2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

AsFault: Testing Self-Driving Car Software Using Search-based Procedural Content Generation

Alessio Gambi
University of Passau, Germany
alessio.gambi@uni-passau.de

Marc Müller
BeamNG GmbH, Germany
mmueller@beamng.gmbh

Gordon Fraser
University of Passau, Germany
gordon.fraser@uni-passau.de

***Abstract*—Ensuring the safety of self-driving cars is important, but neither industry nor authorities have settled on a standard way to test them. Deploying self-driving cars for testing in regular traffic is a common, but costly and risky method, which has already caused fatalities. As a safer alternative, virtual tests, in which self-driving car software is tested in computer simulations, have been proposed. One cannot hope to sufficiently cover the huge number of possible driving situations self-driving cars must be tested for by manually creating such tests. Therefore, we developed AsFault, a tool for automatically generating virtual tests for systematically testing self-driving car software. We demonstrate AsFault by testing the lane keeping feature of an artificial intelligence-based self-driving car software, for which AsFault generates scenarios that cause it to drive off the road. A video illustrating AsFault in action is available at: https://youtu.be/lJ1sa42VLDw**

***Index Terms*—Automatic test generation, search-based testing, procedural content generation, self-driving cars**

I. Introduction

Experts forecast that autonomous driving will grow in relevance, increase road safety [1], and profoundly impact society [2]. As of today, companies such as Waymo, Tesla, and Uber, have demonstrated prototypes of self-driving cars which are deployed in everyday urban traffic. However, these prototypes have already caused several car crashes [3] and even fatal accidents [4], [5]. Rigorous testing might have exposed problems before deployment, thus giving a chance to developers to fix them and possibly avoid fatalities.

Naturalistic Field Operational Tests (N-FOT), which put self-driving cars on trial in the real world or tailored driving challenges, are the *de facto* approach to validate self-driving cars [6]. However, N-FOT are expensive, dangerous, and of limited effectiveness in proving that self-driving cars are safer than human drivers [7]. As a consequence, *virtual tests*, in which self-driving cars are fed with synthetic sensory data which simulate driving scenarios [8], are proposed as a more efficient and less risky alternative to N-FOT.

Virtual tests vary in scope and focus: They range from simulating individual sensors to test specific functions of the cars [9], to generating images to test vision components [10], [11]. As a novelty compared to existing work, our AsFault tool tests self-driving car software by automatically generating entire virtual realities for more holistic testing.

Testing self-driving cars using simulations brings two main technical challenges: (A) generating simulation content, which must be valid while accurately representing possibly very complex real-world phenomena; and, (B) creating relevant driving scenarios, which must be selected among the sheer number of possible simulations that can be created. In the current practice, the generation of both simulation content and driving scenarios requires a great deal of manual effort, which is inefficient and might introduce a cognitive bias [12]. AsFault, instead, implements a novel approach to Search-based Procedural Content Generation [13] and applies it for automatically generating tests.

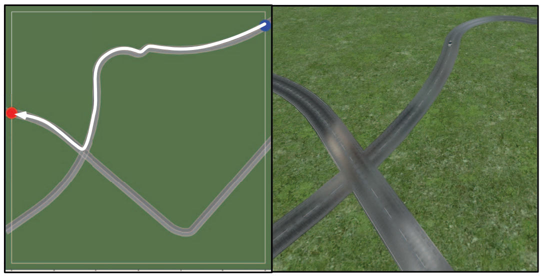
Fig. 1: Example of a test case generated by AsFault.

Test cases are defined as navigation paths inside a road network. The figure shows a sample road network and navigation path as visualized by AsFault (Left) and the corresponding road network in the driving simulator (Right).

AsFault focuses on generating virtual road networks which expose unsafe behaviors, like driving the wrong-way or going off the road, and it does so by combining (i) a genetic algorithm which evolves the virtual road networks to maximize a given fitness function; and, (ii) procedural content generation which implements the virtual road networks in accurate simulations. Figure 1 shows a sample road network generated by AsFault while testing a driving artificial intelligence (D-AI).

