimization Component

Test inputs for testing dispatching algorithms typically encompass thousands of passengers traveling through a building over an entire day. The large size of test input impacts simulation time, thus negatively affecting the efficiency of test generation. To address this issue, our test input minimization component reduces the original test input, while preserving system

behavior at interesting time windows. This component is based on an adaptation of the Delta Debugging algorithm (Zeller and Hildebrandt 2002) for elevator dispatching algorithms, which was originally proposed by Valle et al. (2023). The Delta Debugging algorithm in our context iteratively reduces the test input, while at the same time assessing whether the QoS remains similar to that of the original test input. If so, the test input is reduced again, otherwise, some passengers that have been removed in the previous reduction must be added not to alter the QoS requirements. This process is repeated until the test input cannot be reduced further without altering the QoS requirements.

In our work, we used the Environment-wise Delta Debugging algorithm since it has been identified as one of the most efficient and effective minimization algorithms among those proposed by Valle et al. (2023). This algorithm considers the so-called "*static states*" of the system. These states refer to stable situations of the SUT and the environment. Its two-stage search process works as follows. It first starts by minimizing the test input considering the static states of the system. In a second stage, the test input is further reduced following the same procedure as the traditional delta debugging algorithm. Thus, the algorithm is able to speed-up the minimization process compared to its more basic versions. For a more detailed explanation of this algorithm, refer to Appendix A.

The Delta debugging implementation takes as input the original test input and a performance metric, along with a predefined minimization threshold. The minimization threshold ensures that the QoS metrics remain similar for the minimized test input. As performance metric we employed the Longest Waiting Time (LWT) which measures the longest waiting time experienced by the passengers in the test input, and is a widely used metric to assess elevators' performance by our industrial partner. The resulting minimized test input file is then provided as an input to the test generation component, allowing for a smaller search space and decreased simulation execution time.

3.2 Test Generation Component

The test generation component aims at exploring the behavioral boundary, consisting of input pairs (individuals), whose members are similar, but with notably different QoS values. Unlike focusing on the boundary of the input space, we target the boundary between two test input members, which provide engineers with several insights and advantages. First, it allows us to deal with the challenging QoS Validation under Complex Operational Constraints, as we aim at finding extreme disparities for minor changes in input attributes (e.g., a minor change in the passengers' weight that leads to a huge LWT difference). Second, it enables a high explainability, as it helps engineers find the explicit causes that lead to a significant system degradation easily. While in previous work (Han et al. 2022) if the mass of passengers is selected, the mass of all passengers is changed, LIFTJANUS instead focuses on making minimal changes. On the other hand, our approach also focuses on enhancing the diversity of inputs. That is, we try to find minimal changes among two members, but for different members, this way trying to increase the diversity of the exposed failures.

We designed this component by adapting the DeepJanus algorithm (Riccio and Tonella 2020) to the unique characteristics of elevator systems. DeepJanus uses an original multi-objective search algorithm that combines NSGA-II (Deb et al. 2002) with novelty search (Lehman and Stanley 2011), featuring an archive of solutions and a repopulation operator to avoid stagnation. DeepJanus was originally applied to image classifiers and Lane Keeping Assist Systems for autonomous vehicles, but it has been extended to address other critical systems, such as eye gaze predictors (Riccio et al. 2021). It was used as baseline in the

empirical assessment of state-of-the-art test generators (Riccio and Tonella 2023; Zohdinasab et al. 2023; Fahmy et al. 2023).

In the following, we focus on four key aspects of LIFTJANUS that required us to adapt DeepJanus for our objectives: (1) fitness functions, (2) mutation operator, (3) initialization of the population, and (4) archive of solutions.

3.2.1 Fitness Functions

LIFTJANUS optimizes two fitness functions. The first one rewards the quality of an individual, by maximizing diversity to other solutions and similarity between the members of the pair. The second one promotes the differential behavior exhibited by the members of the pair when exercising the elevator system, by maximizing the difference in overall QoS.

Quality of an individual The quality of an individual is measured as a combination of two metrics, i.e., (1) the distance between the members of an individual and (2) the sparseness of an individual with respect to the individuals in the archive, i.e., the best solutions found during the search. As described by Riccio and Tonella (2020), both of these metrics rely on a distance function $dist$, which is problem- and domain-specific. In our case, we adopt the Hamming distance (Hamming 1986) to calculate the distance between two test inputs. In particular, the Hamming distance measures the distance between two members in terms of passengers' attribute values. It ranges between 0 and 1, where 0 indicates that the test inputs are identical and 1 is the maximum difference they can have. Our approach avoids the addition or removal of passengers during the mutation process. Based on preliminary experiments, we observed that adding/removing passengers greatly facilitates the algorithm to induce large LWT differences. However, the resulting input pairs are more difficult to debug and increase the chance of having false positives, as the situation between a member and another one could be extremely different, potentially resulting in unrealistic passenger profiles. Instead, our industrial partners were interested in how minimal changes to passenger attributes produce large LWT differences. This research is aligned with prior studies (Han et al. 2022). Without mutation operators that alter the number of passengers, our approach allows for practical failure debugging and reduces the probability of false positives.

Given two test inputs A and B that contain the same number of passengers, with two corresponding passengers occupying the same row in the two files, we compute the Hamming distance between A and B as follows:

$$d_H(A, B) = \frac{\sum_{i=1}^{Pass} \delta(a_i, b_i)}{Pass} \tag{1}$$

where $Pass$ is the number of passengers. The function $\delta(a_i, b_i)$, further explained in (2), is the distance between the values of the input attributes of two corresponding passengers $\langle a_i, b_i \rangle$, defined as:

$$\delta(a_i, b_i) = \frac{\sum_{j=1}^{Attrs} \frac{|a_{ij} - b_{ij}|}{maxVal_j - minVal_j}}{Attrs} \tag{2}$$

Equation (2) calculates the Hamming distance between two passengers, i.e., the i-th passenger of each test input. Each passenger is represented by a number of attributes ($Attrs$) from which we selected 5 as described in Section 2.

Hence, we compute the difference for each attribute (i.e., j-th attribute), normalized by the value range of that attribute ($maxVal_j$ and $minVal_j$). In practice, these ranges were

manually set based on the constraints of the problem, the specific characteristics of each of the installations (for instance, the number of floors is specific for each installation) and previous works (Han et al. 2023).

Differential behavior Unlike previous case study systems addressed by DeepJanus, for elevator systems we cannot leverage expected behaviors (as there is no ground-truth behavior) or acceptable performance thresholds to identify the behavioral boundary. For this reason, our search strategy encourages differential behavior between the two members (test inputs) of the same pair, by maximizing the difference between the LWT values achieved by such members.

3.2.2 Mutation

The mutation genetic operator explores new regions of the search space by manipulating the individuals of the current population. We designed a novel mutation operator that perturbs one or more attributes of one or more passengers. The operator takes one member from each individual in the population and applies a mutation to it, resulting in a mutated version as output. This procedure is carried out for each individual in the population by randomly selecting the member to be mutated.

Algorithm 1 shows the mutation process we used in our approach. The mutation operator first randomly selects a passenger from the input (Algorithm 1 Line 4). Then, it selects an attribute of that passenger (Algorithm 1 Line 7) and mutates it by assigning a random value within its numerical limits (Algorithm 1 Line 8). Each mutation performed by our test generator is grounded in practical considerations, as the mutation ranges are carefully selected to reflect real-world scenarios. For instance, a passenger's weight is not randomly altered to extreme values (e.g. 350 Kg), but instead is constrained within a reasonable range depending on the operational domain, such as 60 to 90 kg, consistent with other works in the same field (Han et al. 2023). These ranges were discussed with domain experts prior to be implemented and are easily configurable. For example, in Asian countries, where the average passenger weight is lower than in Europe, practitioners can adjust the attribute ranges accordingly. After each attribute perturbation, the operator decides whether another attribute of the same passenger should be mutated with probability 2^{-N}, N being the number of attributes mutated so far for that passenger: as the number of mutated attributes increases, the probability to mutate a new attribute decreases exponentially. This mutation operator enhances exploration and improves the overall efficiency of the algorithm, while ensuring that the majority of mutations are subtle, similarly to previous studies (Valle et al. 2023; Abdessalem et al. 2020). Our approach always ensures that at least one mutation is performed (i.e., initial $p = 0.5^0 = 1$).

Likewise, after finishing the mutation of a passenger from the input, the operator decides whether to mutate another passenger or not (Algorithm 1 Line 16) with probability 2^{-M}, M being the number of already mutated passengers.

3.2.3 Initial Population

LIFTJANUS aims to create a diverse initial population consisting of individuals (input pairs). In our context, there is scarcity of realistic inputs that can be used as a starting point for the search. In fact, we rely on a single representative and large test input that is minimized by our test minimization component. To address this limitation, we implemented a seed

Algorithm 1 Test Input Mutation algorithm.

