**Use of Hill Climbing Search for finding Crash scenarios in Autonomous Cars**

*Nima Meghdadi*

*Nmegh082@uottawa.ca*

*300368777*

Abstract

This report presents a novel testing approach for Advanced Driver Assistance Systems (ADAS), which are complex software systems that assist drivers in autonomous cars. Our approach combines multi-objective search and surrogate modeling based on neural networks to test ADAS in a simulated environment. The surrogate modeling allows us to efficiently explore a large input space with limited computational resources. We apply our approach to an industrial ADAS system and demonstrate its effectiveness in generating test cases that reveal critical ADAS behaviors. We also show that our approach improves the quality of the test cases compared to using only the multi-objective search, especially when the computational resources are scarce. The aim of this project is to generate test cases for a crash scenario problem using a surrogate model and an optimization algorithm. A surrogate model is a simplified approximation of the original problem that can be evaluated faster and cheaper. We use a neural network as our surrogate model, which is trained on synthetic data generated by a simulator. We implemented a hill climbing algorithm as our optimization algorithm. A hill climbing algorithm is a simple and efficient method that can find local optima. We compare the performance of the hill climbing algorithm with a random search algorithm, which is a naive technique that samples solutions randomly from the search space. We evaluate the quality of the solutions found by both algorithms and analyze their characteristics and implications. The source code is available in Github (https://github.com/NimaMeghdadi/Carla\_scenario\_test\_HillClimbing).

**Keywords**

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

1. Introduction

Autonomous vehicles require reliable and safe software systems to operate. Testing these systems is a crucial challenge that can be addressed by using simulation techniques. Simulation allows testing the system models at the design time, before deploying them on real vehicles. This is a practical and effective method for verifying the functionality and performance of software systems for autonomous driving.Simulation is a powerful technique for testing software systems for autonomous driving. However, two major challenges hinder its full potential:

1) The lack of intelligent and automated simulation tools that can generate and guide simulation scenarios towards revealing faults. Therefore, engineers have to manually select and design the scenarios, and use simulation only to verify them.

2) The high computational cost of running simulation scenarios. Therefore, only a limited number of scenarios can be explored within a given testing time.

In this paper, we address these two challenges by proposing novel solutions for simulation-based testing. We demonstrate how our solutions can improve the effectiveness and efficiency of testing software systems for driver assistance. Figure 1 shows the map and environment of CARLA [1], a synthetic dataset generated by an open simulator . CARLA provides various sensors and measurements for testing the control and perception software components of self-driving. Our project consists of three main sections: 1) Creating dataset, 2) Training neural network, and 3) Searching with hill climbing. In the first section, we use CARLA, an open-source simulator for autonomous driving research, to generate synthetic data for our problem. We set up a test scenario in map1 of CARLA, where we have two main actors: a car and a pedestrian. We assign different parameters to these actors, to control their behavior in the scenario. We then save the results of the simulation. In the second section, we train a neural network to approximate the results of the simulation without actually running it. We use the synthetic data from the first section as the input and output of the neural network. In the third section, we use a hill climbing algorithm, a simple optimization technique, to search for critical scenarios that can cause a crash. We use the neural network from the second section as a surrogate model, meaning that we evaluate the scenarios using the neural network instead of the simulator.



Figure 1: Map1 in CARLA Simulator that is used for scenarios

**Challenges**: One of the limitations of the simulation platform is the lack of adequate documentation. The simulator does not have clear and comprehensive instructions that would facilitate its use. Another limitation of the simulation platform is the absence of pedestrian collision scenarios in the predefined set of scenarios. A third limitation of the simulation platform is the high computational cost of physics-based simulations. These simulations require a lot of processing power and memory to model the realistic interactions between the car, the pedestrian, and the environment. Therefore, running these simulations can be time-consuming and resource-intensive.

2.Related works

Abdessalem et al. [2] tested a new approach PeVi that focuses on the Pedestrian Detection Vision (PeVi) system that uses multi-objective search and neural networks to generate test cases for Advanced Driver Assistance Systems (ADAS). The approach automatically found test cases that showed critical ADAS behavior, such as failures or weaknesses.

Abdessalem et al. [3] evaluated the performance of the hybrid test objectives on testing autonomous cars. The hybrid test objectives combined multiple criteria, such as coverage, diversity, and severity, using multi-objective search and neural networks. The experiment compared the hybrid test objectives with two standard test objectives used in the software testing literature (i.e., coverage-based and failure-based test objectives).

