Testing Advanced Driver Assistance Systems using Multi-objective Search and Neural Networks

Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand
SnT / University of Luxembourg, Luxembourg
{raja.benabdessalem,shiva.nejati,lionel.briand}@uni.lu

Thomas Stifter
IEE S.A. Contern, Luxembourg
thomas.stifter@iee.lu

ABSTRACT

Recent years have seen a proliferation of complex Advanced Driver Assistance Systems (ADAS), in particular, for use in autonomous cars. These systems consist of sensors and cameras as well as image processing and decision support software components. They are meant to help drivers by providing proper warnings or by preventing dangerous situations. In this paper, we focus on the problem of design time testing of ADAS in a simulated environment. We provide a testing approach for ADAS by combining multi-objective search with surrogate models developed based on neural networks. We use multi-objective search to guide testing towards the most critical behaviors of ADAS. Surrogate modeling enables our testing approach to explore a larger part of the input search space within limited computational resources. We characterize the condition under which the multi-objective search algorithm behaves the same with and without surrogate modeling, thus showing the accuracy of our approach. We evaluate our approach by applying it to an industrial ADAS system. Our experiment shows that our approach automatically identifies test cases indicating critical ADAS behaviors. Further, we show that combining our search algorithm with surrogate modeling improves the quality of the generated test cases, especially under tight and realistic computational resources.

CCS Concepts

•**Software and its engineering** → **Software testing and debugging;**

Keywords

Advanced Driver Assistance Systems, Multi-Objective Search Optimization, Simulation, Surrogate Modeling, Neural Networks

1. INTRODUCTION

With the challenge of developing software for autonomous vehicles comes the challenge of testing this software to ensure vehicles' safety and reliability. Simulation, i.e., design time testing of system models, is arguably the most practical and effective way of testing software systems used for autonomous driving. Rich simulation environments are able to replicate various real world traffic situations. They enable engineers to execute scenarios describing different ways in which pedestrians interact with the road traffic or vehicles interact with one another. Recent years have seen major improvements in accuracy and usability of simulation tools. Their full exploitation, however, is hampered by two main factors: (1) Existing simulation tools lack the intelligence and automation necessary to guide the simulation scenarios in a direction that would be likely to uncover faulty behaviors. Hence, engineers have to identify critical system behaviors manually, with simulation being used only to execute (re-play) these behaviors. (2) Executing simulation scenarios is computationally expensive. Hence, given a limited time budget for testing, only a small fraction of system behaviors can be simulated and explored. In this paper, we provide solutions to specifically address these two limitations when testing is performed via simulation. We show how our proposed solutions enable effective testing of software systems used to assist drivers.

Motivation. We motivate our work using an advanced driver assistance system (ADAS) case study. ADASs (e.g., collision avoidance systems) are developed to assist drivers to properly react to risky situations [37]. These systems constitute one of the fastest growing segments in the automotive industry. ADASs are typically based on vision systems and sensor technologies.

Our case study is a Pedestrian Detection Vision based (PeVi) system. Its main function is to improve the driver's view by providing proper warnings to the driver when pedestrians (people or animals) appear to be located in front of a vehicle in particular when the visibility is low due to poor weather conditions or due to low ambient light. PeVi consists of a CCD camera, and software components implementing image processing and object recognition algorithms as well as algorithms that determine when and which warning message should be shown to the driver.

Testing PeVi with real hardware and in the real environment (e.g., by making a person or an animal cross a road while a car is approaching) is obviously dangerous, time-consuming, costly and to a great extent infeasible. Hence, a large part of testing for ADAS has to be carried out using what is known as *physics-based simulation platforms* [1]. These simulation platforms are able to replicate a wide range of virtual traffic and road environments. This includes simulating various weather conditions, road types and topologies,

ASE'16, September 3–7, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-3845-5/16/09...$15.00
http://dx.doi.org/10.1145/2970276.2970311

intersections, infrastructures, vehicle types and pedestrians. Further, such platforms are able to simulate sensor technologies such as radar, camera and GPS [1]. Figure 1 shows a snapshot of a simulation environment called PreScan [1] that is used to test PeVi. The physics-based simulation function of PreScan is enabled via a network of connected Matlab/Simulink models [3] implementing dynamic behavior of vehicles, pedestrians and sensors. The software under test (PeVi in our work) is also developed in Simulink and is integrated into PreScan using inter-block connections between the Simulink models of the car and PeVi.

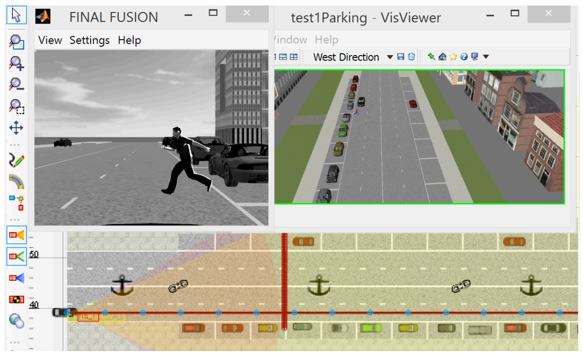


Figure 1: A snapshot of the simulation platform used to test the Pedestrian Detection Vision based (PeVi) system.

Challenges. Provided with a physics-based simulation platform, test case execution is framed as executing models of the system under test and its environment. Existing simulation platforms are able to simulate ADAS behaviors with reasonable accuracy when provided with a set of input test data. However, they have two important limitations: The *first limitation* is that simulation platforms provide no guidance to engineers as to which test scenarios should be selected for simulation. Since test inputs are specified manually in current platforms, simulation is limited to a small number of scenarios hand-picked by engineers. Manual test generation is expensive and time-consuming, and further, manually picked test cases are unlikely to uncover faults that the engineers are not aware of a priori. Hence, it is important to augment these simulation platforms with some automated *test strategy* technique, i.e., a sampling strategy in the space of all possible simulation scenarios [14], which attempts to build a sufficient level of confidence about correctness of the system under analysis through exercising only a small fraction of that space. The key question is how to choose an effective test strategy for testing ADAS.

We note that the space of all possible test scenarios for ADAS is very large. Traditional test coverage measures, which are common for small-scale, white-box testing, are infeasible and impractical for testing applications with large test spaces. Following the intuitive and common practice of system test engineers, we develop a *test strategy* that focuses on identifying high-risk test scenarios, that is scenarios that are more likely to reveal critical failures [14]. In particular, we rely on search techniques to devise a test strategy that focuses testing effort on an effective and minimal set of scenarios. The search is guided by heuristics that characterize high risk scenarios on which testing should focus. We develop meta-heuristics based on system requirements and critical environment conditions and system behaviors.

The *second limitation* is that physics-based simulations are computationally expensive because they often involve executing high-fidelity mathematical models capturing continuous dynamic behaviors of vehicles and their environment. To address this limitation, we rely on surrogate models [34] built based on machine learning techniques. Surrogate models are mathematical relations and aim to reduce computational cost by approximating high-fidelity but computationally expensive models of physical phenomena [8, 11]. They are able to predict simulation outputs within some confidence interval allowing us, under certain conditions, to bypass the execution of expensive simulations.

Contributions. We propose an automated technique to test complex ADASs based on physics-based executable models of these systems and their environments. Our technique builds on the intuitions illustrated on the motivating example described above, utilizing meta-heuristic search techniques guided by quantitative fitness functions capturing relevant and critical aspects of the system under test and its environment. We make three contributions in this paper. The *first* contribution is that we formulate our testing approach as a *multi-objective search technique.* We use multi-objective search to obtain test scenarios that stress several critical aspects of the system and the environment at the same time. For example, PeVi test scenarios should exercise behaviors during which a pedestrian appears in front of a car in such a way that the possibility of a collision is high and the chance of detecting the pedestrian is low because the pedestrian is very close to the car or because the camera's field of view is blocked. We compute such desired test scenarios by minimizing the following three fitness functions: a function measuring the distance between the pedestrian and the PeVi warning areas in front of a car, a function estimating the time to collision, and a function measuring the distance between the car and the pedestrian (see Section 2 for detailed information about PeVi). Note that combining fitness functions into one function and using single-objective search is less desired in this situation because: First, our three fitness functions are about different concepts (i.e., time and distance). Second, engineers are typically interested in exploring interactions among critical factors of the system and the environment. For example, they might be particularly interested to inspect if PeVi is able to detect pedestrians when they are located on the borders of the camera's field view. For this purpose, a multi-objective search algorithm that produces several test cases that exercise different and equally critical interactions of the system and the environment is preferred to a single-objective search algorithm that generates a single test case.