Input: testInput = $\{p_1, p_2, ..., p_{np}\}$ *//Test Input*
threshold *//Attribute thresholds*
Output: mutTestInput = $\{p'_1, p'_2, ..., p'_{np}\}$ *//Mutated Test Input*

```
numOfMutPass ← 0
mutTestInput ← testInput
do
    passToMutate ← randSelect(mutTestInput)
    numOfMutAtt ← 0
    do
        attToMutate ← randSelect(passToMutate)
        passToMutate ← mutate(passToMutate, attToMutate, threshold)
        numOfMutAtt ← numOfMutAtt +1
        pp ← rand()
    while pp < 0.5^numOfMutAtt;
    mutTestInput ← update(mutTestInput, passToMutate)
    numOfMutPass ← numOfMutPass +1
    p ← rand()
while p < 0.5^numOfMutPass;
return mutTestInput
```

generation strategy that generates a predefined number of diverse seeds starting from a single minimized test input. Our seed generator iteratively produces an initial population by applying the mutation operator to the original test input multiple times until the minimum Hamming distance between the new member and the already generated seeds is greater than 0.0001. This threshold was chosen to balance the trade-off between seed diversity and execution time during seed initialization. In fact, we observed that increasing the threshold improves diversity among seeds but also significantly increases the time required to identify diverse seeds. As the number of seeds in the initialization archive increases, the number of mutations required to get a seed that surpasses the Hamming distance threshold increases too. The same happens with the Hamming distance threshold, i.e., the higher the threshold, the harder it is to find a seed whose Hamming distance from all existing seeds exceeds the threshold, increasing the execution time. Conversely, a lower threshold could negatively affect diversity and may lead to convergence on similar types of failure-inducing inputs. We defined this threshold through preliminary experiments to evaluate the balance between diversity and execution time. Further information about the preliminary experiments can be found in Appenix B

Then, similar to the original DeepJanus algorithm (Riccio and Tonella 2020), the two members of each individual are obtained by copying a seed, which becomes the first member of the pair, and by applying the mutation operator to the first member, to get the second one.

3.2.4 Archive of Solutions

The archive of solutions maintains the best individuals encountered during the search. At the end of the last search iteration, it contains the final solutions.

In contrast to the original DeepJanus implementation (Riccio and Tonella 2020), which relies on an unbounded archive managed through an empirically defined threshold, we use a fixed-size archive of 20 solutions. In fact, performing costly simulations to identify the best distance threshold is unfeasible in our case, while our industrial partner could easily specify the desired number of solutions to be inspected by their engineers. When the archive is not

full, all candidates in the current population, i.e., individuals with highest LWT difference and lowest Hamming distance between members, are included into the archive until the archive is full. If the archive is full, a new candidate input must locally compete with its nearest neighbor in the archive based on their fitness. If a candidate individual exhibits better scores in the differential behavior fitness compared to the individual in the archive, LIFTJANUS replaces the latter with the former.

3.3 System Repair Component

Manual system repair conducted by domain experts can be costly due to the huge configuration space of elevator systems. The boundary values identified by the test generation component can identify failures due to misconfigurations of the elevator system. LIFTJANUS incorporates an automated system repair component based on the misconfiguration repair algorithm proposed by Valle et al. (2023). In this work, we adapted the algorithm to address input pairs belonging to the behavioral boundary.

The dispatching algorithm is highly configurable to deal with the demands of different types of installations and traffic patterns (Valle et al. 2023). Its configuration is handled by an XML file (i.e., the configuration file) that encompasses several configurable parameters. The system repair component takes as input the archive of solutions produced by the test generation component, the original configuration file, and the threshold for the QoS of that installation. The system repair component selects the N most critical individuals from the archive, i.e., those with the largest difference in LWT between their members. The value of N is selected by the test engineers based on the allocated budget for the repair process. Then, the repair algorithm searches a better system configuration for each of the given individuals Depending on the specific algorithm, there can be between 40 and 80 parameters, whose types may be boolean, double, or integer, each with defined minimum and maximum values. The repair component has access to these parameters and aims at searching for a repaired solution, i.e., a configuration for which the LWT for both members of the individual is below a certain threshold. The repair process is performed by changing the aforementioned parameters of the dispatching algorithm until the QoS metric LWT for both members of the individual is below a certain threshold which is installation-specific. Therefore, as output, we get one new configuration per individual, N in total. Let's have an individual whose members scored a LWT of 200 for member 1 and 300 for member 2. If the QoS threshold for LWT of that installation is 250, the repair algorithm tries to find a configuration in which the QoS of both members is below 250, i.e., reduce the QoS of member 2 while maintaining the same QoS score for member 1.

Notably, any configuration that improves the QoS of one member while worsening the QoS of the other is discarded as suboptimal. The termination criteria (Valle et al. 2023) include either (1) fixing the misconfiguration or (2) exceeding the maximum running time.

If the system repair component fails to find a better configuration within the timeout, the members of the individual undergo manual evaluation by an engineer from our industrial partner. If no better system configuration is found even by the engineer, the individual is discarded as likely invalid (possibly because it is physically unfeasible to improve the QoS of the worse member). Valid tests are retained, along with the improved configuration proposed either by the repair component or by the engineer. This final set of tests and improved configurations is given to the development and validation team for analyzing the possible bugs and misconfigurations of the current dispatching algorithm.

4 Empirical Evaluation

4.1 Goal and Research Questions

In our study, we evaluate the efficiency and effectiveness of LIFTJANUS with an industrial traffic dispatching algorithm. Moreover, we assess the usefulness of our solution with domain experts. Therefore, we answer the following research questions (RQs):

RQ1 - Fitness guidance *Does the fitness guidance improve the effectiveness of* LIFTJANUS? LIFTJANUS includes a search-based test generator specifically designed for identifying the behavioral boundary of elevators' dispatching algorithms. In this RQ, we assess whether the guidance provided by our fitness-based genetic operators actually helps in generating input pairs whose members are similar but result in notable differences in the selected QoS, i.e., LWT.

RQ2 - Test input *Can* LIFTJANUS *generate valid test inputs pairs?*
Automatically generated tests are valid to the developers if they represent physically feasible scenarios and provide a large QoS degradation with minimal changes. In this RQ, we assess the ratio of valid test inputs among the automatically generated tests, where an input is deemed valid if LIFTJANUS is able to propose a new configuration that repairs the QoS requirement for the test input by restoring the QoS of the input pair.

RQ3 - Minimization component influence *How does the test input minimization component affect the efficiency and effectiveness of* LIFTJANUS?
The introduction of test input minimization before test generation promises higher effectiveness due to the reduced search space, despite the overhead it introduces. In this RQ, we investigate the impact of the test input minimization component on the overall test generation process through an ablation study.

RQ4 - Qualitative analysis *To what extent do domain experts perceive the generated test inputs pairs as non trivial to find and useful?*
Effectively evaluating the usefulness of LIFTJANUS requires domain experts in the loop. Through a semi-structured interview, engineers from ORONA evaluated the outputs of our tool in this RQ.

4.2 System Under Test and Original Test Inputs

We considered ORONA'S most common traffic dispatching algorithm (Barney and Al-Sharif 2015), which is a well-known algorithm in the domain. To assess our approach, we used two real installations with ORONA'S elevators. The first installation encompasses a total of 3 elevators traveling along 12 floors, whereas the second one has 6 elevators operating on 10 floors. We selected these installations to ensure that our evaluation is conducted in diverse and realistic settings, as ORONA regularly uses them to test their dispatching algorithms. Moreover, ORONA provided us with real operational data from those installations, which allowed us to assess our approach with real passenger data. Table 2 summarizes the key characteristics of the used installations and test inputs. As depicted in Fig. 3, both installations have different passenger flows. While the test input for Installation 1 lasts about 55,000

Table 2 Characteristics of the considered installations and original test inputs

Test	# of elevators	# of floors	# of passengers	Execution Time in speed-up simulation (s)
Installation 1	3	12	3,769	159.16
Installation 2	6	10	6,558	221.46

seconds for 3,769 passengers, the test input for Installation 2 lasts about 31,000 seconds for 6,558 passengers (i.e., Installation 2 faces a higher passenger density).

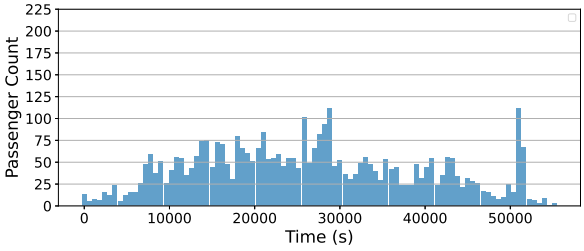
4.3 Evaluation Metrics

To assess our approach, we used the following evaluation metrics:

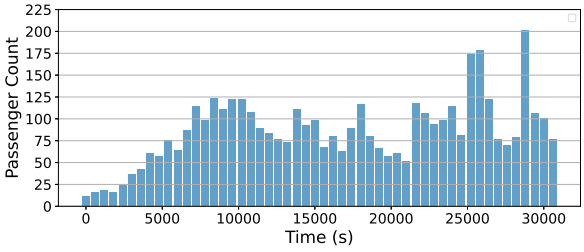
LWT difference (LWT_{Δ}) We measure the difference in the LWT between two members of each individual in the archive of solutions (i.e., the difference in the performance of the SUT according to the LWT QoS metric). The larger the LWT_{Δ}, the larger the difference in the performance of the SUT when exposed to similar inputs. We used LWT as QoS performance metric as it is one of the most commonly employed metrics to assess the quality of dispatching algorithms in the considered domain (Barney 2010).

Hamming distance Boundary individuals should be characterized by a small difference between their members. The lower this metric is, the more similar the input pair members are.

Test input validity rate The test input validity rate represents the proportion of new, improved configurations found by the repair component. A higher validity rate indicates



(a) Installation 1



(b) Installation 2

Fig. 3 Passenger flows for each Installation

a greater ability of our approach to generate valid test inputs, which are possibly useful for the engineers to improve the system.

LWT_Δ reduction ratio This metric measures the reduction ratio of LWT_Δ between the execution of a test input pair with the new configuration and the execution of the same test input pair with the original configuration. A higher LWT_Δ reduction ratio indicates a better performance of the system with the proposed new configuration.

Algorithm execution time We evaluate the efficiency of LiftJanus by measuring the time it requires to generate test inputs: lower execution times indicate higher efficiency. We relate this metric to the use of the minimization component (ablation study).

As reported in Table 3 we used five different evaluation metrics to evaluate our approach. For the first RQ, we used the LWT_Δ and the Hamming distance. With these metrics we wanted to assess the effectiveness of LiftJanus by measuring whether the generated test inputs effectively identified the behavioral boundary of the system.