Abdessalem et al. [4] presents a system testing algorithm for Advanced Driver Assistance Systems (ADAS) that combines evolutionary search algorithms and decision tree classification models. The algorithm aims to generate critical test scenarios efficiently and accurately characterize critical regions in the input space of ADAS systems. The evaluation of the algorithm on an industrial ADAS demonstrates its superiority over a baseline algorithm, generating a significantly higher number of distinct critical test scenarios. The paper also highlights the practical benefits of the algorithm, as reported by interviews with engineers, including the ability to debug systems, identify hardware changes for increased safety, and specify conditions that may lead to ADAS failures. Overall, this paper contributes to the understanding and improvement of ADAS systems by providing a comprehensive approach to system testing.

A. Assadi et al. [5] uses microscopic simulation to examine how network-level traffic flow descriptors vary under different conditions, and how this affects the safety of Advanced Driver Assistance Systems (ADAS) and highly automated vehicles.

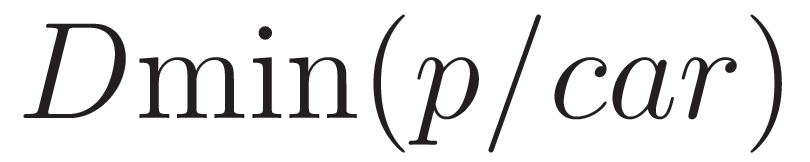
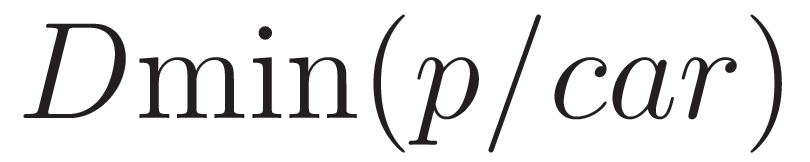
J. E. Stellet et al. [6] This paper introduces a three-dimensional taxonomy for testing advanced driver assistance systems (ADAS), which are technologies that assist drivers in various tasks, such as lane keeping, collision avoidance, and parking. The taxonomy consists of three axes: the level of automation, the test environment, and the test objective. The paper argues that this taxonomy can help to organize and compare different testing methods and challenges for ADAS, especially as they evolve towards higher levels of automation.

3.Proposed method

In this section, we present our technical details and our proposed method. We first explain the fitness functions that we used to evaluate the solutions for our problem. Then, we discuss the surrogate models that we considered and the advantages of the one that we selected. Finally, we describe our method for searching in the search space and compare it with other methods.

3.1.Fitness functions

This project uses two fitness functions to generate test cases for a crash scenario problem. The fitness functions are: 1) Minimum distance between the car and the pedestrian when the car tries to avoid collision. 2) Minimum time to collision. These fitness functions measure how close the car and the pedestrian are to each other and how much time they have to react. The lower the values of these fitness functions, the higher the risk of a crash.

1. One of the criteria we used to evaluate the crash scenarios was the minimum distance between the car and the pedestrian: We defined a function, [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=D_%24min%24%20(p%2Fcar%20)#0), that calculates this distance based on the simulation time. The function takes the distance, [](https://www.codecogs.com/eqnedit.php?latex=D(p%2Fcar)(t)#0), between the car and the pedestrian at any given time, *t*, and returns the smallest value within the simulation time interval, *T*. The function [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=D_%24min%24%20(p%2Fcar)#0) is expressed as follows:  
   [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=D_%24min%24%20(p%2Fcar)%20%3D%20Min%7BD(p%2Fcar%20)(t)%7D%20(0%20%3C%20t%20%3C%20T)%20#0).
2. Another criterion we used to evaluate the crash scenarios was the minimum time to collision, denoted by [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=TTC_%24min%24%20#0): This is the time it would take for the car to collide with the pedestrian, assuming they maintain their speed and direction at any given time, t. We calculated [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=TTC_%24min%24%20#0) by finding the smallest value of [](https://www.codecogs.com/eqnedit.php?latex=TTC(t)#0) within the simulation time interval, *T*. [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=TTC%24min%24%20#0) is a useful indicator of the collision risk and the severity of the traffic situation. Therefore, we aimed to generate scenarios with a low [](http://www.sciweavers.org/tex2img.php?bc=Transparent&fc=Black&im=jpg&fs=100&ff=modern&edit=0&eq=TTC_%24min%24%20#0), as these are the most dangerous ones.