In this demonstration, we utilize AsFault to test the fundamental feature which defines self-driving cars, i.e., *lane keeping*, and show that AsFault can generate roads which cause the D-AI to drive out of its lane. We configure AsFault to generate roads with one lane per direction, which is the simplest road configuration for testing lane keeping with respect to unsafe behaviors such as driving off the road and invading the opposite traffic lane, and maximize the observed distance between the lane center and the simulated car.

By automatically generating virtual tests which trigger unexpected behaviors, AsFault not only helps developers in exposing safety critical bugs, but also supports them in identifying those environmental conditions (e.g., road shapes) which cause self-driving car software to fail.

II. Testing Self-Driving Cars by Procedurally Generating Roads

AsFault generates virtual tests by procedurally generating road networks within a fixed-size map of configurable size to fit with the capabilities of current simulation software. AsFault represents roads as set of *polylines*, i.e., discrete sequences of points, and generate valid road networks in a bottom-up fashion (see Figure 2): AsFault starts by generating an initial road segment at the boundary of the map (a); then, it attaches additional road segments to it and forms a gapless road (b–c). If the added segments violate road validity constraints, i.e., the roads partially overlap or self-intersect, AsFault backtracks and replaces the offending road segments with freshly generated ones until a valid road is generated. Finally, AsFault combines roads to form a road network (d); if roads partially overlap, rather than creating proper intersections, AsFault replaces them until it obtains a valid configuration.

AsFault defines test cases on the road networks as driving tasks: the self-driving car software under test, i.e., the *ego-car*, must drive from a starting point to a goal location following a given navigation path chosen by AsFault from the map. A test passes only if the ego-car reaches the goal location within a given timeout and without violating any constraints, i.e., it follows the given navigation path without driving off the lane. Since the number of possible navigation paths increases exponentially with the complexity of the road networks, AsFault selects navigation paths according to a heuristic: AsFault randomly samples a number of *simple paths*[1] from the map and then selects the longest path among them. With this heuristic, AsFault aims to maximize the proportion of the network traversed during test execution, while keeping the path selection fast despite its complexity [14]. Testers can control the potential bias towards generating tests which the take longer to complete by specifying the size of the fixed-size map over which AsFault generated road networks.

After selecting the navigation paths, AsFault generates the simulation code which implements the virtual roads and automatically sets the simulator up such that the ego-car is placed in the starting position. Next, it starts the simulator in a separate process, and binds to its interface to monitor the test execution and collect live execution data. At this point, AsFault triggers also the ego-car which receives the live simulation data, e.g., video frames, computes the driving actions, and sends them back to the simulator effectively driving the virtual car. AsFault monitors the position of the virtual car every 250 ms and stops the simulation as soon as the virtual car reaches the goal location or the timeout triggers. Further, AsFault elaborates the positional data to compute how many times the virtual car broke out of lane bounds, i.e., the amount of out of bound episodes (OBE).

AsFault targets lane keeping, hence we characterize effective tests as those which cause the self-driving car software to break out of the lane bounds. Consequently, AsFault's goal is to generate test suites which cause large amounts of OBEs. To generate effective tests, AsFault follows a standard evolutionary approach which aims to maximize the *maximum* distance between the car position and the center of the lane. This captures the intuition that tests which caused the ego-car to drive away from the lane center might contain road segments which stress the lane keeping ability of the ego-car more. Eventually, by selecting and recombining those segments in new configurations, AsFault leads the ego-car to drive out of its lane. We opted for using the maximum distance as fitness function instead of the average distance under the assumption that OBEs are exceptional cases; that is, we expect that ego-car would mostly keep the lane. Under this assumption, using the average distance, instead of the maximum distance, would smooth the observations and effectively filter out short OBEs.

AsFault creates an initial population of random road networks and navigation paths by following the procedure we described above; next, it continuously evolves the population by recombining and mutating the road networks using a number of search operators triggered according to configurable probabilities until it exhausts the specified generation budget.