For RQ2, we used the test input validity rate, LWT_Δ Reduction Ratio aiming to measure the ability of LiftJanus to generate valid individuals that are relevant for the engineers from Orona.

For RQ3, we used LWT_Δ and the algorithm execution time to assess the effect of the test input minimization component on the effectiveness and efficiency of LiftJanus. An effective minimization of the test input should result in a larger LWT_Δ and a shorter algorithm execution time. Differently from RQ1, we do not consider the Hamming distance as effectiveness indicator in RQ3, because input minimization makes the Hamming distance values non comparable between minimized and non-minimized test inputs, as they have a different number of passengers.

For RQ4, we did not use quantitative metrics, whereas we conducted a qualitative analysis through a semi-structured interview with experts from the domain as explained in Section 4.8.

4.4 Algorithm Setup

The three algorithms used in our approach were configured as follows:

Test input minimization algorithm Among the configurations proposed by Valle et al. (2023), we selected the best performing configuration from a previous study (Valle et al. 2023). In particular, our reduction targeted the LWT metric with a 10% threshold. Despite this threshold, in both of the employed installations, the LWT between the original and minimized test input was identical.

Table 3 Overview of the used evaluation metrics per RQ

	RQ1	RQ2	RQ3	RQ4
LWT_Δ	+	–	+	–
Hamming distance	+	–	–	–
Test input validity rate	–	+	–	–
LWT_Δ Reduction Ratio	–	+	–	–
Algorithm execution time	–	–	+	–

Test input generation algorithm We configured the population size to 8, the number of generations to 100, and the archive size to 20. These relatively small values are consistent with those used by Riccio and Tonella (2020) in scenarios involving resource-intensive simulation-based input evaluations.

System repair algorithm Our configuration is based on that proposed by Valle et al. (2023). We carried out two changes: (1) based on preliminary experiments, we reduced the maximum repair time to 4 hours; (2) we limited the repair archive size to 1 solution, as we have a single repair objective (i.e., LWT), and unlike the previous work (Valle et al. 2023), we do not need an archive of Pareto-optimal solutions.

4.5 Ablation Baselines

To assess the effectiveness of LIFTJANUS in RQ1, we implemented BASEJANUS, an unguided version of the test input generator component. Being unguided refers to the absence of a selective mechanism when choosing the next population. Unlike LIFTJANUS, which applies a guiding strategy (i.e., a tournament selection based on Pareto dominance), BASEJANUS retains all the offspring individuals in the population for the next iteration, without any filtering or prioritization.

In addition, we also used the approach UNCERROBUA proposed by Han et al. (2023) as an additional baseline. This technique generates test inputs by mutating a specific attribute of all the passengers in the original test inputs by following advanced statistical techniques. We adapted the configurations of their algorithms to make the comparison with our approach fair. To this end, we generated the same number of test inputs as generated by LIFTJANUS (i.e., 8 generations $\times$ 100 iterations, resulting in a total of 800 test inputs). To evaluate the distance between pairs, we measured the distance relative to the original test input, since there are no pairs in this approach. To compare the outcomes of both approaches, we followed the same archive criteria as LIFTJANUS, maintaining an archive of the 20 best solutions, updated through tournament selection based on Pareto dominance. This way, we ensure to get the best 20 solutions out of the generated 800. Based on the results of the original study (Han et al. 2023), we used the best three configurations according to their evaluation, i.e., *Mass (C1), Capacity factor (C2)* and their interaction (*C3*)

In RQ3, we ran LIFTJANUS without the test input minimization component to evaluate its influence in both the effectiveness and efficiency of our approach.

4.6 Experimental Runs and Statistical Tests

Due to the stochastic nature of the considered algorithms, we executed them multiple times within our time constraints.

For RQ1, we executed 8 runs of each component of LIFTJANUS for each test installation. Each run had a budget of 100 iterations, resulting in approximately 7 hours of simulation time per run for Installation 1 and 8 hours for Installation 2. We also executed 8 runs for each configuration of UNCERROBUA for each test installation. Each run had a budget of 800 generations, resulting in approximately 3.5 hours of simulation time per run for Installation 1 and 4 hours for Installation 2. In total, we needed 420 hours for the first RQ (8 (runs) $\times$ 2 (BASEJANUS and test input generation algorithms of LIFTJANUS) $\times$ 7 hours for Installation

1 + 8 × 2 × 8 hours for Installation 2 + 8 × 3 (UNCERROBUA configurations) × 3.5 hours for Installation 1 + 8 × 3 × 4 hours for Installation 2).

For the system repair component, we selected the 5 most significant individuals (i.e., the ones with the largest LWT_{Δ}) from each archive produced by the test input generation component and applied the system repair component with a time budget of 4 hours per pair. In total, we required around 320 hours for RQ2 (5 (individuals) × 2 (installations) × 8 (runs) × 4 hours (time budget)).

For RQ3, we ran the test input generation component without the minimization component 4 times for each installation. A lower number of runs was employed because in RQ3, we did not use the test input minimization procedure and the execution time of the generation component was significantly higher (around 70 hours for Installation 1 and 78 hours for Installation 2). The total execution time for this RQ was around 592 hours, i.e., $4 \times 70 + 4 \times 78$.

In summary, our evaluations required a cumulative execution time of around 1,332 hours, precluding the execution of many runs. Notably, parallelizing these experiments was not feasible due to the requirement of a simulator license for each execution instance on personal computers.

In RQ1 and RQ3, we assessed the statistical significance of the difference between the results achieved by our test input generation component and the baselines. We first analyzed the data distribution using the Shapiro-Wilk test. Since the data was normally distributed, we employed the Anova test. A p-value below 0.05 was considered indicative of a significant distinction between the techniques. Additionally, we evaluated effect sizes through the Vargha and Delaney's $\hat{A}_{12}$ value and the Cohen's d-value (Cohen 1987). According to Romano et al. (2006), the effect size of the $\hat{A}_{12}$ value can be categorized as *negligible* if $d < 0.147$, *small* if $d < 0.33$, *medium* if $d < 0.474$ and *large* if $d \geq 0.474$, where $d = 2|\hat{A}_{12} - 0.5|$. Regarding the effect sizes of Cohen's d value, according to Sawilowsky (2009), the effect size can be categorized as *negligible* if $d < 0.01$, *very small* if $d < 0.2$, *small* if $d < 0.50$, *medium* if $d < 0.80$, *large* if $d < 1.2$, *very large* if $d < 2$ and *huge* if $d \geq 2$.

4.7 Execution Environment

The experiments were conducted using a PC with a Windows 10 operating system, with a dual-core CPU Intel Core i5 7th generation, and 16 GB RAM. We used Elevate 8.19 as a simulator for executing the tests.

4.8 Qualitative Analysis

For RQ4, we conducted a human-in-the-loop qualitative analysis through semi-structured interviews to ORONA'S engineers. During these interviews, the experts analyzed elevator configuration files and the corresponding pairs of test inputs generated by LIFTJANUS. In particular, we conducted an interview with two domain experts specialized in the development of traffic dispatching algorithms. The experience in the domain of the two engineers was 15 and 6 years.

The experts received a replication package containing a tool-repaired situation (i.e., test input pair and repaired configuration) and a tool-unrepaired situation (i.e., test input pair for which the repair component failed to find a better system configuration), automatically generated by LIFTJANUS. Conducting a qualitative analysis and interviews with engineers is onerous and time-consuming. We therefore decided to limit the qualitative analysis to test inputs generated by LiftJanus, and we prepared questionnaires with a reasonable number of

examples to evaluate. While it might be interesting to also conduct a human study evaluation of the test inputs generated by the considered baselines, the goal of our evaluation was to assess the developers' perception of the pairs of inputs produced by LiftJanus, which are characterized by minimal variations, but result in a large LWT difference.

The engineers were asked to simulate, analyze, and manipulate the test inputs, and to propose better configurations at their convenience using the simulator. The interview was guided through a Google Forms questionnaire with a total of 14 questions: 7 questions for the tool-repaired situation, 3 questions for the tool-unrepaired situation and 4 questions related to the usefulness of LIFTJANUS in general.

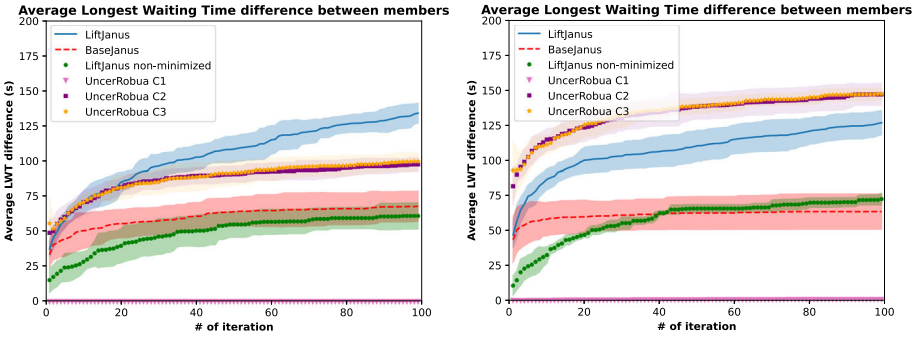
Specifically, the interviews aimed at assessing the following aspects:

- The ease with which participants would perform the same task as LIFTJANUS, i.e., identifying similar inputs exhibiting significant differences in LWT, to assess whether such inputs are trivial to find or not;
- Their ability to explain performance differences when presented with a pair of input files;
- Whether they could suggest configurations to enhance the SUT performance, while adhering to QoS requirements;
- Determining if test inputs not automatically repairable represent scenarios beyond the capability of the dispatching algorithm;
- Assessing the usefulness of LIFTJANUS in the daily tasks of dispatching algorithm developers.