3.2.Surrogate model

In this project, we employed surrogate models to reduce the computational cost of running physics-based ADAS simulations. To evaluate the crash scenarios, we needed to run two fitness functions, as explained before. However, these functions required costly physics-based simulations. Therefore, we built a surrogate model for each function that could estimate the fitness values without performing the simulations. We considered different neural network methods for developing the surrogate models, such as AdaBoostRegressor, Regression, Nearest Neighbors Regression, and MLPRegressor. We chose MLP regressor. MLPRegressor as our surrogate model because it can capture complex patterns in the data. MLP regressor is a type of artificial neural network that consists of multiple layers of neurons with nonlinear activation functions. We compared MLP regressor with other regression models, such as AdaBoostRegressor, LinearRegression, and KNeighborsRegressor. We found that AdaBoostRegressor was sensitive to noisy data and produced unstable results. LinearRegression was unable to model complex patterns and had a high error rate. KNeighborsRegressor was sensitive to irrelevant or redundant features and required careful feature selection and scaling. Therefore, we concluded that MLP regressor was the best choice for our problem.

We developed a surrogate model for each fitness function by using a neural network technique. We trained the neural network with a dataset of input-output pairs obtained from the Carla simulator. The dataset consisted of 1000 observations, which we split into a training set (80%) and a test set (20%) for each fitness function.

Figure 2 shows the feature that can influence the collision outcome. In our case, the feature fragment consists of the parameters that affect the distance and time to collision, such as the speed, position, and direction of the car and the pedestrian. Our project uses the following features for the simulation: 1) Pedestrian position 2) Vehicle speed 3) Road type (fixed) 4) Weather type (fixed). These inputs affect the control and perception software components of the self-driving car.

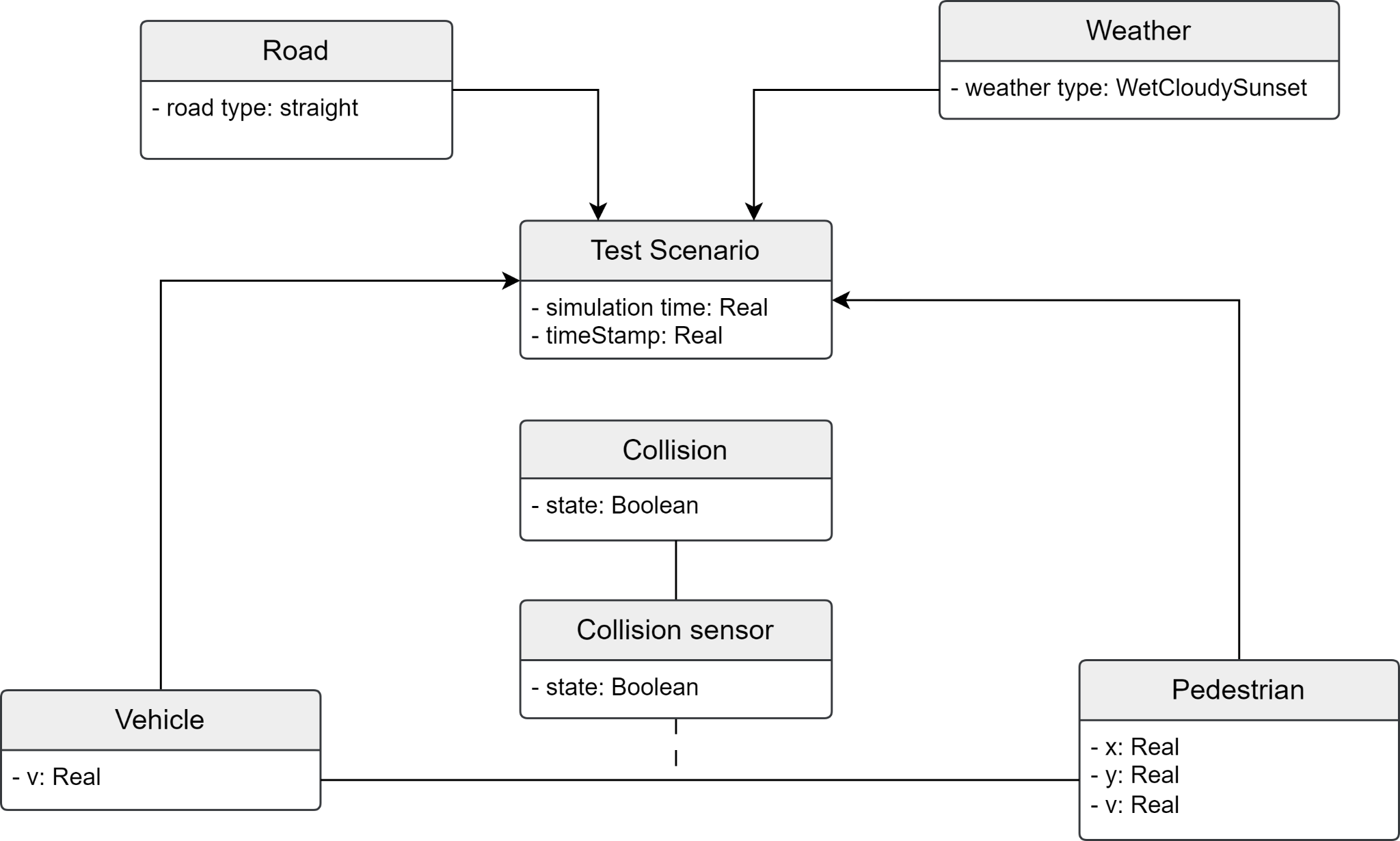
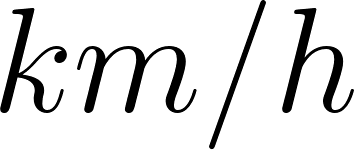
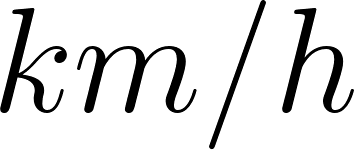
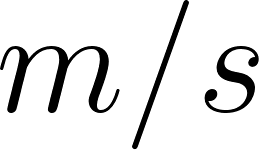
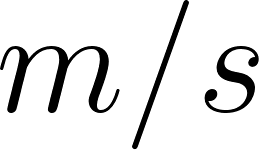


