Titre: Title:	A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems
Auteur: Author:	Dmytro Humeniuk
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Humeniuk, D. (2021). A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/9144/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems

DMYTRO HUMENIUK
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems

présenté par **Dmytro HUMENIUK**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Gabriela NICOLESCU, présidente
Foutse KHOMH, membre et directeur de recherche
Giuliano ANTONIOL, membre et codirecteur de recherche
Mohammad HAMDAQA, membre

DEDICATION

To my family, especially my grandfather,
who introduced me to the world of engineering ...

ACKNOWLEDGEMENTS

In the first place I would like to thank my research director Foutse Khomh, for believing in me and accepting to be my supervisor in this study program. Throughout my studies he was very supportive of my research work and always had time to discuss it, giving his guidance, no matter how busy he was.

Further I am very grateful to my co-director Giuliano Antoniol. He was always there to help me, not hesitating to share a piece of code or provide some very useful resources.

I am also thankful to all the SWAT research team members for insightful discussions during the meetings and on-line, especially to Amin Nikanjam, who would always provide very detailed answers to all my questions.

Last, but not least, I would like to thank my committee members, Dr. Nicolescu Gabriela and Dr. Hamdaqa Mohammad, for accepting to evaluate my master's thesis.

RÉSUMÉ

De nombreux systèmes cyber-physiques modernes intègrent des technologies de vision par ordinateur, des capteurs complexes, et des logiciels de contrôle avancés, leur permettant d'interagir de manière autonome avec l'environnement. Les exemples incluent les essaims de drones, les véhicules autonomes, les robots autonomes, etc. Tester de tels systèmes pose de nombreux défis : non seulement les entrées du système doivent être variées, mais également l'environnement doit être pris en compte. De nombreux outils ont été développés pour tester les modèles de tels systèmes pour les entrées possibles falsifiant leurs exigences. Cependant, ils ne sont pas directement applicables aux systèmes cyber-physiques autonomes, car les entrées pour de tels systèmes sont générées à partir d'environnements virtuels.

Dans ce travail, nous proposons AmbieGen, un cadriciel pour la generation de scénarios de test défiant, pour les systèmes cyber-physiques autonomes. Les scénarios représentent un environnement dans lequel un agent autonome navigue. Ce cadriciel est applicable à la génération de différents types d'environnements.

Pour générer les scénarios de test, nous utilisons l'algorithme NSGA-II avec deux objectifs. Le premier objectif évalue l'écart du comportement du système observé par rapport à son comportement attendu. Le deuxième objectif vise a maximiser la diversité des cas de test, calculée comme une distance de Jaccard avec un cas de test de référence. Pour guider le premier objectif, nous utilisons un modèle de système simplifié plutôt que le modèle complet. Le modèle complet est utilisé pour exécuter le système dans un environnement de simulation et peut prendre un temps considérable à exécuter (plusieurs minutes pour un scénario). Le modèle de système simplifié est dérivé du modèle complet et peut être utilisé pour obtenir une approximation des résultats à partir du modèle complet sans exécuter la simulation.

Nous évaluons AmbieGen sur trois cas de génération de scénarios: un thermostat intelligent, un système d'évitement d'obstacles robotique, et un système d'aide au maintien dans la voie pour les véhicules. Pour toutes les études de cas, notre approche surpasse les approches existantes avec le code disponible en ligne en détection de défauts et autres métriques telles que la diversité des défauts révélés et la proportion de scénarios valides.

AmbieGen a pu trouver des scénarios, révélant des défaillances pour les trois agents autonomes considérés dans nos études de cas. Nous avons comparé trois configurations d'AmbieGen: basée sur un algorithme génétique à objectif unique, à objectif multple et sur recherche aléatoire. Les configurations à objectif unique et à objectif multiple surpassent la recherche aléatoire. La configuration à objectif multiple peut trouver les individus de même niveau de

qualité que l'objectif unique, produisant plus des scénarios uniques, pour le même budget de temps. Notre cadriciel peut être utilisé pour générer des environnements virtuels de différents types et complexité et révéler les défauts du système au début de l'étape de la conception.

ABSTRACT

Many modern cyber-physical systems incorporate computer vision technologies, complex sensors and advanced control software, allowing them to interact with the environment autonomously. Examples include drone swarms, self-driving vehicles, autonomous robots, etc. Testing such systems poses numerous challenges: not only should the system inputs be varied, but also the surrounding environment should be accounted for. A number of tools have been developed to test the system model for the possible inputs falsifying its requirements. However, they are not directly applicable to autonomous and vision based cyber-physical systems, as the inputs to their models are generated while operating in a virtual environment.

In this work, we aim to design a search-based framework, named AmbieGen, for generating diverse fault-revealing test scenarios for vision-based and autonomous cyber-physical systems. The scenarios represent an environment in which an autonomous agent navigates. The framework is applicable to generating different types of environments.

To generate the test scenarios, we leverage the NSGA-II algorithm with two objectives. The first objective evaluates the deviation of the observed system's behaviour from its expected behaviour. The second objective is maximization of the test case diversity, calculated as a Jaccard distance with a reference test case. To guide the first objective we are using a simplified system model rather than the full model. The full model is used to run the system in the simulation environment and can take substantial time to execute (several minutes for one scenario). The simplified system model is derived from the full model and can be used to get an approximation of the results obtained from the full model without running the simulation.

We evaluate AmbieGen on three scenario generation case studies, namely a smart-thermostat, a robot obstacle avoidance system, and a vehicle lane-keeping assist system. For all the case studies, our approach outperforms the available baselines in fault revealing and several other metrics such as the diversity of the revealed faults and the proportion of valid test scenarios.

AmbieGen could find scenarios, revealing failures for all the three autonomous agents considered in our case studies. We compared three configurations of AmbieGen: based on a single objective genetic algorithm, multi-objective, and random search. Both single and multi objective configurations outperform the random search. Multi objective configuration can find the individuals of the same quality as the single objective, producing more unique test scenarios in the same time budget. Our framework can be used to generate virtual environments of different types and complexity and reveal the system's faults early in the design stage.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

CPS	Cyber-Physical System
GA	Genetic Algorithm
MOEA	Multi Objective Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm
RS	Random Search
SO	Single Objective
MO	Multi Objective
SBSE	Search-Based Software Engineering
SUT	System Under Test
DNN	Deep Neural Network
XML	Extensible Markup Language
ADAS	Advanced Driver Assistance System
AEB	Automated Emergency Braking system
LKAS	Lane Keeping Assist System
PPP	Path Planning Problem
UAV	Unmanned Aerial Vehicle

CHAPTER 1 INTRODUCTION

1.1 Context and Motivation

One of the rapidly developing families of cyber-physical systems (CPS) are autonomous and vision based CPS. Examples include drone swarms, self driving cars, cave or underwater exploring robots. Typically, in the CPS development process the systems are validated and verified according to the V-model approach [1]. Prior to running the tests on a real system, the V-model includes model-in-the-loop and software-in-the-loop testing stages. In these stages the simulations of the system are run in a virtual environment. The goal is to model the real environment effect on CPS(s) and generate test scenarios violating some critical properties of CPS. However, during these simulations, engineers often lack tool support for generating the scenarios [2]. For particular applications there exist content generation techniques, like a Kruskal's algorithm for maze generation [3] or pre-configured scenarios, like the virtual worlds used in computer games [4]. However, they do not always provide the needed scenario complexity and oftentimes the scenarios have to be designed manually. Consider an autonomous robotic system, that should navigate to a goal destination in an environment with obstacles. The robots interact with the physical world via sensors and actuators in a feedback loop, avoiding the obstacles and searching the goal destination. Test scenario for such system includes virtual environment, obstacle positions, moving and unexpected obstacles, changing terrain structure and environmental conditions. Manually designing all the possible scenarios in the virtual environment is a tedious task.

1.2 Research objectives

The high level goal of our study is to design a search-based framework for generating diversified fault-revealing test scenarios for vision-based and autonomous cyber-physical systems. The framework should be applicable to generating different types of environments. It should aid the researchers in creating the test environments automatically optimizing them in order to reveal the system faults more effectively. It includes such sub-objectives as:

1. Definition of the fitness functions, scenario encoding and search operators allowing to represent and evolve different types of environments for autonomous CPS systems;
2. Selection of a search strategy allowing to find more fault revealing and diversified test scenarios given a certain time budget. In our work we are considering random, single

(GA), and multi objective (NSGA-II) search algorithms.

1.3 Thesis overview

We develop a prototype of a search based framework, further referred to as "AmbieGen", to automatically generate test scenarios for cyber-physical systems. In the literature, typically one objective, accounting for the scenario fault revealing power [5], or two objectives, accounting for fault revealing power and diversity [6], are used. To evaluate the contribution of adding the first and the second objectives, we consider three configurations of AmbieGen: based on random search, single-objective genetic algorithm (AmbieGen SO) and multi-objective genetic algorithm NSGA-II (AmnbieGen MO). Preliminary results confirm that using the two objectives, maximizing both: scenario fault revealing power and diversity, allows to find more unique fault revealing scenarios given the same time budget. To calculate the first objective, maximizing the fault revealing power, we are using the simplified system model, derived from the full model of the system, as suggested by Menghi et al. [7]. This allows to reduce the computational and time resources needed to produce the test scenarios. The full model is used to execute the scenarios in the simulation environment and is usually computationally expensive. The simplified model doesn't require the simulator to run and provides the approximated outputs of the full model in the reduced amount of time.

We evaluate our approach on three test generation case studies. In the case studies we are testing autonomous agents with control algorithms of different complexity. In the first case study, the goal is to generate the temperature schedule and a combination of environmental conditions, so that a smart-thermostat agent, using a simple PID controller, cannot follow the schedule with expected precision. For the second case study, the aim is to design a navigation map with obstacles for an autonomous robot, maximizing the difficulty for the robot to reach the goal. The autonomous robot is using a nearness diagram control technique for navigation [8]. For the third, the task is to create a virtual road, that forces the self-driving car model to drive off its lane without creating invalid roads. The car uses a Deep Neural Network (DNN) model to infer the control signals from the images of the surrounding environment. AmbieGen could reveal on average 14 failures in two hours and 40 failures in 5 hours for the self-driving car model. It also revealed 9 failures in two hours for the autonomous robot model. Given the same evaluation budget, in all the problems, multi-objective and single-objective configurations of AmbieGen produced fitter solutions with a large effect size, in comparison to the random search. Multi-objective configuration allows to produce more diverse scenarios with medium to large effect size in comparison to single-objective, finding the solutions of the same or similar quality.

1.4 Thesis contributions

This thesis presents the following contributions:

1. We design a prototype of a search-based framework for generating customized environments for testing autonomous and vision based CPS.
2. We propose a novel technique for generating the virtual roads and robot navigation maps.

Researchers and practitioners can leverage AmbieGen to automatically generate scenarios for autonomous CPS that will be further passed to the simulators to run tests on the full CPS models.

The code for replication of all of our experiments is available at [9].

1.5 Novelty of the work and main achievements

A number of frameworks exist for generating test scenarios for testing perceptive components of autonomous cyber-physical systems. They generate the static images of the environment to test the DNN model controlling the system. However, to test the perceptive as well as the control part of the system, dynamic scenarios, where the environment is changing in time should be used. To the best of our knowledge, no framework exists for generating dynamic scenarios for testing different types of autonomous cyber-physical systems. Existing frameworks are fine tuned for testing a specific autonomous CPS.

The novelty of our work, is in designing a framework that can be used to generate virtual environments and dynamic scenarios for different types of autonomous CPS. It leverages evolutionary search algorithm, where the scenario representation and search operators can be customized to evolve different virtual environments. In this work we demonstrate how the framework prototype can be applied to generate test scenarios for thee different autonomous CPS and reveal failures.

Earlier studies in the thesis were published/submitted as follows:

1. *A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems*

   Humeniuk, D., Antoniol, G. and Khomh, F., Submitted to the Information and Software Technology, July 2021.

2. *SWAT tool at the SBST 2021 Tool Competition*

   Humeniuk, D., Antoniol, G. and Khomh, F., In Proceedings of the IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST) (pp. 42-43), May 2021.

3. *Data Driven Testing of Cyber Physical Systems*

   Humeniuk, D., Antoniol, G. and Khomh, F., In Proceedings of the IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST) (pp. 16-20), May 2021.

4. *Double Cycle Hybrid Testing of Hybrid Distributed IoT System*

   Zid, C., Humeniuk, D., Khomh, F. and Antoniol, G., In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (pp. 529-532), June 2020.

1.6 Thesis plan

The remainder of the thesis is organized as follows. In Chapter 2 we present the core concepts of genetic algorithms and their applications. Chapter 3 discusses the related works in the domain of CPS testing. In Chapter 4, we formalize the problem of scenario generation and provide the description of the AmbieGen framework. Chapter 5 describes the test generation case studies used to evaluate our approach. In Chapter 6 we formulate the research questions and our evaluation methodology. The same chapter reports our results and answers to research questions. We also discuss the results and the main challenges of this study. The possible threats to the validity of our results are described in Chapter 7. Chapter 8 concludes the thesis and discusses some avenues for future works.

CHAPTER 2 BACKGROUND

2.1 Search based optimization algorithms

Many activities in software engineering can be stated as optimization problems. The domain that aims to formulate software engineering problems as search problems is known as Search-based Software Engineering (SBSE). The evolutionary search algorithms are one of the most commonly used SBSE tools. Their main advantage is the possibility to perform a global search by sampling many points in the search space at once [10].

Applying an evolutionary SBSE to a software engineering problem requires three main components [11]:

1. a solution representation;
2. a mechanism for making changes to that solution;
3. means of measuring the solution's quality.

The representation encodes candidate solutions to the problem in the form of binary/integer vectors, tree, graph, string, etc. The search operators define how the solutions are varied in order to effectively explore the search space. The fitness function evaluates how good a candidate solution is.

Our study relates to automatic test scenario generation, where the evolutionary search techniques, most notably the genetic algorithms, are typically used [12]. In the next subsection we discuss in a more detail the key concepts of single and multi-objective genetic algorithms.

2.1.1 Genetic algorithms

Genetic Algorithms (GA's) are general purpose stochastic search and optimization algorithms, based on genetic and evolutionary principles [13]. The basic idea is to start with a set of individuals (candidate solutions) representing initial population, usually generated randomly from the allowable range of values. Each individual is encoded in a dedicated form, such as a bitstring, and is called a chromosome. Chromosome is composed by genes. Each individual is evaluated and assigned a fitness value. Some of the individuals are selected for mating. The search is continued until a stopping criterion is satisfied or the number of iterations exceeds a specified limit. Three genetic operators are used to evolve the solutions: selection, crossover, and mutation.

Selection. Selection is an operator that gives solutions with higher fitness a higher probability of contributing to one or more children in the succeeding generation. The intuition is to give better individuals more opportunities to produce offsprings. One of the commonly used selection operators is tournament selection [14]: a small subset of individuals is chosen at random, and then the best n individuals in this set are selected for the mating. The selection pressure can be adjusted by controlling the size of the subset used. Another commonly used technique is a proportional-based selection, also known as a roulette wheel selection, where the probability of choosing an individual depends directly on its fitness. *Crossover operator.* The crossover operator is used to exchange characteristics of candidate solutions among themselves. In our approach we are using a one-point crossover, illustrated in Fig. 2.1. We can see how the parent chromosomes are exchanged genes t_i to produce the offsprings. Other types of crossover include multi point crossover and uniform crossover, however, they are only applicable to fixed size individuals.

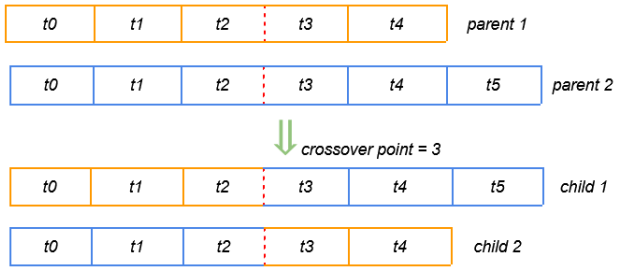


Figure 2.1 One point crossover operator

Mutation operator. The mutation operator has been introduced to prevent convergence to local optima; it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bitstring) [15].

Crossover and mutation are performed with probability *pcross* and *pmut* respectively, where $pmut < pcross$. The rates at which mutation and crossover are applied are an implementation decision.

After the new offspring have been created via the genetic operators the two populations of parents and children must be merged to create a new population [13]. The abbreviation $(\mu + \lambda)$ denotes an evolutionary search that generates λ offspring from μ parents and selects the μ best individuals from the $\mu + \lambda$ individuals (parents and offspring) in total. The selection process is performed by one of several general techniques including: (1) the best μ solutions are retained to become the parents for the next generation (elitist), or (2) μ of the best solutions are statistically retained (tournament), or (3) proportional-based selection. Other approaches include insertion of the children in place of the least fittest individuals in

the population. The least fit individuals can be selected with one of the techniques described above, such as tournament or roulette wheel selection.

It is natural that during initial population generation, crossover or mutation invalid individuals will be produced. Two strategies can be adopted: a repair strategy, when the invalid individual is modified i.e "repaired" to become valid or a penalty strategy, when infeasible individuals are rejected. The latter is more simple and efficient. In our study we adopt the penalization strategy.