Our *second* contribution is concerned with the large execution time of our testing approach. The execution time of our search-based testing technique is large because physics-based simulations are computationally expensive. We reduce the execution time of our search algorithm by proposing a *new* combination of multi-objective search with surrogate models built based on supervised learning techniques [31]. Surrogate models are able to predict the fitness function values within some confidence intervals. The time required for surrogate models to predict values is significantly less than the time required to run simulations of physical models.

The combination of multi-objective search with surrogate modeling proposed in this paper is not tied to our particular search-based testing algorithm and is applicable to any multi-objective search algorithm that computes a set of *Pareto optimal* solutions [38, 27]. A solution is called Pareto optimal if none of the fitness functions used by the search can be improved in value without degrading some of the other fitness values [58]. In our work, we use optimistic and pessimistic fitness function predictions computed based on surrogate models and a given confidence level to rank Pareto fronts during search. We identify and prune from the search space the candidate solutions that have a low probability to be selected in the best ranked Pareto front. We show that when actual fitness values are not better than their respective optimistic predictions, the search algorithm with surrogate modeling behaves the same as the original search algorithm without surrogate modeling. Specifically, under this condition and provided with the same set of candidate solutions at each iteration, search with and without surrogate modeling select the same solutions, but the search with surrogate modeling is likely to call less simulations per iteration than the search without surrogate modeling. Note that our proposed combination of multi-objective search with surrogate modeling is more accurate than existing alternatives [40, 25] as it eventually uses the actual simulations instead of the predictions to compute Pareto optimal fronts.

Our *third* contribution is focused on demonstrating the effectiveness of our search-based testing technique by applying it to an industrial cases study, i.e., the PeVi system. Our results show that: (1) Our search-based testing technique for ADAS outperforms a random test case generation strategy (baseline). (2) Combining multi-objective search with surrogate modeling improves the quality of the generated test cases within a limited time budget for test generation. (3) Our search-based testing technique is able to produce several test scenarios indicating potential errors in the PeVi system. These test scenarios had not been previously found by manual testing based on domain expertise.

Structure. Section 2 describes the PeVi system. Section 3 outlines our approach to developing surrogate models. Section 4 provides our multi-objective search algorithm that uses surrogate modeling. Section 5 tailors our search algorithm to the PeVi system. Section 6 presents our empirical evaluation. Section 7 compares our work with the related work. Section 8 concludes the paper.

2. THE PEVI SYSTEM

In this section, we provide some further background on the PeVi system, its important inputs and outputs and the fitness functions that we design to guide our test strategy to exercise PeVi's most critical behaviors.

PeVi Requirements. Based on the Pedestrian Detection Vision based (PeVi) specification, the cone-shaped space in front of a car that is scanned by the PeVi camera is divided into three *warning areas* illustrated in Figure 2. The size of the cone vertex, α, is a feature of the camera and is called *camera's field of view*. The warning areas are described as follows. (1) The *acute warning area (AWA)* is the red rectangle in Figure 2. (2) The *warning area (WA)* is the orange area in Figure 2. (3) The *cross warning area (CWA)* refers to the two yellow right-angled rectangles on the two sides of WA in Figure 2.

The size and the position of the above three areas depend on the type of the car on which PeVi is deployed, the type

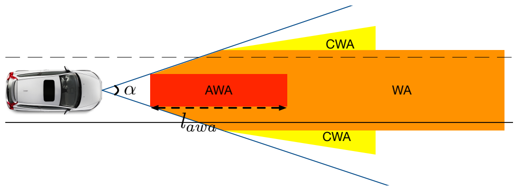


Figure 2: PeVi's warning areas.

of the camera used for PeVi, and the OEM (i.e., car maker) preferences. For example, in our case study, the field of view α is set to 40°, and the length of AWA (l_{awa}) ranges between 60m to 168m depending on the car speed.

The main requirement of PeVi is stated as follows: R = "*The PeVi system shall generate a red, orange, or yellow alert when it detects an object in AWA, WA, and CWA warning areas, respectively. Further, PeVi shall fulfill this requirement under different weather conditions and while the car runs on different types of roads with different speeds.*" The requirement R, although summing up the main function of PeVi, is still very broad. There are several scenarios where a pedestrian may end up being in one of the dangerous area in front of a car when crossing a road. To focus testing on the most high risk test scenarios among the numerous possibilities that requirement R characterizes, we identified the following specific situations after discussions with engineers at our partner company: *"It is more critical for PeVi to detect pedestrians in the warning areas when pedestrians are closer to the car and when the chance of collision is higher."*. We use these specific situations to define fitness functions for our multi-objective search algorithm.

PeVi Input and Output. In general, PeVi's function is impacted by several physical phenomena and environment factors. For example, road friction or wind may affect vehicle speed, which in turn, influences PeVi's behavior. However, given that the testing budget both in terms of manual and computational effort is limited, we identified, through our discussions with the domain expert, the most essential elements impacting the PeVi system. We developed a *domain model* to precisely capture these elements. This domain model essentially specifies a restricted simulation environment that is sufficient for testing PeVi. Further, this domain model characterizes the PeVi inputs and the outputs generated after simulating PeVi.

The domain model is shown in Figure 3. Based on this model a test scenario for PeVi contains the following input: (1) the value of the scene light intensity; (2) the weather condition that can be normal, foggy, rainy, or snowy; (3) The road type that can be straight, curved, or ramped; (4) the roadside objects, namely, trees and cars parked next to the road; (5) the camera's field of view; (6) the initial speed of the vehicle; and (7) the initial position, the orientation (θ) and the speed of the pedestrian. All these input elements except for the vehicle and the pedestrian properties are static (i.e., immobile) during the execution of a test scenario. The vehicle and the pedestrian (human or animal) are dynamic (i.e., mobile).

Our domain model makes some simplifying assumptions about PeVi's test scenarios. For example, we assume that the test scenarios contain only one pedestrian and one vehicle, and the vehicle and the pedestrian speeds are constant. These assumptions are meant to reduce the complexity of

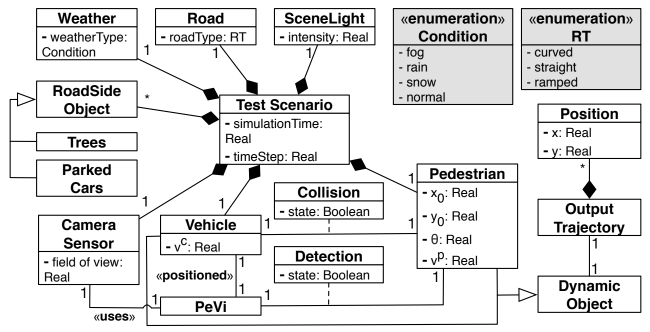


Figure 3: A fragment of the PeVi domain model.

test scenarios and were suggested by the domain expert. However, we note that our search-based test generation approach is general and is not restricted by these assumptions.

Each of the input elements in Figure 3 (i.e., the elements related by a composition relation to the test scenario element in Figure 3) impacts PeVi's behavior. For example, the weather condition and the scene light intensity impact the quality of images and the accuracy of PeVi in detecting pedestrians. The camera field of view (α) and the road shape (e.g., straight, curved and ramped) impact the topological positions of the three warning areas, which in turn, impact the pedestrian detection function of PeVi. Roadside objects may block the camera's field of view, hence leaving PeVi with little time to detect a pedestrian and to react with a proper warning message. Finally, PeVi's function should be tested for various scenarios by varying the speed of the vehicle and the pedestrian, and the position and orientation of the pedestrian.

Each test scenario is associated with a simulation time denoted by T and a time step denoted by Δt. The simulation time T indicates the time we let a test scenario run. The simulation interval $[0..T]$ is divided into small equal time steps of size Δt. In order to execute a test scenario, we need to provide the simulator with the values of the input elements described in Figure 3 as well as T and Δt.