The search operators defined by AsFault are novel and reflect the hierarchical structure of the road networks: AsFault *mutates* tests only at the level of roads by picking a random road segment and replacing it with a new one. Crossover, instead, targets both roads and road networks: the *join* operator splits roads at a random point and recombines them (see Figure 3), while the *merge* operator targets road networks and recombines random subsets of their roads (see Figure 4). The search operators can produce invalid configurations, such as roads which self-intersect and roads which partially overlap. When this happens, AsFault discards the invalid configurations and retries the same operation which generated them; since the search operators are random this might produce a different, hopefully, valid configuration. To avoid getting stuck while generating valid offspring, AsFault increases a probability of giving up after each failure; hence, either a valid configuration is found quickly, or the operation is aborted and no offspring are created. It follows that the amount of individuals in new generations might be smaller than the target population size; in this case, new generations are padded with the most fit individuals from the previous generation.

At every step of evolution, AsFault persists all the generated test cases on disk. Test cases are stored as JSON files, which encode the structure of the road network and the definition of the navigation path, hence, AsFault can re-execute them at developers' will. At the end of the process, AsFault outputs the final test suite and visualizes the results, i.e., the OBE count and the coverage of road segments, which the test suite achieved.

III. Using AsFault

This section describes where to obtain, install, and use AsFault to generate tests. Additional details on how to use

[1]Simple paths are navigation paths where each road segment is traversed at most once.

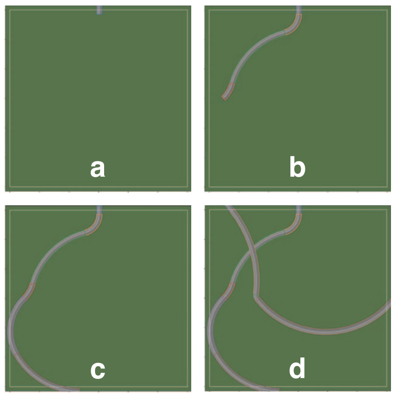

Fig. 2: Road network generation

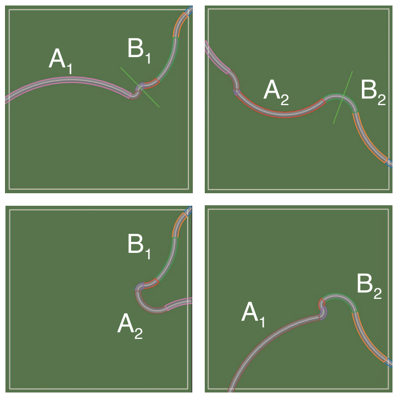

Fig. 3: Join Crossover

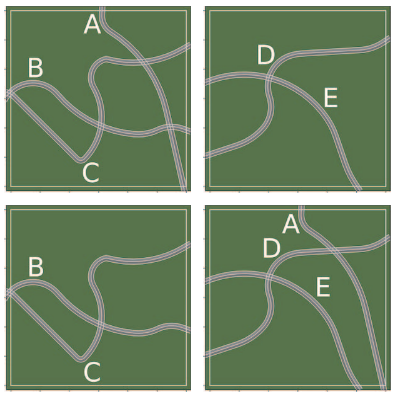

Fig. 4: Merge Crossover

The figures illustrate AsFault's "bottom up" road network generation (Figure 2), and its two novel crossover operators: *join*, which recombines road segments to form new roads (Figure 3), and *merge*, which recombines roads to form new road networks (Figure 4).