Finally, there are two types of algorithms depending on the number of offsprings produced: *steady state* and *generational GA*. In steady state GA, one or two off-springs are generated in each iteration and they replace one or two individuals from the population. A steady state GA is also known as Incremental GA. In a generational GA, n offsprings are generated, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

2.1.2 Single objective genetic algorithms

The single objective algorithm includes the steps described above. It evaluates the individuals considering only one fitness function value. It pseudo-code, as presented in [13], is shown below:

Algorithm 1 GA

1: **procedure** SINGLE OBJECTIVE GA
2: Initialize population P
3: # T - a termination criterion
4: **while** not (T) **do**
5: Evaluate objective F values over population P
6: Select individuals for mating from $P \rightarrow P'$
7: Recombination, mutation of individuals in $P' \rightarrow P''$
8: Evaluate fitness values of children P''
9: Construct new population$(P' \cup P'') \rightarrow P'''$ from parents and offsprings
10: Select the fittest individuals from P''' of the size of P, $P''' \rightarrow P$
11: **end while**
12: **end procedure**

2.1.3 Multi objective genetic algorithms

Evolutionary algorithms seem particularly suitable to solve multiobjective optimization problems, because they deal simultaneously with a set of possible solutions. This allows to find several members of the Pareto optimal set in a single "run" of the algorithm. Under Pareto optimality, one solution is better than another if it is better according to at least one of the individual fitness functions and no worse according to all of the others. The use of Pareto optimality is an alternative to simply aggregating fitness using a weighted sum of the n fitness functions. When searching for solutions to a problem using Pareto optimality, the search yields a set of solutions that are non dominated. That is, each member of the non-dominated set is no worse than any of the others in the set, but also cannot be said to be better.

One is the most popular MOEA is the non-dominated sorting algorithm-II (NSGA-II). It builds a population of competing individuals, ranks and sorts each individual according to nondomination level, applies Evolutionary Operations (EVOPs) to create new pool of offspring, and then combines the parents and offspring before partitioning the new combined pool into fronts. The NSGA-II then assigns a crowding distance to each member. The crowding distance value of a particular individual is the average distance to its two neighboring individuals in the Pareto front. It uses the crowding distance in its selection operator to keep a diverse front by making sure each member stays a crowding distance apart. This keeps the population diverse and helps the algorithm to explore the fitness landscape. The pseudo code for NSGA-II presented by Coello et al., [16] is shown below.

Algorithm 2 NSGA-II

```
1: procedure NSGA-II ALGORITHM
2:     Initialize population P
3:     Evaluate objective F values over population
4:     Assign rank based on Pareto dominance
5:     Assign crowding distance
6:   # T - a termination criterion
7:     while not (T) do
8:         Select individuals for mating from P → P'
9:         Recombination, mutation of individuals in P' → P''
10:        Evaluate objecive values of children P''
11:        Rank (P' ∪ P'') → P''' based on Pareto Dominance
12:        Assign crowding distance
13:        # remove or repair infeasible or dominated individuals
14:        Reduce P''' → P
15:    end while
16: end procedure
```

2.1.4 Statistical testing

In the order to compare two search based algorithms, Algorithm A and Algorithm B, we should perform statistical testing to ensure that Algorithm A outperforms, or not, Algorithm B with a certain confidence. Search based algorithms are highly randomized, therefore it is advised to perform at least 30 runs before comparing the two algorithms [17]. Typically, in the experimental sciences, the level of acceptable error is chosen to be either 1% or 5%. This is the chance of making a so-called "Type I" error, leading to concluding that some Algorithm A id better than Algorithm B when, in fact, it is not. The statistical test will result in a p-value. The p-value is the chance that a Type I error has occurred. To obtain the p-value, SBSE researchers often use non-parametric statistical tests that make fewer assumptions about the distribution of the data. In our study we used a common non-parametric Mann-Whitney U-test. In addition to assessing whether an algorithm is statistically better than another, it is crucial to assess the magnitude of the improvement. To analyze such property, effect size measures are needed [17]. For the effect size, we are using the non-parametric measure such as Cliff's delta (d). The magnitude is assessed using the thresholds provided in [18] , i.e. [d] < 0.147 "negligible", [d] < 0.33 "small", [d] < 0.474 "medium", otherwise "large".

2.2 Model-Based design of Cyber-Physical systems

Model-Based Development is a widely employed development technique in industry for CPS design [19]. In Fig.2.2 you can see a diagram of a traditionally used V-model design cycle. The left side of the "V" identifies steps that lead to code generation, including system specification and detailed software design. The right side of the V focuses on the verification and validation of steps cited on the left side, including software and system integration. After im-

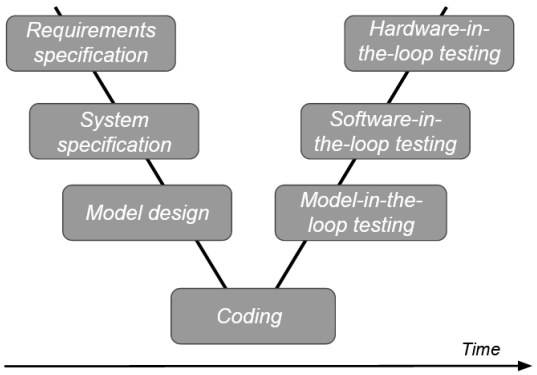


Figure 2.2 The traditional V-model design cycle

plementing the model according to the established requirements, comes the verification part of Model-Based Design approach, where the model-in-the-loop (MIL), software-in-the-loop (SIL), processor-in-the-loop (PIL) and hardware-in-the-loop (HIL) testing is performed [20].

First, a model of the actual hardware (also named plant) in developed in a simulation environment. Then the controller model is created and verified on the plant plant model i.e. assuring the contoller works correctly with the hardware model. This step is called Model-in-Loop (MIL) and the controller logic is tested with the simulated model of the plant. Once the model is verified (i.e., MIL in the previous step is successful), the next stage is SIL testing, where the controller model is replaced by a corresponding software code. The next step is Processor-In-Loop (PIL) testing. In this step, the controller code is uploaded into an embedded processor board and run a close loop simulation with the simulated plant (hardware). This step will help identify if the processor is capable of running the developed control logic code. Once the plant model has been verified using PIL, the plant model can be replaced with the original hardware. This step is known as HIL testing and it helps to identify issues related to the communication channels.

Our test scenario generation approach can be leveraged by researchers and practitioners in the MIL, SIL and PIL testing stages, when the scenarios are run in the simulation environment.

CHAPTER 3 LITERATURE REVIEW

The cyber-physical systems are developed using a model-based design approach [1]: after establishing the requirements of the system, model-in-the-loop testing is performed. In this step, models of the hardware part and the controller are created and tested in the simulation environment. One of the limitations of the simulation platforms is that they don't provide clear guidance to engineers as to which test scenarios should be selected for simulation. Therefore a number of approaches were developed to generate the testing scenarios.

3.1 General approaches for cyber-physical systems testing

In the classical approach, the exhaustive exploration of the state-space of the model is performed [21]. It uses the abstract model, created strictly according to the system requirements, to generate the test cases for the model of the system under test (SUT). If the outputs of the SUT model and abstract model are different, the fault in the SUT is revealed. As the system models get more complex, the search space becomes infeasible. More recently, falsification based approaches have been proposed verifying whether the model meets specific requirements, specified in a temporal logic notation such as metric temporal logic (MTL) or signal temporal logic (STL). The Upaal-tool performs the statistical model checking (SMC) of the MTL or STL of a given model [22]. The core idea of SMC is to monitor some simulations of the system, and then compute the probability along with confidence intervals that a specific requirement holds for the SUT. A number of tools were developed that instead of calculating the probability that a system satisfies the property with a certain confidence, compute the worst expected system behaviour as a quantitative value, called robustness. Examples of such tools are S-Taliro [23], Breach [24] and ARIsTEO [7]. Differently from others, ARIsTEO propose to apply falsification testing to the surrogate i.e. approximated model of the SUT, that closely mimics its behaviour but is significantly cheaper to execute. Arrieta et al., [25] proposed a search based approach, that doesn't use the system model . Authors define three cost effectiveness measures to guide search towards generating optimal test cases: requirements coverage, test case similarity (effectiveness) and test execution time (cost). The common disadvantage of the mentioned approaches is that they search for a combination of fault revealing control inputs, not relating the real world to the models under test.

Testing autonomous systems requires more sophisticated scenarios, where the search is performed over the surrounding environment, rather than the system control inputs. A number

of approaches has been proposed for testing vision-based vehicle driving systems and robotic systems accounting for interaction with the environment.

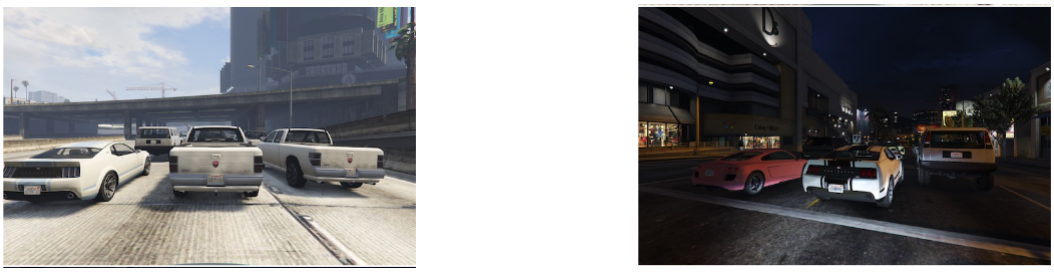
3.2 Autonomous cyber-physical systems testing

Autonomous cyber-physical systems rely on complex planning and control algorithms, including modern machine learning methods such as deep neural networks to control their interactions with the physical world. Testing of such intelligent cyber-physical systems is a challenge due to the huge state space associated with high-resolution visual and sensor inputs [26]. We divide the research works for testing autonomous CPS in two groups: autonomous driving systems testing, including testing of self-driving vehicles and driving assist systems, and autonomous robotic system testing, including autonomous robots, robot swarms and drones.

3.2.1 Autonomous driving system testing

Static scenario encoding. Undoubtedly, the scenarios for testing such systems should be as realistic as possible. Zhang et al. [27] develop a framework for automatic generation of large amounts of accurate driving scenes to test the DNN-based autonomous driving systems. Deep-Road delivers driving scenes with various weather conditions (including those with rather extreme conditions) by applying the Generative Adversarial Networks (GANs) along with the corresponding real-world weather scenes.

To have even more control over the generated results, Fremont et al. propose SCENIC [28], a programming language for scenario specification and scene generation. Authors represent the environment as a scene i.e. configuration of objects in space (including dynamic agents, such as vehicles) along with the distribution of their features. It allows, for instance, to put distributions on the time of day and the weather conditions during the scene. Examples of the scenarios generated with SCENIC are shown in Fig. 3.1. We can see three cars on a road in different times of the day. Further, Dreossi et al [29] develop a toolkit for the design and analysis of artificial intelligence-based systems, named VERIFAI. It creates a scenario with the SCENIC language and then uses Bayesian optimization sampling to find small perturbations of the initial scene (generated by SCENIC) which cause the vehicle to violate the specification. Perturbations are performed over the initial positions and orientations of all objects, the cruising speed and reaction time of the vehicle. It allows to perform a better search over the possible scenarios, than the SCENIC itself, however the initial scenario to be optimized should still be provided by the developer.



(a) Scenario with three cars on a road in daylight (b) Scenario with three cars on a road at night

Figure 3.1 Examples of the scenarios generated with SCENIC

Both Deep-Road and SCENIC can be leveraged to improve the training datasets for the ML based autonomous driving systems by adding some rare scenarios to it. Moreover, once one scenario is created, it can be easily augmented by changing environmental conditions, time of the day, etc. However, these frameworks don't provide enough guidance as to what scenarios should be added to retrain the model. Furthermore, they only test the perceptive part of the system, and not the whole system, which also includes the driving controllers. In the next section we discuss the approaches that generate dynamic scenarios for testing the CPS as a closed loop system, having perceptive and control components.

Dynamic scenario encoding. Althoff et al., develop a CommonRoad project, which includes a number of scenarios to benchmark the autonomous systems [30]. The scenarios are specified in an XML file, which is composed of 1) a formal representation of the road network, 2) static and dynamic obstacles, and 3) the planning problem of the ego vehicle(s). However, at the current stage, most of the scenarios should be handcrafted and no optimization is used to improve them.

Nalic et al. [31] implement a framework for automated scenario generation, based on a realistic traffic model built using measured data from a real world test road. Traffic configuration, driving behaviour and driver models can be edited using the MATLAB GUI connected to the VISSIM simulator [32]. During testing the vehicle models are sent repeatedly through the virtual motorway sections, which is updated with the continuously produced random scenarios. In 4542 kilometres the vehicle travelled in simulation 26 collisions and 378 near-collisions were found.

Both frameworks provide the possibility to implement the dynamic scenarios and reveal system faults, however they don't guide the developer towards the fault revealing scenarios to be selected and executed in the first place.

Mullins et al. [33] indicate that more information about the system is gained by testing in

regions where critical decisions must be made by the autonomous system. According to the authors, it is ineffective to test the system in regions where performance is constant and known. They use the Gaussian processes to drive the search towards yet unexplored regions of the input space. In addition, they group the scenarios by performance using unsupervised clustering techniques and sort them by the effectiveness at diagnosing changes in the autonomous system's behaviour.

Abdessalem et al. in [2] use a multi-objective search to obtain test scenarios for Advanced Driver Assistance System (ADAS), specifically the Automated Emergency Braking (AEB) system, helping avoid collisions with pedestrians. They implement an NSGA-II based search with three objectives: minimizing the distance to the pedestrian, minimizing acute warning area and the time to collision. To mitigate the computation cost of executing physics-based simulations, they use the simplified models of ADAS built with neural networks (NN). The search is performed over the properties of the vehicle and the pedestrian. More specifically, the test input is defined as a vector, where the speed of the car, position, orientation and speed of the pedestrian are specified. The following NSGA-II configuration was used: crossover rate of 0.9, mutation rate of 0.5 and the population size of 10, duration of the experiment - 150 min. In [34] authors continue their work, proposing a new learnable evolutionary algorithm, which combines NSGA-II and decision tree classification models. Results show that this algorithm generates 78% more distinct, critical test scenarios compared to NSGA-II. Both static and dynamic environmental parameters of the system are varied, such as the weather conditions (snow, fog, rain), road type (curved, straight, ramped), the pedestrian location. They define an ADAS scenario as a tuple containing the static variables (precipitation, fogginess, road shape, visibility range), dynamic variables (vehicle and pedestrian location), allowable ranges for the variables and constraints. The first fitness function computes the minimum distance between the pedestrian and the field of view of the vehicle, and the second that computes the speed of the car at the time of collision. The following NSGA-II configuration was used: population size of 100, mutation rate of 0.11, the crossover rate of 0.6, duration of the experiment - 24 hours. The simulations were run in a commercial PreScan simulator [35].

Gambi et al., [5] develop a tool to generate the road configuration to test car Lane Keeping Assist System (LKAS). It combines procedural content generation with a single-objective search based technique. The test road is represented by road segments, which are obtained by applying affine transformations to points of the initial road segment. The road network is represented by a graph, in which edges model road segments and nodes model either road intersections (internal nodes) or intersections between roads and map boundaries (source and destination nodes). The scenario represents a navigation path between randomly chosen source and destination nodes. The randomly selected navigation paths are evolved by

applying mutation and crossover operators. Road network mutation randomly replaces road segments in a road, "join" crossover operator recombines road segments from parent roads to form new roads, and "merge" crossover operator recombines roads in parent road networks to form new road networks. The fitness function maximizes the distance of ego-car from the center of the road lane. Intuitively, the tests which cause the ego-car to drive away from the lane center might contain road segments which stress the self-driving car software more. To improve the efficiency of test generation and promote scenario diversity, AsFault identifies and filters out similar tests before executing them. AsFault computes the similarity between tests by means of the Jaccard Index of their road segments. Given tests T1 and T2 their similarity is defined as:

$$similarity(T_1, T_2) = \frac{[C_{T_1} \cap C_{T_2}]}{[C_{T_1} \cup C_{T_2}]} \tag{3.1}$$

where C_{T_i} refers to sequences of consecutive road segments of a given size. Value closer to 1 corresponds to a high similarity between tests. The following genetic algorithm configuration was used: mutation rate 0.05, crossover rate 0.5, population size - 25 and evaluation budget - 24 hours. Every scenario is evaluated using a self driving car model in the BeamNg simulation environment. After one hour of execution AsFault detected on average 30.1 failures, increasing to an average number of 42.1 after 3 hours, and then stabilizing to an average amount of 47.5 failures after 24 hours. We surmise that the small increase after 3 hours of execution, comparing to 1 hour, is due to the convergence of the algorithm to a local maximum. Possibly, periodic repopulation of the algorithm could increase the number of revealed faults, however this point is not discussed in the paper.

Tonella et al., address the idea of repopulation in their work [36]. They use a multi-objective search to test the car LKAS system by generating virtual roads. They use two fitness functions, one maximizing the car deviation of the car from the lane center, as in [5], and the second maximizing the diversity. The difference between two virtual roads is calculated using a weighted Levenshtein distance [37]. The virtual roads are saved in an SVG format, which is an XML-based image format that can represent shapes as the combination of cubic and quadratic Bézier curves. The mutation operator randomly chooses a start point, an end point or a control point of the road shape and applies a displacement to it in the two-dimensional space. To prevent the algorithm from getting stuck in a local optima, authors define a repopulation operator. It replaces n of the most dominated individuals in the current population with individuals newly generated from the seeds. For evaluation, authors used 12 individuals in initial population and 100 generations of time budget. Authors report the effectiveness of the approach for exploring the behaviour frontiers of the system, however don't report the number of revealed faults (car going out of the lane) for a given a time budget.

