AmbieGen tool at the SBST 2022 Tool Competition

Dmytro Humeniuk
dmytro.humeniuk@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Giuliano Antoniol
giuliano.antoniol@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Foutse Khomh
foutse.khomh@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Abstract

AmbieGen is a tool for generating test cases for cyber-physical systems (CPS). In the context of SBST 2022 CPS tool competition, it has been adapted to generating virtual roads to test a car lane keeping assist system. AmbieGen leverages a two objective NSGA-II algorithm to produce the test cases. It has achieved the highest final score, accounting for the test case efficiency, effectiveness and diversity in both testing configurations.

CCS Concepts: • **Software and its engineering** → *Object oriented frameworks*; • **Theory of computation** → **Theory of randomized search heuristics**.

Keywords: test cases, virtual roads, competition, genetic algorithm

ACM Reference Format:
Dmytro Humeniuk, Giuliano Antoniol, and Foutse Khomh. 2022. AmbieGen tool at the SBST 2022 Tool Competition. In *The 15th Search-Based Software Testing Workshop (SBST'22), May 9, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3526072.3527531

1 Introduction

Self-driving cars have a perspective of becoming a part of our lives in the near future. These systems are safety-critical and should undergo a rigorous testing process. One of the stages is testing in a virtual environment. However, manually designing all the possible scenarios in the virtual environment is a tedious task.

In this paper we describe the principle of operation of the AmbieGen tool for automated generation of test cases, submitted to the SBST2022 tool competition. You can can find a more detailed information about AmbieGen as a test generation framework in our previous work [5]. To generate the test cases for the vehicle lane-keeping assist system (LKAS), AmbieGen leverages a two objective NSGA-II genetic algorithm. The goal of the genetic algorithm is to increase the fault revealing power of test cases, preserving their diversity. We further provide a more detailed description of the tool operation. The detailed results of the performance of the AmbieGen tool are outlined in the competition report [6].

SBST'22 , May 9, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9318-8/22/05…$15.00
https://doi.org/10.1145/3526072.3527531

2 AmbieGen tool description

The goal of the competition is to generate test cases for an autonomous vehicle, with a lane-keeping assist system, that should follow a road lane of the given trajectory, without going out of its bounds. The testing approach should generate a virtual road that forces the car to go out of the road. The road is represented as a sequences of points, defining the road spine. To generate the virtual roads the AmbienGen tool leverages a search based approach. Each run of the search algorithm produces a set of Pareto optimal solutions, which are then used as the test cases for the LKAS system. It starts by generating an initial population of the individuals. Following the crossover, mutation and selection operators, the fitter individuals are produced, which correspond to more challenging test cases to test the LKAS. Below we provide a more detailed description of our tool.

Table 1. Example of individual representation

	S_1	S_2	S_3	S_4	S_5
A_1, *road type*	0	1	1	1	0
A_2, *straight road length*	15	-	-	-	5
A_3, *turning angle*	0	60	60	75	0

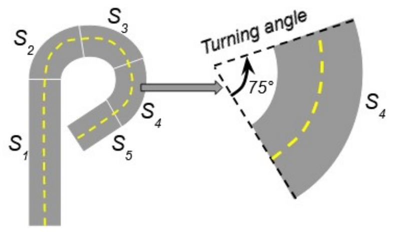

Figure 1. An example of the test case represented by an individual in Table 1 with a demonstration of a turning angle parameter

2.1 Individual representation

Each individual corresponds to one test case, i.e., one road topology. We suggest representing the road topology as a combination of road segments of different length and curvature. We encode each individual as a matrix of size nxm, where m corresponds to the number of road segments S_j and n to the number of attributes A_i describing their shape and type. In the context of LKAS testing, we define three attributes: A_1, A_2 and A_3. A_1 corresponds to the road type and it is sampled from the three possible values 0, 1, and 2 corresponding to the straight road, turning right and turning left road respectively. A_2 corresponds to the straight road length in meters and is sampled from the interval [5, 50]. A_3 corresponds to the road turning angle in degrees sampled from the interval [5, 85]. In Table 1 you can see an example of encoding a road topology shown in Fig. 1. This road consists of 5 road segments of straight and turning right types.

