A^{2}CoST: An ASP-based Avoidable Collision Scenario Testbench for Autonomous Vehicles

Ruolin Wang, **Yuejiao Xu**, **Jie Peng**, **Jianmin Ji**[*]
University of Science and Technology of China
{rl_wang, yuejiao, pengjieb}@mail.ustc.edu.cn, jianmin@ustc.edu.cn

Abstract

This paper addresses the challenge of generating safety-critical scenarios with multiple adversarial vehicles for testing autonomous vehicles. Such scenarios must be plausible and collision-avoidable while resulting in a collision with the vehicle-under-test. However, the tremendous number of scenarios and the low ratio of plausible scenarios makes previous methods squander primary resources on implausible scenarios, degenerating their efficiency. We propose a two-stage framework called the ASP-based Avoidable Collision Scenario Testbench (A^{2}CoST) to overcome this obstacle and improve efficiency. In the former stage, we apply Answer Set Programming (ASP) for generating plausible logical scenarios. In the latter stage, we use a search algorithm to refine logical scenarios into safety-critical concrete scenarios. We also compute collision-free trajectories in these concrete scenarios while the vehicle-under-test fails to avoid the collision. We empirically show the A^{2}CoST significantly decreases the time consumption for simple scenarios while still effectively generating complex critical scenarios. The comparison with real-world traffic data further demonstrates the value of A^{2}CoST in generating plausible scenarios. The source codes of our method and the baselines are opened at https://github.com/Autonomous-Driving-Safety-Project/AACoST.

1 Introduction

Autonomous Vehicle Safety Assessment (AVSA), as one of the critical autonomous driving technology, is a validation system to ensure the safety of autonomous vehicles (AVs). Validating the safety of these vehicles requires more than 275 million failure-free miles of testing to demonstrate that their failure rate is lower than that of human drivers with a 95% confidence level (Kalra and Paddock 2016). However, most of the existing safety validation methods used in industry and literature are insufficient as they only test AV systems in a finite number of fixed scenarios (Hauer, Pretschner, and Holzmüller 2020).

To shorten the testing process and identify defects before deploying the AVs on the road, researchers have explored generating safety-critical scenarios in simulators. For example, Beglerovic, Stolz, and Horn (2017); Gangopadhyay et al. (2019); Koschi et al. (2019); and Calò et al. (2020) use search-based algorithms to generate collision scenarios, and Koren et al. (2018); Chen et al. (2021); and Kuutti, Fallah, and Bowden (2020) employee Reinforcement Learning (RL) based methods to perform adversarial attacks in complex scenarios. However, search-based methods cannot handle complex scenarios due to the enormous search space involved, and RL-based algorithms often generate implausible scenarios which introduce inefficiency in preventing adversarial vehicles from colliding with each other. Note that, finding collision scenarios is more like a search problem rather than control, while RL algorithms intend to obtain a control policy.

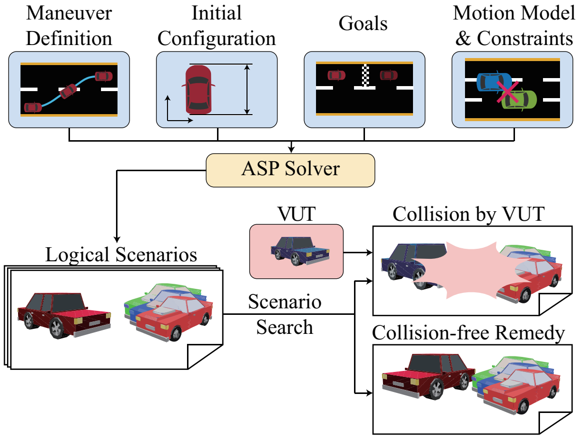
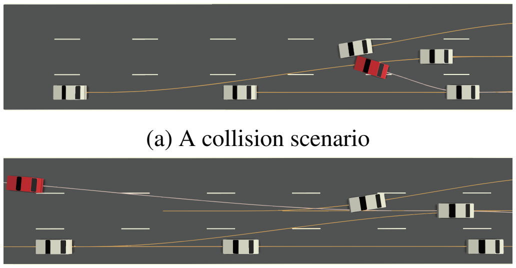


Figure 1: The framework of ASP-based Avoidable Collision Scenario Testbench (A^{2}CoST).

From the above observations, the critical point is the tremendous number of scenarios and the low ratio of plausible scenarios. To address this problem, we propose a two-stage framework, the ASP-based Avoidable Collision Scenario Testbench (A^{2}CoST), as shown in Fig. 1. As defined by Menzel, Bagschik, and Maurer (2018), we call a parameterized scenario with the range of parameters as *logical scenario*, and those scenarios with fixed parameters as *concrete scenario*. We firstly apply Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011) to generate logical scenarios that contain possible traffic environments around the vehicle-under-test (VUT) and the maneuvers of multiple adversarial vehicles (vehicles other than the VUT). ASP is a non-monotonic logic programming language that

[*]The corresponding author.