Ideally, PeVi should be tested by varying properties of all the input elements. However, due to technical limitations of PreScan, the simulation platform used for testing PeVi, only properties of the pedestrian and the vehicle can be directly manipulated by dynamically modifying the Simulink models implementing the vehicle and the pedestrian. Other elements have to be configured manually through the tool user interface. Hence, in this paper, we limit the PeVi test input to the properties of the vehicle and the pedestrian, and leave the problem of testing PeVi for other input properties to the future. Specifically, we define the PeVi's test input as a vector $(v^c, x_0, y_0, \theta, v^p)$ where v^c is the car speed, x_0 and y_0 specify the position of the pedestrian, θ is the orientation of the pedestrian, and v^p is the speed of the pedestrian. We assume that the initial position of the vehicle (x^c_0, y^c_0) is fixed in test scenarios (i.e., $x^c_0 = 0m$ and $y^c_0 = 50m$). We denote the value ranges for v^c, x_0, y_0, θ and v^p, respectively, by R_{v^c}, R_{x_0}, R_{y_0}, R_θ and R_{v^p} such that R_{v^c}=[3.5km/h, 90km/h], R_{v^p}=[3.5km/h, 18km/h], R_θ=[40°, 160°], R_{x_0}=[x^c_0+20m, x^c_0+85m], and R_{y_0}=[y^c_0-15m,y^c_0-2m]. Note that variables v^c, x_0, y_0, θ and v^p are all of type float.

Having provided the input, PreScan simulates the behavior of PeVi, the vehicle and the pedestrian, and generates the following output elements: (1) output trajectory vectors: each dynamic object (i.e., the vehicle and the pedestrian) is associated with an output trajectory vector that stores the position of that object at each individual time step. The size of the trajectory vectors, denoted by k, is equal to the number of time steps within the simulation time, i.e., $k \approx \frac{T}{\Delta t}$. We denote the trajectory output of the pedestrian by the following two functions: $X^p, Y^p : \{0, \Delta t, 2 \cdot \Delta t, \ldots, k \cdot \Delta t\} \to \mathbb{R}$. We write $X^p(t)$ and $Y^p(t)$ to denote the pedestrian position on x- and y-axes at time t, respectively. Similarly, we define functions $X^c, Y^c : \{0, \Delta t, 2 \cdot \Delta t, \ldots, k \cdot \Delta t\} \to \mathbb{R}$ for the trajectory output of the car; (2) collision: this is a Boolean property indicating whether there has been a collision between the vehicle and the pedestrian during the simulation; and (3) detection: this is a Boolean property generated by PeVi indicating whether PeVi has been able to detect the pedestrian or not.

Fitness Functions. Our goal is to define fitness functions that can guide the search into generating test scenarios that break or are close to breaking the requirement R mentioned earlier. Based on our discussions with the domain expert, we identify the following three fitness functions.

1. Minimum distance between the car and the pedestrian. The first function, denoted by $D^{min}(p/car)$, computes the minimum distance between the car and the pedestrian during the simulation time. We denote the Euclidean distance between the pedestrian and the car at time t by $D(p/car)(t)$. The function $D^{min}(p/car)$ is then defined as follows: $D^{min}(p/car) = Min\{D(p/car)(t)\}_{0 \le t \le T}$.

The test scenarios during which the pedestrian gets closer to the car are more critical. Hence, our search strategy attempts to minimize the fitness function $D^{min}(p/car)$.

2. Minimum distance between the pedestrian and AWA. The second function, denoted by $D^{min}(p/awa)$, computes the minimum distance between the pedestrian and AWA during the simulation time. We denote the distance between the pedestrian and AWA at time t by $D(p/awa)(t)$. This function depends on the shape of the road, the orientation of the pedestrian, and her position at time t. The value of $D(p/awa)$ for time steps in which the pedestrian is inside AWA is zero. The function $D^{min}(p/awa)$ is then defined as follows: $D^{min}(p/awa) = Min\{D(p/awa)(t)\}_{0 \le t \le T}$

The goal of our search strategy is to minimize the second fitness function as well. This is because in order to test PeVi's function for AWA, we need to generate scenarios during which the pedestrian crosses AWA or gets close to it. To test PeVi for the two other warning areas, WA and CWA, we modify $D^{min}(p/awa)$ to compute the distances between the pedestrian and WA and CWA, respectively.

3. Minimum time to collision. The third fitness function is referred to as the minimum time to collision and is denoted by TTC^{min}. The time to collision at time t, $TTC(t)$, is the time required for the car to hit the pedestrian if both the car and the pedestrian continue at their speed at time t and do not change their paths. The function TTC^{min} is then defined as the minimum value of $TTC(t)$ when t ranges from 0 to T. TTC^{min} has proven to be an effective measure to estimate the collision risk and to identify critical traffic situations [51]. We are interested to generate scenarios that yield a small TTC^{min} since these scenarios are more risky.

3. SURROGATE MODELS

We use surrogate models in our work to mitigate the computation cost of executing physics-based ADAS simulations.

Specifically, in order to compute the three fitness functions described in Section 2, we have to execute expensive physics-based simulations. We create a surrogate model for each fitness function to predict the fitness values without running the actual simulations. Such surrogate models are often developed using machine learning techniques such as classification, regression or neural networks [5]. Given that we are dealing with real-valued functions, *regression* or *neural network* techniques are more suitable for our purpose because classification techniques are geared towards functions with categorical outputs. Many studies have shown that neural networks perform better than regression techniques in particular when the input space under analysis is large and when the relationship between inputs and outputs is complex [44, 26]. Hence, we use neural networks to build surrogate models. Neural networks can be used with supervised or unsupervised training algorithms [33]. In our work, we are able to obtain output values for training input data by running simulations. Hence, we use neural networks in a supervised training mode.

Neural networks consist of a number of artificial nodes, called *neurons*, connected via weighted links, forming a network of neurons. The neurons are organized in several *layers*. The first layer is the input layer followed by one or more hidden layers. The last layer is the output layer. Given a network with a predefined number of neurons and layers, the training process aims to synthesize a network by learning the weights on links connecting the neurons. Learning is carried out in a number of iterations known as *epochs*. In this paper, we consider the following well-known training algorithms to develop our surrogate models: Bayesian regularization backpropagation (BR) [39], Levenberg-Marquardt (LM) [30], and Scaled conjugate gradient backpropagation (SCG) [43].

Given a fitness function F, we build a surrogate model of F by training a neural network. To do so, we use a set of observations containing input values and known output values [56]. We divide the observation set into a *training* set and a *test* set. The training set is used to infer a predictive function $\hat{F}$. This is done by training a neural network of $\hat{F}$ such that $\hat{F}$ fits the training data as well as possible, i.e., for the points in the training set, the differences between the output of F and that of $\hat{F}$ are minimized. The test set is, then, used to evaluate the accuracy of the predictions produced by $\hat{F}$ when applied to points outside the training set. Training neural networks requires tuning a number of parameters, particularly the number of (hidden) layers, the number of neurons in each hidden layer and the number of epochs. Further, we need to choose among the three training algorithms (i..e, BR, LM, and SCG). Finding the best values for these parameters and selecting the best performing algorithm in our case is addressed in our empirical evaluation (Section 6).

In addition to building function $\hat{F}$ to predict the values of a fitness function F, we develop an error function $\hat{F}_\epsilon^{cl}$ that estimates the prediction error based on a given confidence level cl. The value of cl is a percentage value between 0 and 100. For example, let $\hat{F}_\epsilon^{cl}$ be the error function computed for $\hat{F}$ with respect to $cl = 95$. This implies that with a probability of 95%, the actual value of $F(p)$ lies in the interval of $\hat{F}(p) \pm \hat{F}_\epsilon^{cl}$. We compute $\hat{F}_\epsilon^{cl}$ based on the distribution of prediction errors obtained based on the test sets.

4. SEARCH WITH SURROGATE MODEL

We cast the problem of test case generation for ADAS as a multi-objective search optimization problem [38]. Specifically, we identified three fitness functions in Section 2 to characterize critical behaviors of the PeVi system and its environment. The solutions to our problem are obtained by minimizing these three fitness functions using a multi-objective *Pareto optimal* approach [38, 27] that states that "A solution p is said to dominate another solution p', if p is not worse than p' in all fitness values, and p is strictly better than p' in at least one fitness value". The solutions on a Pareto optimal front are non-dominating, representing best found test scenarios that stress the system under analysis with respect to the three identified fitness functions.