2.2 Initial population generation

The initial population is generated by assigning values to the cells of the test case matrix, such as in Table 1. The attribute values can be randomly sampled from the allowable value ranges.

2.3 Fitness function evaluation

To evaluate the test case we are using two fitness functions F_1 accounting for the test case fault revealing power and F_2 accounting for the test case diversity. More precisely, F_1 is evaluated as the maximum deviation from the lane center after executing the test case. For the test cases, that violate the established requirements, i.e., produce road topologies that go out of the map bounds, are too sharp or intersect, the fitness value is set to 0.

First fitness function. To calculate F_1 we first transform the test case matrix, such as shown in Table 1, to a set of points, defining the road topology. This procedure is described in a more detail in our previous work [3]. In brief, we start with building a base vector, located in the center of the map with a norm equal to the road width. Further, we apply affine transformations to this vector, such as parallel transitions, for straight road segments, and rotations for curved road segments. After the road topology is generated, we use a simplified car model to evaluate its maximal deviation from the road lane center given this road topology. We built the simplified car model based on the bicycle car model and a Stanley controller path tracking [7]. To execute this model, the simulator environment is not needed and the model behaviour, i.e., the maximum deviation from the road lane is estimated based on the equations describing the car movement. During the evaluation of the test case TC, the deviation from the lane $D = d_1, d_2, ..., d_n$ is recorded at regular time intervals. F_1 is evaluated as follows:

$$F_1 = max(D, TC),$$

where D is a list of car model distance from the lane center during the test case TC execution.

Second fitness function. From our experience, it's important to add the diversity preservation during the test generation, otherwise a high number of similar test cases is produced. To this end, we added a second objective function accounting for diversity. We compute it as the Jaccard distance between a given test case TC and a reference test case TC_{ref}. Given that the test cases represent a set of road segments with certain properties, Jaccard distance can be evaluated as [2]:

$$F_2 = \frac{S}{T},$$

where T corresponds to the total number of road segments in TC and TC_{ref} and S - to the number of similar or same segments in TC and TC_{ref}. As a reference test case we can use the fittest individual in terms of the F_1 fitness function, for instance.

2.4 Crossover operator

We are using a one point crossover operator, which is one of the commonly used operators for variable-length solution representation. This operator creates two new test cases by exchanging information between two existing test cases TC_1 (parent 1) and TC_2 (parent 2), with corresponding lengths of m_1 and m_2. An illustration of the crossover operation between two individuals is shown in Fig. 2. Individual TC_1 length is 4 elements and individual TC_2 is 3. Both individuals have three attributes A_1 (road type), A_2 (straight road length) and A_3 (turning angle). The crossover point is chosen to be 2 and is shown as a dotted line.

2.5 Mutation operator

We define two mutation operators:

- *exchange operator*: two randomly selected segments of the individual are exchanged the positions. You can see an example in Fig. 3.
- *change of variable operator*: a road segment S_j in the individual is randomly selected, then for one of the attributes A_i the value is changed according to its type and maximum as well as minimum values. An example is shown in Fig. 4.

2.6 Evaluation setup

We used the following genetic algorithm configuration: population size: 100, number of generations: 75, mutation rate: 0.4, crossover rate: 1, algorithm type: generational.