In both previous works authors indicate the importance to promote the diversity. Loiacono et al., propose novel metrics to promote the diversity to generate tracks for a high-end racing games [38]. In particular, they describe the diversity of a track in terms of its shape (i.e., the number and the assortments of turns and straights it contains), and in terms of driving experience it provides (i.e., the range of speeds achievable while driving on the track). They define two fitness functions: the entropy of the track's curvature and entropy of speed profiles. A track is represented as a set of control points that the track has to cover; the track is generated as a sequence of Bezier curves connecting the control points. However, the main goal of the search is to make the tracks more interesting to the gamers and not reveal the failures.

In the following subsection we discuss the dynamic scenario generation techniques dedicated to testing autonomous robotic systems.

3.2.2 Autonomous robotic systems testing

Arnold et al. [39] designed a tool to produce navigation maps to test autonomous robot control algorithms. They use the a procedural content generation technique to randomly generate 2-D maps with obstacles and then select the ones, that correspond to a defined set of unwanted behaviours, such as stalling or colliding with a wall. More specifically, to generate an environment their tool first creates a 2D noise map using a Perlin noise process [40]. Then two filters are applied: a pixelisation filter to make the noise more granular and a thresholding filter to convert the noise into binary occupied/unoccupied. The experiments were conducted in the Player/Stage simulation environment [41], evaluating 500 unique maps with Pioneer 3-AT robot. Authors provide analysis of the several revealed high-risk scenarios.

Sotiropoulos et al. [42] generate navigation maps of different difficulty levels. A map is characterized by its size, its percentage of obstruction (due to objects), and its degree of smoothness (resulting from the ground local deformations). The inputs to robot navigation testing are a world instance and a navigation mission in this world, defined by a starting position and a target arrival position. Depending on the collision events and timeouts (when robot fails to reach the goal in the time budget) three levels of difficulty are assigned to a scenario: easy, challenging and very difficult. According to the study, the size of obstacles is one of the most important factors to control the difficulty level.

Even though the algorithms specified above may reveal system faults, they achieve it randomly, which might take more time and produce less effective scenarios comparing to the search based techniques. We further discuss the works leveraging evolutionary search algorithms to improve the effectiveness of the test scenarios.

Ashlock et al., [43] apply a single objective genetic algorithm to generate grid robot path planning problems. A path planning problem (PPP) for a grid robot is specified by the size of the grid, the position of obstructions within the environment, and an initial and goal position for the grid robot. This study uses a cellular [44] representation for PPPs. In a cellular representation, directions for constructing an object of interest are evolved, rather than parts of the object directly. The cellular representation for a PPP is constructed of a set with four parameters (x, y, l, t), called descriptor. A single descriptor specifies a group of obstructions laid out according to a simple rule. It specifies obstacle starting position (x, y), the maximum number of obstacle blocks l and t the type or pattern of the descriptor (type of the obstacle). Six types of single descriptors were used in this study, such as obstacle above/below/left/right of the current positing, zig-zag obstacle. A PPP is made of 4 - 50 single descriptors. A PPP has a maximum number of obstructions that are permitted. The single descriptors are processed in the order they appear in the PPP's chromosome. In order to evolve PPPs the variation operators are applied. Namely a two point crossover of the list of descriptors, treating single descriptors as atomic objects, and a mutation operator. With equal probability the mutation operator changes the values in descriptors by adding/removing obstacles, changing the obstacle type, etc. Algorithm was evaluated using one of the three fitness functions: maximizing the number of turns, maximizing number of forward moves, and maximizing the total moves (turn left, turn right or advance). The disadvantage of this approach is that only produces simplistic 2-D scenarios not integrated with a robotic simulator.

Nguyen et al., propose an approach to generate scenarios to test an autonomous robotic cleaner agent [45]. The agent needs to collect all the objects, not bumping into obstacles and keeping the charge level higher than 10 %. The artificial environment is defined as a square area, A. In the area A there can be obstacles, dustbins, waste, and charging stations. Different locations of the objects pose different levels of difficulty in which the cleaner agent must operate. The test scenario (and a chromosome) is encoded as a matrix A of size RxR cells, where R is a resolution. Objects such as obstacles, waste, bins, charging stations are placed into cells. A cell containing an object is denoted by 1, while a content–free cell is denoted by 0. As a fitness function, they minimize the distance to obstacles encountered during the operation of the agent. The simulations are performed in JADEX simulation environment [46]. Their experiments confirm that evolutionary testing, guided by fitness functions derived from soft-goals, outperforms random testing given the same time budget.

Zou et al., [47] target testing the conflict resolution algorithms for unmanned rotary wing aircrafts (helicopters, multi-copters, drones). The environment in the simulations is a 3-D cuboid flight area. An unmanned aerial vehicle (UAV) is placed in this environment and is

given a navigation task i.e. go from the start location to goal location. The goal is to find such positions of other UAVs (the intruders) that the target UAV collides with one or several of them. In total 7 parameters are varied to describe the intruder and perform the search. The search is performed using the NSGA-II algorithm with two objectives: first minimizing the distance between the UAV and intruder UAV and second minimizing the number of UAVs involved in collision. When smaller number of UAVs cause the collision it constitutes simpler counterexamples to correct operation of the resolution algorithm. The NSGA-II algorithm was set to evolve for 20 generations, each generation having a population of 5,000, evaluation 100,000 scenarios in one run. To the experiments the MASON simulator was used [48]. The results show that the proposed method can find encounters meeting the two objectives more efficiently than the random search based approach.

Considering the discussed approaches for autonomous CPS testing, we can see that frameworks already exist to test the autonomous agents DNN models allowing to generate realistic images of the scenarios. However, these scenarios are static and only test the perceptive part of the system. Search based tools have been developed to generate dynamic scenarios for different types of autonomous systems, such as: car breaking, lane keeping assist system, UAVs, autonomous robots. However, the proposed evolutionary search algorithms have different scenario representation (such as graphs, tuples, SVG files) and search operators that are fine tuned to testing a particular autonomous system. Redesigning the solution encoding and search operators for each new scenario generation task is tedious. Our vision is that there should be standardized frameworks for generating dynamic scenarios for different systems, similar to frameworks, where only static scenarios are generated. In our work we aim to design a search based testing framework for generating dynamic scenarios for autonomous CPS with customizable scenario encoding and search operators. It could be used by other researches and practitioners as a guidance to design the virtual environments for their testing tasks.

3.3 Chapter summary

Testing autonomous systems requires varying the environment where the system operates and not only its inputs. The approaches to generate static images of the scenes can be used to test the perception part of autonomous system. However, to test the system completely, including its driving controllers, dynamic scenarios should be developed. Evolutionary search is used in majority of the works to generate the fault revealing dynamic scenarios and outperforms the random search. During the search it is important to promote the fault revealing power of the scenarios as well as their diversity. In our study we present an approach that

uses a standardized environment representation, selection and mutation operators for generating different types of environment. As a fitness function, we are using both scenario fault revealing power and diversity.

CHAPTER 4 AMBIEGEN: SEARCH BASED FRAMEWORK FOR VIRTUAL TEST ENVIRONMENT GENERATION

In this chapter we formulate the problem of the scenario generation as a search problem and present the genetic algorithm configuration used by AmbieGen. We also discuss some implementation details of AmbieGen.

4.1 Problem formulation

The test scenario generation problem can be formulated as follows: given the parameters that describe the environment for a cyber-physical system, generate such an environment, that forces the system to falsify the requirements. For example, to test a self-driving vehicle we should consider such parameters as the road type and size, the location of other vehicles or pedestrians, the driving weather conditions, etc. Evidently, one of the most important requirements would be the collision avoidance. Generating such scenarios manually is an extremely time consuming task: the engineers would need to list the precise positions positions and heading directions of the moving objects, specify different types of environmental conditions, etc. Therefore it is preferable to generate such scenarios automatically. To control their quality, optimization techniques should be used.

4.1.1 Scenario representation

In this subsection we formalize the definition of the test scenarios. A cyber-physical system is a reactive system, consisting of a collection of computing devices interacting with the environment via inputs and outputs [49].

We propose to encode the test cases TC as a matrix of size MxN, such as shown in Table 4.1, where M is the number of elements E_i constituting the environment and N, the number of attributes describing each element A_i. Therefore, each column describes a particular part of the environment. The attributes, i.e. rows provide more details about the environment part e.g. they can represent the location, size and type of the obstacle in a particular space. Each attribute has its allowable range $[A_{nMIN}, A_{nMAX}]$. Finally, the test scenarios can have restrictions R, which limit the scenario length M or particular combinations of attributes. We provide examples of application of this representation to generate different environments in the chapter 5.

With such representation it's easy to implement the crossover and mutation operators as well

Table 4.1 Scenario TC representation

E_1	E_2	E_3	...	E_n
A_{1h1}	A_{1h2}	A_{1h3}	...	A_{1hi}
A_{2h1}	A_{2h2}	A_{2h3}	...	A_{2hi}
A_{nh1}	A_{nh2}	A_{nh3}	...	A_{nhi}

as generate the initial population. To perform the crossover, we can simply exchange the columns of the individuals. To perform mutation, we can change the values in the cells.

4.1.2 Search objectives definition

The main goal is to find scenarios producing system faults. At the same time, the scenarios should be diverse, uncovering different types of faults. From our experience, using only one objective results in producing many similar test cases in the last generation. Therefore we suggest adopting a multi-objective algorithm, where one of the objectives is accounting for the diversity of the test cases. The idea of adding a second objective for diversity was addressed in the novelty search research works, such as [50] as well as test scenario generation tools [6].

To estimate the fault revealing power φ of the test case we compute the difference between the expected $B(TC)$ and observed system behaviour $B_o(TC)$ after executing the test case:

$$\varphi(TC) = \delta(B(TC), B_o(TC)), \tag{4.1}$$

where δ is a developer defined function for computing the deviation between the expected and observed system behaviour and TC is the test scenario specification. The expected behaviour $B(TC)$ is typically defined in the system requirements or formulated by the developers e.g. "the car should not deviate from the lane center more than 1 meter". The observed behaviour corresponds to the model under test (MUT) outputs after scenario execution. However, the models of autonomous systems are rather complex and take long time to execute in the simulators i.e. up to several minutes for one scenario. Moreover, executing the full models in the simulation environments requires additional system resources such as a GPU and high amount of RAM. Therefore we suggest estimating the observed behaviour $B_o(TC)$ using the approximated (surrogate) system models. Such models can be built based on the grey-box modelling approach [51], where model structure is chosen from system knowledge and parameters are selected to match sampled data. When little knowledge is available about the model, system identification techniques [52] can be used, where the modelled system is considered as a black box.

To estimate the variability v of the test case we compute the Jaccard distance between it and a reference test case:

$$v(TC, TC_{ref}) = 1 - \frac{TC \cap TC_{ref}}{TC \cup TC_{ref}}, \tag{4.2}$$

where $TC \cup TC_{ref}$ corresponds to the total number of unique inputs in both test cases and $TC \cap TC_{ref}$ to the number of inputs with similar or unique attributes.

Finally, we define our search objectives:

$$maximize : (\varphi(TC),\ v(TC, TC_{ref}))$$
$$subject\ to : C_1(TC) = \varphi(TC) - \alpha > 0,$$

where C_1 is a constraint for the minimum value of the first objective, α is the developer defined threshold to identify the test cases as having a risk of producing a failure. This constraint is introduced to avoid producing test cases with low fault revealing power.

In our study we consider two configurations of our approach: AmbieGen MO, decribed above and AmbieGen SO based on a single objective genetic algorithm (GA) with F_1 as a fitness function.

4.2 AmbieGen description

To perform the search we are using evolutionary search algorithms NSGA2 and GA [16], which have proven to be effective at similar tasks [25, 34]. Below, We present the GA and NSGA2 configurations used in AmbieGen.

4.2.1 Genetic algorithm configuration

We implemented the AmbieGen MO and AmbieGen SO using a python Pymoo framework [53]. The framework provides the possibility to define custom solution representations, crossover and mutation operators.

Solution representation. Each individual in the population corresponds to a test case. Individuals can have a variable number of genes i.e. environment elements depending on the application. We suggest representing the the test scenario matrix as a dictionary, as shown below:

```
{"E1": { "A1:" A1, "A2": A2,
"An": An }, "Ei": {...} },
```

where "Ei" corresponds to the element of the environment that is described.

Algorithm 3 Test case generation algorithm

Input: - population size P_s
- crossover rate C_r
- mutation rate M_r
- number of offsprings N
- termination criteria T
- fitness functions F_1, F_2
- constraints C

Output: a set of critical scenarios $CS = (TC_1, TC_2, TC_i)$, represented by the Pareto optimal solutions found by the algorithm

procedure NSGA2 SCENARIO GENERATION(P, C_r, M_r, N)
  # Generate an initial population set P randomly or using a Markov chain
  population $\mu \leftarrow PopulationSampling(P_s)$
  $Evaluate(\mu, F_1, F_2)$
  #while termination criteria is not reached
  while not (T) **do**
    # select offsprings for crossover and mutation
    $offsprings\ Q \leftarrow BinaryTournament(N, \mu)$
    children $\lambda \leftarrow OnePointCrossover(Q, C_r)$
    children $\lambda \leftarrow Mutation(\lambda, M_r)$
    # add offsprings to the population
    population $\mu \leftarrow \mu + \lambda$
    $Evaluate(\mu, F_1, F_2)$
    # update the population by selecting top P_s
    individuals from the old population and offsprings
    population $\mu \leftarrow UpdatePopulation(P_s)$
  end while
  return CS
end procedure

Initial test case generation. The search begins by generating the initial test cases. One of the options is to randomly assign values A_{ij} to environmental attributes A_i from their allowable ranges $[A_{ijMIN},\ A_{ijMAX}]$. When some distribution of attribute values is known

to produce better test cases, from both semantical and fault-revealing point of view, we suggest using the Markov chain to assign the values. For example, when generating the road segments, the road consisting of only the straight segments is very unlikely to produce faults. Such cases can be avoided by assigning values with the Markov chain.

Fitness evaluation. We use two fitness functions to evaluate each individual: F_1, corresponding to the function in (4.1) and F_2 corresponding to (4.2).

F_1 is calculated after executing the test case with a surrogate model M of the system. This function is problem specific and should be proportional to the unwanted behaviour of the system. For example, for evaluation of a self-driving car test case we can compute the maximum deviation from the road lane center, where bigger deviation is likely to produce more faults.

In our implementation we compute F_2 as the Jaccard distance between the individual and its parent, which acts as a reference test case. The intuition is to promote the modifications done to the test cases. However, a different reference test case can be used, such as the closest individual from the Pareto optimal solutions.

As the Pymoo framework minimizes the fitness functions, in our implementation we multiply F_1 and F_2 actual values by (-1).

Mating selection. To select the individuals for crossover and mutation the binary tournament selection is used, which is implemented by default in Pymoo. N individuals are selected, producing N new individuals after crossover and mutations.

Crossover operators. We are using a one point crossover operator, which is one of the commonly used operators for variable-length solution representation. It exchanges the environment elements E_i between two different test cases.

Mutation operators. We define three mutation operators:

- *exchange operator*: two states of a chromosome are randomly selected and exchanged the positions;
- *change of variable operator*: a state in a chromosome is randomly selected, then for one of the state variables (temperature, duration, model) value is changed according to its type and maximum as well as minimum values.
- *scramble operator*: a number of states in a chromosome is selected, then their positions in the chromosome are randomly exchanged.

Individual insertion. To insert the individuals the $\mu + \lambda$ approach is employed [54]. The

idea is to merge the population and offsprings together, and then from the merged set, select the best possible non-dominated solutions of the population size.

4.2.2 Software implementation

In this subsection we provide more details about the software implementation of AmbieGen. As mentioned earlier, we use the Pymoo framework, which provides the key components for implementation of single and multi objective evolutionary algorithms. The key components of AmbieGen are summarized in Fig. 4.1. We implemented customized modules for creating initial population (MySampling), crossover (MyCrossover), mutation (MyMutation) and the solution representation. "Solution" is an object containing all information about the solution: its fitness, novelty and corresponding scenario. Initial encoding of the scenario is the matrix TC, containing a high-level description of the environment. With such representation it's easy to implement the search operators by simply exchanging the matrix columns and cell values. To evaluate its fitness it first needs to be converted to the environment configuration for the approximated model ("TC to environment" module). For example, the scenarios for the smart-thermostat should be converted to a list of temperatures to follow. Then an approximated model is used to execute the scenario and the fitness function is calculated based on the execution results ("Fitness function" module). Approximated model can be created either from real data or from the full model data. Another possibility is to use already implemented simplified models of cyber-physical systems, such as those available at python robotics project [55].

Overall, to generate the scenarios the developer needs to provide the list of attributes and their allowable ranges, an implementation of the "TC to environment" and "fitness function" modules. The AmbieGen will integrate the modules and implement the initial population generation, crossover and mutation operators.