Figure 2: Features can be effective in collision .Between vehicle and pedestrian collision can be happen and this collision can be detected by collision sensor

We defined the test input for the Carla simulator as a vector of four variables:[](https://www.codecogs.com/eqnedit.php?latex=s_c#0), *d*, and [](https://www.codecogs.com/eqnedit.php?latex=s_p#0). These variables represent the speed of the car, the distance of the pedestrian, and the speed of the pedestrian, respectively. We also specified the value ranges for each variable as follows: [](https://www.codecogs.com/eqnedit.php?latex=%20R_s_c%20#0)= [15 [](https://www.codecogs.com/eqnedit.php?latex=km%2Fh#0), 80 [](https://www.codecogs.com/eqnedit.php?latex=km%2Fh#0)] , [](https://www.codecogs.com/eqnedit.php?latex=R_y#0) = [5 [](https://www.codecogs.com/eqnedit.php?latex=m#0), 25 [](https://www.codecogs.com/eqnedit.php?latex=m#0)], and [](https://www.codecogs.com/eqnedit.php?latex=R_s_p#0) = [0.1 [](https://www.codecogs.com/eqnedit.php?latex=m%2Fs#0), 0.5 [](https://www.codecogs.com/eqnedit.php?latex=m%2Fs#0)]. All the variables are of type float. We generated the input dataset for the simulator by dividing the value ranges of [](https://www.codecogs.com/eqnedit.php?latex=R_y#0), [](https://www.codecogs.com/eqnedit.php?latex=R_s_c%20#0), and [](https://www.codecogs.com/eqnedit.php?latex=R_s_p#0) into 100, 200, and 100 equal intervals, respectively. This resulted in a total of 2,000,000 possible combinations of the input variables. Figure 3 illustrates the process of how the project uses fitness functions.