(a) A collision scenario

(b) Collision-free remedy of (a)

Figure 2: A critical scenario generated by A^{2}CoST with 5 adversarial vehicles. (a) shows that the VUT (driven by IDM (Treiber, Hennecke, and Helbing 2000) and MOBIL (Kesting, Treiber, and Helbing 2007)) causes a collision in the scenario. (b) shows that the scenario is collision-avoidable while the VUT follows the given trajectory. The thin lines indicate the vehicles' trajectories.

can encode various driving rules, motion models, and constraints of vehicles effectively. Then, we apply a search algorithm to refine them into several concrete scenarios. These concrete scenarios are required to be: (*i*) plausible, i.e., adversarial vehicles behave normally following real-world behavioral regularities; (*ii*) critical, i.e., the VUT collides with another vehicle; and (*iii*) collision-avoidable, i.e., there are deliverable trajectories for the VUT to avoid collisions.

To showcase the effectiveness of A^{2}CoST in producing plausible scenarios, this study examines the acceleration and lateral velocity distributions of vehicles in the generated scenarios. The comparison is made against the NGSIM dataset (U.S. Department of Transportation Federal Highway Administration 2016), which is comprised of vehicle trajectories recorded on the U.S. I-80 freeway. To measure the dissimilarity between the dataset and generated scenarios, the Wasserstein distance (Vallender 1974) is utilized. The findings indicate that the scenarios generated by A^{2}CoST are more closely aligned with real-world scenarios compared to the baselines.

As illustrated in Fig. 2, we also investigate a lightweight simulator, **L**ightweight **A**utomobile **S**imulator based on Open**S**CENARIO (LASS), to evaluate and visualize various scenarios generated by A^{2}CoST. Notice that both the logical scenarios from the first stage of A^{2}CoST and the concrete scenarios from the second stage can be converted to corresponding OpenSCENARIO[1] files. Then LASS reads such an OpenSCENARIO file to generate an OpenAI Gym-like simulation environment, which can be used not only to evaluate the scenario but also to implement an RL-based method for collision scenario generation. LASS uses a simplified physics model, sacrificing some simulation accuracy for faster processing speed, which is suitable for early validation of autonomous driving planning algorithms. For end-to-end autonomous driving systems, we have also developed an alternative high-precision distributed simulation framework based on CARLA. The LASS simulator and the distributed simulation framework will be released after acceptance.

In summary, the main contributions of this paper are:

1. We propose an effective two-stage framework, A^{2}CoST, for the critical scenario generation problem. The former stage provides a formalization of plausible scenarios, which can rule out many invalid scenarios. The latter stage uses two search processes to refine these plausible scenarios to identify whether they are critical and collision-avoidable, respectively.
2. We develop a lightweight simulator, LASS, that allows customizing a simulation environment by an OpenSCENARIO file and supports OpenAI Gym's API for RL training. A distributed simulation framework is developed alternatively to meet the needs of end-to-end and high-precision simulation.
3. Our comprehensive experiments show that A^{2}CoST can generate plausible, critical, collision-avoidable scenarios effectively compared with previous methods.

2 Related Work

2.1 Scenario Description Language

The scenario-based approaches are widely applied to the algorithm development of automated vehicles. A typical scenario-based approach often requires recording and restoring scenarios, which gives birth to various traffic scenario description languages. Depending on the level of abstraction and presentation ability, the scenarios can be classified into *functional scenarios*, *logical scenarios*, and *concrete scenarios* (Menzel, Bagschik, and Maurer 2018).

Scenic (Fremont et al. 2020) is a logical scenario description language based on Python. However, specifying dynamic scenarios in Scenic is elaborate, especially when the entities have complex actions. Paracosm (Majumdar et al. 2019) describes logical scenarios by parameters. External controllers can be imported to depict dynamic details. Geoscenario (Queiroz, Berger, and Czarnecki 2019) is a domain-specific language used to describe concrete scenarios. Waypoints, maneuvers, and triggers are used to depict the behavior of the entities in a scenario. OpenSCENARIO is a widely used concrete and logical scenario description language. Similar to Geoscenario, it describes dynamical contents with parameterized maneuvers and triggers. OpenSCENARIO is also compatible with OpenDRIVE[2], a widely used road network description language, and is supported by multiple simulators. Thus, we choose OpenSCENARIO to represent the logical and concrete scenarios in our method.

2.2 Critical Scenario Generation

Critical scenario generation is a problem in finding failures of the VUT under simulated environments. This technique allows testing the VUT as a black box or gray box, which makes it suitable for testing complex systems.

A survey by Riedmaier et al. (2020) thoroughly examined and classified recent scenario-based safety assessments of AVs. In this survey, the authors call the criti-

[1] https://www.asam.net/standards/detail/openscenario/

[2] https://www.asam.net/standards/detail/opendrive/

cal scenario generation problem as *simulation-based falsification*. In general, several early approaches use searching and optimization methods to find failures in static or low DOF scenarios (Beglerovic, Stolz, and Horn 2017; Gangopadhyay et al. 2019; Koschi et al. 2019). However, these methods have limited applicability in complex scenarios, making them less effective. To address this issue, researchers have investigated Reinforcement Learning (RL) algorithms to generate complex scenarios. An example is Adaptive Stress Testing (Koren et al. 2018; Koren and Kochenderfer 2019), which utilizes Monte Carlo Tree Search and RL to generate adversarial scenarios. Despite their success, finding critical scenarios using RL is a sparse reward task and requires a lot of extra interactions. Researchers have attempted to improve data efficiency by incorporating domain expert knowledge (Du and Driggs-Campbell 2021), employing ensemble reinforcement learning (Chen et al. 2021), and backward algorithm (Koren, Nassar, and Kochenderfer 2021). Other approaches such as multimodal deep generative network (Ding et al. 2021) and grammar-guided learning-based fuzz testing (Zhong, Kaiser, and Ray 2022) have also been investigated.