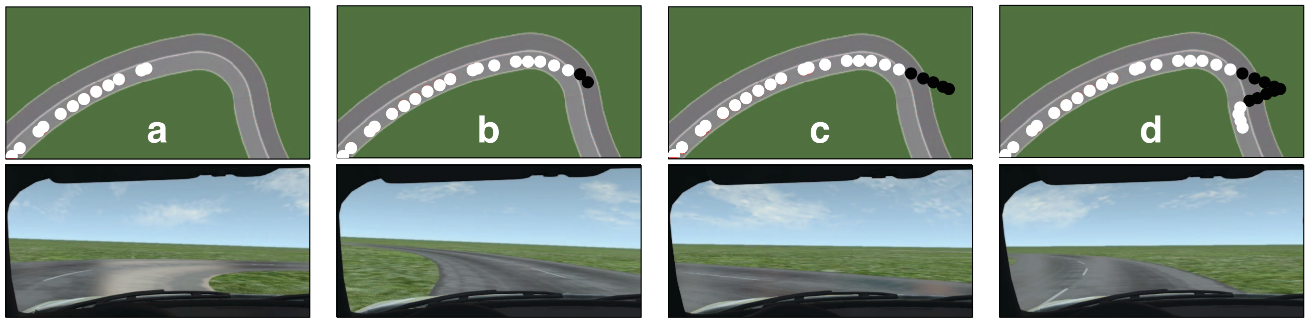

Fig. 5: Time lapse of an OBE

The figure illustrate a time lapse of an OBE caused by AsFault: the ego-car approached a sharp right turn (a), but understeered, which caused the ego-car to invade the opposite lane (b) and drive out of the road (c). Eventually, the ego-car turned around and regained the original lane (d).

AsFault for re-executing tests and visualizing test data can be found in its documentation.

A. Requirements and Installation

AsFault is open-source and freely available at:

https://github.com/Signaltonsalat/AsFault

AsFault requires the availability of external libraries. AsFault is implemented in Python 3; hence, it requires a corresponding interpreter to be installed. After obtaining its source code, AsFault can be installed as a standard Python package using the `pip` utility.

AsFault uses BeamNG.research [15], a freely-available driving simulator for executing the tests. We opted for BeamNG.research instead of other available driving simulators (e.g., [16], [17]) because it is physically accurate and exposes an intuitive API for programmatically generating roads. Consequently, a copy of BeamNG.research must be installed and configured as described in the official documentation in order to run AsFault. Additionally, since BeamNG.research supports only Windows, AsFault cannot be executed on other platforms (e.g., Linux).

B. Configuration and Execution

Once installed, AsFault can be called using the command:

```
asfault evolve {bng|ext <setup>} [options]
```

which starts the test generation process using the genetic algorithm (`evolve`) and either uses the BeamNG D-AI (`bng`) or an external D-AI configured via a setup script (`ext <setup>`) as test subject. Several options can be passed to the `evolve` command including a predefined seed for the search (`--seed`) and the search budget (`--budget`). Additionally, testers can control the size of the map, the size of the target test suite, and the probabilities of triggering the various search operators by means of various configuration files located in the AsFault's home folder (i.e., `~/.asfaultenv/`).

IV. Evaluation

Driving within a lane is the fundamental task in driving: if a car cannot keep its lane, it becomes a safety hazard for its passengers, passers-by, and other traffic participants. We demonstrate that AsFault can generate effective test cases by testing the lane keeping feature of *BeamNG.AI*, the driving AI which is shipped with BeamNG.research. BeamNG.AI has perfect knowledge of the surrounding environment, i.e., the road network, and follows an ideal trajectory along a

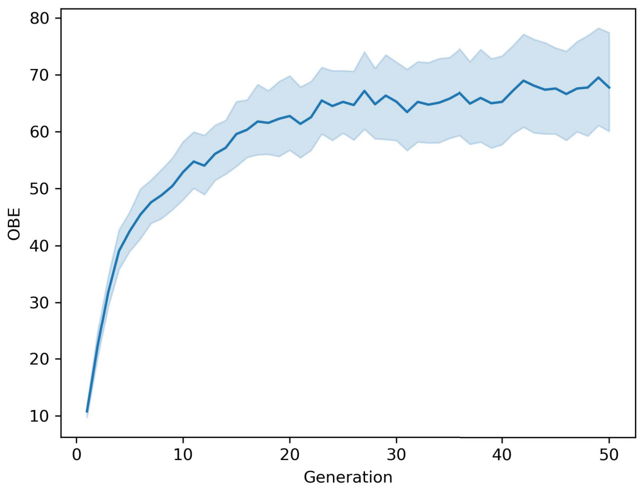

Fig. 6: Evolution of test case efficiency

The plot shows how the OBE count changes as the test generation process develops. The plot reports the average amount of OBE across 30 repetitions of the tests generation for each generation as thick line and the confidence interval around that.

provided navigation path. BeamNG.AI computes the actual driving actions (i.e., accelerate, brake, and turn) by solving a complex optimization problem which is controlled by the so-called "aggression" factor. In this demo, we set BeamNG.AI with a low aggression value which results in a *careful* driver which drives smoothly and close to the centre of the lane.