In our work, we rely on population-based and multi-objective search optimization algorithms [17, 19]. In this class of algorithms, the dominance relation over chromosome populations is used to guide the search towards Pareto-optimal fronts. In our work, we choose the Non-dominated Sorting Genetic Algorithm version 2 (NSGAII) [22, 38] algorithm which has been applied to several application domains and has shown to be effective in particular when the number of objectives is small [47].

Figure 4 illustrates the NSGAII algorithm. The algorithm works as follows: Initially, a random population P is generated (Line 2). After computing fitness functions $F_1, \ldots, F_k$ for each individual in P (Line 4), the individuals in P as well as those in the archive A from the previous iteration are sorted based on the non-domination relation (Line 6). In particular, a partial order relation *rank* is computed to sort elements in $Q = P \cup A$ based on the fitness functions $F_1, \ldots, F_k$. Assuming that the goal of optimization is to minimize the fitness functions F_1 to F_k, the partial order $rank \subseteq Q \times Q$ is defined as follows:

$$\forall p, p' \in Q \cdot rank(p, p') \quad \Leftrightarrow \quad \begin{array}{l} \forall i \in \{1, \ldots, k\} \cdot F_i(p) \leq F_i(p') \bigwedge \\ \exists i \in \{1, \ldots, k\} \cdot F_i(p) < F_i(p') \end{array}$$

Specifically, $rank(p, p')$ if and only if p dominates p' at least in one fitness value, i.e., p is not worse (higher) than p' in all the fitness values and p is strictly better (less) than p' in at least one fitness value. Note that it might happen for a given pair of individuals that neither of them dominates the other. For these individuals, we use the notion of crowding distance [22], denoted by cd, to be able to partially rank them. We write *non-dominating*(p, p') when p and p' are non-dominating, and say that *non-dominating*(p, p') holds iff:

$$\begin{array}{l} \forall i \in \{1, \ldots, k\} \cdot F_i(p) = F_i(p') \bigvee \\ \exists i, i' \in \{1, \ldots, k\} \cdot F_i(p) < F_i(p') \land F_{i'}(p) > F_{i'}(p') \end{array}$$

Let $R \subseteq Q$ be such that for all $p, p' \in R$, we have *non-dominating*(p, p'). For the elements in R, a crowding distance function cd is defined that assigns a value to $p \in R$ based on the distance between p and other $p' \in R$. The definition of the relation $rank \subseteq Q \times Q$ is then extended as follows: For every $p, p' \in Q$, we have $rank(p, p')$ iff

$$\begin{array}{l} (\forall i \in \{1, \ldots, k\} \cdot F_i(p) \leq F_i(p') \land \exists i \in \{1, \ldots, k\} \cdot F_i(p) < F_i(p')) \bigvee \\ (\textit{non-dominating}(p, p') \land cd(p') < cd(p)) \end{array}$$

Having computed the *rank* partial order, the NSGAII algorithm then creates a new archive A of the best solutions found so far by selecting the best individuals from Q based on *rank* (Lines 9-10). Note that $BestRanked(Q, rank)$ returns an element $p \in Q$ such that no other $p' \in Q$ dominates p, i.e., $\neg rank(p', p)$ for every $p' \in Q \setminus \{p\}$. When there are

Algorithm. NSGAII
Input: - m: Population and archive size /*|A| = |P| = m*/
- g: Maximum number of search iterations
Output: - *BestSolution*: The best solutions found in g iterations.

```
1.  $A = \emptyset$ /* Empty archive*/
2.  $P = \{p_1, \ldots, p_m\}$ /*Initial population (randomly selected)*/
3.  for $g$ iterations do
4.    $ComputeFitness(P)$ /* For all $p \in P$, fitness values
      $F_1(p), \ldots F_k(p)$ are computed*/
5.    $Q = P \cup A$ /* |Q| = 2m*/
6.    $rank = ComputeRanks(Q)$
7.    $A = \emptyset$
9.    while $|A| < m$ do
8.      $p = BestRanked(Q, rank)$
10.     $A = A \cup \{p\}$
11.   $BestSolution = A$
12.   $P = Breed(A)$ /*breeding a new population $P$ from the
      parent archive $A$*/
13. return $BestSolutionFound$
```

Figure 4: NSGAII Algorithm

more than one individual that are not dominated by any element in Q, $BestRanked(Q, rank)$ returns the one that appears first in the lexicographic ordering.

After computing the archive A, the algorithm breeds a new children population P from the parent archive A by calling the *Breed* procedure (Line 12). This is done by selecting $m/2$ individuals as parents from A using the tournament selection technique [38], and then creating m offsprings from the $m/2$ selected parents using the *crossover* and *mutation* genetic operators. The details of the *Breed* procedure can be found in [22]. We will describe the genetic operators used in our approach in Section 5. The algorithm terminates by returning the best solutions found within g generations.

In this paper, we change the NSGAII algorithm in Figure 4 to use, instead of the actual fitness values, the predicted fitness values obtained from surrogate models to compute the partial order *rank* and to select the best individuals A. We refer to our algorithm as NSGAII-SM. The main goal of NSGAII-SM is to speed up the search by selecting an archive of best individuals A from the set Q without the need to run costly simulations for every element in the newly bred children population P. Specifically, we bypass execution of the simulation for any individual $p \in P$, if we can conclude using predicted fitness values that p has a low probability to be included in A.

Recall that the surrogate model for any fitness function F comprises a prediction function $\hat{F}$ and an error function $\hat{F}^{cl}_{\epsilon}$ that estimates the prediction errors of $\hat{F}$ within the confidence level cl. Since our optimization problem aims to minimize fitness values, for any individual p, we have a most optimistic fitness value $\hat{F}(p) - \hat{F}^{cl}_{\epsilon}(p)$ and a most pessimistic fitness value $\hat{F}(p) + \hat{F}^{cl}_{\epsilon}(p)$. The gap between these two values widens by increasing the confidence level cl, and decreases by lowering cl.

Our NSGAII-SM algorithm is shown in Figure 5. The algorithm computes predicted fitness values for every individual $p \in P$ (Line 4). The algorithm, further, uses a set *Predicted* to keep track of elements for which only predicted fitness values are known, i.e., the elements for which the actual simulation has not yet been executed. The set *Predicted* is initially set to P since for the elements in P, actual fitness values have not yet been computed. Then, the algorithm computes two partial order relations $rank^-$ and $rank^+$. The relation $rank^-$ is computed based on optimistic fitness values ($\hat{F}(p) - \hat{F}^{cl}_{\epsilon}(p)$) for individuals in *Predicted* and actual

Algorithm. NSGAII-SM
Input: - m: Population and archive size /*|A| = |P| = m*/
- g: Maximum number of search iterations
Output: - *BestSolution*: The best solutions found in g iterations.

```
1.  $A = \emptyset$ /*archive*/
2.  $P = \{p_1, \ldots, p_m\}$ /*initial population (randomly selected)*/
3.  for $g$ iterations do
4.    $PredictFitness(P)$ /*For every $p \in P$ and every fitness function $F_i$ s.t.
      $i \in \{1, \ldots, k\}$, compute $\hat{F}_i(p) - \hat{F}^{cl}_{i,\epsilon}(p)$ and $\hat{F}_i(p) + \hat{F}^{cl}_{\epsilon,i}(p)$*/
5.    $Predicted = P$
6.    $Q = P \cup A$ /* |Q| = 2m*/
7.    $rank^-, rank^+ = ComputeRanks(Q)$
8.    $A = \emptyset$
9.    while $|A| < m$ do
10.     $p = BestRanked(Q, rank^-)$
11.     while $p \notin Predicted \wedge |A| < m$ do
12.       $A = A \cup \{p\}$
13.       $Q = Q \setminus \{p\}$
14.       $p = BestRanked(Q, rank^-)$
15.     if $|A| = m$ then break
16.     $p = BestRanked(Predicted, rank^+)$
17.     $ComputeFitness(\{p\})$ /*Run simulation and compute
        actual fitness values $F_1, \ldots, F_k$ for $p$*/
18.     $Predicted = Predicted \setminus \{p\}$
19.     $rank^-, rank^+ = ComputeRanks(Q)$ /*re-rank the remaining
        elements in $Q$ after computing the actual fitness values for $p$.*/
20.   $BestSolution = A$
21.   $P = Breed(A)$ /*breeding a new population from the parent archive*/
22. return $BestSolutionFound$
```

Figure 5: NSGAII-SM Algorithm

fitness values (F) for other individuals. Dually, the relation $rank^+$ is computed based on pessimistic fitness values ($\hat{F}(p) + \hat{F}^{cl}_{\epsilon}(p)$) for individuals in *Predicted* and actual fitness values (F) for other individuals.