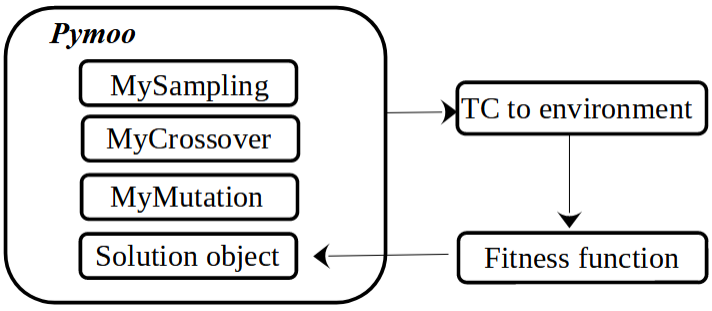


Figure 4.1 AmbieGen software implementation structure

Moreover, it's simple to control the level of complexity needed for the environment. By

adding more attributes, the complexity can be increased. The limitation is the possibility of the simulator to interpret more complex environment configurations, such as the terrain type, the weather conditions, etc.

4.3 Chapter summary

In this chapter we propose the key components needed to implement the evolutionary search algorithm: the encoding of the individuals, implementation of the crossover and mutation operators. We also define two fitness functions, accounting for the scenario fault revealing power, as the deviation of the observed system behaviour from the expected, and diversity, as the Jaccard index between the scenarios. Finally, we describe some details of the implementation of our framework. We provide examples of application of AmbieGen to generation of scenarios for three different autonomous CPS the chapter 5.

CHAPTER 5 TEST SCENARIO GENERATION CASE STUDIES

In this section we demonstrate how AmbieGen can be applied to three different types of environment. We consider the following test generation case studies: a smart-thermostat, robot obstacle avoidance system and vehicle lane-keeping assist system. In every case study the autonomous agent controller has a different level of complexity: simple PID controller for the thermostat, a robot controller based on the nearness diagram navigation approach [8] and a deep neural network based controller for the vehicle. We evaluate AmbieGen by comparing the results obtained with random search for all the three problems. For the last case study we also compare our results with state-of-the art approach, presented at SBST2021 tool competition [1].

5.1 Wireless thermostat case study

Nowadays, home automation becomes more and more popular. Automatic temperature control systems are one of the most commonly used. Such systems consist of a controller, temperature sensor and a heating element. The controller goal is to keep the room temperature according to the programmed schedule. The simplest solution is to send "ON" and "OFF" commands to the heater, when the temperature needs to be increased or decreased. More sophisticated thermostats implement PID controllers to achieve smoother operation. Testing the controllers in the simulators for different temperature schedules is necessary in order to ensure their precision and reveal the possible limitations. In this study, the test generation goal is to create scenarios accounting for the scheduled temperature as well as environmental conditions.

System under test description

In our case study we consider a simple wireless thermostat system. It consists of of one room with a heater, sensor and controller and is part of a larger system, described in a more detail in [56]. The room dimensions are approximately 2.5 m $\times$ 4 m and the height is about 2.6 m. The heating element is a Steelpro 1.5 Kw electronic convector [2], which is controlled via a wireless Z-wave protocol based switch. The temperature is measured by a Aeotec MultiSensor 6 device [3], placed at about 2.2 m from the floor. The controller is a

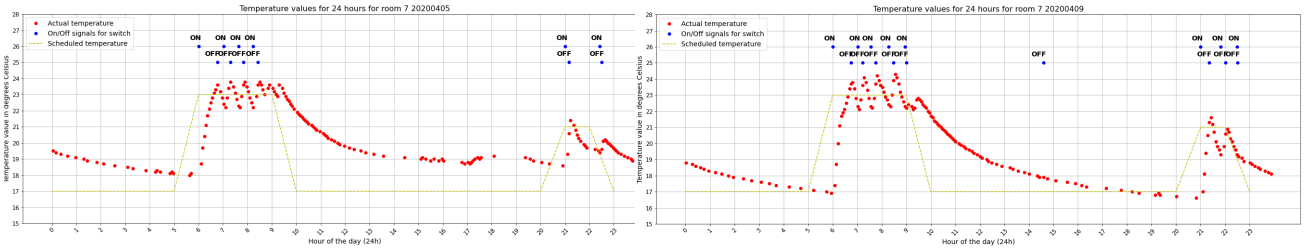
[1] `https://sbst21.github.io/tools/`

[2] `2https://www.stelpro.com`

[3] `https://aeotec.com/z-wave-sensor`

Raspberry Pi 3B running Z-wave.me with a RaZberry 4 [4] daughter card acting as Z-wave network controller. The Raspberry Pi has a user defined schedule of temperature levels; it reads the thermometer measured values and if needed (according to the schedule and required temperature) it switches on (off) the heating.

The data for the room temperature was collected for the period from December 2019 to May 2020. To collected the data we created a python script to parse the system log files. From the data, we could observe 7 patterns of temperature dynamics after ON/OFF commands of the thermostat. As an example, in the Fig. 5.1 you can see that the temperature decreases with different rates, which depends on such factors as indoor and outdoor temperature, humidity, etc. We represent different temperature dynamics patterns with different thermostat models.



(a) Slower decreasing temperature pattern (b) Faster decreasing temperature pattern

Figure 5.1 Different temperature dynamics patterns

The goal of the search is to find the schedule and the corresponding thermostat operating mode, when the error of following the schedule is more than 1 degree.

The case study description is summarized in the table 5.1 and described in a more detail in the following paragraphs.

Problem representation

For this problem we define three high-level input attributes: A_1 the goal temperature value, A_2 the duration of this temperature and the thermostat operation mode A_3, corresponding to one of the identified patterns. The allowable ranges for the variables are shown in the Table 5.2. We define two restrictions R_1 and R_2 for the test scenarios. For R_1, the duration of the schedule T cannot exceed 24 hours:

$$\sum_{i=1}^{n} A_{2i} < T,\ T = 24 \tag{5.1}$$

[4] `https://z-wave.me/products/razberry`

Table 5.1 Summary of the smart thermostat case study parameters

The problem	Find a schedule and environmental conditions combination, where the smart thermostat cannot follow the schedule with a given precision.
Environment element	One part of the temperature schedule with the goal temperature, duration of the given temperature and thermostat operation mode specified.
Test case restrictions	Total schedule duration less than 24 hours, temperature cannot change faster than 5 degrees per hour.
Fitness function F1	Maximize the root mean square error between the scheduled and simulated temperature.

Table 5.2 Attributes to generate scenarios for the smart thermostat case study

$A_1, temperature, T°$	$A_2, duration, min$	$A_3, operation\ mode$
[16, 17,...,25],	[60, 75,..., 240]	[1, 2,..., 7]

For R_2, the temperature cannot change too sharply i.e. more than 5 degrees between two inputs:

$$A_{1h_i} - A_{1h_{i+1}} < 5, i \in [0, m], \tag{5.2}$$

where m is the number of inputs in the test case.

Fitness function definition

To calculate one of the fitness functions we need to create a surrogate i.e simplified model of the system. To this end, we extracted the data from the experimental measurements and selected the series of data points, corresponding to behaviour of the thermostat after “switch on” and “switch off” commands in different thermostat operation modes. The next challenge is to select the model structure. In our case it is possible to build a first-principles model, as the heating and cooling of a closed space is guided by physical laws, such as Newton Law of cooling [57]. The law has an exponential nature, therefore our model structure is based on increasing and decreasing exponential function.

We propose the following time-discreet model structure for the M_1 ("on") mode:

$$Y = k_{on1} * (1 - e^{-k_{on2} * t_i}) + T_0 \tag{5.3}$$

and for the M_2 ("off") mode:

$$Y = k_{off1} * (e^{-k_{off2} * t_i}) + T_0 - k_{off1} \tag{5.4}$$

Here k_{on1}, k_{on2}, k_{off1}, k_{off2} are the unique coefficients defining the model behaviour in a particular environment. T_0 - is the starting temperature and t_i - the discreet time step value, Y - the output temperature. We keep the coefficients in a table, such as table 5.3, where coefficients for the three models are shown . As an example, in Fig. 5.2 you can see how the model 1 with the coefficients from the table, fits the data from real measurements. One model includes two equations describing behaviour in “on” and “off” modes. In total, we identified 7 models having different coefficients in the equations, corresponding to the thermostat operating in different environmental conditions.

Table 5.3 Smart thermostat simplified model coefficients

Model	k_{on1}	k_{on2}	k_{off1}	k_{off2}
1	7.7	0.11887928	5.6	0.02929884
2	7.9	0.11180434	5.2	0.04803319
3	6	0.14704908	4.8	0.1203876

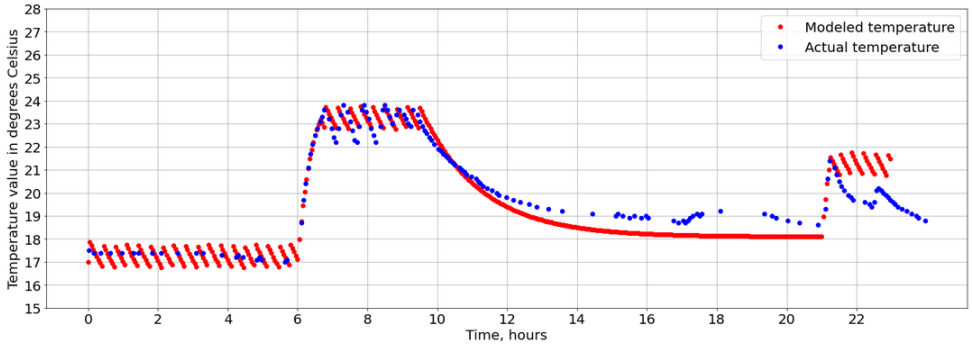


Figure 5.2 Model (red points) fitting the experimental data (blue points)

To obtain the coefficients, we fit the experimental data by a curve with minimal deviation. We used python SciPy library, namely *curve_fit* function from *Optimize* class, which is based on non-linear least squares method [58]. The average root mean square error between original and approximated data did not exceed 0.5 degrees.

Finally, to calculate the first fitness function we execute the test scenario TC using the surrogate model. We obtain the output values of the room temperature set by the thermostat Y and calculate the root-mean square error between Y and the temperature values defined

in the schedule S:

$$F_{1therm} = \sqrt{\sum_{i=1}^{n} \frac{(Y_i - S_i)^2}{n}}, \tag{5.5}$$

where n is the number of datapoints in the output. For the test cases that don't satisfy the requirements (5.1) and (5.2) we set the F_{1therm} to 0.

We calculate the second fitness function, F_{2therm} according to (4.2). In order to prevent obtaining the test cases with low fault revealing power, we also add a search constraint C_{therm}:

$$C_{therm} : |F_{1therm}| - 1.5 > 0, \tag{5.6}$$

AmbieGen configuration

We used the following GA (AmbieGen SO) and NSGA-II (AmbieGen MO) configurations for the smart thermostat problem: population size: 250, number of generations: 200, mutation rate: 0.4, crossover rate: 1, algorithm type: generational, number of evaluations: 50 000.

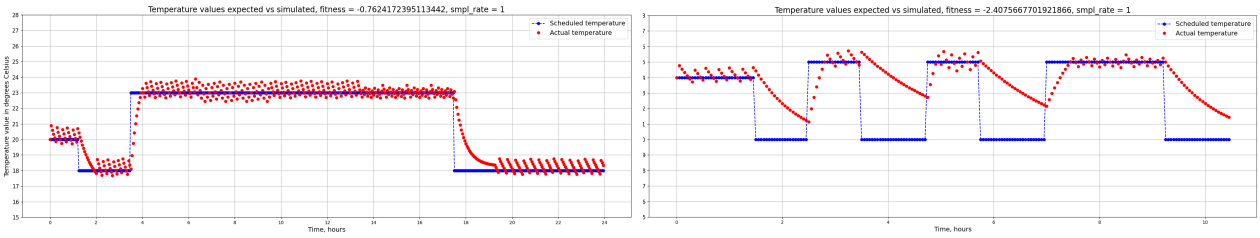
We are using a high mutation rate, as from our experience, it allowed to converge to better solutions faster. In our implementation a $\mu+\lambda$ insertion approach is used, where only the best individuals from previous generation and offsprings are inserted to the next generation. In the generational GA the number of offsprings inserted in the population is equal to the population size. The average time to run 50000 evaluations was 136.691 sec for GA and 123.665 sec for NSGA2.

Scenario generation

Finally, we discuss an example of the produced scenarios. In Fig.5.3a you can see a scenario with a low fitness value of 0.76 degrees, indicating that the temperature deviates from the schedule 0.76 degrees on average. On the contrary, in Fig.5.3b the scenario produced by AmbieGen search has a higher fitness of 2.4 degrees. Clearly, this scenario is more likely to be unacceptable to the user, comparing to the first one.

5.2 Autonomous robot case study

The autonomous robotic systems are used in many domains: from everyday tasks such as room cleaning to critical missions such as navigation to harsh environments. For every application, we need to have a high confidence that their behaviour will be safe. Running the simulations of the system in various virtual environments can uncover the possible failures



(a) Randomly produced scenario (b) Search produced scenario

Figure 5.3 Examples of scenarios for the smart thermostat

of the robot in the early design stage.

In this case study we consider an autonomous mobile robot, navigating in a space with obstacles. The robot has to reach the goal location, relying only on its range sensors and the planning algorithm. The goal is to generate the environment, i.e., a room with obstacles that forces the robot to fail. Similar test generation problems were addressed by [42], [39]. In [39] the navigation maps are created using the procedural content generation technique. Then robots are assigned a randomized route to follow. The test scenario is an environment populated with robots, obstructions, and mission allocations. Sotiropoulos et al. characterize a map by its size, percentage of obstruction (due to objects), and its degree of smoothness (resulting from the ground local deformations). The robot is given a navigation mission, defined by a starting position and a target arrival position, situated in the map boundaries. Both approaches only consider the random generation.

System under test description

We ran the simulations in the Player/Stage simulation environment (see Fig. 5.4), which is one of the most commonly used in the robotics field [41]. We also considered using such simulators as Gazebo[5], MORSE[6], and Argos[7]. One of the advantages of Player/Stage for our study was the possibility to load the automatically generated environment configuration files as well as the big number of implemented models and controllers. For Gazebo and MORSE the environments have to be manually created in a dedicated 3D design tool. For Argos, the maps can be generated automatically, however the number of implementation examples is limited. One of them, which includes a planning algorithm implementation, is dedicated to robot swarms, which we plan to explore more in the future [59].

[5] `http://gazebosim.org/`
[6] `https://github.com/morse-simulator/morse`
[7] `https://www.argos-sim.info/`

For simulations we used a Pioneer 3-AT mobile robot [8] model provided by the Player/Stage simulator. The robot is equipped with a SICK LMS200 laser with the sensing range of 10 meters, it has four wheels and is capable of speeds of up to 0.8m/s. One of the planning algorithms provided by the simulator is using the nearness diagram (ND) navigation method. This is a reactive navigation method, where the motion commands are computed based on the robot sensor data. The method computes the optimal motion command to avoid collisions while moving the robot toward a given goal location. Before the robot mission starts, it runs the A* planning algorithm to obtain the route towards the goal. Then it uses its ND algorithm for navigation. To increase the challenge for the robot we applied the Ramer–Douglas–Peucker algorithm [9] to reduce the number of waypoints in the created route.

Our objective is to find environments, when the navigation algorithm fails and the robot doesn't reach the goal, getting stuck during the navigation. The scenario is represented by a bitmap, where the location of obstacles is specified, as well as by a set of waypoints for the robot to follow. Failures are detected by a daemon script that continuously monitors the simulation environment.

The case study parameters are summarized in table 5.4 and described in more detail in the following paragraphs.

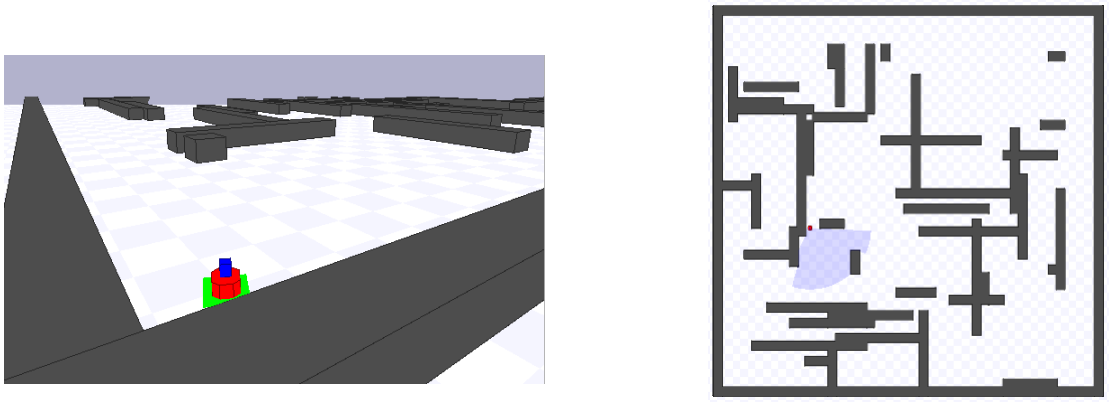
Table 5.4 Summary of the autonomous robot case study parameters

The problem	Generate a navigation map with obstacles, that a robot cannot complete without collisions.
Environment element	One part of the map, where obstacle type, obstacle size and position are specified.
Test case restrictions	Obstacles cannot block completely the path from the start to the goal position.
Fitness function F1	Maximize the length of the robot path from the start to the goal position.