5 Experimental Results and Discussion

5.1 RQ1 - Fitness Guidance

Figure 4 shows the obtained LWT_{Δ} by the individuals of the archive over the iterations of LIFTJANUS and the selected baselines. The graphs depict the Average LWT_{Δ} across all individuals in the archive. In both Installations, LIFTJANUS (with minimization) clearly outperformed BASEJANUS. From both graphs, an interesting observation is that, even at the last iterations, the Average LWT_{Δ} of the guided approach seems not to converge, with the LWT_{Δ} steadily increasing throughout the entire search process. Conversely, particularly in



(a) Installation 1 (b) Installation 2

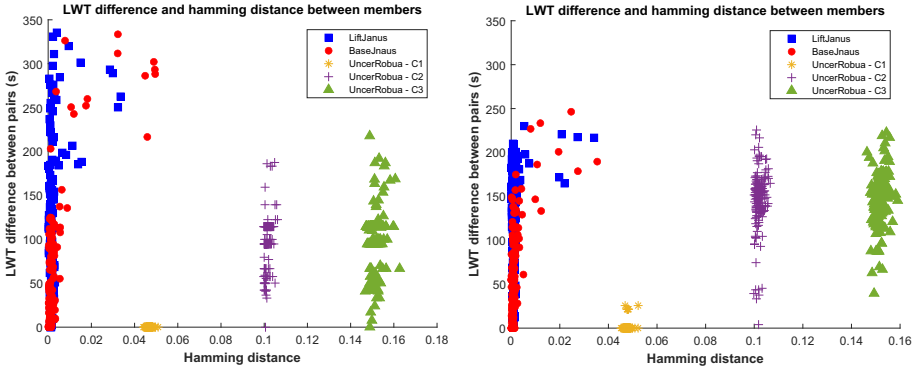
Fig. 4 Average LWT_{Δ} between members per iteration of LIFTJANUS and BASEJANUS (both with minimization), LIFTJANUS without minimization component and UNCERROBUA approaches

the second installation, BASEJANUS appears to converge after approximately 20 iterations. When compared to UNCERROBUA, we can clearly see that LIFTJANUS outperformed it for *C1* in both Installations. To be more precise, UNCERROBUA-*C1* barely increased the LWT_{Δ} in both Installations. However, when comparing LIFTJANUS against UNCERROBUA *C2* and *C3*, it can be observed that it clearly outperformed for Installation 1 after the 40th iteration, where UNCERROBUA converged. However, for Installation 2, UNCERROBUA *C2* and *C3* outperformed LIFTJANUS when considering the Average LWT_{Δ} across all individuals in the archive. This can be due to the passenger profile characteristics. As depicted in Fig. 3 the passenger flow for Installation 2 is more dense than the one for Installation 1, i.e., there are more passengers in a shorter period of time. This makes it easier to degrade the system QoS when altering the attributes of all passengers in the test input. The system is especially sensitive when changing the capacity factor of all passengers (i.e., the attribute that makes a passenger enter or not depending on how many people are already in the cabin attending the passenger) in a high density profile, as there will be a large portion of passengers that will not enter the elevator due to reducing this attribute.

The scatter plots shown in Fig. 5 show how the similarity between the members of the input pairs (in terms of Hamming distance) affects the LWT_{Δ} for LIFTJANUS, BASEJANUS and UNCERROBUA approaches. Figure 6 zooms this scatter plot to better differentiate the results from LIFTJANUS and BASEJANUS. In the case of LIFTJANUS, even minimal changes resulted in a high LWT_{Δ}. For instance, in both installations, some members with very low Hamming distance exhibited LWT_{Δ} of nearly 200 seconds. As expected, in general, if the distance between members increased, the LWT_{Δ} also increased. For instance, in three cases of the first installation, a Hamming distance greater than 0.002 led to an LWT_{Δ} of 300 seconds, a significant difference according to domain experts.

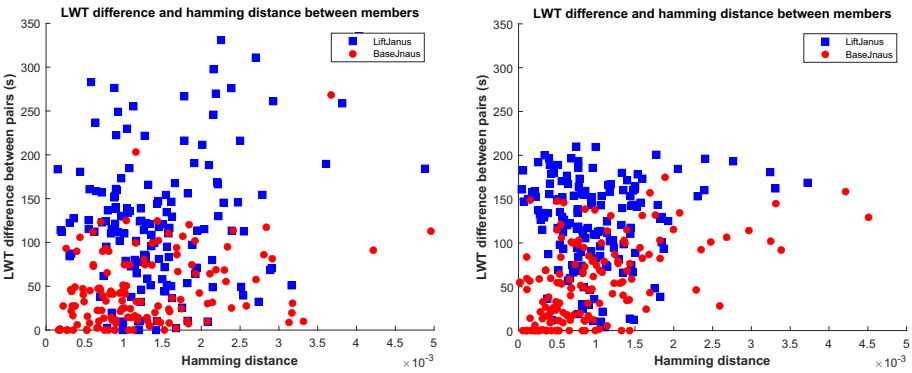
In contrast, the Hamming distance for the three configurations of UNCERROBUA were notably higher compared to those for BASEJANUS and LIFTJANUS. Within UNCERROBUA, configurations *C1* and *C2* obtained lower Hamming distances than *C3*. This was expected because *C3* changed both the mass and the capacity factor of the passengers in the test inputs. However, LIFTJANUS consistently outperformed UNCERROBUA across all configurations, achieving comparable LWT_{Δ} results but at approximately 80 times lower Hamming distances.

Table 4 summarizes the results of the statistical tests used to compare the effectiveness of LIFTJANUS with the the selected baselines. When comparing LIFTJANUS with BASEJANUS



(a) Installation 1 (b) Installation 2

Fig. 5 Scatter plot of LWT_{Δ} vs pairwise Hamming distance for LIFTJANUS, BASEJANUS and UNCERROBUA approaches



(a) Installation 1 (b) Installation 2

Fig. 6 Scatter plot of LWT_{Δ} vs pairwise Hamming distance for LIFTJANUS and BASEJANUS approaches

and UNCERROBUA *C1* for the LWT_{Δ} and Area Under Curve (AUC - LWT_{Δ}) metrics, there was statistical significance in favor of LIFTJANUS, with large effect sizes in both installations. There was statistical significance also when comparing LIFTJANUS with UNCERROBUA *C1* and *C2* for Installation 2, where these two configurations outperformed LIFTJANUS with large effect sizes. As explained before, this was due to the high effect of reducing the capacity factor of many passengers in a scenario of high passengers density.

When considering the other metric, i.e., the Hamming distance, LIFTJANUS outperformed all configurations of UNCERROBUA in both installations. This was expected as LIFTJANUS aims at making minimal changes on a few passengers' attributes, whereas UNCERROBUA makes changes in one or two attributes of all passengers. LIFTJANUS also showed statistical significance when compared to BASEJANUS for Installation 1, although the results were statistically indistinguishable for Installation 2.

In conclusion, LIFTJANUS is better than BASEJANUS in all cases, although the similarity between pairs can be comparable in some installations. On the other hand, LIFTJANUS can be better than UNCERROBUA at producing higher LWT_{Δ} in some installations, while in others UNCERROBUA can produce higher LWT_{Δ} at the cost of increasing the Hamming distance between members, which may have a negative impact in the debugging process. We can therefore answer the first RQ as follows:

> **RQ1:** LIFTJANUS is effective at generating boundary input pairs, yielding a high performance discrepancy between similar inputs while having a low distance between member pairs when compared to the selected baselines.

5.2 RQ2 - Test Input Validity

Table 5 summarizes the test input validity results achieved by LIFTJANUS in the evaluated installations. Specifically, we measured to what extent LIFTJANUS was effective in finding a configuration that aligns with the QoS requirements specified by our industrial partner , i.e., that each test input must not exceed a certain (predefined) threshold for the LWT metric. Notice that this threshold was established based on domain experts' feedback, similar to what we did in our previous study (Valle et al. 2023). This evaluation was conducted using 80 generated test input pairs (i.e., 5 per run and installation).

Table 4 Statistical tests comparing the archives of LIFTJANUS, BASEJANUS and UNCERROBUA approaches

			LWT_{Δ}			AUC-LWT_{Δ}			Hamming distance		
			$\hat{A}_{12}$	p-value	Cohen's d	$\hat{A}_{12}$	p-value	Cohen's d	$\hat{A}_{12}$	p-value	Cohen's d
		BASEJANUS	1.00	< 0.0001	6.74	1.00	< 0.0001	4.86	0.09	0.0032	-1.77
Install. 1	LIFTJANUS	UNCERROBUA_{C1}	1.00	< 0.0001	24.80	1.00	0.0003	24.30	0.00	< 0.0001	-127.61
		UNCERROBUA_{C2}	1.00	< 0.0001	5.36	1.00	0.0007	2.84	0.00	< 0.0001	-287.12
		UNCERROBUA_{C3}	1.00	< 0.0001	4.75	0.95	0.0002	2.46	0.00	< 0.0001	-303.13
		BASEJANUS	1.00	< 0.0001	5.43	1.00	< 0.0001	4.13	0.42	0.8016	-0.12
Install. 2	LIFTJANUS	UNCERROBUA_{C1}	1.00	< 0.0001	18.86	1.00	0.0007	16.22	0.00	< 0.0001	-104.55
		UNCERROBUA_{C2}	0.04	0.0004	-2.28	0.03	0.0016	-2.79	0.00	< 0.0001	-210.37
		UNCERROBUA_{C3}	0.00	< 0.0001	-2.80	0.00	< 0.0001	-3.40	0.00	< 0.0001	-289.99

Table 5 Summary of the test input validity results for Installation 1 and Installation 2

	Test Input Validity Rate (%)	LWT_{Δ} reduction ratio		LWT_{Δ} pre-repair		LWT_{Δ} post-repair	
		m	σ	m	σ	m	σ
Installation 1	90.0	0.88	0.11	247.24	54.15	26.33	24.47
Installation 2	62.5	0.93	0.05	186.39	17.45	12.55	11.33

Overall, our approach demonstrated high effectiveness at generating valid test inputs, as reflected in the test input validity rate and the improvement in QoS requirements for the proposed configurations in Table 5. For Installation 1, the system repair component successfully identified better configurations for 90% of the generated individuals, while for Installation 2, the success rate was 62.5%.