Let *rank* be the partial order over Q computed based on the actual fitness values for every element in Q (assuming that the actual fitness values are known for elements in Q). Then, we show the follow lemma.

Lemma. Let $BestRanked(Q, rank^-) \notin Predicted$. Suppose for every $p \in Predicted$ and every fitness function $F_i \in \{F_1, \ldots, F_k\}$, we have $\hat{F}_i(p) - \hat{F}^{cl}_{i,\epsilon}(p) \leq F_i(p)$. Then, $BestRanked(Q, rank^-) = BestRanked(Q, rank)$.

Proof. By our assumption, the actual fitness value for any element $p \in Predicted$ is higher than their optimistic fitness value $\hat{F}_i(p) - \hat{F}^{cl}_{i,\epsilon}(p)$, which is the value used to create $rank^-$. Hence, none of the elements in *Predicted* could be ranked higher than $BestRanked(Q, rank^-)$ when we use the partial order *rank*.■

The above lemma states that assuming that actual fitness values are not better than the optimistic predictions and if $BestRanked(Q, rank^-) \notin Predicted$, then $BestRanked(Q, rank^-)$ is equal to the best ranked element computed in NSGAII where we do not use predictions.

Given the above lemma, in NSGAII-SM, we first add the elements of Q that are ranked best by $rank^-$ and are not in *Predicted* to the archive of best elements A (Lines 10-14). After that, if A is still short of elements (i.e., $|A| < m$), we compute actual fitness values for the predicted element that is ranked highest by $rank^+$, i.e., the partial order based on pessimistic fitness values of predicted elements (Lines 16-18). We then recompute $rank^-$ and $rank^+$ (Line 19), and continue until we select m best elements from Q into A. For all the elements in A, the actual fitness values are already computed (i.e., $A \cap Predicted = \emptyset$).

Upon termination of the while loop (Lines 9-19) in Figure 5, the set $Predicted \subseteq Q$ contains those elements that NSGAII-SM was able to discard without the need to compute the actual fitness values for them. Hence, at each iteration, the size of *Predicted* indicates the number of simulation calls that our algorithm has been able to save. This is in contrast to the original NSGAII algorithm (Figure 4) where at each iteration, simulation is called for m times. According to the above Lemma, if actual fitness values are not better than the optimistic predictions, then NSGAII and NSGAII-SM behave the same. That is, assuming that NSGAII and NSGAII-SM are provided with the same set P at each iteration, they select the same candidate solutions (set A), but NSGAII-SM is likely to perform less simulations per iteration than NSGAII. The probability of actual fitness values being better than the optimistic predictions depends on the confidence level cl. For example, for $cl = 95\%$, with a probability of 2.5%, the actual fitness values are better than their optimistic predictions, and hence, NSGAII-SM might select less optimal solutions compared to NSGAII given the same set P. In Section 6, we empirically compare the quality of the solutions generated by NSGAII-SM and NSGAII in particular by accounting for the randomness factor in generating P and by executing the two algorithms within a limited and realistic time budget.

5. TAILORING SEARCH TO PEVI

The algorithms NSGAII and NSGAII-SM described in Section 4 are generic. We tailor them to our search-based test generation problem by specifying the search input representation, the fitness functions, and the genetic operators.

Input representation. The input space of our search problem consists of vectors $(v^c, x_0, y_0, \theta, v^p)$. The variables v^c, x_0, y_0, θ and v^p and their ranges are defined in Section 2. Each value assignment to the vector $(v^c, x_0, y_0, \theta, v^p)$ represents a *chromosome*, and each value assignment to the variables of this vector represents a *gene*.

Fitness functions. We use the three fitness functions, $D^{min}(p/car)$, $D^{min}(p/awa)$ and TTC^{min}, defined in Section 2 for our search algorithm. A desired solution is expected to minimize these three fitness functions.

Genetic operators. The *Breed*() procedure in the NSGAII and NSGAII-SM algorithms is implemented based on the following operators:

- *Selection.* We use a binary tournament selection with replacement that has been used in the original implementation of NSGAII algorithm [22].

- *Crossover.* We use the Simulated Binary Crossover operator (SBX) [10, 20]. SBX creates two offsprings from two selected parent individuals. The difference between offsprings and parents is controlled by a distribution index (η): The offsprings are closer to the parents when η is large, while with a small η, the difference between offsprings and parents will be larger [21]. In this paper, we chose a high value for η (i.e., $\eta = 20$) based on the guidelines given in [20].

- *Mutation.* Mutation is applied after crossover to the genes of the children chromosomes with a certain probability (mutation rate). Given a gene x (i.e., any of the variables v^c, x_0, y_0, θ and v^p), our mutation operator shifts x by a value x' selected from a normal distribution with mean $\mu = 0$ and variance σ^2. To avoid invalid offsprings, if the result of a crossover or a mutation is greater than the maximum, it is set to the maximum. If the result is below the minimum, it is clamped to the minimum.

6. EVALUATION

In this section, we investigate the following Research Questions (RQs) through our empirical evaluation applied to the PeVi case study.

RQ1. (Comparing Random Search, NSGAII and NSGAII-SM) *How do NSGAII, NSGAII-SM and random search perform compared to one another?* We start by comparing the time performance and the quality of solutions obtained by our test generation strategy when we use NSGAII and NSGAII-SM. Our goal is to determine whether NSGAII-SM is able to generate results with higher quality than those obtained by NSGAII within the same time period. We then compare the algorithm that performs better, between NSGAII and NSGAII-SM, with a random test generation algorithm (the baseline of comparison typically adopted in SBSE research [32]).

RQ2. (Usefulness) *Does our approach help identify test scenarios that are useful in practice?* This question investigates whether the test scenarios generated by our approach were useful for the domain experts and how they compared with the test scenarios that have been previously devised by manual testing based on domain expertise.

Metrics. We evaluate the prediction accuracy of surrogate models using the coefficient of determination (R^2) [56] that measures the predictive power of a surrogate model by identifying how well a test set fits the model. Specifically, R^2 measures the proportion of the total variance of F explained by $\hat{F}$ for the observations in the test set where F is a fitness function and $\hat{F}$ is its corresponding predictive function. The value of R^2 ranges between 0 and 1. The higher the value of R^2, the more accurate the surrogate model is.

To assess and compare the quality of Pareto fronts obtained by our alternative search algorithms, we use two well-known quality indicators [36], hypervolume (HV) and generational distance (GD). HV [59] measures the volume in the solution space that is covered by members of a non-dominated set of solutions. The larger the volume (i.e., the higher the value of HV), the better the results of the algorithm. GD [52] compares the pareto front solutions computed by an algorithm with an *optimal pareto front* (or *true pareto front*), i.e., the best non-dominated solutions that exist in a given space of solutions for a given problem. In particular, GD [52] is the average distance between each point in a computed Pareto front and the closest optimal pareto front solution to that point. A value of 0 for GD indicates that all the obtained solutions by a search algorithm are optimal. The lower GD, the better the results of the algorithm. Computing an optimal pareto front is usually not feasible. As suggested in the literature [53], instead, we use a reference pareto front that is a union of all the non-dominated solutions computed by our search algorithms (i.e., NSGAII, NSGAII-SM and random search). The HV and GD are selected from the combination and convergence quality indicator categories, respectively. As discussed in [53], to assess the quality of computed pareto fronts with respect to combination and convergence indicators, it is sufficient to choose only one indicator from each of these two categories.