However, existing RL-based approaches have been limited by the few adversarial agents used to expose issues in intricately designed VUTs. The collisions between adversarial agents and their lack of knowledge on how to attack the VUT without running into themselves have made it challenging to extend these algorithms to more complex scenarios. Additionally, the curse of dimensionality has led to a complicated policy and applying tricks like hierarchical RL is difficult due to the asynchronous high-level maneuvers of different vehicles.

In addition, ontology-based approaches have been applied to construct a test scenario database to be utilized by downstream tasks such as combinatorial testing (Bagschik, Menzel, and Maurer 2018; Li, Tao, and Wotawa 2020; Bannour, Niol, and Crisafulli 2021). It is important to note that these methods aim to generate coarse-grained scenarios and do not consider the VUTs, in contrast to critical scenario generation.

2.3 Answer Set Programming and Planning

Answer Set Programming (ASP) is a logic language with stable model semantics, thus a non-monotonic reasoning mechanism (Gelfond and Lifschitz 1989; Brewka, Eiter, and Truszczyński 2011). As a form of declarative programming, ASP uses logical expressions to describe a problem instead of the algorithm for solving it. Therefore, it is convenient to specify constraints for dynamic systems.

An answer set for an ASP program is a set of atoms that satisfies the stable model semantics. Intuitively, if the body of a rule is satisfied by an answer set, then its head should also be satisfied. Answer sets are considered solutions of an ASP program, which can be computed by an ASP solver.

By regarding the timestep as a series of incremental atoms, the ASP can be used to solve planning problems (Subrahmanian and Zaniolo 1995). Dimopoulos, Nebel, and Koehler (1997) have proved its efficiency, especially in solving NP-hard ones. We have seen its success in the action planning of service robots (Chen et al. 2011), autonomous driving (Kothawade et al. 2021), and multi-agent path finding (Nguyen et al. 2017).

3 Method

In this section, we specify details of A^{2}CoST, our two-stage framework for critical scenario generation. This section is organized as follows: Subsection 3.1 reviews the language specifications of ASP, and Subsection 3.2 and Subsection 3.3 explain the two stages of A^{2}CoST, respectively.

3.1 Language Specifications of ASP

To aid comprehension of the forthcoming ASP code, this subsection briefly reviews the language specifications of ASP. Further details can be found in the Potassco clingo document[3] (Gebser et al. 2019).

An ASP program consists of rules like:

$$A_0 \texttt{ :- } L_1, \ldots, L_N \texttt{.}$$

where the *head* A_0 is an *atom* like `p` or `p(t₁, ..., tₘ)`, which can be denoted by `p/m`. $\texttt{t}_\texttt{i}$ $(1 \leq i \leq m)$ represents a *term*, which can be a number, a *constant* leading with a lowercase letter, or a *variable* leading with an uppercase letter, standing for all variable-free terms. The *body* is the conjunction of *literals* L_j $(1 \leq j \leq N)$ of the form A or `not` A, where A is an atom. A rule can have an empty head or body, representing a constraint or a fact, respectively. An empty head denotes a constraint that the body can never be satisfied, while an empty body represents a fact that is always true.

An atom begins with a dash "–" is called *classical negation*. For example, `-p` has the same meaning as `neg_p` with a constraint:

```
:- p, neg_p.
```

3.2 Scenario Modeling and Generation

Here we specify the first stage of A^{2}CoST, i.e., the formalization of logical scenarios. Notice that logical scenarios are plausible scenarios for vehicles with maneuvers. These scenarios should adhere to real-world behavioral regularities and ensure collision-free movements. Our experimental focus is on a straight and unidirectional freeway scene. In the following, we detail how the problem of logical scenario generation can be formalized as a planning problem in this specific scene.

As illustrated in Fig. 4, given $N+1$ vehicles $\{o_0, \ldots, o_N\}$ where o_0 denotes the vehicle-under-test, we specify their initial longitudinal positions $p(o_0, 0), p(o_1, 0), \ldots, p(o_N, 0) \in \mathbb{P}$, velocities $v(o_0, 0), v(o_1, 0), \ldots, v(o_N, 0) \in \mathbb{V}$ and locating lanes $l(o_0, 0), l(o_1, 0), \ldots, l(o_N, 0) \in \mathbb{L}$, where $\mathbb{P}$, $\mathbb{V}$ and $\mathbb{L}$ are sets of valid positions, velocities, and lanes, respectively. Notice that, these longitudinal positions, velocities, and locating lanes for all vehicles specify the initial state of the scenario.