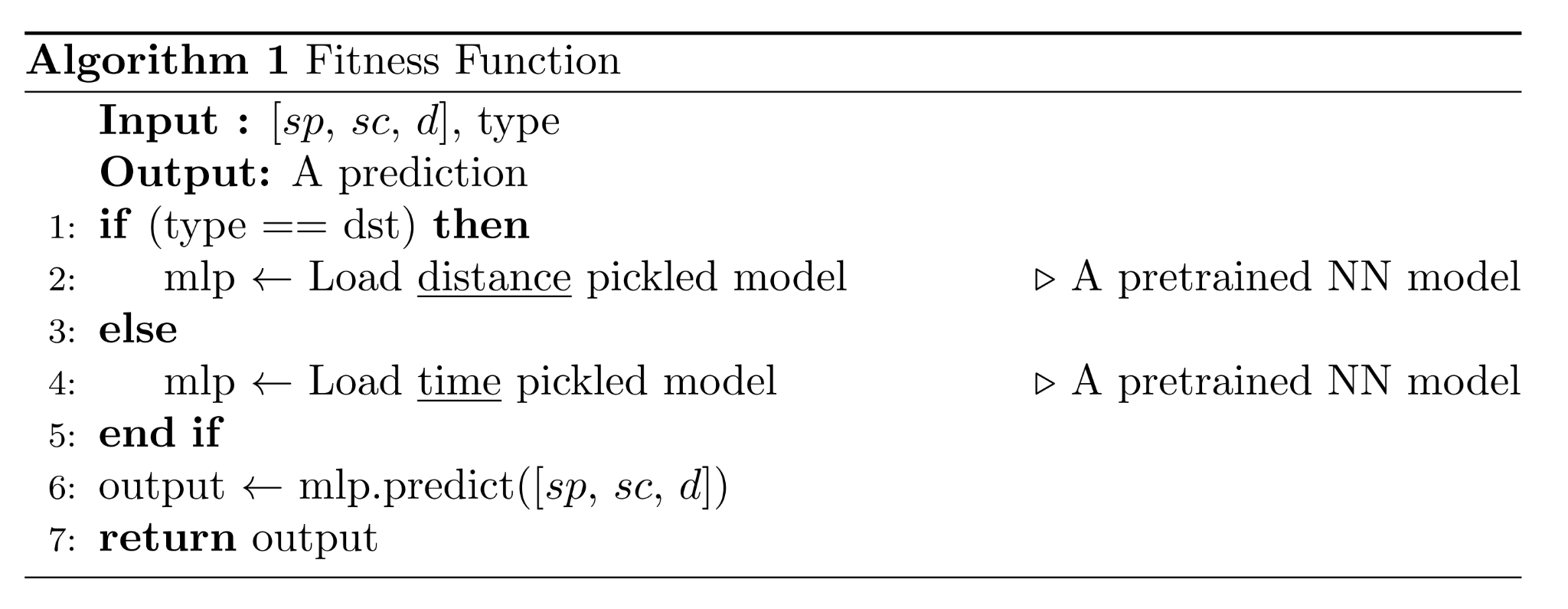


Figure 3: The process of calculating Fitness functions

3.3.Search with surrogate model

Our goal was to generate test cases for ADAS that would reveal the most critical behaviors of the system and its environment. To achieve this, we formulated the test case generation as a multi-objective search optimization problem. We defined two fitness functions that measured the severity and the likelihood of the crash scenarios.We use the concept of dominance to compare different solutions in terms of their fitness values. A solution *n* dominates another solution *m* if *n* has better fitness values than *m*. Figure 4 shows the steps of the hill climbing [7] algorithm that we used to generate test cases for ADAS. The algorithm starts with a random population *m* of potential solutions. Then, it compare the value of fitness functions for *m* and neighbors. Next, it iterates over the solutions in *m* and tries to find better solutions in their neighborhood. A solution is considered better than another if it has lower values for both fitness functions. The algorithm stops when no improvement can be found in the neighborhood of any solution in *m* also in Figure 4 fitness function calculated by a model that is generated by MLPRegressor method.