Experiment Design. We implemented the NSGAII and NSGAII-SM algorithms and the neural network surrogate models in Matlab. In addition, we implemented a test generation strategy based on random search. Random search [38] and our NSGAII-based search algorithms require to interact with PreScan to execute simulations of the Simulink models of the pedestrian, the car and the PeVi system embedded into the car (see Section 2). The NSGAII-SM algorithm, in addition to calling the PreScan simulator, calls neural networks that are developed to serve as surrogate models. We ran all the experiments on a laptop with a 2.5 GHz CPU and 16GB of memory. Based on our experiments, each PeVi simulation (i.e., each call to PreScan), on average, takes 2 min with a min value of 1.2 min and a max value of 3.4 min. The simulation time variations are due to the variations in the car and the pedestrian speeds and positions. Further, we may stop simulations before completion at a point where we can conclusively determine the fitness function values.

To answer our research questions, we designed and performed the following experiment. First, we identified the training algorithm and the configuration values that lead to the most accurate neural network-based surrogate models for the PeVi case study. To do so, for each of our three PeVi fitness functions, we compared 18 different neural network configurations. The comparison is based on a k-fold cross validation with $k = 5$ [4, 24]. Specifically, we first selected (using adaptive random search [38]) 1000 observation points from the input search space of the PeVi system. Adaptive random search was used to maximize diversity in our training and test sets. It is an extension of the naive random search that attempts to maximize the Euclidean distance between the points selected in the input space. Recall from Section 2 that each PeVi input point is a vector $(v^c, x_0, y_0, \theta, v^p)$ selected from a five dimensional space. We simulated each point to obtain the actual values for each fitness function. In the experiment, we refer to fitness function $D^{min}(p/car)$ by F_1, to $D^{min}(p/awa)$ by F_2, and to TTC^{min} by F_3. The 1000 observation points are randomly partitioned into five disjoint subsets with 200 points in each. We then randomly selected four subsets to create a training set with 800 points, and the remaining subset is used as the test set. This process is repeated for five times so that for each 5-fold cross validation, the R^2 values are computed on a test set containing the entire 1000 points. To account for randomness, we repeated the 5-fold cross validation ten times.

To develop neural networks with high prediction accuracy, we considered three training algorithms, BR, LM and SCG, and we set different values to the following parameters: the number of hidden layers (nl), the number of neurons in each hidden layer (nn) and the number of epochs (ne). Specifically, we set $nl = 2$ as is common in the literature [28, 35]. There are various recommendations for setting nn. In particular, nn is recommended to be less than twice or equal to $\frac{2}{3}$ of the size of the input vector [9, 35], or to be a number between the input and the output size [9, 35]. We considered the values 3 and 4 for nn. In addition, we considered the value 100 for nn because in some cases the accuracy may improve when nn is set to values considerably larger than the input size [49]. Finally, we set ne to 10 and 100.

In total, we developed and trained 18 different neural network configurations. We computed R^2 for 10 different repetitions of 5-fold cross validations of the 18 neural network configurations related to our three fitness functions. We selected the following neural network configurations with highest predictive accuracy (highest R^2) for our three fitness functions: For F_1, we selected the configuration that was developed by the BR algorithm with $nn = 100$ and $ne = 100$ ($R^2 = 0.99$). For F_2, we selected the configuration that was developed by the BR algorithm with $nn = 100$ and $ne = 100$ ($R^2 = 0.84$), and for F_3, we selected the configuration that was developed by the LM algorithm with $nn = 3$ and $ne = 100$ ($R^2 = 0.89$). Note that $R^2 = 0.84$ indicates that 84% of the variance in the test set is explained by the predictive model. The high R^2 values of the selected configurations indicate their high predictive accuracy.

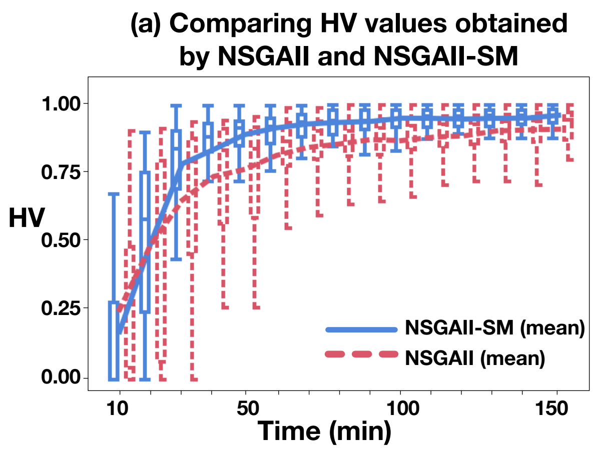
Having obtained the most accurate surrogate models to be used by the NSGAII-SM algorithm, we now discuss value selection for the search algorithms' parameters. Since our experiments, which involve running physics-based simulations, are very time-intensive, we were not able to systematically tune the search parameters (e.g., based on the guidelines provided in [6]). Instead, we selected the parameters based on some small-scale preliminary experimentations as well as existing experiences with multi-objective search algorithms. Specifically, we set the crossover rate to 0.9, the mutation rate to 0.5 and the population size to 10. Our choice for the crossover rate is within the suggested range of $[0.45, .., 0.95]$ [12]. Our preliminary experimentations showed that more explorative search may lead to better results. Hence, we set the mutation rate to 0.5 which is higher than the suggest value of $1/l$ where l is the length of chromosome [6]. Finally, we chose a relatively small population size to allow for more search iterations (generations) within a fixed amount of time.

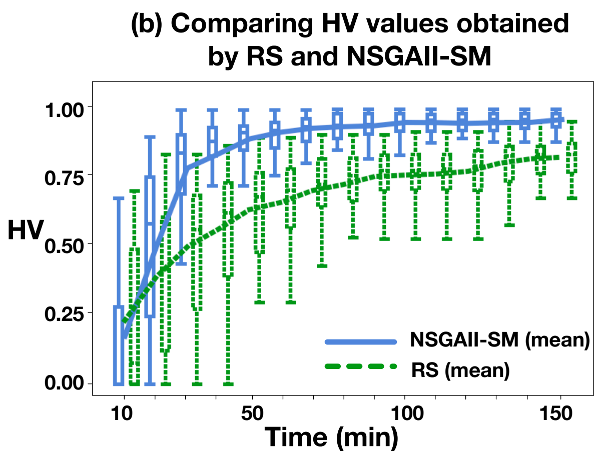
Results. Next, we discuss our RQs:

RQ1 (Comparing Random Search, NSGAII and NSGAII-SM). To answer RQ1, we ran Random search, NSGAII and NSGAII-SM (with cl =.95) 40 times for 150 min. We computed the HV and the GD values for the pareto front solutions obtained by these alternative search algorithms at every 10 min interval from 0 to 150 min. We used the resulting HV and GD values and the changes in these values over time to first compare NSGAII and NSGAII-SM by focusing on their performance when the algorithms are executed within a practical execution time budget (i.e., 150 min). Second, we compare the better algorithm between NSGAII and NSGAII-SM with Random search. To statistically compare the HV values, we performed the non-parametric pairwise Wilcoxon Paired Signed Ranks test [16], and calculated the effect size using Cohen's d [18]. The level of significance (α) was set to 0.05, and, following standard practice, d was labeled "small" for $0.2 \leq d < 0.5$, "medium" for $0.5 \leq d < 0.8$, and "high" for $d \geq 0.8$ [18].

Comparing NSGAII and NSGAII-SM. Figure 6(a) shows the HV values obtained by 40 runs of NSGAII and NSGAII-SM up to 150 min. As shown in the figure, at the beginning both NSGAII and NSGAII-SM are highly random. After executing these two algorithms for 50 min, the degree of variance in HV values across NSGAII-SM runs reduces faster than the degree of variance in HV values across NSGAII runs. Further, the average HV values obtained by NSGAII-SM grows faster than the average HV values obtained by NSGAII. After executing the algorithms for 120 min, both search algorithms converge towards their pareto optimal solutions and the difference in average HV values between the two algorithms tends to narrow.