We configured AsFault to evolve a population of 25 tests defined over a map of 400 hectare, and repeated the test generation 30 times to collect statistically robust observations. For each test suite we consider the cumulative number of observed OBE, and aggregate the observation by averaging the OBE count across all the test generations.

Figure 5 shows a time lapse of an OBE that AsFault caused during one of the experiments. The top part of the figure shows AsFault's visualization; dots correspond to observations of the virtual car position which is tracked in real time by AsFault. The bottom part shows the corresponding frames as rendered by BeamNG from the driver's perspective. From this figure we can see that the ego-car (a) approached a sharp right turn; (b) understeered, hence invading the opposite lane; (c) drove out of the road; and, eventually, (d) drove back to the lane.

Figure 6, instead, shows the average count of OBEs (*OBE*) that AsFault achieved during the search process (*Generation*) as a thick line surrounded by a confidence band (to show variation from repetition of the experiment). As can be observed from this plot, the number of OBEs caused by the tests consistently increased over the search and plateaued towards the end at about 65 OBEs. This shows that tests generated by AsFault are effective.

V. Conclusion and Future Work

In this paper, we showed how our tool AsFault can be used to test self-driving car software, specifically their lane keeping features, using a novel search-based procedural content generation for simulation-based testing. As our evaluation shows, AsFault can generate effective tests which cause self-driving car software to drive out of the lane.

We plan to extend AsFault in several ways in the near future, for example by incorporating obstacles, traffic, and weather to improve the realism of the generated roads. Additionally, we plan to devise additional search strategies which, on one side, consider passenger comfort in addition to safety during the search (i.e., multi-objective search), and, on the other side, reward diversity of the generated test suites (i.e., novelty search).

References

[1] M. Bertoncello and D. Wee, "Ten ways autonomous driving could redefine the automotive world | McKinsey & Company."

[2] P. Bansal and K. M. Kockelman, "Forecasting Americans' long-term adoption of connected and autonomous vehicle technologies," *Transportation Research Part A: Policy and Practice*, vol. 95, pp. 49–63, Jan. 2017.

[3] A. Davies, "Google's Self-Driving Car Caused Its First Crash," *Wired*, Feb. 2016.

[4] CNBC, "Uber fatal crash: Self-driving SUV saw pedestrian, didn't brake: feds," May 2018.

[5] S. Levin, "Tesla fatal crash: 'autopilot' mode sped up car before driver killed, report finds," *The Guardian*, June 2018.

[6] M. Bertozzi, A. Broggi, A. Coati, and R. I. Fedriga, "A 13,000 km Intercontinental Trip with Driverless Vehicles: The VIAC Experiment," *IEEE Intelligent Transportation Systems Magazine*, vol. 5, no. 1, pp. 28–41, 2013.

[7] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, Dec. 2016.

[8] S. Masuda, "Software Testing Design Techniques Used in Automated Vehicle Simulations," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 300–303, Mar. 2017.

[9] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the International Conference on Software Engineering*, ICSE '18, pp. 1016–1026, 2018.

[10] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the International Conference on Software Engineering*, ICSE '18, pp. 303–314, 2018.

[11] 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 *Proceedings of the International Conference on Automated Software Engineering*, ASE '18, pp. 132–142, 2018.

[12] I. Salman, "Cognitive biases in software quality and testing," in *Proceedings of the International Conference on Software Engineering*, ICSE '16, pp. 823–826, 2016.

[13] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural Content Generation: Goals, Challenges and Actionable Steps," in *Artificial and Computational Intelligence in Games*, vol. 6, pp. 61–75, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[14] R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms, Third Edition*. Addison-Wesley Professional, third ed., 2001.

[15] BeamNG GmbH, "BeamNG.research – BeamNG."

[16] 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*, pp. 1–16, 2017.

[17] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics* (M. Hutter and R. Siegwart, eds.), (Cham), pp. 621–635, Springer International Publishing, 2018.