At each timestep t, a vehicle o can perform a lateral maneuver $m_{la}(o, t) \in \mathbb{M}_{la}$ with a duration of T_m steps to

[3] https://potassco.org/doc/

change the present locating lane $l(o,t)$ of o to the resulting lane $l(o,t+T_m)$. We use $\mathbb{M}_{la}$ to denote the set of valid lateral maneuvers. We can also specify the changing process $l(o,t+1),\dots,l(o,t+T_m)$ of o's lane locations in T_m steps, where $l(o,t+i)$ has a slight lateral movement w.r.t. $l(o,t+i-1)$ for $1 \leq i \leq T_m$.

Similarly, a vehicle o can also perform a longitudinal maneuver $m_{lo}(o,t) \in \mathbb{M}_{lo}$ with a duration of T_n steps to change the present velocity $v(o,t)$ and the present longitudinal position $p(o,t)$ of o to the resulting velocity $v(o,t+T_n)$ and the position $p(o,t+T_n)$, where $\mathbb{M}_{lo}$ denotes the set of valid longitudinal maneuvers. We also specify the corresponding changing process $v(o,t+1), p(o,t+1),\dots,v(o,t+T_n), p(o,t+T_n)$, where $v(o,t+i)$ has a slight variation w.r.t. $v(o,t+i-1)$ and $p(o,t+i)$ is determined by the following equation ($1 \leq i \leq T_n$):

$$p(o,t+i) = p(o,t+i-1) + v(o,t+i-1)\Delta t, \qquad (1)$$

where Δt denotes the length of the timestep.

Note that, the duration T_m and T_n depend on the specific lateral and longitudinal maneuvers, respectively.

We can set the target longitudinal positions, velocities, and locating lanes for all vehicles to specify the goal state, i.e., the set

$$\{\, p(o_i,t), v(o_i,t), l(o_i,t) \mid p(o_i,t) \in \mathbb{P},\ v(o_i,t) \in \mathbb{V},\ l(o_i,t) \in \mathbb{L},\ 0 \leq i \leq N \,\}.$$

We can also specify a set $\mathbb{C}$ of constraints to find preferred solutions. These constraints can be specified by longitudinal positions, velocities, and locating lanes of corresponding vehicles in ASP. The constraint set $\mathbb{C}$ also defines the collision rules of the vehicles. These rules ensure the scenarios are valid without collisions thus excluding a great number of invalid scenarios.

Based on the above initial state, the definitions of lateral and longitudinal maneuvers, the target state, and constraints, we can specify a planning problem in ASP. The ASP solver can compute its solution that contains a sequence of lateral and longitudinal maneuvers for each vehicle, such that the goal state is achieved at the last time and all constraints are satisfied. Given a compatible initial configuration and a reachable goal condition, it is guaranteed that at least one solution exists.

The primary role of the goal state is to avoid generating unsolvable scenarios, as illustrated in Fig. 3. The vehicle o_0 will be replaced by the VUT during the next stage. By specifying a goal for the vehicle o_0 (e.g., requiring all the other vehicles to be located behind it), an achievable trajectory will be planned for it, ensuring there are no unpreventable collisions in the generated scenarios. It is also convenient to customize the generated scenarios by adding more requirements to the goal.

We use ASP to encode the planning problem. For example, a maneuver accelerating by $1\,\mathrm{m/s}$ in one timestep can be described as follows:

```
h(speed(Car, Spd+1), t) :-
    occurs(accelerate, Car, t-1),
    is_car(Car),
    is_speed(Spd),
    h(speed(Car, Spd), t-1).
-h(during_longitudinal_maneuver(Car),t):-
    occurs(accelerate, Car, t-1),
    is_car(Car).
```

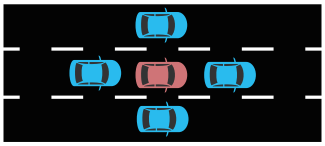


Figure 3: An unsolvable scenario: the VUT is surrounded.

Where the atom `h/2` stands for "hold", which means that the property is true at the timestep. The position update rule corresponds to Equ. (1) is:

```
h(position(Car, Pos+Spd), t) :-
    is_car(Car),
    is_position(Pos),
    is_position(Pos+Spd),
    is_speed(Spd),
    h(speed(Car, Spd), t-1),
    h(position(Car, Pos), t-1).
```

More examples and explanations of encoding can be found in our demo code[4].

We use Potassco clingo (Gebser et al. 2019), a state-of-the-art ASP solver, to compute answer sets of the program. Each answer set specifies a logical scenario and can be translated into an OpenSCENARIO file. The maneuver sequence for every vehicle is determined in a logical scenario. However, the parameters of each maneuver (e.g., start time, duration, acceleration) can vary in a narrow range, thus leading to various detailed concrete scenarios.

3.3 Critical Scenario Searching

After the first stage, we obtain a group of plausible logical scenarios under the given initial Configuration and goal. Here we specify the second stage of A^{2}CoST, i.e., two search phases that refine generated logical scenarios into concrete scenarios and identify whether they are critical and collision-avoidable, respectively. In the case of collision scenario generation applications, the second stage commences by randomly selecting a logical scenario. However, for comprehensive testing of the VUT, all logical scenarios need to be tested during the second stage.