Problem representation

In this scenario generation case study the environment is represented by a map with obstacles. We define the map size to be 50 x 50 m. Each environment part E_i corresponds to a space of the size 1 x 50 m. In total there are 50 E_i elements, therefore the scenario matrix size M is fixed and is equal to 50. We define three attributes describing the environment: A_1, the

[8] `https://www.generationrobots.com/media/Pioneer3AT-P3AT-RevA-datasheet.pdf`
[9] `https://rdp.readthedocs.io/en/latest/`



(a) A perspective view in the simulation environment

(b) A top view in the simulation environment, with the robot navigating in the environment using the laser range sensors (blue)

Figure 5.4 Player/Stage simulation environment for autonomous robots

type of the obstacle, A_2 position of the obstacle and A_3 the size of the obstacle. This gives the size N of the scenario matrix of 3. The values for the attributes are specified in Table 5.5. We use two types of obstacles - vertical and horizontal walls. The size - is the total obstacle length in meters. The position - is the obstacle center location in the element E_i. We distribute the obstacles by assigning to each row the position of the obstacle center.

We define two restrictions. First, R_1: only one obstacle per element E_i. Second, R_2: the obstacles cannot cover completely or intersect with the initial and target robot location points.

Table 5.5 Attributes to generate scenarios for the autonomous robot case study

A_1, *obstacle type*	A_2, *obstacle size*	A_3, *obstacle position*
[horizontal, vertical]	[5,6, ..., 15]	[1, 2, ..., 50]

Fitness function definition

The intelligent robotic systems are typically equipped with a planning algorithm that builds a path to the goal location as the robot moves through the environment. The trajectory is adjusted as the new obstacles are discovered by the robot.

In the simplified case, the robot knows about the location of all the obstacles in advance. Therefore, as the robot approximated model we are using the Python robotics implementation

of A* planning algorithm [55], which creates the route given the map, start and destination location. We have selected the A* because it is a deterministic algorithm and always finds a route, if it exists. The disadvantage is that the computations take longer time, than for non-deterministic planning algorithms such as RRT*.

The first fitness function, F_{1robot}, maximizes the distance the robot would have to travel to find the goal. For the test cases that don't meet the requirements, F_{1robot} is set to 0. The second fitness function is calculated according to (4.2).

AmbieGen configuration

We used the following GA (AmbieGen SO) and NSGA2 (AmbieGen MO) configurations: population size: 100, number of generations: 400, mutation rate: 0.4, crossover rate: 1, algorithm type: steady state with 50 offsprings, number of evaluations: 20 000.

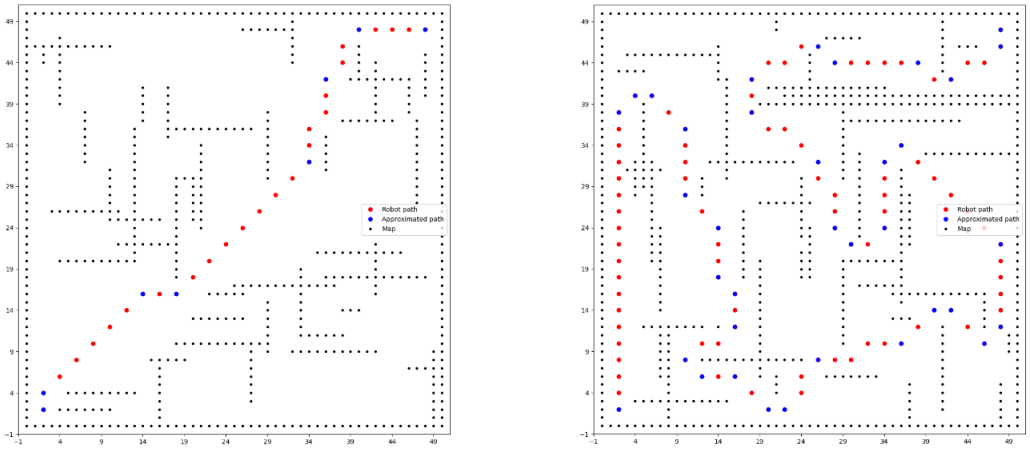
For this problem we used a smaller number of offsprings to run more generations for the same time budget. The A* algorithm implementation was computationally expensive to execute. The average time to run 20000 evaluations was 2727.2 sec for AmbieGen SO and 2394.9 sec for AmbieGen MO.

Scenario generation

In Fig.5.5 we show examples of the generated scenarios, i.e., rooms with obstacles obtained by random generation Fig. 5.5a and with AbmieGen Fig. 5.5b. In Fig. 5.5a the length of the robot path towards the goal is 78.76 meters, while in the Fig. 5.5a - it is 202.36 meters. Evidently, the second scenario poses a more challenging navigation environment for the robot, than the first scenario. The video demonstration of the fault revealed for the robot model in the Player/Stage environment can be found via the link: `https://figshare.com/s/7208f6d5ce19e1476474`.

Next we describe our navigation map generation approach in a more detail.

Approach description. The virtual map is represented by a PxP matrix, where P is the resolution. In our case study we chose to have two obstacle types: vertical and horizontal wall. The width of the wall corresponds to the map resolution. Each row of the map contains P cells. We distribute the obstacles by assigning to each row the position of the obstacle center, selected from the range $[1,\ P-1]$. Given the location of the centers we build the obstacles, removing the parts that are going out of the map bounds. Another limitation is that obstacles cannot instersect or block the start of or the target location of the robot. Appart from obstacles, it's possible to assign the terrain types, the initial location of dynamic



(a) Random scenario (b) Search produced scenario

Figure 5.5 Examples obtained robot navigation maps

objects, such as other robots. Moreover, different types of obstacles can be used.

5.3 Lane keeping assist system case study

Self-driving cars have a perspective of becoming a part of our lives in the near future. These systems are safety-critical and should be well tested to avoid unwanted consequences. Running the simulations in the virtual environments can reveal the possible faults of their control algorithms.

In this case study we generate the virtual roads to test car Lane Keeping Assist System (LKAS). The goal is to generate the roads that force the ego-car, i.e., the test subject, to drive off its lane without creating invalid roads. A number of tools were suggested for automatic generation of virtual roads, such as DeepJanus [36] and AsFault [5]. This year, four tools, such as Frenetic, Deeper, Swat, and GA-Bézier were presented at the SBST2021 tool competition [60]. The SWAT tool is the submission of the random generator based implementation of our approach for virtual road generation.

System under test description

For simulating the car and the environment, we used the simulation pipeline initially provided by [36] and adapted for the SBST2021 tool competition. This environment uses the BeamNG.tech driving simulator [61], a freely available research-oriented version of the commercial game BeamNG.drive (see Fig. 5.6). The test subject is the builtin driving agent,

BeamNG.AI. This driving agent is omniscient, i.e., it knows the geometry of the whole road and utilizes a complex optimization process to plan trajectories that drive the ego-car as close as possible to the speed limit while keeping the vehicle inside the lane. The car controller adopts a behavioural reflex approach, i.e., the deep learning component (DL) learns a direct mapping from the sensor camera input to the steering angle value to be passed to the actuators [62].

Figure 5.6 The screenshot from a BeamNG simulation environment

The case study parameters are summarized in table 5.6 and described in a more detail in the following paragraphs.

Table 5.6 Summary of the lane keeping assist system case study parameters

The problem	Generate a valid road with a trajectory that forces the car to go out of the lane.
Environment element	One road section, where the road type, its length or curvature angle are specified.
Test case restrictions	The road cannot go out of the map bounds, cannot intersect or be too sharp.
Fitness function F1	Maximize the deviation of the car from the road lane center.

Problem representation

In this case study, the test scenario is a flat road surrounded by plain green grass with the fixed weather conditions: sunny clear day. The road layout (i.e., number and width of lanes) is fixed and consists of two lanes.

Each environment element E_i corresponds to one road section. To describe the road section we define three attributes: the type of the road A_1: going straight, turning right and turning left. A_2: the length of the straight road segment, and A_3: the angle of the turn of the curved segment. The attributes representation is shown in Table 5.7. The test scenario contains 3 rows ($N = 3$) and a variable number of columns M, depending on how many road segments fit in a map. In our scenarios and at SBST competition, the map size was 200 x 200 m.

Table 5.7 Attributes to generate scenarios for the vehicle LKAS system case study

A_1, *road type*	A_2, *straight road length*	A_3, *road turn angle*
["straight", "turn left", "turn right"]	[5, 6, ..., 50]	[5, 10,..., 85]

The test cases have the following limitations: the roads can't be too sharp, can't intersect and shouldn't go out of the map bounds. Examples of valid and invalid roads are shown in Fig. 5.7.

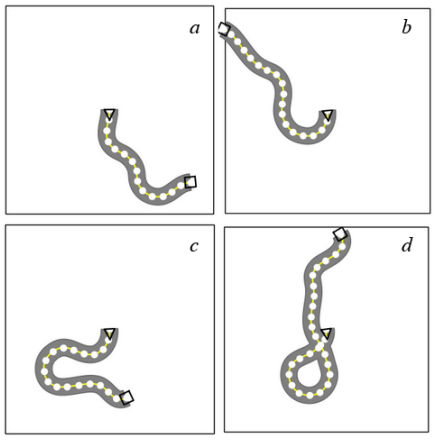


Figure 5.7 Examples of valid (a) and invalid roads: (b) - out of bounds, (c) - too sharp, (d) - intersecting

Fitness function definition

To calculate the test scenario fitness we need to create the simplified model of the car. Similarly to the thermostat problem, we built the car model from the first principles as the car movement can be described by a well known car kinematic model [63]. To describe the car movement we use the equations from [49], see Fig.5.8. To keep the car close to the lane center we adopt Stanley control [64].

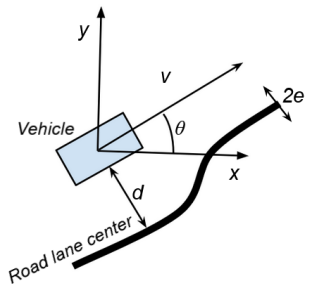


Figure 5.8 The simplified car model parameters

In the equations (5.7 - 5.10) below, x, y - are the current coordinates of the car on the map, θ is the angle between car direction and a reference plane, α and β - constants, corresponding to velocity value, d - the distance of the car from the closest point on the road. When d is smaller than a certain threshold e, the car goes straight, when d is larger than e - the car turns either left or right. The turn angle is adopted depending on the car speed and the deviation from the road lane center.

Therefore we have the following fine-tunable parameters: k, α, β and the initial speed ν_0. In order to fine tune the parameters, we created a dataset with the road points and the corresponding car model path S recorded by the simulator while executing the scenarios. Then we compared the outputs of our model with the simulated car path using such metric as a "Hausdorff distance". A similar metric, Frechet distance, was used in [65] to compare the similarity between roads. The goal was to minimize the Hausdorff distance. To perform the optimization we use the sci-py Nelder-Mead algorithm implementation. However, other optimization algorithms can be used, such as genetic algorithms. The set of the parameters that indicated the lowest average Hausdorff distance of 13.74 is shown in Table 5.8.

Table 5.8 Autonomous vehicle simplified model coefficients

ν_0	k	α	β
7	3.5	0.3	0.1

$$\dot{x} = \nu \cdot cos\theta \tag{5.7}$$

$$\dot{y} = \nu \cdot sin\theta \tag{5.8}$$

$$\dot{\theta} = \begin{cases} \tan^{-1}(\frac{k}{\nu(t)}) & if\ d < -e \\ -\tan^{-1}(\frac{k}{\nu(t)}) & if\ d > e \\ 0 & if -e \leq d \leq e \end{cases} \tag{5.9}$$

$$\dot{\nu} = \begin{cases} -\alpha & if\ d < -e, d > e \\ \beta & if -e \leq d \leq e \end{cases} \tag{5.10}$$

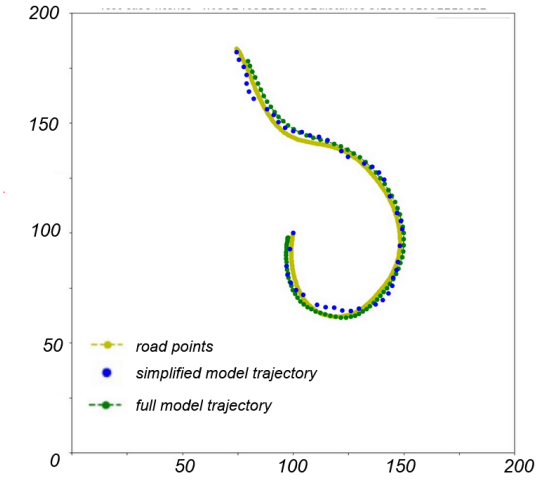


Figure 5.9 The simplified and full car model trajectory given the same road points

In Fig.5.9 you can see how the surrogate (blue points) and the full model (green points) follow the interpolated road points (yellow). The Hausdorff distance between the two roads is 5.153.

Finally, as the fitness function F_{1veh} we maximized the biggest deviation d from the lane center reached while executing the test case, as in [5] and [6]. The second fitness function was calculated according to (4.2). For all the invalid test scenarios the F_{1veh} was set to 0.

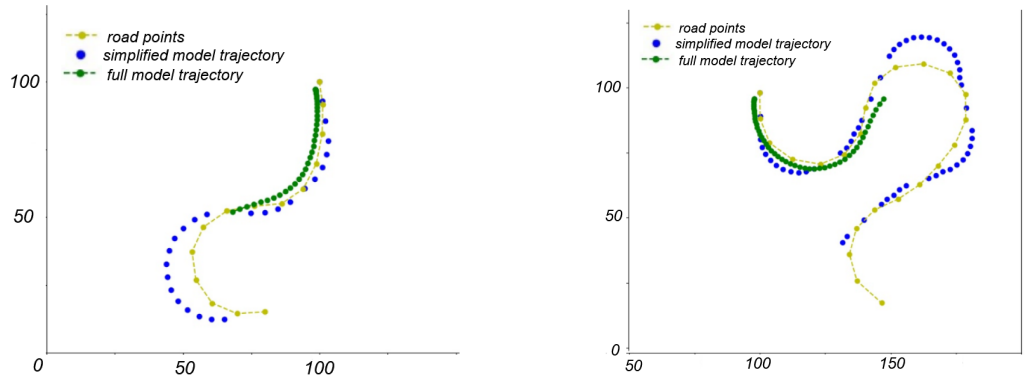
AmbieGen configuration

We used the following GA and NSGA-II configurations: population size: 500, number of generations: 200, mutation rate: 0.4, crossover rate: 1, algorithm type: generational, number of evaluations: 100 000.

We are using a higher population size, rather than the bigger number of generations, as from

our experience, with bigger population the results were more consistent across different runs. The average time to run 100000 evaluations was 1522.405 sec for GA and 1380.66 sec for NSGA2.

Scenario creation



(a) Scenario forcing the car to drive off the lane

(b) Scenario forcing the car to drive off the lane

Figure 5.10 Examples of fault revealing scenarios for vehicle lane keeping assist system

In Fig. 5.10 we show examples of the generated test cases, that forced the car to go out of the lane. The golden points correspond to the road lane center, the blue points - to the surrogate model path, green points - the full model path. When the virtual car went out of the lane bounds, the simulation recording stopped, therefore we see the full model path only for the part of the road. The video demonstration of the failure, when the car model is going out of the lane bounds during execution of one of our scenarios can be found via the link: `https://figshare.com/s/b4a096f0a66e0abbe7b1`.

Below we describe in a more detail our virtual road generation approach.

Approach description. In the current case study, the scenario is represented as a flat road surrounded by green grass. The environmental conditions and the road layout are fixed and predefined: sunny day and a two lane road. Our goal is to produce a sequence of points, defining a road lane center. It will be further interpolated with cubic splines to obtain the final road geometry. A diagram, showing our road generation process is shown in Fig. 5.11. The road generation starts from the initial vector ν_1, placed in the middle of the map. From our experience, initial placing in the middle of the map, produced better results than at the borders or random location. To create the road points $p_1 - p_7$, we then apply affine transformations to the initial vector ν_1, according to the road types specified in the

generated scenario, i.e., “straight N meters”, “turn right/left N degrees”. We use three types

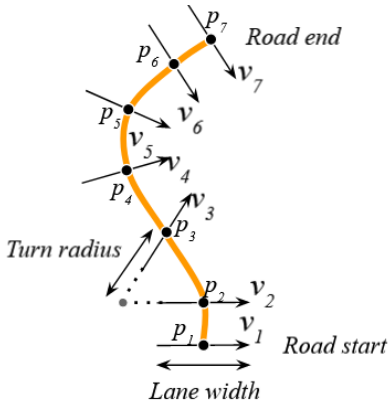


Figure 5.11 The road points (black) generated after applying affine transformations to the initial vector ν_1

of transformations:

- parallel transition, corresponding to road going straight;
- clockwise rotation, corresponding to road turning right;
- anticlockwise rotation, corresponding to road turning left.

For example, to obtain ν_2 we moved the ν_1 parallelly N meters (“straight N meters”). To obtain ν_3 we rotated ν_2 N degrees anticlockwise (“turn left N degrees”).