3 Evaluation results

In this section we report the evaluation results provided by SBST2022 tool competition team [6]. The tool was evaluated on the two subject systems in two different experimental setups: BeamNG.AI and Dave-2. Each setup was given 1 hour

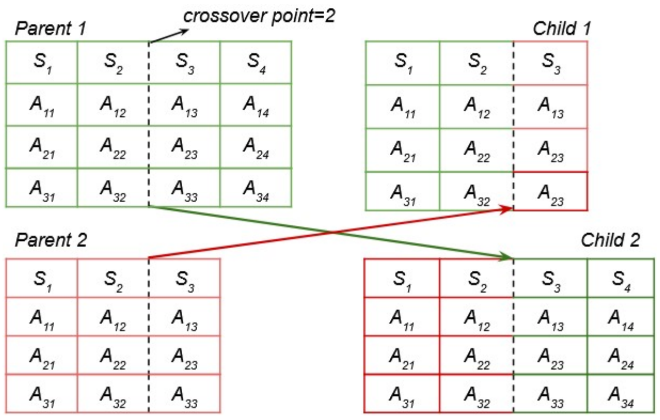

Figure 2. An example of the crossover operator functioning, crossover point is selected equal to 2

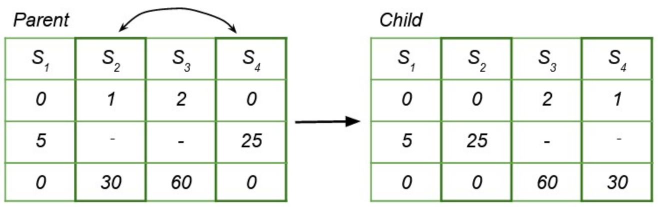

Figure 3. An example of the exchange mutation operator, where segments 2 and 4 are exchanged

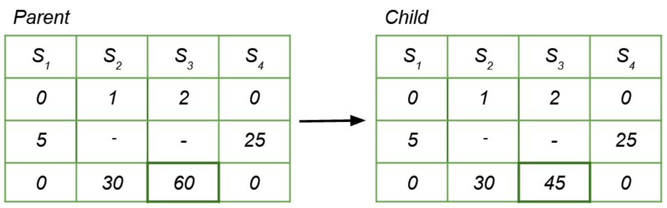

Figure 4. An example of the change of variable mutation operator, where the value of the third attribute of the third segment is changed

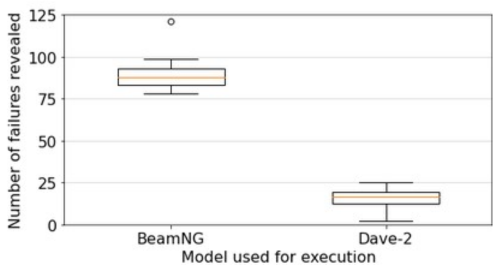

Figure 5. Number of failures revealed

test generation budget and 2 hour test execution budget. BeamNG.AI configuration features the BeamNG.AI agent

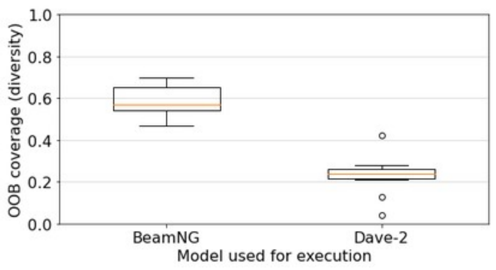

Figure 6. The diversity of the revealed failures

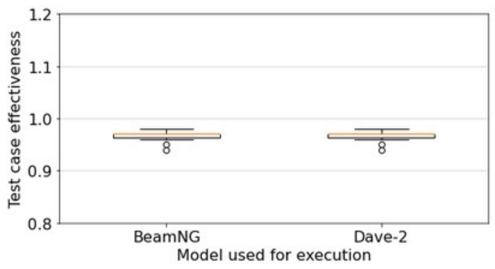

Figure 7. Effectiveness of the generated test cases

driving up to 70Km/h, an OOB (out of bound episodes) tolerance value of 0.85, meaning that the the failure is detected when more than 0.85 of the car area goes out of the road bounds. BeamNG.AI 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.

Dave-2, instead, is an end-to-end approach that uses a deep learning (DL) architecture consisting of three convolutional layers, followed by five fully-connected layers [1] to predict steering angles from camera images. Dave-2 was trained at low speed, therefore this configuration's maximum speed is 35Km/h. Consequently, a lower tolerance value (0.1) was used to make the OBB monitor more sensible.