As we show in Alg. 1, the first phase describes the search process for parameters of macro-maneuvers that result in a collision happening on the VUT, i.e., critical scenarios, and the second phase describes the search process for finding a collision-free trajectory for the VUT, i.e., collision-free remedy.

Here we formally define a logical scenario as a quintuple $\mathcal{S} = \{O, M_{la}, M_{lo}, \Theta, \Phi\}$, where $O = \{o_0,\dots,o_N\}$ is the set of all vehicles, M_{la} and M_{lo} are sets of sequences

[4] https://github.com/Autonomous-Driving-Safety-Project/ASP-GEN-demo

Algorithm 1: Critical Scenario Searching

```
Data: logical scenario $\mathcal{S}$, vehicle-under-test $o_{VUT}$,
      gradient-free optimizer $\mathcal{O}$
Result: critical concrete scenario $s_c$ for VUT,
        collision-free remedy scenario $s_r$
// Phase 1
$\mathcal{S}^* \leftarrow$replace vehicle $o_0$ in $\mathcal{S}$ by $o_{VUT}$;
construct a concrete scenario $s(\theta) \in \mathcal{S}^*$ with random
  parameters $\theta \in \Theta$;
do
    run $s(\theta)$ in simulator and get the minimal
      distance $d(s(\theta))$ between the VUT and the
      adversarial vehicles;
    update $\theta$ with $\mathcal{O}$ by minimize $d(s(\theta))$;
while $d(s(\theta)) > 0$;
$s_c \leftarrow s(\theta)$;
// Phase 2
do
    construct a concrete scenario $s(\theta, \varphi) \in \mathcal{S}$ with
      random parameters $\varphi \in \Phi$ for $o_0$'s maneuvers;
while $s(\theta, \varphi)$ has collision;
$s_r \leftarrow s(\theta, \varphi)$;
```

of lateral and longitudinal maneuvers of each vehicle, respectively. Θ denotes the domain of maneuver parameters of adversarial vehicles $\{o_1, \dots, o_N\}$, and Φ denotes the domain of maneuver parameters of o_0. In the first phase of our method, we replace the vehicle o_0 in the logical scenario $\mathcal{S}$ with the VUT and evaluate the scenario in a simulator to measure the collision risk of the VUT with the adversarial vehicles. Due to the gap between the simulator and optimization objective, we use a gradient-free optimization algorithm like stochastic hill-climbing or pattern search (Hooke and Jeeves 1961), to maximize the collision risk. Moreover, there will still come up with a collision among adversarial vehicles occasionally after the first phase. We address the problem by restarting the search progress with other randomly initialed values.

Note that we do not determine whether the VUT is responsible for the collisions. Liability determination is a complex issue that goes beyond technical considerations. To circumvent this issue, we require that collisions be avoidable, which is in line with the well-known RSS model (Shalev-Shwartz, Shammah, and Shashua 2017).

Thus, in the second phase, we aim to find a collision-free trajectory for the VUT by using the planned maneuver sequence of vehicle o_0 in the logical scenario. We randomly assign parameters to o_0's maneuvers while keeping the behavior of the adversarial vehicles fixed until a collision-free remedy scenario is found, which proves that the collision found in the first phase is avoidable. Note that this is straightforward as the logical scenarios generated in the former stage are designed to be collision-free.

Algo.	Param.	Value	Description
IDM	v_0	25m/s	desired velocity
	a	3m/s^2	maximum acceleration
	δ	4	exponent
	s_0	1m	minimum spacing
	T	1.5s	desired time headway
	b	1.6m/s^2	comfortable braking deceleration
MOBIL	b_{safe}	4m/s^2	safe limit
	p	0.2	politeness factor
	Δa_{th}	0.5m/s^2	lane-changing threshold
PID	k_p	0.2	proportional gain
	k_i	0	integral gain
	k_d	20	derivative gain

Table 1: Hyper-parameters of the VUT's algorithms

4 Experiment

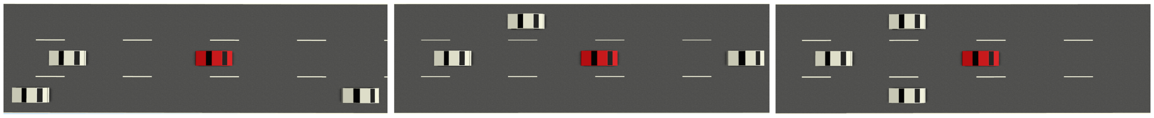
We conduct an evaluation of the effectiveness of A^{2}CoST in the LASS simulator. To assess the performance of A^{2}CoST, we compare it with two other algorithms: an RL-based method proposed in (Chen et al. 2021) and a random search algorithm. We measure the performance of each algorithm in terms of the time taken to identify the first collision in the simulation. To ensure the reliability of our results, we run each algorithm five times with different random seeds (i.e., 10, 20, 30, 40, and 50). The experiments are conducted using scenarios involving three adversarial vehicles, each with initial states as shown in Fig. 4.

4.1 Experiment Configurations

Vehicle-Under-Test In our experiments, the VUT is controlled by the Intelligent Driver Model (IDM) (Treiber, Hennecke, and Helbing 2000) and the MOBIL lane-changing algorithm (Kesting, Treiber, and Helbing 2007). A PID controller is used to generate low-level lateral actions. The hyper-parameters of the VUT's algorithms are shown in Table 1.