Figures 7 and 8 show a significant reduction in the LWT_{Δ} difference between members once a new configuration was produced for both installations. As shown in Table 5, the mean LWT_{Δ} decreased from 247.24 seconds to 26.33 seconds, i.e., a reduction ratio of 88%, with a corresponding reduction in the standard deviation. Similarly, for Installation 2, the mean LWT_{Δ} shows a reduction ratio of 93%, reducing the LWT_{Δ} from 186.39 seconds to 12.55 seconds. These results indicate that LIFTJANUS is effective at repairing the system configuration so as to reduce the LWT difference in the input pair.

> **RQ2:** For the majority of the generated boundary test inputs (61 out of 80 pairs), LIFTJANUS is able to generate valid test inputs and provides a system configuration for each individual that fixes the QoS requirements by reducing the LWT difference (resp. by 88% and 93% in the two installations).

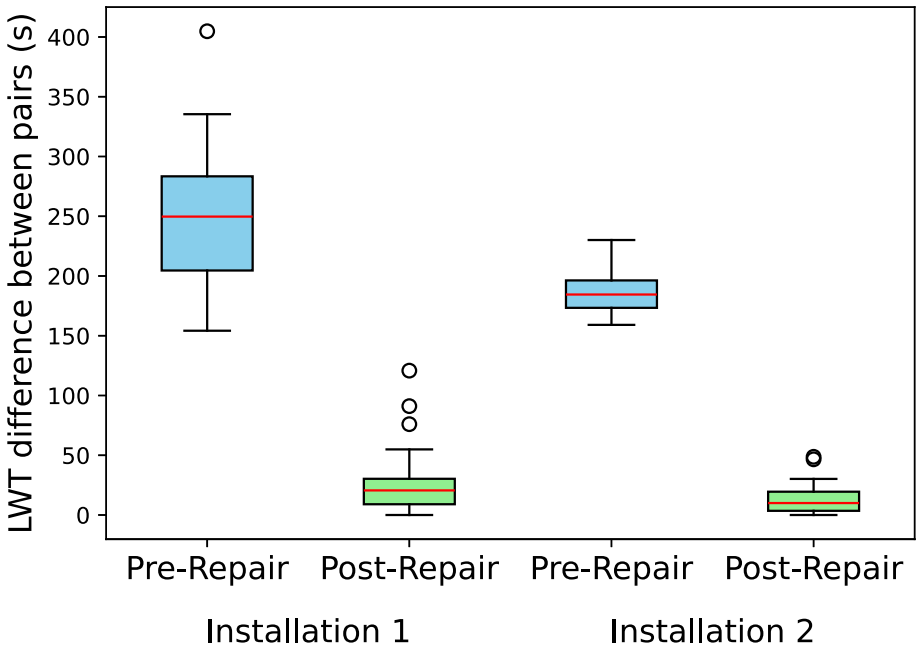


Fig. 7 LWT_{Δ} between members before and after repair component

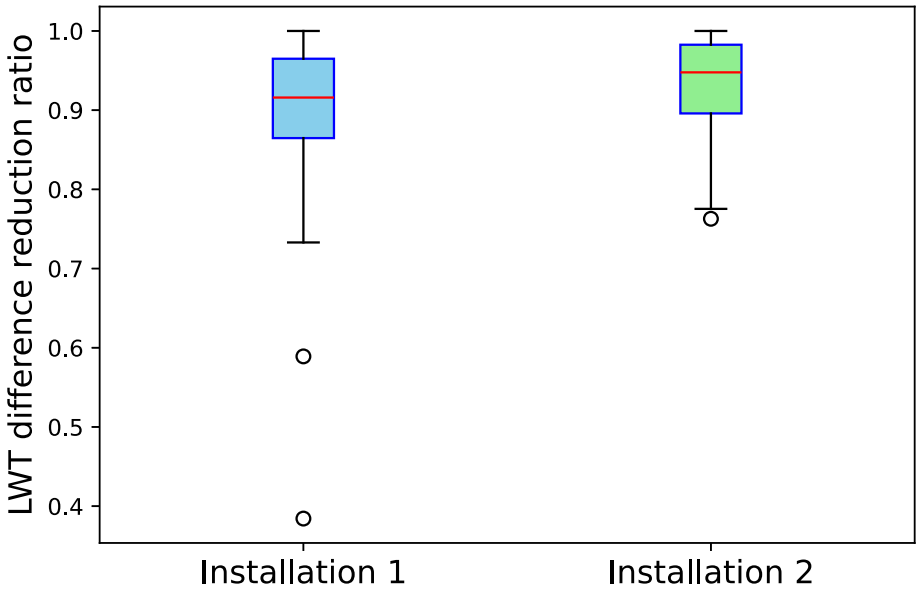


Fig. 8 LWT_{Δ} reduction ratio

5.3 RQ3 - Minimization Component Influence

Figure 4 shows the comparison between LIFTJANUS (Minimized) and the LIFTJANUS without the test input minimization component (Non-minimized). The minimization component clearly enhances the effectiveness of LIFTJANUS in both installations. In both cases, the approach with the minimization component obtained twice the Average LWT difference compared to the approach without it. In fact, for Installation 1 the LIFTJANUS non-minimized approach was outperformed by all approaches but *C1* for UNCERROBUA, and for Installation 2, it barely outperformed BASEJANUS after 40 iterations.

Regarding the effect of the minimization component on the efficiency of LIFTJANUS, Table 6 compares the execution time of the algorithm with both the test input minimization component and without it. The test input minimization component decreased the number of passengers in the test input file, i.e., 11.08 times for Installation 1 and 12.44 times for

Table 6 Comparison of the efficiency of LIFTJANUS with and without the minimization component, mean (m) and standard deviation (σ) values

		# of passengers	Total execution time of LIFTJANUS (h)	
			m	σ
Installation 1	Minimized	340	7.028	0.002
	Non-minimized	3769	71.574	0.092
Installation 2	Minimized	527	8.136	0.020
	Non-minimized	6558	78.336	0.109

Installation 2. This led to a reduction in the execution time of LIFTJANUS by 10 times in Installation 1 and 9.66 times in Installation 2.

> **RQ3:** The test input minimization component significantly improves both the effectiveness and efficiency of LIFTJANUS, producing boundary pairs with larger discrepancies in LWT within a shorter execution time.

5.4 RQ4 - Qualitative Analysis

In RQ4, we conducted a semi-structured interview with two domain experts from ORONA to qualitatively assess our approach and the usefulness of its integration into their daily work.

When assessing ***whether it is trivial for them to find similar input pairs with such a large difference in performance metrics***, both engineers agreed that without an automated framework, it is highly unlikely to find inputs that represent those boundaries with minimal input variations. The most experienced engineer mentioned the challenge of manually identifying a diverse set of representative test inputs, highlighting that LIFTJANUS enables this process.

When assessing ***how easy it is to explain QoS performance differences when comparing both input files,*** both engineers agreed that identifying the precise cause on discrepancy when provided with a pair of input files is challenging. The most experienced engineer highlighted that there could be a large number of possible causes that could contribute to performance degradation based on the variation.

We also assessed ***whether the engineers could suggest configurations to enhance the SUT performance, while adhering to QoS requirements***. The less experienced engineer claimed that without knowing the exact root cause of performance differences, it was nearly impossible to propose SUT configurations. Instead, the more experienced engineer suggested a configuration but the performance improved only for one of the members, while deteriorating for the other one. This led to the conclusion that manually repairing the system is extremely complex due to the large number of configuration parameters involved. Their answers support the usefulness of the repair component.

They examined ***pairs of inputs that could not be automatically repaired***. Both engineers concurred that, while it is impossible to certainly know it, these non-repairable cases might represent corner cases that surpass the capabilities of the dispatching algorithm. One of them also did not discard that there could be instances that could eventually be repaired, but given that the configuration space is vast, both the repair algorithm and the human expert may be unable to find the exact configuration needed for the repair.

Finally, when assessing ***the utility of LIFTJANUS in the daily tasks of dispatching algorithm developers***, both engineers concurred that LIFTJANUS is highly beneficial for testing new dispatching algorithms and correcting misconfigurations. The most experienced engineer emphasized the importance of manually assessing the boundary inputs identified by LIFTJANUS before concluding that they expose an unforeseen situation that necessitates a repair.

> **RQ4:** The interviewed domain experts agreed that the test inputs generated by LIFTJANUS could ease their daily tasks in the development and maintenance of dispatching algorithms.

6 Threats to Validity

To address potential threats to **internal validity** due to the multitude of configurations of the considered techniques, we set up their configuration parameters based on those suggested in their corresponding papers.

An **external validity threat** in our evaluation relates to the generalizability of the results. This can be distinguished in two different dimensions: (1) obtaining the same level of benefits observed in our evaluation and (2) applicability of our approach beyond our case study system. While additional case studies are necessary to further validate our approach, our case study was performed in a representative industrial setting involving a real software system from ORONA, as well as realistic building installations. Furthermore, we evaluate our approach in two different installations with real operational data.