In Fig. 5 - Fig, 7 you can see results for evaluating such metrics as the number of failures revealed, OOB coverage (test case diversity) and test case effectiveness after running the tool for 10 times.

Number of failures corresponds to the number of test cases, where the car went out of the lane. From Fig. 5 we can see that in BeamNg setup our tool reveals 90.4 OOBs on average, while in Dave-2 - 15.3.

OOB coverage captures the effectiveness of the generated tests to expose as many different failures as possible. First, structural and behavioral features of the tests portion relevant to the OOBs is extracted. Then a feature map is populated with the extracted values. Finally, the OOB coverage is defined by counting the cells in the map covered by the exposed OOB. Larger values of OOB coverage identify better test generators. From Fig. 6 we can see that our tool produces test cases with average diversity of 0.585 for BeamNG setup and 0.231 for Dave-2.

Test case effectiveness corresponds to the ratio of valid tests over all the generated tests. Effective test generators produce valid tests, i.e., they do not waste the generation budget in generating invalid tests. From Fig. 7 we can see that our tool has a similar effectiveness of 0.966 in both BeamNG and Dave-2 setups.

4 Discussion

We can see from the results that our tool is able to reveal the failures of both models used in the setups. For the BeamNg setup on average 6 times more failures are revealed, than for the Dave-2. We attribute it to the fact that the Dave-2 agent maximal driving speed is twice smaller than that of the BeamNg, reducing the chance to go out of the bounds.

In both setups the tool allowed to explore diverse failures. We surmise that promoting explicitly the test case diversity plays an important part. Moreover, we are using a simplified system model, allowing to faster evaluate the solutions and run more iterations of the search algorithm. While using the tool, the algorithm can be run for a number of times. Each run starts from producing a new initial population of candidate solutions. This allows our tool to continuously explore the search space and escape the local minima. For BeamNG setup, the diversity score was around 2.5 higher than for the Dave-2. We assume that for the BeamNg model a broader space of possible failures was explored as more failures were detected.

Finally, our tool achieves a high effectiveness, i.e., valid to total test case ratio of 0.96. We have internal verification mechanisms that assign a low fitness function to the test cases not satisfying the established requirements, preventing the production of invalid test cases.

5 Conclusions

In this paper we present our tool, AmbieGen, applied to the generation of test cases for a vehicle LKAS system. It leverages evolutionary search guided by two objectives accounting for the test case fault revealing power and diversity. To evaluate the first fitness function we use the simplified model of the system, which does not require running the system model in the driving simulator.

Using a simplified system model is effective in revealing faults of the real system model. In two hour execution budget our tool could reveal on average 90 failures for the BeamNG setup and 15 failures for the Dave-2 setup.

Adding a fitness function to promote diversity improves the diversity of the test cases produced by the search algorithm. AmbieGen achieved the average diversity of 0.585 for BeamNG setup and 0.231 for Dave-2.

In our future work we plan to extend the test generation to produce more complex scenarios, that include static and moving obstacles, various weather conditions.

6 Using the tool

The tool is open source and available at [4], where the instructions on how to run it are provided. Currently it is integrated in the SBST 2022 CPS competition pipeline.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[2] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 318–328.

[3] Dmytro Humeniuk, Giuliano Antoniol, and Foutse Khomh. 2021. SWAT tool at the SBST 2021 Tool Competition. In *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. 42–43. https://doi.org/10.1109/SBST52555.2021.00019

[4] Dmytro Humeniuk, Giuliano Antoniol, and Foutse Khomh. 2022. AmbieGen tool. https://github.com/dgumenyuk/tool-competition-av.git.

[5] Dmytro Humeniuk, Foutse Khomh, and Giuliano Antoniol. 2022. A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems. https://arxiv.org/a/0000-0002-2983-8312.html.

[6] Vicenzo Riccio and Alessio Gambi. 2022. SBST Tool Competition 2022. In *International Conference on Software Engineering, Workshops, Pittsburgh, PA, USA, 2022*. ACM.

[7] Jarrod M Snider et al. 2009. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).