The sequences of transformations were generated automatically, using the Markov chain with three states: “straight”, “turn right” and “turn left”. The designed Markov chain with the probabilities for changing states is shown in Fig.5.12. We fine-tuned the probabilities empirically, so that on average, longer loads are produced. Adjusting the probabilities of transition allowed to avoid generating useless test scenarios, such as completely straight roads. To each state the value is then randomly assigned from the list of accepted values.

The next challenge is to construct valid roads from the given sequences of road sections, i.e. vector transformations. Examples of performing “left turn 15 degrees”, “right turn 15 degrees” and “straight N meters” transforms to a vector are shown in Fig.5.13.

Parallel transition is performed by moving the vector by N points in the orthogonal direction. To perform the rotation, it’s important to correctly select the rotation axis, which can be located next to the vector start or end point. The choice depends on the direction in which the vector moves along the map. For a left turn, the vector is always rotated anti-clockwise around the axis, for the right turn - clockwise.

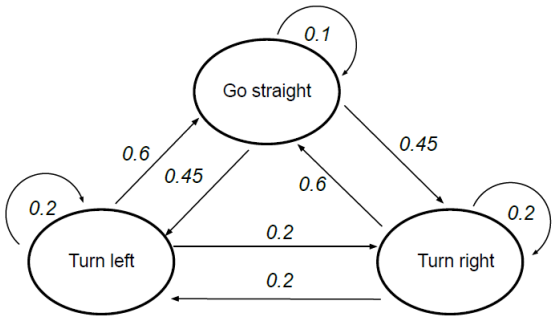


Figure 5.12 Markov chain for generating virtual roads

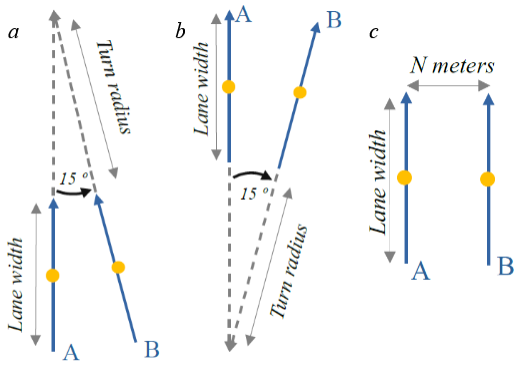


Figure 5.13 Left turn (a), right turn (b) and parallel transforms (c)

To select the rotation axis, our initial strategy was to infer the direction of the movement from the location of the previous vector. This approach, however, produced a relatively high number of invalid roads. The strategy that worked well and that we selected for our approach is illustrated in the figures 5.14 - 5.15, where a left turn is performed and vector is moving from position A to B. The idea is that first we select the axis position arbitrary. If it produces a valid transformation - we keep it, otherwise we select another position.

In Fig. 5.14, firstly, the rotation axis is selected to be on the vector end side (b). We do a small perturbation by rotating this vector anti-clockwise (clockwise for right turn) for a small angle value (2 degrees). The rotated vector appears inside the polygon defined by the current and previous vectors, which is an invalid position. Therefore, the location of rotation axis is changed to be at the vector start side. Then the rotation is performed using the new axis location (a). The intuition is that the new vector position should not intersect with the path defined by previous vectors. The road generation stops, when the road vector goes out of the road map bounds. The same steps apply to the transformation in Fig.5.15.

The implementation of this approach is openly available [66]. We submitted it to the

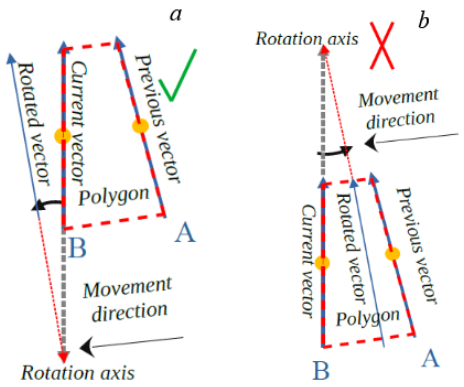


Figure 5.14 Correct choice of the axis (a) and incorrect choice of the axis (b)

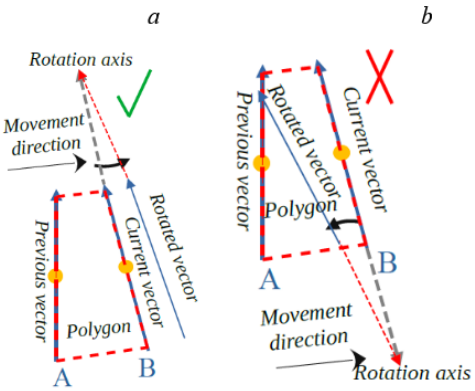


Figure 5.15 Correct choice of the axis (a) and incorrect choice of the axis (b)

SBST2021 tool competition under the name "SWAT". It achieved the highest ratio of the validly generated scenarios i.e. 95 % among other four tools. However, it only used random generation and revealed low number of failures comparing to the best results.

Below we also present other approaches described in the literature for virtual road generation. *Related works.* In the literature a number of approaches were presented to generate the virtual roads. They are using Beizer curves [65], Frenet frames [67], Catmull-Rom cubic splines [6], lanelets [68] and polylines [5]. The advantage of our approach is the possibility to encode, along with the virtual road specification, a more detailed description of the environment, such as the terrain/surface type of a particular road segment, as well as the position of static/dynamic objects on it. These descriptions can be evolved by the search algorithm together with the combination of road segments, allowing to produce more complex environments.

5.4 Chapter summary

In this chapter we present a more detailed description of the case studies we used to evaluate AmbieGen, namely a smart-thermostat, autonomous robot and a vehicle lane keeping assist system. For each of the systems we created a simplified model to calculate the first fitness function, accounting for the scenario fault revealing power. We also present the examples of the scenarios generated by AmbieGen and a more detailed description of the approach to generate them.

CHAPTER 6 EMPIRICAL EVALUATIONS

In this chapter we formulate our research questions and evaluation methods. We then present the main results of our evaluations and answers to research questions. In the end, we discuss the results.

6.1 Research questions

We evaluate our approach using the three test case generation case studies described above. For each of the case studies we answer the following research questions.

RQ1. (Comparing random, single-objective and multi-objective search). *Considering the single and multi objective versions of AmbieGen as well as the random search, which configuration produces the test scenarios with the higher fault revealing power values given the same time budget*?

Motivation: Firstly we would like to know if the use of evolutionary search is beneficial and allows to produce better solutions, than simple random search. Next, we want to know if adding additional fitness function for diversity allows to find better solutions. We expect AmbieGen MO to produce at least as good solutions as AmbieGen SO. Previous works on novelty search [69], [50] have shown that adding a fitness function for diversity may increase the convergence speed.

Experiment design: We give the same time budget to all the three algorithms in terms of number of evaluations and compare the average F_1 fitness function value of the best solutions found. We repeat the measurements 30 times.

RQ2. (Comparing diversity of the solutions found by the single-objective and multi-objective search). *To what extent the diversity of the solutions found by the multi objective AmbieGen configuration is higher than the diversity of the single objective configuration solutions?*

Motivation: This research question is aimed to quantify the difference between diversity of the solutions produced by AmbieGen So and AmbieGen MO. We expect the AmbieGen MO to produce more diverse scenarios.

Experiment design: Given the same time budget, we compare the average diversity of the best 5 solutions found by the single-objective algorithm and the average diversity of the Pareto optimal solutions found by the multi-objective algorithm. We repeat each measurement 30

times.

For the autonomous robot and lane keeping assistant case study we also answer the following question:

RQ3. (Comparing our AmbieGen with the available baselines) *To what extent does our approach perform better in generating test scenarios for the full model in comparison with the available baselines?*

Motivation: This research question is aimed to quantify the effectiveness of AmbieGen in the number of revealed failures for the full models used in simulations.

Experiment design:

Autonomous robot case study. To the best of our knowledge, there are no available test generation baselines for the autonomous robot system. Therefore we compare the generated scenarios with the random search by giving the same time budget of two hours and executing the generated environments in the robotic simulator. We repeat the experiment 30 times.

Lane keeping assist system. For the lane keeping assist system, we compare AmbieGen with the open-source approach that showed the best results in the SBST2021 tool competition [60], i.e. Frenetic tool [67]. In the competition the same test evaluation pipeline was provided to all the participants. It allowed to compare the generated test cases for the number of faults revealed (forcing the ego-car to go out of the lane), the diversity of the revealed faults and the proportion of the valid test cases. We perform the same 2 hour experiment as in the competition, averaging the results over 30 runs. We further perform additional 5 hour experiment to compare AmbieGen and Frenetic.

For all the research questions, to confirm the statistical significance of the results we performed a two-tailed non-parametric Mann-Whitney U test and measured the effect size using the non-parametric measure such as Cliff's delta (d). The magnitude is assessed using the thresholds provided in [18] , i.e. [d] $<$ 0.147 "negligible", [d] $<$ 0.33 "small", [d] $<$ 0.474 "medium", otherwise "large".

We ran all the experiments on the PC running Microsoft Windows 10 Home and featuring a quad-core AMD Ryzen 7 4800HS CPU @ 2.90 GHz, 16 GB of Memory, and an NVidia GeForce GTX 1660 GPU @ 6GB.

6.2 Results

RQ1.(Comparing random, single-objective, and multi-objective search) In the Fig. 6.1, Fig. 6.2, and Fig. 6.3, we present the best fitness value found over generations by

Random search (green boxplots), AmbieGen SO (red boxplots) and AmbieGen MO (blue boxplots) averaged over 30 runs for the three problems. We considered the fitness function accounting for the fault revealing power and described in Equation (4.1).

We compare the fitness function values found after the allowed number of evaluations with a two-tailed non-parametric Mann-Whitney U test. The obtained p-values and effect sizes of the problems are shown in the Tables 6.1, 6.2, and 6.3, respectively.

Thermostat case study. From Fig. 6.1 we can see that on average random search (yellow) converges to values of -1.608, while AmbieGen SO (red) and AmbieGen MO (blue) find the solutions with twice higher fitness value of -3. Statistical tests confirm that AmbieGen outperforms the random search with $p < 0.01$. We can observe that on average the SO converges faster than MO, however, the difference between the converged values is negligible.

Table 6.1 Results of two-tailed non-parametric Mann-Whitney U test and Cliff's delta effect sizes for the smart thermostat case study

	SO (GA)	*MO (NSGA2)*	*Random*
SO (GA)			
MO (NSGA2)	$p = 0.378$ $0.133, negligible$		
Random	$p < 0.01$ $1, large$	$p < 0.01$ $1, large$	

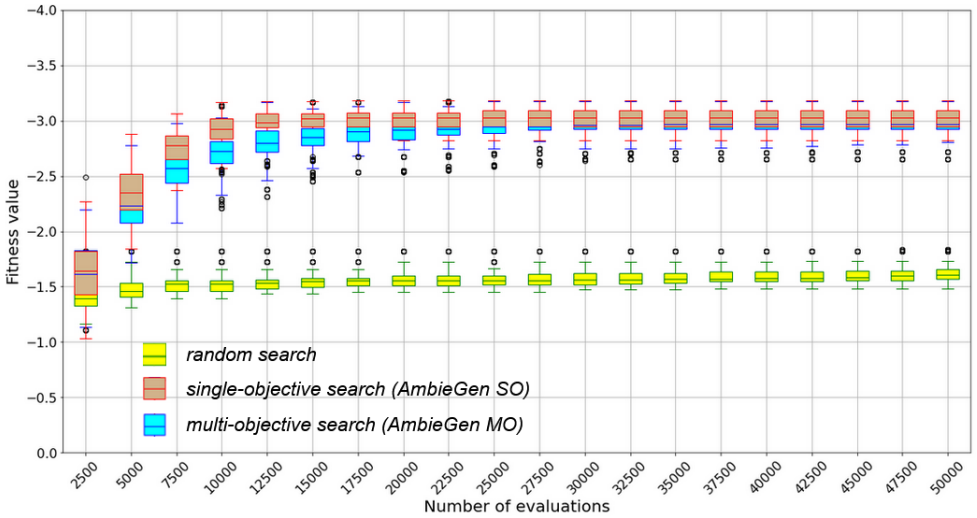


Figure 6.1 Best fitness function value over evaluations for the thermostat case study

Autonomous robot case study. After 20000 evaluations, on average, random search produced solutions with the highest fitness value of -158.89. AmbieGen outperforms the random search,

with the SO configuration producing 42 % fitter solutions of -278.2 and the MO configuration producing solutions of -251.6 fitness value. Given the same time budget, AmbieGen SO produces almost 10 % fitter solutions than AmbieGen MO.

Table 6.2 Results of two-tailed non-parametric Mann-Whitney U test and the Cliff's delta values for the autonomous robot case study

	SO (*GA*)	*MO* (*NSGA*2)	*Random*
SO (*GA*)			
MO (*NSGA*2)	$p < 0.01$ $0.772, large$		
Random	$p < 0.01$ $1, large$	$p < 0.01$ $0.978, large$	

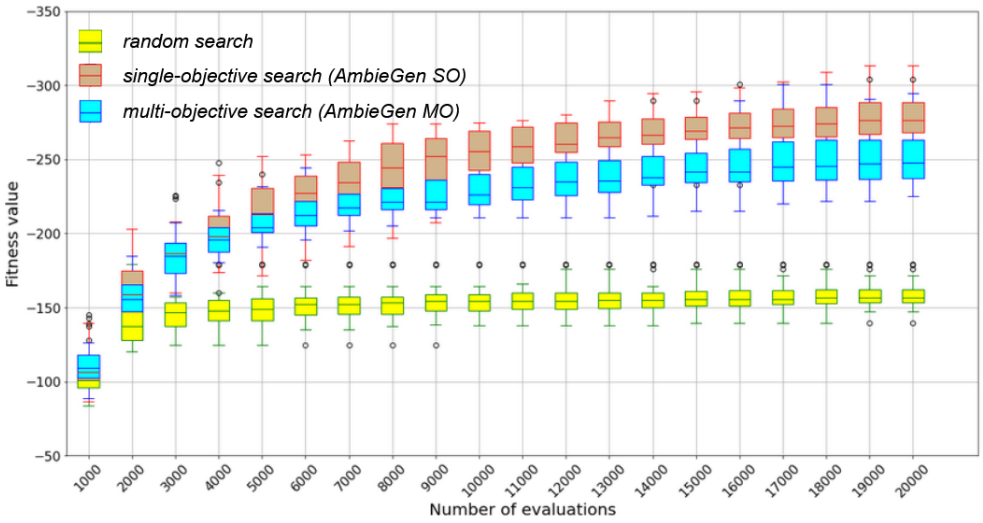


Figure 6.2 Best fitness function value over evaluations for the autonomous robot case study

Lane keeping assistant case study. In 100000 evaluations random search produced scenarios with the average highest fitness of -9. AmbieGen MO and SO produced almost 50 % fitter solutions of -17 and -16, respectively. There was no statistical difference between the best solutions of SO and MO.

Overall, We can see that for all the problems, AmbieGen finds on average from 40 % to 50 % better solutions, than the random search. AmbieGen SO and AmbieGen MO show no statistical difference in the produced solutions for the thermostat and lane keeping assistance problem. For the autonomous robot problem, the AmbieGen SO produces better solutions with a large effect size given 20000 evaluations.

RQ1 summary. AmbieGen SO produced scenarios with highest fault revealing power for the

Table 6.3 Results of two-tailed non-parametric Mann-Whitney U test and the Cliff's delta values for lane keeping assistant case study

	$SO\ (GA)$	$MO\ (NSGA2)$	$Random$
$SO\ (GA)$			
$MO\ (NSGA2)$	$p = 0.175$ $0.215, small$		
$Random$	$p < 0.01$ $0.877, large$	$p < 0.01$ $0.886, large$	

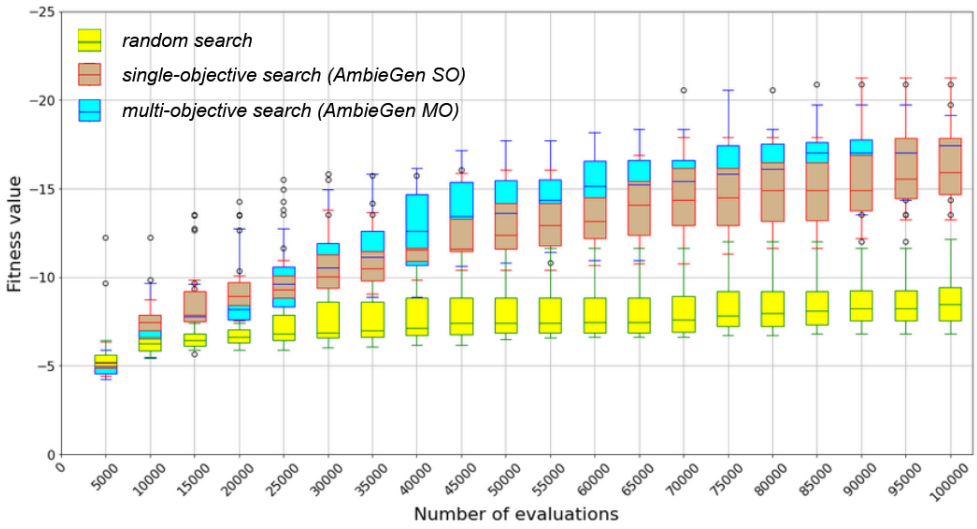


Figure 6.3 Best fitness function value over evaluations for the lane keeping assistant case study

autonomous robot case study. For the thermostat and LKAS case studies the difference in the solution fitness of AmbieGen SO and MO was negligible. Ovrerall, AmbieGen outperforms the random search with "large" effect size in all case studies.