Random variations might have affected the results, given the non-deterministic nature of the techniques. We mitigated this **conclusion validity** threat by running each algorithm multiple times and employing statistical tests to ensure the results' significance. Another threat to the **conclusion validity** raises from the limited number of engineers who participated in the qualitative assessment (RQ4) of our approach. It is important to highlight that these two domain experts are the only engineers working on this module of the elevator, so, we could not do anything to palliate this issue. Furthermore, the experience of these engineers warrants a reliable and informed evaluation, as they possess in-depth knowledge of the dispatching algorithm and its requirements. Future work will involve adapting the approach to other industrial contexts and trying to generalize our findings with more domain experts.

7 Lessons Learned

Our development, application, evaluation and contrast with practitioners led to a set of key lessons learned that are applicable to other CPS domains beyond the one considered in this paper.

Lesson 1 – Slight input changes may lead to significant system behavior degradation, indicating that the generated system boundary values highlight severe robustness issues. However, finding such effective inputs at the frontier requires appropriate search mechanisms By applying LIFTJANUS to our industrial case study system, we find that small variations in the test inputs (i.e., passengers) may produce a significant degradation in the time spent by passengers to wait. However, the comparison with BASEJANUS suggests that finding such slight input changes that lead to behavior degradation is challenging. Indeed, in both tested installations, we see that without appropriate guidance, the Average LWT difference does not change much, suggesting that advanced search mechanisms are needed. Because of this, LIFTJANUS integrates an adapted version of Riccio and Tonella (2020) to our domain, integrating NSGA-II (Deb et al. 2002) with novelty search (Lehman and Stanley 2011). We believe we can find several analogies in other CPSs from other domains. For instance, in the context of drones, the introduction or slight modification of an obstacle may cause unstable and potentially unsafe behaviors of the SUT (Khatiri et al. 2023). In the context of autonomous vehicles (AVs), changes in the road shape (Riccio and Tonella 2020), initial driving conditions (position, velocity and orientation) (Biagiola and Tonella 2023), or weather and light conditions can lead to system misbehaviors (Stocco et al. 2020). While our approach's system inputs encompassed passengers, it could easily be adapted to other

CPS test inputs based on other domains, such as, road shapes (lane/trajectory keeping for AVs (Gambi et al. 2019; Riccio and Tonella 2020)), input signals (Matinnejad et al. 2018, 2016), placement of obstacles (obstacle avoidance for AVs) (Khatiri et al. 2023).

Lesson 2 – Minimizing the test input helps increase both the effectiveness and efficiency of the search process In our approach, we decided to reduce the test inputs by using a delta debugging approach (Valle et al. 2023), with the goal of bringing to the search process only relevant parts of the test inputs. This provides two core advantages. On the one hand, it significantly reduces the test execution time every time we invoke the simulator, thereby, producing a quicker fitness evaluation (the main bottleneck of the search process). On the other hand, the test input is smaller (i.e., in our case, fewer passengers are involved in the input), which significantly reduces the search space. We believe this can also be applied in other systems beyond our domain. For instance, in the context of AVs, the road could be minimized to only consider cases in which the AV was closer to be out of bounds. In the context of UAVs (Khatiri et al. 2023), the scenario and the number of obstacles could be reduced to only consider those that are relevant.

Lesson 3 – Automatically repairing misconfigurations enhances trust in the test input validity A core issue when generating test cases for CPSs is the generation of invalid test inputs. For instance, in the context of autonomous vehicles, test input generators typically generate unavoidable obstacles (Calò et al. 2020) or non-realistic tests (Wu et al. 2024). By taking advantage of the configurable nature of our system, we leverage an automated misconfiguration repair algorithm to check whether a new configuration can reduce the behavioral difference between the two members. As CPSs are typically highly configurable (Arrieta et al. 2019, 2017; Calò et al. 2020), this approach could be adopted by CPS developers to gain confidence on the validity of their generated test cases and to obtain repair hints. Moreover, our semi-structured interview with domain experts indicates that even if not repaired, a boundary test input pair could be still valid, e.g. because an unforeseen situation has been discovered or because it is due to bugs in the code that trigger a differential behavior. In addition, it also showed that manually proposing a new configuration to fix the found issue is extremely complex, giving further value to our automated solution.

Lesson 4 – It is difficult to explain which are the explicit causes that led to a system degradation and LIFTJANUS is beneficial for this task The interviewed domain experts claimed that identifying the specific cause that leads to a system degradation is challenging, since it could be related to different reasons like wrong parametrization or sub-optimalities when implementing the software (or both). The experts also claimed that manually assessing the boundary pairs produced by LIFTJANUS is useful in their daily tasks. In our future work, we will further improve the feedback provided to engineers to help them debug the system degradation, e.g., by incorporating decision tree-based learning techniques (Kampmann et al. 2020).

8 Related Work

This section summarizes prior research relevant to our approach, highlighting how the different components of LIFTJANUS correspond to existing works in the literature. Table 7 lists the key components of our approach alongside related research works that address similar challenges or propose comparable solutions.

Table 7 Correspondence of the related work with each component of our approach

Related Papers	Test Oracle	Test Input Minimizator	Terst Input Generator	System Repair
Nicolas et al. (2016)	+	–	–	–
Ayerdi et al. (2020)	+	–	–	–
Gartziandia et al. (2022)	+	–	–	–
Valle et al. (2023)	–	+	–	–
Sagardui et al. (2017)	–	–	+	–
Han et al. (2023)	–	–	+	–
Han et al. (2022)	–	–	+	–
Mandrioli et al. (2024)	+	–	+	–
Mullins et al. (2018)	–	–	+	–
Riccio and Tonella (2020)	–	–	+	–
Tuncali and Fainekos (2019)	–	–	+	–
Valle et al. (2023)	–	–	–	+
Biagiola and Tonella (2022)	–	–	+	+
LIFTJANUS (This paper)	+	+	+	+

8.1 Testing Elevator Systems

Different approaches have been proposed over the last few years for testing systems of elevators. Nicolas et al. (2016) proposed an FPGA-based testing approach for hardware-in-the-loop testing of elevator controllers in charge of handling the position and speed of elevators. Ayerdi et al. (2020) proposed the use of metamorphic testing for tackling the test oracle problem in the context of elevator dispatching algorithms. Afterwards, they proposed automating the generation of metamorphic relations to reduce manual effort when defining them Ayerdi et al. (2021). Gartziandia et al. (2022) proposed a machine-learning based approach to automate the detection of performance problems in elevator dispatching algorithms. All these studies target the automation of either the execution of test cases or their evaluation (i.e., the test oracle problem). In contrast, our approach targets the automated generation of test cases.

As for test generation, Sagardui et al. (2017) combined model-based testing with variability modeling to test the controllers of elevators' doors in the context of regression testing in Simulink. Conversely, our approach relies on the combination of several techniques (e.g., delta debugging, search-based approaches) for testing another component of the elevator, i.e., the dispatching algorithm. In the context of elevator dispatching algorithms, Han et al. (2023, 2022) proposed a series of methods (named as Han et al. (2022) and Han et al. (2023)) to systematically detect uncertainties in passenger files in order to test the robustness of the system. Similar to our approach, they change different passengers' attributes (e.g., their mass). Their techniques focus on changing the attributes for all passengers in the files (e.g., the mass of all passengers) by following a uniformly random strategy, without the guidance of a simulator. In contrast, our approach aims at producing minimal changes to selected passengers using a fitness guided approach. Mandrioli et al. (2024) introduced a method for testing CPSs using design assumption-based metamorphic relations (MRs) and genetic programming to generate input-output trace pairs. In contrast, our work focuses on testing a high-level CPS controller, the elevator dispatching algorithm, which involves complex domain-specific logic and a large configuration space (40-80 parameters). Unlike

low-level controllers, which are typically analyzed with control-theoretic methods, high-level controllers are often tuned manually or via optimization, making traditional control-theoretic guarantee approaches hardly applicable.

8.2 Search-Based Boundary Testing for CPS

Several testing approaches focused on boundary inputs (Pezzè and Young 2008) as they can facilitate debugging by isolating portions of inputs responsible for notably different system behaviors, e.g., correct behavior vs misbehavior. Many of these approaches resort to search-based techniques to identify boundary inputs, although their scalability to complex contexts, like CPS, including elevator systems, might require special techniques and specific adaptations.

Mullins et al. (2018) used an adaptive search algorithm to identify performance boundaries of Autonomous Driving Systems. Tuncali and Fainekos (2019) designed an approach to generate pairs of configurations in which a vehicle collision is avoidable for one member, unavoidable for the other. Biagiola and Tonella (2022), generated pairs of environment configurations for testing the plasticity of Reinforcement Learning (RL) based systems. For one member of the pair the RL SUT can adapt to the new environment, while in the other it cannot. Riccio and Tonella (2020) proposed DeepJanus, a multi-objective search algorithm designed to generate boundary inputs for various systems, including CPS like ADS. It generates similar road shapes for which the ADS behaves differently. We chose to adapt this algorithm since it promotes both the quality and diversity of boundary inputs. Thus, we adapted it for systems of elevators and augmented it with a test input minimization component and a repair algorithm.

Unlike all these papers, to the best of our knowledge, this is the first industrial experience report applying boundary value detection to elevator systems. To this end, our approach required tailored adjustments to suit the domain-specific characteristics of these systems. Moreover, our approach incorporates novel aspects, such as the combination of boundary testing with test input minimization and automated repair. The lessons learned from our combination of these techniques could be promising also in other contexts beyond elevator systems.

9 Conclusion and Future Work

LiftJanus is the first test generator for elevator systems that integrates test input minimization, boundary value detection, and automated system repair. Our experiments involving real-world systems provided by our industrial collaborator, Orona, demonstrated its effectiveness in generating boundary test input pairs that are valid to expose repairable issues of the system configuration. The insights gathered from domain experts through interviews confirmed that LiftJanus is a valuable end-to-end technique for enhancing the quality of complex elevator systems.