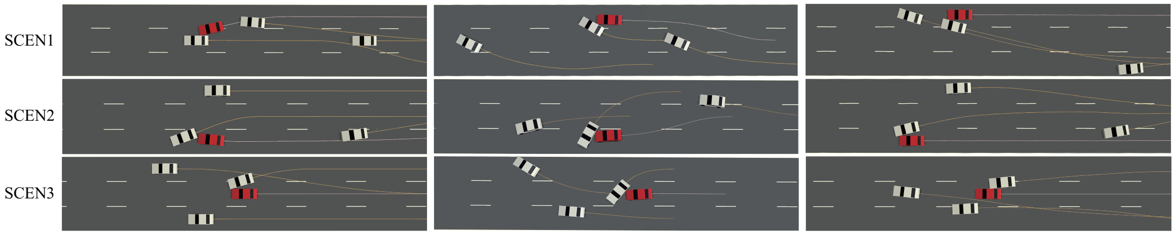
Baselines As there are limited open-source implementations and details on the domain knowledge used in some existing methods, a direct comparison of our results with most of those previous works is infeasible. Therefore, we establish an RL-based method proposed in (Chen et al. 2021) and a random search method as our baselines for comparison purposes.

1. **RL baseline**: The RL-based method is trained in the LASS simulator with a 15-dimensional observation space (containing the lateral coordinate of the VUT, three adversarial vehicles' relative positions to the VUT, and the headings and velocities of all vehicles) and a 6-dimensional action space (containing three adversarial vehicles' throttle and steering values). The reward function is
$$r = -r_{VUT} + r_{rule} + r_{shaping},$$



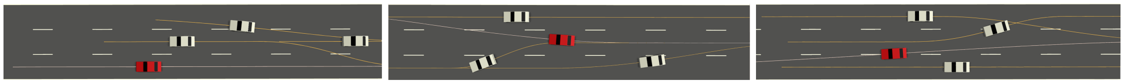
(a) Scenario 1 (SCEN1) (b) Scenario 2 (SCEN2) (c) Scenario 3 (SCEN3)

Figure 4: Bird's-eye views of the initial states of the three experimental scenarios. The VUT is marked in red.



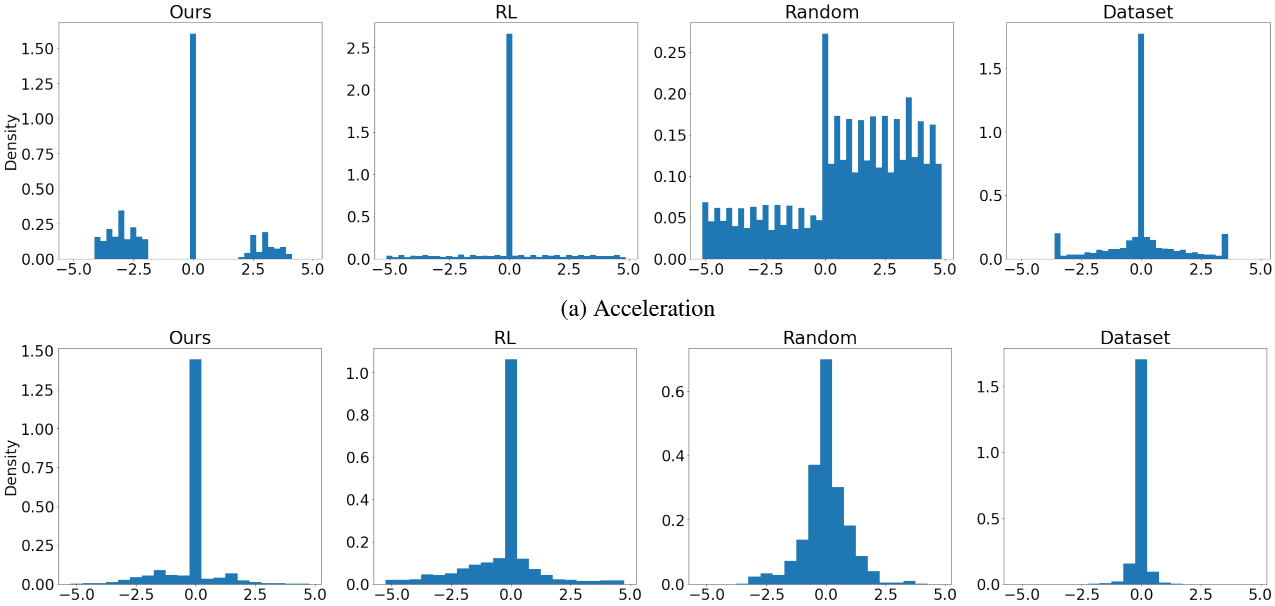
(a) A^{2}CoST (ours) (b) RL-based (c) Random search

Figure 5: Collision scenarios found by our framework and the baseline methods.



(a) SCEN1 (b) SCEN2 (c) SCEN3

Figure 6: Collision-free remedies for Fig. 5a found by A^{2}CoST. Notice that the adversarial vehicles follow the same trajectories shown in Fig. 5a.