RQ2.(Comparing diversity of the solutions found by the single-objective and multi-objective search). In Fig. 6.4, Fig.6.5, and Fig. 6.6, we compare how diverse are the produced solutions by AmbieGen SO and AmbieGen MO.

For SO, we select 10 fittest individuals and compute the diversity according to (4.2) between each pair of individuals. We report the average value. For NSGA2, we compute the diversity (4.2) between each pair of Pareto optimal solutions. The size of the Pareto front was 7 individuals on average. All the solutions in the Pareto front have a fault revealing (F_1) fitness function value higher than a certain fault-revealing threshold, established by the developer.

For all the problems, the two-tailed non-parametric Mann-Whitney U test confirmed that

AmbieGen MO produces more diverse solutions, than AmbieGen SO. For the smart thermostat problem, the MO scenarios are more diverse with a p-value smaller than 0.01 and a "large" effect size of 0.852. For the autonomous robot problem, AmbieGen MO scenarios are more diverse with a p-value smaller and a "large" effect size of 1. For the lane keeping assist system, MO scenarios are more diverse with a p-value of 0.0109 ($p \leq 0.05$) and a "medium" effect size of 0.383.

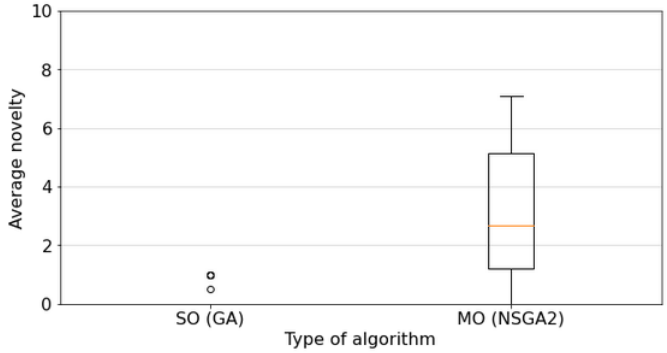


Figure 6.4 Diversity of the test cases in the last generation for the thermostat case study

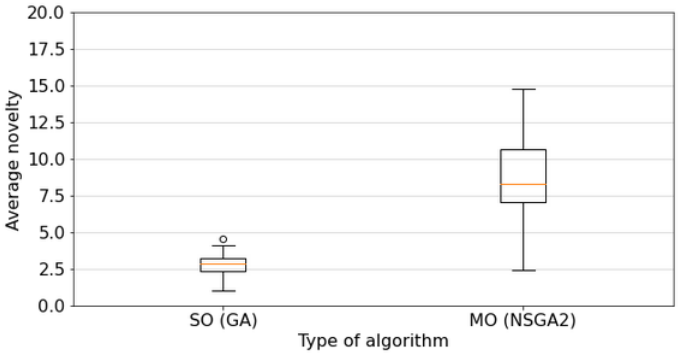


Figure 6.5 Diversity of the test cases in the last generation for the autonomous robot case study

RQ2 summary. In all the considered problems AmbieGen MO produced more diverse test cases: with "large" effect size for thermostat and robot case study and "medium" effect size for the LKAS case study.

RQ1, RQ2 summary. AmbieGen MO can find scenarios of the same quality as AmbieGen SO and better scenarios with a large effect size than the random search. Moreover, AmbieGen MO produces a more diverse set of scenarios, than AmbieGen SO. Overall, we recommend using the AmbieGen MO configuration.

RQ3. (Comparing AmbieGen with the available baselines)

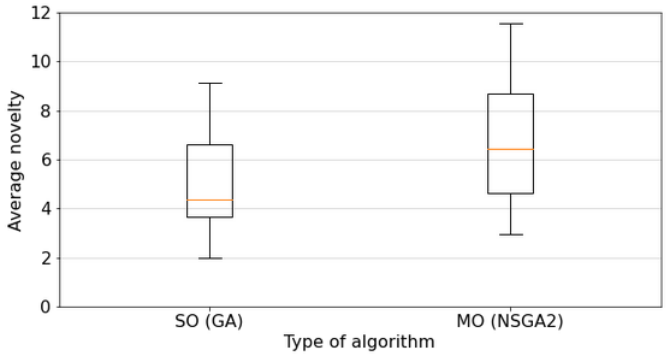


Figure 6.6 Diversity of the test cases in the last generation for the lane keeping assistant case study

Autonomous robot case study. In this subsection we compare the number of faults revealed by the NSGA2 configuration of AmbieGen (AmbieGen MO) and the random search. We created a scenario evaluation pipeline, where firstly a two hour budget is given to produce the scenarios. Then all the scenarios are passed to the simulator and executed. The daemon script monitors the execution and reports a failure when the robot stalls and doesn't reach a goal. We repeated the experiment 30 times in both configurations. You can see the average number of failures detected in Fig. 6.7. AmbieGen produced on average 9 failures in two hours, in comparison to the 2 failures of random search. AmbieGen outperforms the random search with a p-value less than 0.01 and a large effect size of 1.

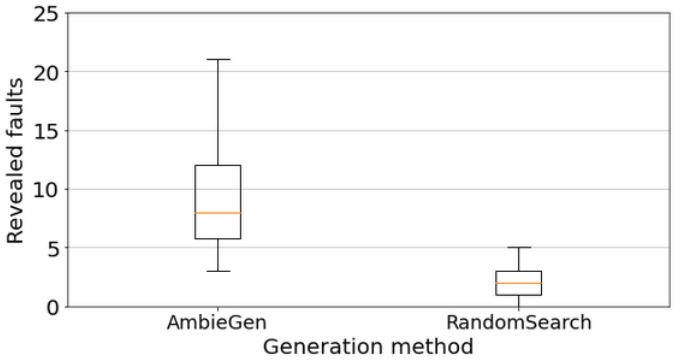


Figure 6.7 Number of faults revealed for the autonomous robot

Lane keeping assist system case study. In this subsection we report results of evaluating AmbieGen MO (AmbieGen) and the Frenetic tool (Frenetic). In addition we evaluate the random search (RS) configuration of AmbieGen and the AmbieGen MO configuration based on the full model (Full).

For AmbieGen we used a simplified configuration for virtual road generation, where only,

5100 evaluations are performed (population size 100, number of generations 200, number of offsprings - 25) in order to produce more test scenarios given a limited time budget. We gave the same time budget (5090 evaluations) for the random search to produce the solutions.

Finally, for the full model we used a configuration previously suggested by Gambi et al. [5] for Asfault tool, that also uses the full model to guide the search. In this configuration the population size is 25, number of offsprings is 4 and the number of generations is limited by the time budget, i.e, two hours.

Approaches were evaluated using the SBST2021 code pipeline [6], that integrates the test generators with the BeamNG simulator by validating, executing, and evaluating the generated test cases. We executed the SBST21 2 hour experiment, where the fault is revealed when 0.85 percent of the car area goes out of the lane. Also, the driving agent travels up to 70 Km/h.

The test cases are compared in terms of the number of faults Fig. 6.8, the diversity of the faults Fig. 6.9, and the proportion of the valid test cases Fig.6.10. The corresponding statistical test and effect size measures (Cliff's delta) are shown in the Tables 6.4, 6.5 and 6.6.

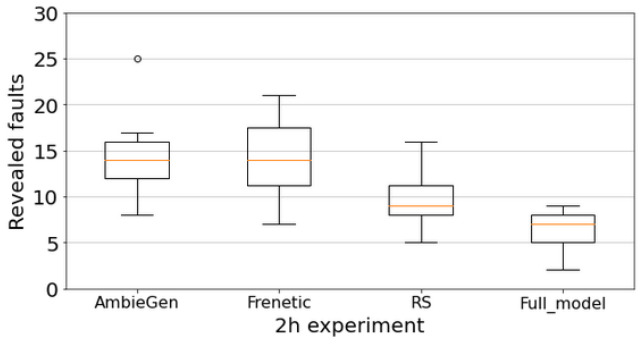


Figure 6.8 The number of revealed faults in 2h experiment

Table 6.4 Mann-Whitney test p value and Cliff's delta for the number of faults

	AmbieGen	*Frenetic*	*Full*
Frenetic	$p = 0.917$ 0.0166, *negligible*	-	-
Full	$p < 0.01$ 0.996, *large*	$p < 0.01$ 0.991, *large*	-
RS	$p < 0.01$ 0.653, *large*	$p < 0.01$ 0.578, *large*	$p < 0.01$ 0.951, *large*

In terms of the number of the revealed faults both, AmbieGen and Frenetic, statistically

outperform the random search and the full model based search. Out of 30 runs, on average, AmbieGen and Frenetic produce almost equal amount of faults, i.e., 14.

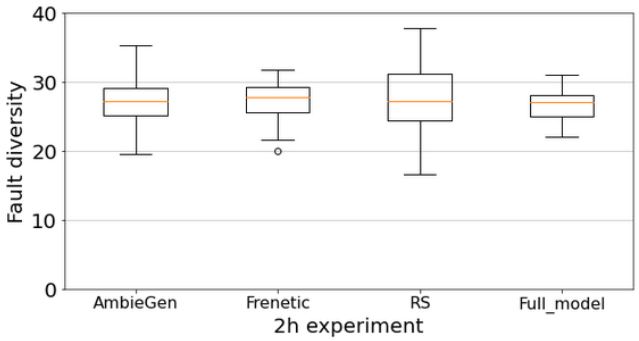


Figure 6.9 The diversity of the revealed faults in 2h experiment

Table 6.5 Mann-Whitney test p value and Cliff's delta for the fault sparsity

	AmbieGen	*Frenetic*	*Full*
Frenetic	$p = 0.897$ 0.020, *negligible*	-	-
Full	$p = 0.0889$ 0.3238, *small*	$p = 0.0998$ 0.315, *small*	-
RS	$p = 0.912$ 0.018, *negligible*	$p = 0.794$ 0.042, *negligible*	$p = 0.147$ 0.285, *small*

Concerning the diversity of the revealed faults, all the approaches have similar performance and don't show a statistically significant difference.

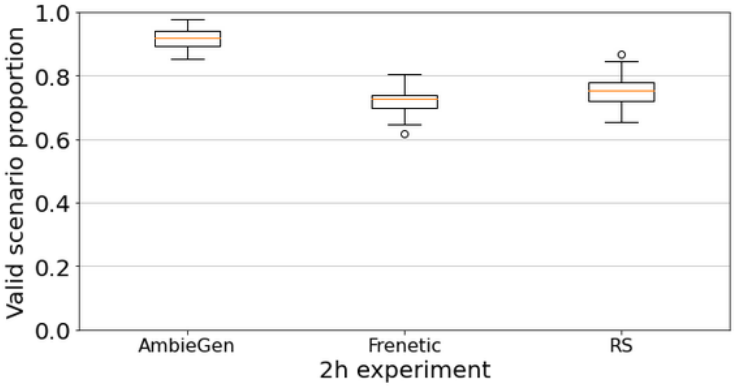


Figure 6.10 The proportion of the valid test cases in the 2h experiment

Another important factor was the proportion of valid test cases out of all the cases produced. From Table 6.6 we see that AmbieGen produces a statistically bigger proportion of the

Table 6.6 Mann-Whitney test p value and Cliff's delta for the proportion of valid cases

	AmbieGen	*Frenetic*
Frenetic	$p < 0.01$ 1, *large*	-
RS	$p < 0.01$ 0.997, *large*	$p = 0.011$ 0.386, *medium*

Table 6.7 Number of total, valid and invalid test cases in 2h experiment

	TCs	*Valid*	*Invalid*
AmbieGen	150	137.8	12.23
Frenetic	190.3	136.46	53.83
RS	86.53	65.32	21.21

valid test cases, than Frenetic and random search. For the full model, the invalid scenarios were assigned the fitness value of 0 and not submitted for evaluation. In Table 6.7 we also indicate the average number of the total produced test cases as well as the number of invalid and valid test cases. For the full number, initially the 25 individuals were produced that were later evolved by the search operators. SBST2021 code pipeline evaluates the test cases procedurally, i.e., as soon as the valid test case is produced it is executed. The new test case can only be produced, when the execution of the previous one stops. The scenario execution time depends on the generated road length, i.e., the longer the road, the more time the car will spend in the simulation. Therefore, we don't evaluate the approaches by the total number of the produced scenarios, as it depends not only on the efficiency of the algorithm, but also on the duration of the generated scenarios. The random search generates the lowest number of solutions as it only provides one solution after 5090 evaluations. AmbieGen, on the contrary, provides around 7 solutions on average, corresponding to the search Pareto front after 5090 evaluations.

Finally, we conducted additional 5 hour experiment to compare the approaches with the highest number of revealed faults from the 2 hour experiment i.e. AmbieGen and Frenetic (30 runs of 5 hours for each tool). The boxplots for the number of revealed faults, their diversity and the proportion of valid to total generated test scenarios is shown in Fig. 6.11, Fig. 6.12 and Fig. 6.13 respectively.

On average AmbieGen revealed 40 faults, comparing to 35 of Frenetic (p value of 0.001 and "large" effect size of 0.48). The average diversity for AmbieGen was 29.95 with negligible difference from Frenetic - 29.18 (p value = 0.488, "negligible" effect size of 0.104). For

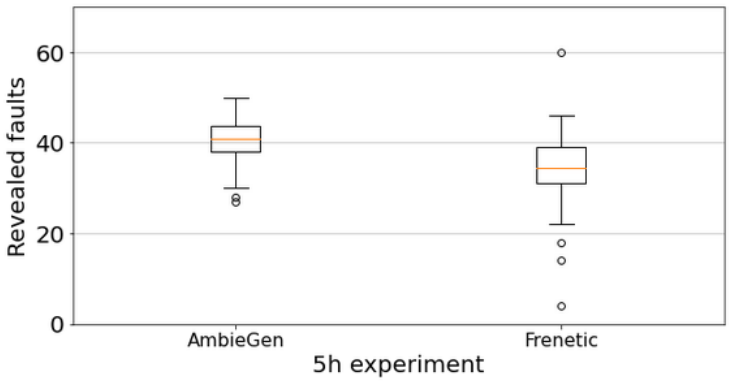


Figure 6.11 The number of revealed faults in 5h experiment

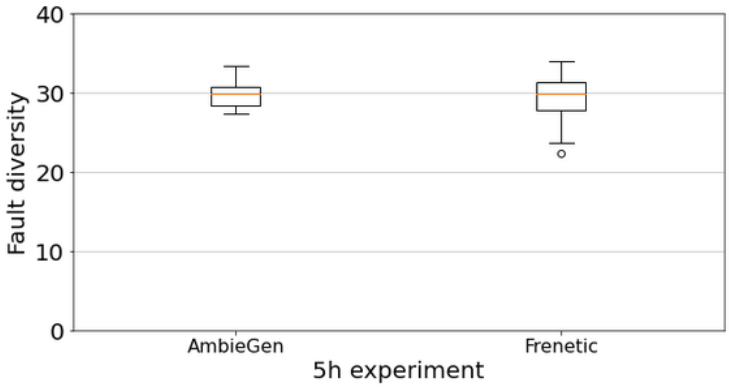


Figure 6.12 The diversity of the revealed faults in 5h experiment

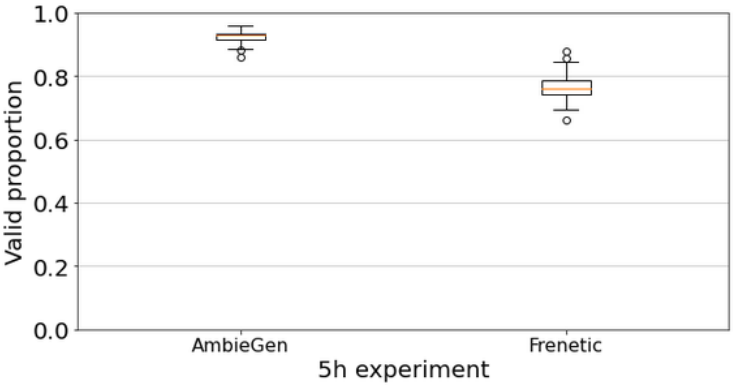


Figure 6.13 The proportion of the valid test cases in 5h experiment

AmbieGen 0.922 of generated scenarios were valid, for Frenetic - 0.766 ($p < 0.001$, "large" effect size of 0.99). Overall, in 5 hour budget AmbieGen reveals 5 more faults, than Frenetic, with the faults being of similar diversity. AmbieGen produces a bigger proportion of valid

scenarios.

Table 6.8 Number of total, valid and invalid test cases in 5h experiment

	TCs	*Valid*	*Invalid*
AmbieGen	377.8	348.36	29.49
Frenetic	422.75	324.21	98.54

RQ3 summary. AmbieGen reveals 9 failures in two hours, in comparison, random search could reveal only 2 failures for the robot case study. Both AmbieGen and state of the art Frenetic tool revealed 14 failures in two hours. In 5 hour budget, AmbieGen revealed on average 40 failures, compared to 35 of Frenetic. The revealed faults have similarly high diversity for both tools. AmbieGen outperforms Frenetic in the proportion of the valid generated scenarios and the number of revealed faults in 5 hour budget. AmbieGen also outperforms the random search and the full model configuration in the number of revealed faults.