In our future work, we plan to generalize our results to a broader spectrum of industrial systems across different domains. We would also like to explore different techniques to improve the feedback provided to engineers by incorporating decision tree-based learning techniques. In addition, we would like to evaluate the quality of the feedback provided to the engineers based on boundary-focused approaches versus diversified approaches, since a recent study demonstrated that test inputs generated through adaptive random search result

in significantly more accurate decision trees than those produced by search algorithms that focus on exploring boundaries (Jodat et al. 2024). An empirical study with the involvement of developers might be useful to compare the perceived quality of the test inputs produced by LiftJanus w.r.t. those produced by the considered baseline tools.

Appendix

A: Input Minimization Algorithm

For the Test Minimization component, explained in Section 3.1, we used the algorithms proposed in our previous work (Valle et al. 2023). In particular, we selected the environment-wise delta debugging algorithm (EWDD) with the event-based minimization technique, as this is the best algorithm according to the results of our previous study. Algorithm 2 shows the EWDD algorithm. This algorithm, first obtains the Environmental States (ES) of the CPS (i.e., an elevator system in this paper) while this is being tested (Line 1). Using this information, the algorithm searches for the so-called *"static states"* of the system. Such static states refer to stable states of the CPS and its environment. For instance, in our industrial case study, we consider the CPS is in a static state when all elevators are stopped, with no passengers inside the cabins and the doors of all elevators are closed. When executing a test, several static states can be found throughout the simulation. After retrieving the environment states, the original failure inducing test input is split (Lines 2 and 3).

Then, the algorithm starts assessing the minimization based on the static states (Lines 4-10). This static state is an object that contains several information (e.g., the time the static state started and finished, position of elevators). With this static state, the CPS and its environment is configured to execute the test (Line 6). It is assumed that the CPS and its environment are configurable so that the simulation can start from such a static state. For instance, in our industrial case study system, it is possible to configure the environment to execute a test with each elevator in a certain position. After the environment is prepared, the algorithm reduces the test input by removing all the passengers before the static state (Line 7). Then, the test is executed (Line 8) and i is reduced by 1, such that the previous static state is kept, in case the execution did not reproduce the failure.

Once the failure is reproduced, our adaptation of the original delta debugging algorithm is executed (Lines 11-21). The following steps aim at reducing TI by removing passengers that are before the conflicting passenger (Lines 12 and 13). This process is carried out by invoking Algorithm 3, which takes as inputs (i) TI' and (ii) the split size. After this, the algorithm enters a while loop (Lines 14-21) that tries to minimize the failure inducing test input as much as possible. To this end, it first executes the test in TI_{NEW} (Line 15). If the test returns a failure, the minimization procedure can continue. The test input in TI_{NEW} is assigned to TI', and the minimization routine is invoked again by means of the *splitMinEvent* function (Lines 18 and 19). If the test has passed, it means that the test input was minimized too much. Therefore, the test input requires to be enlarged (i.e., more passengers are required to reproduce the failure). This is carried out by invoking the routine in Algorithm 4, which adds a number of passengers between the number of passengers in TI_{NEW} and TI'. This procedure returns the maximized test input in TI_{NEW} (Line 21), which is tested in Line 15 of Algorithm 2. This process is repeated until the number of passengers in TI' and TI_{NEW} are the same. When this condition holds, the delta debugging algorithm returns the minimal failure-inducing test input.

Algorithm 2 Environment-wise Delta Debugging Algorithm.

Input: SUT //*System Under Test*
FT // *Failing Time*
SD // *Simulation Data*
TI = $\{p_1, p_2, ..., p_{np}\}$ // *Initial failure inducing test input*
Output: TI' = $\{p'_1, p'_2, ..., p'_{npr}\}$ *Minimized failure inducing test input*

```
ES ← getEnvironmentStatesUntilFailure(SD, FT)
i ← numberOfStaticStates(ES)
TI' ← split(TI,FT)
do
    staticState ← getState(ES, i)
    prepareEnvironment(staticState)
    TI_NEW ← splitStaticState(staticState, TI')
    Verdict ← executeTest(TI_NEW)
    i ← i-1;
while Verdict == Pass;
TI' ← TI_NEW
splitSize ← ⌈TI'.np/2⌉
TI_NEW ← splitMinEvent(TI', splitSize)
while TI'.np ≠ TI_NEW.np do
    Verdict ← executeTest(TI_NEW, SUT)
    splitSize ← ⌈splitSize/2⌉
    if Verdict == Failure then
        TI' ← TI_NEW
        TI_NEW ← splitMinEvent(TI_NEW, splitSize)
    else
        TI_NEW ← splitMaxEvent(TI_NEW.np, TI', splitSize);
return TI'
```

Algorithm 3 SplitMinEvent: Event-based split Minimizing.

Input: splitSize // *Point to split*
TI=$\{p_1, p_2, ..., p_{np}\}$ // *Minimized test input previously selected*
Output: TI_{NEW} // *Minimized test input*

```
for i ← splitSize to TI.np do
    TI_NEW ← TI_NEW ∪ p_i
```

Algorithm 4 SplitMaxEvent: Event-based split Maximizing.

Input: sizeTI // *# of selected passengers*
splitSize // *Point to split*
TI = $\{p_1, p_2, ..., p_{np}\}$ // *Minimized test input*
Output: TI_{NEW} // *Maximized test input*

```
toSplit ← TI.np-(sizeTI + splitSize)
for i ← toSplit to TI.np do
    TI_NEW ← TI_NEW ∪ p_i
```

B: Hamming Distance Threshold Calculation for Population Initialization

Figure 9 depicts the results of the preliminary experiments we performed to assess the impact of Hamming distance thresholds on the execution time required to initialize the initial population. We executed the initialization algorithm with 5 different thresholds: 0.000001, 0.000005,

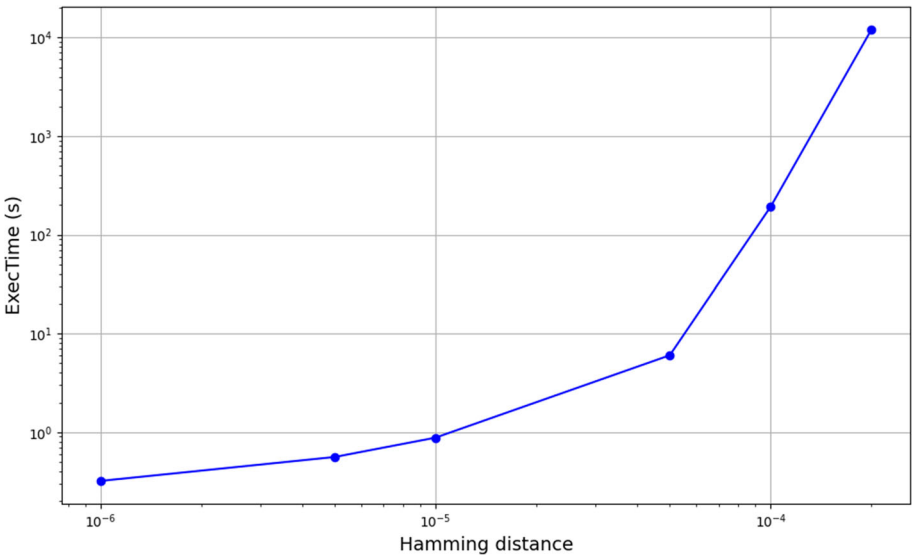


Fig. 9 Initial population generation time for each Hamming distance threshold

0.00001, 0.00005, 0.0001 and 0.0002. As discussed in Section 3.2.3, we selected a threshold of 0.0001. This configuration resulted in an initialization time of approximately 190 seconds. Compared to lower thresholds, the slight increase in execution time is justified by the gain in population diversity. However, increasing the threshold further, e.g., by doubling it to 0.0002, leads to a rise in execution time, reaching around 12,000 seconds. This overhead makes such a threshold impractical for the initialization phase.

Author Contributions All authors contributed to the study conception, design and writing (original draft preparation, review and editing). Aitor Arrieta contributed to funding acquisition of the main project. Pablo Valle implemented the code and launched the experiments, with the supervision of Vincenzo Riccio. All authors read and approved the final manuscript.

Funding Pablo Valle and Aitor Arrieta are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country. Aitor Arrieta is supported by the Spanish Ministry of Science, Innovation and Universities (project PID2023-152979OA-I00), funded by MCIU /AEI /10.13039/501100011033 / FEDER, UE. Pablo Valle is supported by the Pre-doctoral Program for the Formation of Non-Doctoral Research Staff of the Education Department of the Basque Government (Grant n. PRE_2024_1_0014). Vincenzo Riccio is supported by the M4C2 I1.3 "SEcurity and RIghts In the CyberSpace - SERICS" (PE00000014 -CUP H73C2200089001, D33C22001300002) projects, under the National Recovery and Resilience Plan (NRRP) funded by the European Union - NextGenerationEU. Paolo Tonella is supported by the SNSF Project Toposcope (n. 214989).

Data Availability The results obtained from our study and the scripts to analyze the results are provided in the following Github repository (Replication package of liftjanus 2024). However, LIFTJANUS, datasets of the test inputs and scenarios used or generated during this study are not publicly accessible due to the high sensitivity of the data from our industrial partner.

Declarations

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

Ethical Approval According to the regulations of Mondragon's University and Orona, where the study was carried out, the empirical study with developers that we conducted did not require any ethical approval.

Informed Consent Informed consent was obtained from all individual participants involved in the evaluation of this study.