(a) Acceleration

(b) Lateral velocity

Figure 7: Histograms of the acceleration and lateral velocity in generated scenarios and the dataset.

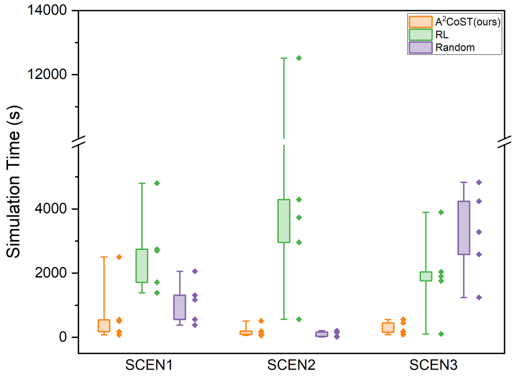


Figure 8: Time consumption before finding the first collision. (Less is better)

where

$$r_{VUT} = \begin{cases} 100, & \text{if episode ends without collision} \\ -50, & \text{if VUT collide with the others} \\ 0.1 v_{VUT} & \text{otherwise,} \end{cases}$$

and

$$r_{rule} = \begin{cases} -50, & \text{if adversarial vehicles rear-end VUT} \\ -500, & \text{if adversarial vehicles out of road,} \\ & \text{or run into themselves} \\ 0 & \text{otherwise,} \end{cases}$$

as well as

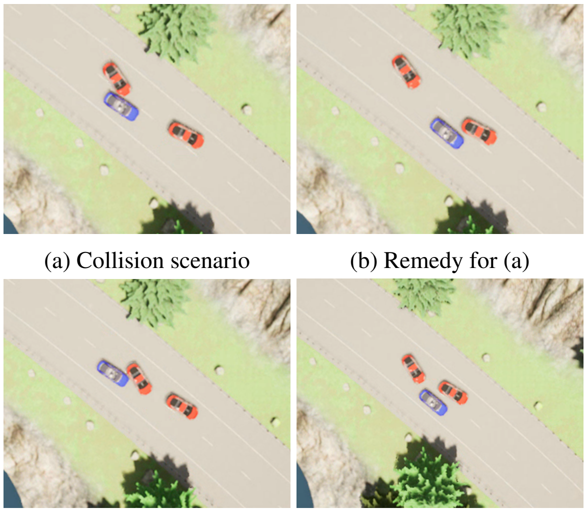
$$r_{shaping} = -0.001 \min distance(o_{VUT}, o_{adversarial}).$$

The original DDPG algorithm in (Chen et al. 2021) is replaced by PPO (Schulman et al. 2017) for better performance and less training time. The hyper-parameters used for training are shown in Table 2.

2. **Random search baseline**: The random search baseline's action space is the same as the RL baseline's. To keep the random action unbiased, the throttle value of each adversarial vehicle is sampled from a zero-expectation distribution instead of a uniform one.

Configurations of A^{2}CoST In the first stage, the configurations are:

1. **Maneuvers**: The allowed maneuvers and their adjustable parameters are listed in Table 3.
2. **Goal**: The goal for o_0 is to overtake all the other vehicles. Besides, to encourage generating more merge maneuvers, the target lane of o_0 is set to be the leftmost lane in SCNE1 and SCEN3, and the rightmost lane in SCEN2. The other vehicles' goals are not specified.
3. **Initial configurations** are the same as those shown in Fig. 4.



(a) Collision scenario (b) Remedy for (a)

(c) Collision scenario (d) Remedy for (c)

Figure 9: Two avoidable collision scenarios found by A^{2}CoST on CARLA simulator.

Parameter	Value
discount factor γ	0.99
policy learning rate	10^{-3}
value function learning rate	10^{-2}
clip ratio	0.2
lambda for GAE	0.97
max episode length	1000
steps per epoch	5000
hidden layers	[64,64]

Table 2: Hyper-parameters for training RL baseline

We generated 1000 logical scenarios for each initial configuration with the timestep $\Delta t = 1$s. In the second stage, we employed a stochastic hill-climbing algorithm as the gradient-free optimizer. The minimal distance between the VUT and adversarial vehicles is used to assess the collision risk. The maximum step size is set to be 0.5, and the batch size is set to be 16. The logical scenario is randomly selected each time.

4.2 Performance of A^{2}CoST

The collision scenarios found in experiments are shown in Fig. 5, and the time consumption in the simulator of each algorithm is shown in Fig. 8. Additionally, the collision-free remedies found by A^{2}CoST are shown in Fig. 6.

We observed that the performance of the RL-based method and random search algorithm is inconsistent and influenced by the initial states and random seeds. On the other hand, our proposed A^{2}CoST framework outperforms the baselines in various scenarios. Additionally, A^{2}CoST can effectively generate critical scenarios for the VUT with up to five adversarial vehicles, a challenging task for the baselines (see Fig. 2).