6.3 Discussion

Evolutionary algorithms for scenario generation. Evolutionary algorithms were proven to be effective, comparing to random generation, to create virtual environments for testing automotive systems in previous works such as [5], [36], [2]. The implementation of such algorithms to generate environments is rather challenging as the customized solution representation and search operators need to be developed.

In our work we apply the evolutionary search for environment generation to such domains as smart-homes, autonomous robots and autonomous vehicles. This work is the first stage in designing a framework for generating virtual environments, AmbieGen. We consider the complete virtual environment to be composed of separate elements. Each element is described with a fixed number of attributes. During the search we recombine the elements as well as their attributes. One of the advantages of such representation is the simplicity of implementation of initial population generation, crossover and mutation search operators. Therefore the developer only needs to consider a high level description of the problem and not concentrate on the design of search operators and solution representation. By adding more attributes, the scenario complexity can be increased. For the smart-thermostat, for example, we can add such attributes as the humidity inside the room and temperature outside the room for each time period. For the autonomous robot - the terrain type and indicate the presence of other robots. For the car, for each road section we can indicate the terrain type,

the road ramp slope, the location of the other vehicles, etc.

Overall, our study confirms the effectiveness of search based approaches for environment generation. Our framework is aimed to reduce the effort of the developers of evolutionary algorithms to test the autonomous CPS. We provide the structure of the solution representation and search operators, which can be applied to generation of different types of environments. We provide examples of generating smart-thermostat schedules, maps with obstacles and virtual roads with search algorithm implementation based on Pymoo framework.

Using simplified system models. We explore the possibility to use the approximated system models, rather than the full models to compute the fitness function. Evidently, full models can detect failures with a higher precision, however they are more expensive to execute in terms of resources and time budget. For instance, the recommended requirements for running BeamNg simulator are 16 GB RAM, Nvidia GeForce GTX 970 videocard and Intel Core i7-6700 3.4Ghz processor or better. Our evaluations have shown that the full model failures can be detected by an approximated model. Moreover, given the same time budget the search guided be the approximated model may reveal more faults, than when guided by the full model. We advocate for the development of more precise simplified CPS models and making them open source, so that they can be easily used by researchers to calculate the search fitness functions. The possibility to use the surrogate models was first suggested by [7], however it was used only to generate the CPS inputs. In [2] the approximated models were used to generate the environments, however no comparison with the full model configuration was provided.

In conclusion, we advocate for creation of open source approximated and full models of CPS. Moreover, it is important to establish more test evaluation pipelines and baselines, similar to LKAS system for other domains of CPS. Finally, we surmise that the CPS simulators should provide a possibility to create environment from configuration files or an API to automate the design of environments.

6.4 Chapter summary

In this chapter we presented our research questions as well as the results of our experiments. Overall, we recommend using the multi-objective configuration of AmbieGen, AmbieGen MO, as it can find scenarios of the same quality as AmbieGen SO and better scenarios with a large effect size than the random search. Moreover, AmbieGen MO produces more diverse scenarios from “large” to “medium” size, than the single-objective configuration. Finally, we present the comparison with existing baselines. Comparing with the Frenetic tool AmbieGen

revealed the same amount of failures in two-hour time budget and 12.5 % more failures in 5 hours. Our approach also outperforms the random search and the full model based search in the number of revealed failures.

We discussing our results we advocate for the development of more precise simplified CPS models and making them open source, so that they can be easily used by researchers to calculate the search fitness functions. Moreover, it is important to establish more test evaluation pipelines and baselines, similar to LKAS system test evaluation pipeline, used in SBST2021 competition, for other domains of CPS.

CHAPTER 7 THREATS TO VALIDITY

In this chapter we discuss the possible threats to the validity of our study as well as our strategies for mitigating them.

Internal validity. To minimize the threats to internal validity, relating to experimental errors and biases, whenever available, we used standardized frameworks for development and evaluation. We implemented all the evolutionary search algorithms (GA and NSGA2) using a standartd Python Pymoo framework. To evaluate the scenarios for the LKAS case study we used a standardized test pipeline used for SBST2021 workshop tool competition. For autonomous robot case study we created a customized test evaluation
pipeline, which is based on the open source Player/Stage robotic simulator. It provides implementations of the widely used robotic models, such as Pioneer 3-AT, and planning algorithms. This simulator was previously used by researchers to conduct similar evaluations, as in [39].

Conclusion validity. Conclusion validity is related to random variations and inappropriate use of statistics. To mitigate it, we followed the guidelines in [17] for search-based algorithm evaluation. We ran each evaluation at least 30 times and ensured the statistical significance of the results by using a two-tailed non-parametric Mann-Whitney U test and Cliff's delta.

Construct validity. Construct validity is related to the degree to which an evaluation measures what it claims. To compare the test generation algorithms we gave the same time budget to all the algorithms to produce the solutions. For all the algorithms we evaluated the best fitness found, accounting for the scenario fault revealing power. To compare the tools in terms of number of revealed faults we gave each tool the same time budget to produce the scenarios. To measure the diversity of the test scenarios we used a standard metric such as Jaccard distance, previously used in other studies to compare the difference between the test cases. The exact implementation of this metric is, however, case study specific and thus can introduce some additional bias. Furthermore, the results produced by AmbieGen depend on the implementation of the approximated model. Presumably, higher quality surrogate models can produce more failures of the full model and improve the AmbieGen performance.

External validity. External validity relates to generalizability of our results. We demonstrated how our framework can be applied to generate environments for three different autonomous cyber-physical agents. However, we only considered a limited number of test subjects and limited levels of environment complexity. Therefore more problems should be addressed with different agents and higher environment complexity to make definitive con-

clusions about generalizability of AmbieGen. Nonetheless, our evaluations demonstrated that AmbieGen was effective in revealing unwanted behaviours for all the three considered autonomous CPS agents.

CHAPTER 8 CONCLUSION

8.1 Summary

In this thesis we presented AmbieGen, a framework for generating virtual environments for testing autonomous cyber-physical systems. It leverages evolutionary search guided by the approximated model of system. We applied it to generating scenarios for the smart-thermostat, autonomous robotic system and vehicle lane keeping assist system. Given the same time budget, AmbieGen could generate on average twice fitter solutions, than random search. Moreover, AmbieGen was effective at detecting faults of the full model. In two hours it could find 9 failures of the Pioneer 3-AT mobile robot in the Player/Stage simulator, comparing to only two failures found by random search. For the full model of the vehicle, equipped with lane keeping assist system, AmbieGen found 14 failures on average, the same as the state of the art baseline - Frenetic. Random search only found 8 failures on average. In 5 hour budget AmbienGen produced on average 12.5% more failures than Frenetic. AmbieGen also outperformed the Frenetic in the number of valid generated scenarios with a large effect size.

Comparing the two proposed configurations of AmbieGen, the single objective (AmbieGen SO) and multi objective (AmbieGen MO), AmbieGen SO may find fitter solutions than AmbieGen MO given the same time budget. In two hours, for the autonomous robot case study, AmbieGen So found 10 % fitter soloutins than AmbieGen MO. For the other case studies the difference in the best found solutions fitness was insignificant. Overall, we recommend using the multi objective configuration of AmbieGen, AmbieGen MO, as it always produced a more diverse set of solutions with medium to large effect size and on average could find almost as fit solutions as AmbieGen SO.

8.2 Discussion

In this work we design a prototype of a framework for automatic scenario generation for autonomous cyber-physical systems. We evaluated our framework generating three different types of virutal environments: environmental conditions for a smart thermostat along with a temperature schedule to follow, a map with obstacles for an autonomous robot and virtual roads for a vehicle LKAS system. For each case study, we used the same scenario representation, as well as the crossover and mutation operators. Currently each case study had to be configured manually, with minimal changes to the genetic algorithm implementation. In the

future, when using such framework the researchers and practitioners would have to provide the system model and only the high level information about the system virtual environment. The framework will automatically modify the search algorithm configuration and generate the test scenarios. This is our vision of full functionality of the framework, that we plan to implement in the future work.

Challenges. The challenging part of AmbieGen implementation is in evaluating the test scenario fitness. It consists of two stages: first is to convert the high level description matrix TC to the environment configuration. For the smart thermostat we needed to convert the TC matrix to the list of temperatures to follow, for an autonomous robot - to the coordinates of obstacles in a map, which was rather simple. For the LKAS case study we needed to transform the TC matrix to a set of 2D coordinates, that will produce valid roads after cubic spline approximation. This conversion was more complex and we developed a new technique leveraging affine transformations to vectors. Next challenge was to create an approximated model. For the autonomous robot we used an implementation provided by Python Robotics project. For the LKAS, we implemented the model from scratch. The available open source implementations were rather time consuming to execute. We also created the model from the real data for the thermostat case study as we didn't find any open source full models.

Finally, it was challenging to find baselines and pipelines to evaluate the produced scenarios. For the autonomous robot, we implemented a simple test evaluation pipeline, based on the Player/Stage simulator. More advanced simulators require the manual creation of the scenarios in the 3D design tools. Fortunately, for the LKAS case study we could use the test evaluation pipeline provided by the SBST2021 competition.

8.3 Limitations of the proposed approach

1. In our study we aim to develop a framework for automatic generation of testing scenarios, however the developer stills needs to provide some components such as simplified system model, the fitness function as well as the code to translate the encoded scenario for search algorithm into the input parameters for the simulator.

2. Theoretically, our framework allows encoding complex environments, including different terrain types, various dynamic and static objects, environmental conditions. The practical implementation of more complex scenarios and their effectiveness will be explored in our future works.

8.4 Future research

We plan to continue the research in four directions. First is creating more complex environments, taking into account the weather, environmental conditions and the moving obstacles such as other robots or cars. We also plan to expand the scenario generation to other CPS, such as drone and robot swarms. Secondly, we will explore the possibility to create more precise surrogate model using the system identification techniques, including neural networks and NARIMAX models. Thirdly, it is important to have the pipelines for evaluating the generated scenarios. We plan to improve our evaluation pipeline for autonomous robots by using more sophisticated simulators such as Argos and Gazebo and more complex models of robots. We will also work on developing pipelines dedicated to other types of CPS Finally, we plan to implement AmbieGen as a python framework with an API for generating virtual environments and make it open source.

REFERENCES

[1] A. Aerts, M. Reniers, and M. R. Mousavi, "Model-based testing of cyber-physical systems," in *Cyber-Physical Systems*. Elsevier, 2017, pp. 287–304.

[2] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 63–74.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[4] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf

[5] A. Gambi, M. Mueller, and G. Fraser, "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*, 2019, pp. 318–328.

[6] V. Riccio and P. Tonella, "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*, 2020, pp. 876–888.

[7] C. Menghi, S. Nejati, L. Briand, and Y. I. Parache, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 372–384.

[8] J. Minguez and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[9] D. Humeniuk, G. Antoniol, and F. Khomh, "A search-based framework for automatic generation of testing environments for cyber-physical systems," 2021. [Online]. Available: https://github.com/dgumenyuk/Environment_generation.git

[10] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical software engineering and verification*. Springer, 2010, pp. 1–59.

[11] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–12.

[12] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," *Automated Software Engineering*, vol. 24, no. 2, pp. 233–294, 2017.

[13] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

[14] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.

[15] G. Antoniol, M. Di Penta, and M. Harman, "Search-based techniques applied to optimization of project planning for a massive maintenance project," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 240–249.

[16] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[17] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[18] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, vol. 13, 2006.

[19] B. Liu, H. Zhang, and S. Zhu, "An incremental v-model process for automotive development," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016, pp. 225–232.

[20] A. Roger, "Managing model-based design." The MathWorks, Inc., 2015. [Online]. Available: https://www.mathworks.com/content/dam/mathworks/ebook/gated/MBD_Book_PDF_Version.pdf

[21] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, "Model-based testing of reactive systems," in *Volume 3472 of Springer LNCS*. Springer, 2005.

[22] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.

[23] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[24] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[25] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Search-based test case generation for cyber-physical systems," in *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 688–697.

[26] S. Raj, S. K. Jha, A. Ramanathan, and L. L. Pullum, "Work-in-progress: testing autonomous cyber-physical systems using fuzzing features from convolutional neural networks," in *2017 International Conference on Embedded Software (EMSOFT)*. IEEE, 2017, pp. 1–2.

[27] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 132–142.

[28] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 63–78.

[29] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the design and analysis of artificial intelligence-based systems," *arXiv preprint arXiv:1902.04245*, 2019.

[30] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 719–726.

[31] D. Nalic, A. Eichberger, G. Hanzl, M. Fellendorf, and B. Rogic, "Development of a co-simulation framework for systematic generation of scenarios for testing and validation of automated driving systems," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 1895–1901.

[32] M. Fellendorf and P. Vortisch, "Microscopic traffic flow simulator vissim," in *Fundamentals of traffic simulation*. Springer, 2010, pp. 63–93.

[33] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 1443–1450.

[34] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 1016–1026.

[35] T. International, "Prescan simulation of adas and active safety," 2016.

[36] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE '20. Association for Computing Machinery, 2020, p. 13 pages.

[37] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[38] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *IEEE Transactions on computational intelligence and AI in games*, vol. 3, no. 3, pp. 245–259, 2011.

[39] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2013, pp. 33–44.

[40] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[41] M. Kranz, R. B. Rusu, A. Maldonado, M. Beetz, and A. Schmidt, "A player/stage system for context-aware intelligent environments," *Proceedings of UbiSys*, vol. 6, no. 8, pp. 17–21, 2006.

[42] T. Sotiropoulos, G. Guiochet, I. Ingrand, and W. Waeselynck, "Virtual worlds for testing robot navigation: a study on the difficulty level," in *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 153–160.

[43] D. A. Ashlock, T. W. Manikas, and K. Ashenayi, "Evolving a diverse collection of robot path planning problems," in *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 2006, pp. 1837–1844.

[44] F. Gruau, "Neural network synthesis using cellular encoding and the genetic algorithm," 1994.

[45] C. D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, "Evolutionary testing of autonomous software agents," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 2, pp. 260–283, 2012.

[46] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: A bdi reasoning engine," in *Multi-agent programming*. Springer, 2005, pp. 149–174.

[47] X. Zou, R. Alexander, and J. McDermid, "Testing method for multi-uav conflict resolution using agent-based simulation and multi-objective search," *Journal of Aerospace Information Systems*, vol. 13, no. 5, pp. 191–203, 2016.

[48] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.

[49] R. Alur, *Principles of cyber-physical systems*. MIT press, 2015.

[50] J.-B. Mouret, "Novelty-based multiobjectivization," in *New horizons in evolutionary robotics*. Springer, 2011, pp. 139–154.

[51] R. Romijn, L. Özkan, S. Weiland, J. Ludlage, and W. Marquardt, "A grey-box modeling approach for the reduction of nonlinear systems," *Journal of Process Control*, vol. 18, no. 9, pp. 906–914, 2008.

[52] L. Ljung and T. Glad, *Modeling of dynamic systems*. Prentice-Hall, 1994, no. BOOK.

[53] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[54] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[55] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Pythonrobotics: a python code collection of robotics algorithms," 2018.

[56] C. Zid, D. Humeniuk, F. Khomh, and G. Antoniol, "Double cycle hybrid testing of hybrid distributed iot system," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. New York, NY, USA: Association for Computing Machinery, 2020, p. 529–532. [Online]. Available: https://doi.org/10.1145/3387940.3392218

[57] R. Winterton, "Newton's law of cooling," *Contemporary Physics*, vol. 40, no. 3, pp. 205–212, 1999.

[58] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

[59] V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, "Swarm relays: Distributed self-healing ground-and-air connectivity chains," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5347–5354, 2020.

[60] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "SBST tool competition 2021." in *In International Conference on Software Engineering, Workshops, Madrid, Spain*. ACM, 2021.

[61] BeamNG.tech, "Beamng gmbh." 2021. [Online]. Available: https://www.beamng.gmbh/research

[62] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[63] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, "Guidelines in nonholonomic motion planning for mobile robots," in *Robot motion planning and control*. Springer, 1998, pp. 1–53.

[64] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American control conference*. IEEE, 2007, pp. 2296–2301.

[65] F. Klück, L. Klampfl, and F. Wotawa, "Automatic generation of challenging road networks for alks testing based on bezier curves and search," *arXiv preprint arXiv:2103.01288*, 2021.

[66] D. Humeniuk, G. Antoniol, and F. Khomh, "Swat tool," https://github.com/dgumenyuk/tool-competition-av.git, 2021.

[67] E. Castellano, A. Cetinkaya, C. Ho Thanh, S. Klivovits, X. Zhang, and P. Arcaini, "Frenetic at the SBST 2021 tool competition," *International Conference on Software Engineering, Workshops, Madrid, Spain*, 2021. [Online]. Available: https://github.com/ERATOMMSD/frenetic-sbst21/blob/main/src/frenetic-sbst21-preprint.pdf

[68] M. Althoff, S. Urban, and M. Koschi, "Automatic conversion of road networks from opendrive to lanelets," in *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, 2018, pp. 157–162.

[69] E. D. De Jong, R. A. Watson, and J. B. Pollack, "Reducing bloat and promoting diversity using multi-objective methods," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 2001, pp. 11–18.