References

Derler P, Lee EA, Vincentelli AS (2011) Modeling cyber–physical systems. Proc EEE 100(1):13–28
Arrieta A, Sagardui G, Etxeberria L, Zander J (2017) Automatic generation of test system instances for configurable cyber-physical systems. Softw Qual J 25(3):1041–1083
Arrieta A, Wang S, Sagardui G, Etxeberria L (2019) Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. J Syst Softw 149:1–34
Gambi A, Mueller M, Fraser G (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, pp 318–328
Zohdinasab T, Riccio V, Gambi A, Tonella P (2023) Efficient and effective feature space exploration for testing deep learning systems. ACM Trans Softw Eng Methodol 32(2):1–38
Abdessalem RB, Panichella A, Nejati S, Briand LC, Stifter T (2020) Automated repair of feature interaction failures in automated driving systems. In: Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis, pp 88–100
Nejati S, Gaaloul K, Menghi C, Briand LC, Foster S, Wolfe D (2019) Evaluating model testing and model checking for finding requirements violations in simulink models. In: Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 1015–1025
Barney G (2010) Transportation systems in buildings: CIBSE Guide D: 2010. Chartered institution of building services engineers, London
Barney G, Al-Sharif L (2015) Elevator traffic handbook: theory and practice. Routledge
Biagiola M, Tonella P (2022) Testing the plasticity of reinforcement learning-based systems. ACM Trans Softw Eng and Methodol (TOSEM) 31(4):1–46
Nejati S, Sorokin L, Safin D, Formica F, Mahboob MM, Menghi C (2023) Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial adas and simulink models. Inf Softw Technol 107286
Zhang M, Wu J, Ali S, Yue T (2023) Uncertainty-aware test prioritization: Approaches and empirical evaluation. arXiv preprint arXiv:2311.12484
Stocco A, Pulfer B, Tonella P (2023) Model vs system level testing of autonomous driving systems: a replication and extension study. Empir Softw Eng 28(3):73
Menghi C, Viganò E, Bianculli D, Briand LC (2021) Trace-checking cps properties: Bridging the cyber-physical gap. In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE), IEEE, pp 847–859
Valle P, Arrieta A, Arratibel M (2023) Applying and extending the delta debugging algorithm for elevator dispatching algorithms (experience paper). In: Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis, pp 1055—1067
Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
Derler P, Lee EA, Vincentelli AS (2011) Modeling cyber-physical systems. Proc EEE 100(1):13–28
Fahmy H, Pastore F, Briand L, Stifter T (2023) Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems. ACM Trans Softw Eng Methodol 32(4):1–47
Huai Y, Almanee S, Chen Y, Wu X, Chen QA, Garcia J (2023) sceno rita: Generating diverse, fully-mutable, test scenarios for autonomous vehicle planning. IEEE Trans Softw Eng
Gartziandia A, Arrieta A, Ayerdi J, Illarramendi M, Agirre A, Sagardui G, Arratibel M (2022) Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms. J Softw Evol Process 34(11):e2465
Zohdinasab T, Riccio V, Tonella P (2023) Deepatash: Focused test generation for deep learning systems. In: Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis, pp 954–966
Han L, Ali S, Yue T, Arrieta A, Arratibel M (2023) Uncertainty-aware robustness assessment of industrial elevator systems. ACM Trans Softw Eng Methodol 32(4):1–51

Zhang M, Ali S, Yue T, Hedman M (2016) Uncertainty-based test case generation and minimization for cyber-physical systems: A multi-objective search-based approach. Simula Res Lab
Ayerdi J, Terragni V, Jahangirova G, Arrieta A, Tonella P (2024) Genmorph: Automatically generating metamorphic relations via genetic programming. IEEE Trans Softw Eng
Ayerdi J, Arrieta A, Pobee EB, Arratibel M (2022) Multi-objective metamorphic test case selection: an industrial case study (practical experience report). In: 2022 IEEE 33rd international symposium on software reliability engineering (ISSRE), IEEE, pp 541–552
Humeniuk D, Khomh F, Antoniol G (2022) A search-based framework for automatic generation of testing environments for cyber-physical systems. Inf Softw Technol 149:106936
Jodat BA, Chandar A, Nejati S, Sabetzadeh M (2024) Test generation strategies for building failure models and explaining spurious failures. ACM Trans Softw Eng Methodol 33(4):1–32
Laurent T, Arcaini P, Zhang XY, Ishikawa F (2024) Metamorphic testing of an autonomous delivery robots scheduler. In: 2024 IEEE conference on software testing, verification and validation (ICST), IEEE, pp 361–372
Riccio V, Tonella P (2020) Model-based exploration of the frontier of behaviours for deep learning system testing. In: Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 876–888
Klikovits S, Castellano E, Cetinkaya A, Arcaini P (2023) Frenetic-lib: An extensible framework for search-based generation of road structures for ads testing. Sci Comput Program 230:102996
Barney G (2010) Transportation systems in buildings : CIBSE Guide D: 2010. London: Chartered institution of building services engineers
Lehman J, Stanley KO (2011) Abandoning objectives: Evolution through the search for novelty alone. Evol comput 19(2):189–223
Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
Matinnejad R, Nejati S, Briand LC, Bruckmann T (2018) Test generation and test prioritization for simulink models with dynamic behavior. IEEE Trans Softw Eng 45(9):919–944
Riccio V, Humbatova N, Jahangirova G, Tonella P (2021) Deepmetis: Augmenting a deep learning test set to increase its mutation score. In 2021 36th IEEE/ACM international conference on automated software engineering (ASE), IEEE, pp 355–367
Riccio V, Tonella P (2023) When and why test generators for deep learning produce invalid inputs: an empirical study. In: 2023 IEEE/ACM 45th international conference on software engineering (ICSE), IEEE, pp 1161–1173
Fahmy H, Pastore F, Briand L, Stifter T (2023) Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems. ACM Trans Softw Eng Methodol 32(4):1–47
Mullins GE, Stankiewicz PG, Hawthorne RC, Gupta SK (2018) Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. J Syst Softw 137:197–215
Barney G, Al-Sharif L (2015) Elevator traffic handbook: theory and practice. Routledge
Cohen J (1987) Statistical power analysis for the behavioral sciences (revised edn) In: Laurence erlbaum associates: Hillsdale, NJ, USA
Romano J, Kromrey JD, Coraggio J, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen'sd indices the most appropriate choices. In: Annual meeting of the southern association for institutional research, Citeseer, pp 1–51
Pezzè M, Young M (2008) Software testing and analysis: process, principles, and techniques. John Wiley & Sons
Khatiri S, S. Panichella, and P. Tonella, "Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights. In: 2023 IEEE conference on software testing, verification and validation (ICST), IEEE, pp 281–292
Biagiola M, Tonella P (2023) Boundary state generation for testing and improvement of autonomous driving systems. arXiv preprint arXiv:2307.10590
Stocco A, Weiss M, Calzana M, Tonella P (2020) Misbehaviour prediction for autonomous driving systems. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering, pp 359–371
Matinnejad R, Nejati S, Briand LC, Bruckmann T (2018) Test generation and test prioritization for simulink models with dynamic behavior. IEEE Trans Softw Eng 45(9):919–944
Wu J, Lu C, Arrieta A, Yue T, Ali S (2024) Reality bites: Assessing the realism of driving scenarios with large language models. In: arXiv preprint arXiv:2403.09906
Arrieta A, Wang S, Sagardui G, Etxeberria L (2019) Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. J Syst Softw 149:1–34
Sawilowsky SS (2009) New effect size rules of thumb. J ModernAappl Stat Methods 8:597–599

Stocco A, Pulfer B, Tonella P (2023) Model vs system level testing of autonomous driving systems: a replication and extension study. Empir Softw Eng 28(3):73

Nicolas CF, Ayestaran I, Martinez I, Franco P (2016) Model-based development of an fpga encoder simulator for real-time testing of elevator controllers. In: 2016 IEEE 19th international symposium on real-time distributed computing (ISORC), IEEE, pp 53–60

Ayerdi J, Segura S, Arrieta A, Sagardui G, Arratibel M (2020) Qos-aware metamorphic testing: An elevation case study. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE), IEEE, pp 104–114

Gartziandia A, Arrieta A, Ayerdi J, Illarramendi M, Agirre A, Sagardui G, Arratibel M (2022) Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms. J Softw Evol Process 34(11):e2465

Sagardui G, Etxeberria L, Agirre JA, Arrieta A, Nicolas CF, Martin JM (2017) A configurable validation environment for refactored embedded software: An application to the vertical transport domain. In: 2017 IEEE international symposium on software reliability engineering workshops (ISSREW), IEEE, pp 16–19

Mandrioli C, Shin SY, Bianculli D, Briand L (2024) Testing cps with design assumptions-based metamorphic relations and genetic programming

Mullins GE, Stankiewicz PG, Hawthorne RC, Gupta SK (2018) Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. In: J Syst Softw 137:197–215

Zeller A, Hildebrandt R (2002) Simplifying and isolating failure-inducing input. IEEE Trans Softw Eng 28(2):183–200

Zhang XY, Liu Y, Arcaini P, Jiang M, Zheng Z (2024) Met-mapf: A metamorphic testing approach for multi-agent path finding algorithms. ACM Trans Softw Eng Methodol 33(8):1–37

Ayerdi J, Terragni V, Arrieta A, Tonella P, Sagardui G, Arratibel M (2021) Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study. In: Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 1264–1274

Pezzè M, Young M (2008) Software testing and analysis: process, principles, and techniques. John Wiley & Sons

Zohdinasab T, Riccio V, Gambi A, Tonella P (2023) Efficient and effective feature space exploration for testing deep learning systems. ACM Trans Softw Eng Methodol 32(2):1–38

Replication package of liftjanus (2024) https://github.com/pablovalle/LiftJanus?tab=readme-ov-file