We note that the differences between the HV distributions of NSGAII and NSGAII-SM are not statistically significant. This is likely because the number of runs is rather small (40), thus yielding low statistical power. However, as shown in Figure 6(a), the medians and averages of the HV values ob-





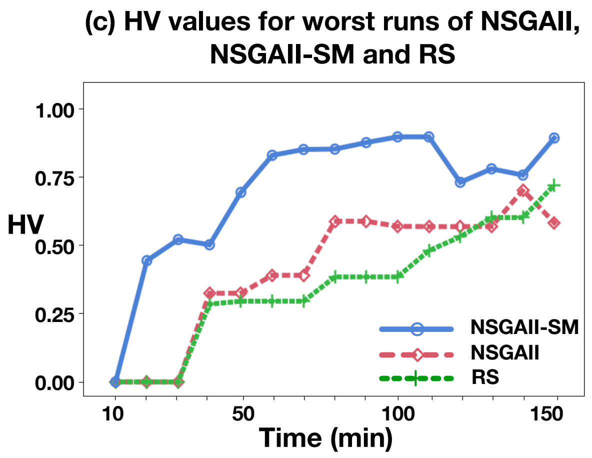


Figure 6: Comparing HV values obtained by (a) 40 runs of NSGAII and NSGAII-SM (cl=.95); (b) 40 runs of random search and NSGAII-SM (cl=.95); and (c) the worst runs of NSGAII, NSGAII-SM (cl=.95) and random search.

tained by NSGAII-SM are higher than the medians and averages of the HV values obtained by NSGAII. Given the large execution time of our test generation algorithm, in practice, testers will likely have the time to run the algorithm only once. With NSGAII, certain runs really fare poorly, even after the initial 50 min of execution. Figure 6(c) shows the HV results over time for the worst run of NSGAII, NSGAII-SM and Random search among our 40 runs. As shown in the figure, the worst run of NSGAII yields remarkably lower HV values compared to the worst run of NSGAII-SM. With NSGAII, the tester might be unlucky, and by running the algorithm once, obtain a run similar to the worst NSGAII run in Figure 6(c). Since the worst run of NSGAII-SM fares much better than the worst run of NSGAII, we can consider NSGAII-SM to be a safer algorithm to use, especially under tight time budget constraints. We note that as shown in Figure 6(c) the HV values do not necessarily increase monotonically over time. This is because HV may decrease when, due to the crowding distance factor, solutions in sparse areas but slightly away from the reference pareto front are favored over other solutions in the same pareto front rank [46]. We further compared the GD values obtained by NSGAII and NSGAII-SM for 40 runs of these algorithms up to 150 min. Similar to the HV results, after 50 min executing these algorithms, the average GD values obtained by NSGAII-SM is better than the average GD values obtained by NSGAII.

In addition, we compared the average number of simulations per generation performed by NSGAII and NSGAII-SM. As expected the average number of simulations per generation for NSGAII is equal to the population size (i.e., ten). For NSGAII-SM, this average is equal to 7.9. Hence, NSGAII-SM is able to perform more generations (iterations) than NSGAII within the same execution time. As discussed in Section 4, for $cl = 95\%$, at a given iteration and provided with the same set P, NSGAII-SM behaves the same as NSGAII with a probability of 97.5%, and with a probability of 2.5%, NSGAII-SM produces less accurate results compared to NSGAII. Therefore, given a fixed execution time, NSGAII-SM is able to perform more iterations than NSGAII, and with a high probability ($\approx 97.5\%$), the solutions generated by NSGAII-SM at each iteration are likely to be as accurate as those would have been generated by NSGAII. As a result and as shown in Figure 6(a), given the same time budget, NSGAII-SM is able to produce more optimal solutions compared to NSGAII.

Finally, we compared NSGAII-SM with three different confidence levels, i.e., $cl = 0.95$, 0.9 and 0.8. The HV and GD results indicated that NSGAII-SM performs best, and better than NSGAII, when cl is set to 0.95.

Comparing with Random Search. Figure 6(b) shows the HV values obtained by Random search and NSGAII-SM. As shown in the figure, after 30 min execution, NSGAII-SM yields better HV results compared to Random search. The HV distributions obtained by running NSGAII-SM after 30 min and until 150 min are significantly better (with a large effect size) than those obtained by Random search. Similarly, we compared the GD values obtained by NSGAII-SM and Random search. The GD distributions obtained by NSGAII-SM after 30 min and until 150 min are significantly better than those obtained by Random search with a large effect size at 100 min and 110 min and otherwise a medium effect size at other times.

To summarize, when the search execution time is larger than 50 min, NSGAII-SM outperforms NSGAII. With less than 50 min execution time, both algorithms show a high degree of randomness. When engineers cannot afford to run the test generation algorithm for a long time, for example because they make a change to the PeVi system and need to rerun the test execution procedure frequently, NSGAII-SM is more likely to provide close to optimal solutions compared to NSGAII. Further, as shown in Figure 6(c), the worst run of NSGAII-SM performs considerably better than the worst run of NSGAII. Finally, NSGAII-SM is able to find significantly better solutions compared to Random search.

RQ2 (Usefulness). To demonstrate practical usefulness of our approach, we have made available at [2] some test scenario examples obtained by our NSGAII-based test generation algorithms. We presented these test scenarios as well as other scenarios to domain experts at our partner company. The scenarios were generated for various stressful weather conditions (e.g., fog, snow and rain) and for situations where roadside objects block the camera's field of view or when ramped and curved roads may interfere with the pedestrian detection function of PeVi. In all the example scenarios at [2] either PeVi fails to detect a pedestrian that appears in the red warning area (AWA) in front of a car, or the detection happens very late and very close to the collision time. As confirmed by our domain expert, such scenarios had not been previously developed by manual testing based on domain expertise. These scenarios particularly helped engineers identify particular car speed and pedestrian speed ranges and pedestrian orientations for which the PeVi's detection function is more likely to fail. In addition, the light scene intensity, the light orientation and reflection may impact the detection capabilities of pedestrian detection algorithms. However, due to the current imitations of PreScan (the PeVi simulation tool) discussed earlier, we were not able to define fitness functions related to the scene light intensity.

To summarize, our NSGAII-based test generation algorithms are able to identify several critical behaviors of the PeVi systems that have not been previously identified based on manual and expertise-based simulations.

7. RELATED WORK

Search-based testing has largely focused on unit testing and has rarely been used for system testing. Exceptions include GUI testing [29, 41] and the generation of system test cases to exercise non-functional behaviors such as quality-of-service constraints [48], computational resources consumption and deadline misses [13]. Embedded software systems and their environments are prevalently captured and simulated using physics-based models such as those captured by MATLAB/Simulink. Some of the test automation techniques for MATLAB/Simulink use metaheuristic search to guide testing towards the maximisation of coverage criteria [55, 57], for example path coverage [57], or towards the generation of input signals that satisfy certain signal shape patterns [7] or temporal constraints [54]. These testing strategies have mostly focused on unit/function-level testing, aiming to maximize coverage or diversity of test inputs. These strategies, however, are inadequate for testing complex physics-based dynamic models such as those used in our case study. Our testing approach, in contrast, is driven by system-level requirements as well as critical and stressful situations of the system and its environment.

Similar to our work, surrogate modeling has been previously used to approximate expensive fitness computations and simulations in the context of evolutionary and metaheuristic search algorithms. Surrogate modeling has been applied to scale up search-based solutions to optimization problems in avionics [45], chemical systems [15], and the medical domain [23]. In particular, combination of surrogate modeling and multi-objective population-based search algorithms has been applied to optimization problems in mobile ad hoc networks [25], manufacturing [50], and optimizing energy consumption in buildings [40]. These techniques, however, solely rely on surrogate model predictions to select best candidate solutions without using the prediction errors and confidence levels. This may lead to a significant deviation between the best solutions selected based on surrogate model predictions and those solutions that would have been selected based on actual fitness computations. In contrast, in our work, we use the prediction errors to decide whether we should compute actual fitness values for candidate solutions or not. Further, we show that when actual fitness values are not better than their respective optimistic predictions, NSGAII and NSGAII-SM behave the same, but NSGAII-SM is likely to call less simulations per iteration than NSGAII. In [42], surrogate modeling has been used in conjunction with single-objective local search such that prediction errors and actual fitness values are used to ensure the search algorithm accuracy. Our work differs from the work of [42] as we combine surrogate modeling with multi-objective population search algorithms.