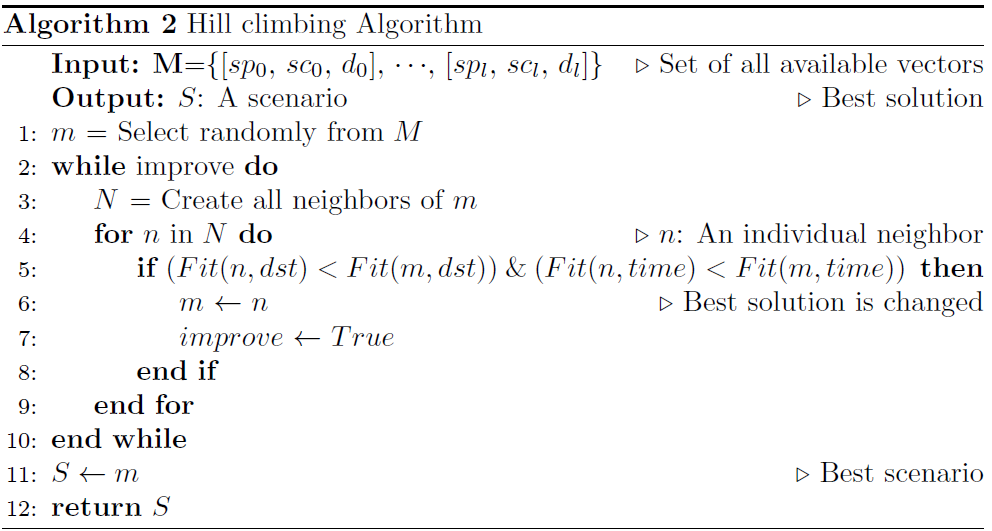


Figure 4: The Hill Climbing Algorithm for Test Case scenario Generation

4.Evaluation

This report describes the experiment that we conducted using a computer with 16GB of memory and an Intel Core i7 8th generation processor with 2.2GHz speed. The operating system was Windows 11. We used Python as the programming language and scikit-learn and CARLA as the main packages. This section presents the Research Questions (RQs) that guide our investigation.RQ1. (Comparing Random Search, Hill Climbing):How do Hill Climbing and random search perform compared to one another? In each iteration, we measure the distance between the car and the pedestrian. Our objective is to compare which method can find the optimal solution in the shortest time. The optimal solution is the one that minimizes the distance and the time to collision. We compare the performance of two search algorithms, Hill Climbing and random search, on the same optimization problem. We start both algorithms from the same initial point and run Hill Climbing until it reaches an optimum. We record the number of iterations that Hill Climbing takes to converge and use it as a fixed budget for random search. We then compare the quality of the solutions found by both algorithms within the same number of iterations.Random search samples solutions from a probability distribution and accepts any improvement over the current solution. However, for the purpose of plotting, we only show the solutions that are better than the previous best solution. This way, we can visualize the progress of the search algorithm.In each iteration, we measure the distance between the car and the pedestrian. Our objective is to compare which method can find the optimal solution in the shortest time. The optimal solution is the one that minimizes the distance and the time to collision.Figure 5 shows the comparison of hill climbing and random search algorithms on the distance fitness function. Hill climbing iteratively improves the current solution by making small changes. Random search is a stochastic algorithm that samples solutions randomly from the search space. The figure 5 and 6 show that hill climbing gradually reduces the distance value and converges to a better solution than random search. This is because hill climbing exploits the gradient of the fitness function and moves towards the optimal solution. We implemented two versions of a random search algorithm for comparison with hill climbing algorithms. The first version of random search selects a random floating-point number between the minimum and maximum values of each parameter.

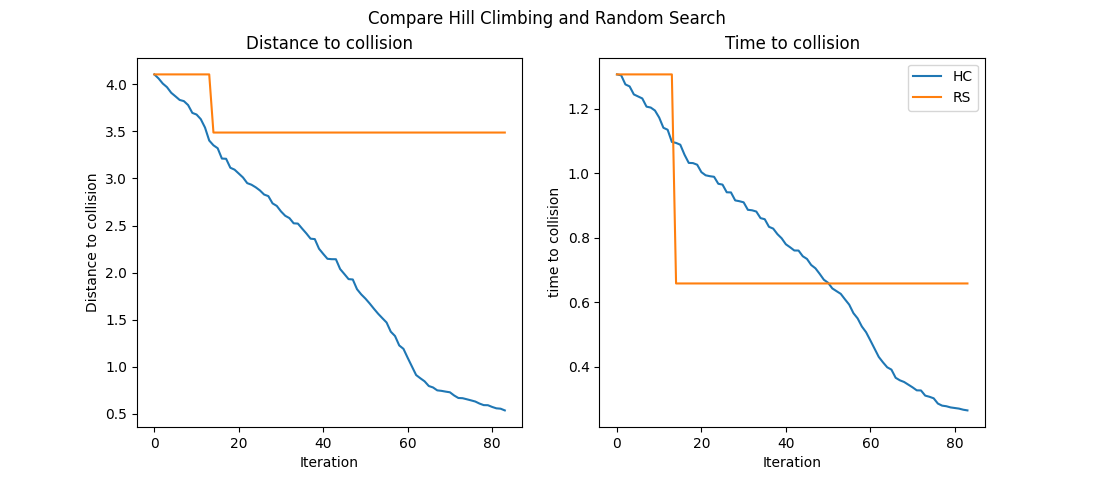


Figure 5: Comparison of hill climbing and random search that is select randomly a float number(version 1)

The second version of random search selects a random value from a predefined range of values for each parameter, as described in section 3.2. The first version cannot find a better solution than hill climbing because it explores the search space without any guidance and wastes time on inferior solutions. On the other hand the second version has a smaller search space than the first one, which makes it more likely to find a better solution that is close to hill climbing in some cases. Figure 6 shows the results of the comparison between the two versions of random search and hill climbing.

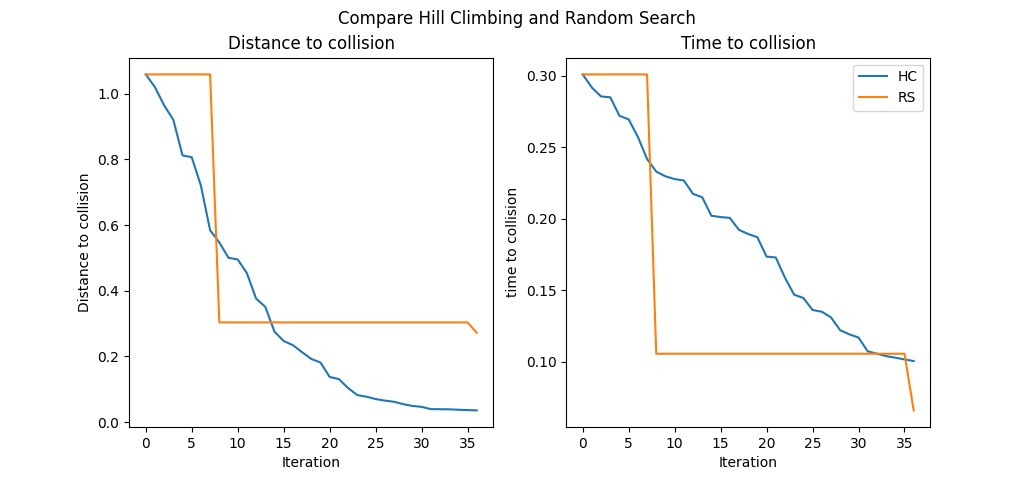


Figure 6: Comparison of hill climbing and random search that is select randomly from predefined search space (version 2)

5.Conclusion

We use physics-based simulation tools to test the control and perception software components of self-driving cars. These tools are feasible and practical, but they have two main limitations: (1) They do not provide enough guidance and automation for generating test cases that can detect faults, and (2) They take a long time to run each test case. To overcome these challenges, we propose an approach that combines multi-objective search and neural networks. Our approach uses meta-heuristics that capture the key features of the system and its environment. These meta-heuristics guide the search to explore behaviors that are more likely to expose faults.

References

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in Proceedings of the 1st Annual Conference on Robot Learning.

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

[3] R. Ben Abdessalem, A. Panichella, S. Nejati, L. C. Briand and T. Stifter, "Testing Autonomous Cars for Feature Interaction Failures using Many-Objective Search," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 2018, pp. 143-154, doi: 10.1145/3238147.3238192.

[4] R. Ben Abdessalem, S. Nejati, L. C. Briand and T. Stifter, "Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms," 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 2018, pp. 1016-1026, doi: 10.1145/3180155.3180160.

[5]A. Assadi, F. Meier and L. del Re, "Flexible Stochastic Microscopic Traffic Model for ADAS Testing," 2020 24th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2020

[6]J. E. Stellet, M. R. Zofka, J. Schumacher, T. Schamm, F. Niewels and J. M. Zöllner, "Testing of Advanced Driver Assistance Towards Automated Driving: A Survey and Taxonomy on Existing Approaches and Open Questions," 2015

[7]L. Hernando, A. Mendiburu and J. A. Lozano, "Hill-Climbing Algorithm: Let's Go for a Walk Before Finding the Optimum," 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 2018.