8. CONCLUSION

Physics-based simulation tools provide feasible and practical test platforms for control and perception software components developed for self-driving cars. We identified the following two key challenges that hinder systematic testing based on these simulation tools: (1) These tools lack the guidance and automation required to generate test cases that would be likely to uncover faulty behaviors, and (2) executing individual test cases is very time-consuming. We proposed an approach based on combination of multi-objective search and neural networks. We developed meta-heuristics capturing critical aspects of the system and its environment to guide the search towards exercising behaviors that are likely to reveal faults. Our proposed search algorithm relies on neural network predictions to bypass actual costly simulations when predictions are sufficient to conclusively prune certain solutions from the search space. Our evaluation performed on an industrial system shows that (1) our search-based algorithm outperforms random test generation, (2) combining our search algorithm with neural networks improves the quality of the generated test cases under a limited and realistic time budget, and (3) our approach is able to identify critical system and environment behaviors.

Due to the current technical limitations of the simulation tool that we used in our study, we had to rely on a subset of the PeVi input elements, i.e., the properties of the vehicle and the pedestrian, to generate test cases. Our approach, however, is general and can account for various critical properties of the environment once the technical limitations of our current simulation tool are resolved. In future, we plan to perform more experiments and to further improve our search algorithm by dynamically refining the neural network models using simulations performed during the search.

9. ACKNOWLEDGEMENT

We gratefully acknowledge funding from IEE S.A. Luxembourg and from Fonds National de la Recherche, Luxembourg under grant FNR/P10/03 - Verification and Validation Lab.

10. REFERENCES

[1] TASS International. PreScan simulation of ADAS and active safety. https://www.tassinternational.com/prescan. Last accessed: March 2016.

[2] Test scenarios. https://sites.google.com/site/testingpevi.

[3] The MathWorks Inc. Simulink. http://www.mathworks.nl/products/simulink. Last accessed: April 2016.

[4] C. Alippi and M. Roveri. Virtual k-fold cross validation: An effective method for accuracy assessment. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'10)*, pages 1–6, 2010.

[5] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[6] A. Arcuri and G. Fraser. On parameter tuning in search based software engineering. In *Proceedings of the International Symposium on Search Based Software Engineering (SSBSE'11)*, pages 33–47, 2011.

[7] A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'03)*, pages 2428–2441, 2003.

[8] R. Barton. Metamodeling: a state of the art review. In *Proceedings of the conference on Winter simulation (WSC'94)*, pages 237–244, 1994.

[9] R. N. Behera. Artificial neural network: A soft computing application in biological sequence analysis. *International Journal of Computational Engineering Research*, 4(4):1–13, 2014.

[10] H.-G. Beyer and K. Deb. On self-adaptive features in real-parameter evolutionary algorithms. *Transactions on Evolutionary Computation*, 5(3):250–270, 2001.

[11] A. J. Booker, J. Dennis Jr, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, 1999.

[12] M. Bowman, L. C. Briand, and Y. Labiche. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837, 2010.

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

[14] L. C. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli. Testing the untestable: model testing of complex software-intensive systems. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 789–792, 2016.

[15] J. A. Caballero and I. E. Grossmann. An algorithm for the use of surrogate models in modular flowsheet optimization. *AIChE journal*, 54(10):2633–2650, 2008.

[16] J. A. Capon. *Elementary Statistics for the Social Sciences: Study Guide*. Wadsworth Publishing Company, 1991.

[17] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems. Genetic and Evolutionary Computation*. Kluwer Academic, 2007.

[18] J. Cohen. *Statistical power analysis for the behavioral sciences (rev)*. Lawrence Erlbaum Associates, Inc, 1977.

[19] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, Chichester, New York, 2001.

[20] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[21] K. Deb and H.-g. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, 9(2):197–221, 2001.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[23] D. Douguet. e-LEA3D: a computational-aided drug design web server. *Nucleic Acids Research*, 38:615–621, 2010.

[24] B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[25] D. Efstathiou, P. McBurney, S. Zschaler, and J. Bourcier. Efficient multi-objective optimisation of service compositions in mobile ad hoc networks using lightweight surrogate models. *Journal of Universal Computer Science*, 20(8):1089–1108, 2014.

[26] D. Emadi and M. Mahfoud. Comparison of artificial neural network and multiple regression analysis techniques in predicting the mechanical properties of {A3} 56 alloy. *Procedia Engineering*, 10:589–594, 2011.

[27] F. Ferrucci, M. Harman, J. Ren, and F. Sarro. Not going to take this anymore: multi-objective overtime planning for software engineering projects. In *Proceedings of the International Conference on Software Engineering (ICSE'13)*, pages 462–471, 2013.

[28] S. Goyal and G. K. Goyal. Article: Study on single and double hidden layers of cascade artificial neural intelligence neurocomputing models for predicting sensory quality of roasted coffee flavoured sterilized drink. *International Journal of Applied Information Systems*, 1(3):1–4, 2012.

[29] F. Gross, G. Fraser, and A. Zeller. Search-based system testing: High coverage, no false alarms. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'12)*, pages 67–77, 2012.

[30] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *Neural Networks, IEEE Transactions on*, 5(6):989–993, 1994.

[31] M. Hall, I. Witten, and E. Frank. *Data mining: Practical machine learning tools and techniques (3rd edition)*. Morgan Kaufmann, 2011.

[32] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11, 2012.

[33] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition, 1998.

[34] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[35] S. Karsoliya. Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture. *International Journal of Engineering Trends and Technology*, 3(6):713–717, 2012.

[36] J. Knowles, L. Thiele, and E. Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. Technical report, Computer Engineering and Networks Laboratory of Zurich, 2006.

[37] A. Lindgren and F. Chen. State of the art analysis: An overview of advanced driver assistance systems (adas) and possible human factors issues. *Human factors and economics aspects on safety*, pages 38–50, 2006.

[38] S. Luke. *Essentials of Metaheuristics.* Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[39] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[40] L. Magnier and F. Haghighat. Multiobjective optimization of building design using TRNSYS simulations, genetic algorithm, and artificial neural network. *Building and Environment*, 45(3):739–746, 2010.

[41] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro. Automatic testing of gui-based applications. *Software Testing Verification and Reliability*, 24(5):341–366, 2014.

[42] R. Matinnejad, S. Nejati, L. Briand, and T. Brcukmann. MiL testing of highly configurable continuous controllers: scalable search using surrogate models. In *Proceedings of the International Conference on Automated Software Engineering (ASE'14)*, pages 163–174, 2014.

[43] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[44] N. Nguyen and A. Cripps. Predicting housing value: A comparison of multiple regression analysis and artificial neural networks. *Journal of Real Estate Research*, 22(3):313–336, 2001.

[45] Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*, 41(4):687–696, 2003.

[46] F. Peng and K. Tang. Alleviate the hypervolume degeneration problem of NSGA-II. In *Proceedings of the International Conference on Neural Information Processing (ICONIP'11)*, pages 425–434, 2011.

[47] A. S. Sayyad and H. Ammar. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *Proceedings of the International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'13)*, pages 21–27, 2013.

[48] M. Shams, D. Krishnamurthy, and B. Far. A model-based approach for testing the performance of web applications. In *Proceedings of the International Workshop on Software Quality Assurance (SOQUA'06)*, pages 54–61, 2006.

[49] K. G. Sheela and S. N. Deepa. Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering*, 2013:1–11, 2013.

[50] A. Syberfeldt, H. Grimm, A. Ng, and R. I. John. A parallel surrogate-assisted multi-objective evolutionary algorithm for computationally expensive optimization problems. In *Proceedings of the Congress on Evolutionary Computation (CEC'08)*, pages 3177–3184, 2008.

[51] R. van der Horst and J. Hogema. Time-to-collision and collision avoidance systems. In *Proceedings of the workshop of the International Cooperation on Theories and Concepts in Traffic Safety (ICTCT'93)*, pages 109–121, 1993.

[52] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Air Force Institute of Technology, 1998.

[53] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the International Conference on Software Engineering (ICSE'16)*, 2016.

[54] B. Wilmes and A. Windisch. Considering signal constraints in search-based testing of continuous systems. In *Proceedings of the International Conference on Software Testing, Verification, and Validation Workshops (ICSTW'10)*, pages 202–211, 2010.

[55] A. Windisch. Search-based test data generation from stateflow statecharts. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, pages 1349–1356, 2010.

[56] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., 3rd edition, 2011.

[57] Y. Zhan and J. A. Clark. The state problem for test generation in simulink. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 1941–1948, 2006.

[58] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[59] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *Transactions on Evolutionary Computation*, 3(4):257–271, 1999.