Maneuver	Parameter	Range
accelerate	acceleration	$2.0 \sim 4.0\,\mathrm{m/s^2}$
	duration	$0.3 \sim 3.0\,\mathrm{s}$
decelerate	acceleration	$-4.0 \sim -2.0\,\mathrm{m/s^2}$
	duration	$0.3 \sim 3.0\,\mathrm{s}$
merge_left	duration	$1.0 \sim 5.0\,\mathrm{s}$
merge_right	duration	$1.0 \sim 5.0\,\mathrm{s}$

Note: the start time of each maneuver can also vary in the range of $\pm 1.0\,\mathrm{s}$.

Table 3: Allowed maneuvers and parameters

	Acceleration	Lateral Velocity
A^{2}CoST	**0.963**	**0.417**
RL	4.708	1.356
Random	4.315	0.519

Table 4: Wasserstein distance between generated scenarios and the dataset

To assess the similarity between the scenarios generated by our proposed method and those in the real world, we conducted statistical analysis on their acceleration and lateral velocity. Specifically, we compared the results to the NGSIM I-80 dataset, which contains trajectories of vehicles collected on the U.S. I-80 freeway. The histograms showing the distributions of acceleration and lateral velocity are presented in Fig. 7. We further calculated the Wasserstein distance between the distribution of these metrics in the generated scenarios and the dataset. The Wasserstein distance is a metric that represents the distance between two distributions. It measures the minimal cost of turning one distribution into the other. Compare with the well-known KL divergence, the Wasserstein distance is symmetrical and finite, thus more suitable here. The results, presented in Table 4, confirm that A^{2}CoST generates the most plausible scenarios.

4.3 Experiment on CARLA Simulator

To fulfill the high-precision and end-to-end simulation needs of our proposed method, we developed a distributed simulation framework based on CARLA, which can be used as an alternative option to LASS. The framework consists of a manager node that accepts simulation requests via the HTTP protocol. Multiple CARLA instances can be registered to the manager node and scheduled by it. The distributed architecture offers two key benefits. First, it speeds up the second stage of A^{2}CoST by running multiple simulations simultaneously. Second, it physically isolates A^{2}CoST and the autonomous driving system, which circumvents security issues that may arise in commercial applications.

Fig. 9 shows two avoidable collision scenarios found by A^{2}CoST on the CARLA simulator. In this experiment, the VUT is controlled by Learning by Cheating (Chen et al. 2020), an end-to-end autonomous driving algorithm. The result indicates the effectiveness of A^{2}CoST on launching adversarial attacks on end-to-end AVs in a high-precision simulator.

5 Conclusion

In this paper, we present A^{2}CoST, a two-stage framework for generating plausible, critical, and collision-avoidable scenarios for the vehicle-under-test. Our proposed framework is designed to improve efficiency by formalizing plausible logical scenarios with Answer set programming in the first stage, which enables the exclusion of invalid scenarios. In the second stage, we refine the logical scenarios into critical and collision-avoidable concrete scenarios with two search processes. We evaluate the efficiency of A^{2}CoST by comparing it with two baselines and demonstrate its plausibility using a dataset from the real world. To speed up the process, we provide a lightweight simulator, and for end-to-end and high-precision requirements, we offer an alternative distributed simulation framework based on CARLA. As a knowledge- and logic-based method, A^{2}CoST offers additional interpretability and mathematical rigor compared to learning-based methods. Therefore, it has significant potential for improving the safety of autonomous vehicles and can pave the way for future research in this area.

However, our proposed method has some limitations that need to be addressed. Firstly, the ASP representation we used is not comprehensive enough to cover all possible logical scenarios. This is due to the approximation involved in describing maneuvers using ASP. While a more fine-grained description may alleviate this issue, it is currently not feasible within our current framework. Secondly, A^{2}CoST is not yet capable of handling urban scenarios. Our future research will focus on investigating better representation that establishes equivalence between answer sets and valid logical scenarios, and extend the method to urban scenarios.

Acknowledgments

The work is partially supported by Guangdong Province R&D Program 2020B0909050001, Anhui Province Development and Reform Commission 2020 New Energy Vehicle Industry Innovation Development Project and 2021 New Energy and Intelligent Connected Vehicle Innovation Project, and Shenzhen Yijiahe Technology R&D Co., Ltd.

References

Bagschik, G.; Menzel, T.; and Maurer, M. 2018. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, 1813–1820. IEEE.

Bannour, B.; Niol, J.; and Crisafulli, P. 2021. Symbolic model-based design and generation of logical scenarios for autonomous vehicles validation. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, 215–222. IEEE.

Beglerovic, H.; Stolz, M.; and Horn, M. 2017. Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 1–6. IEEE.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Calò, A.; Arcaini, P.; Ali, S.; Hauer, F.; and Ishikawa, F. 2020. Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 375–386. IEEE.

Chen, X.; Jin, G.; Ji, J.; Wang, F.; and Xie, J. 2011. Kejia project: Towards integrated intelligence for service robots. *RoboCup@ Home League Team Descriptions, Istanbul.*

Chen, D.; Zhou, B.; Koltun, V.; and Krähenbühl, P. 2020. Learning by cheating. In *Conference on Robot Learning*, 66–75. PMLR.

Chen, B.; Chen, X.; Wu, Q.; and Li, L. 2021. Adversarial evaluation of autonomous vehicles in lane-change scenarios. *IEEE Transactions on Intelligent Transportation Systems.*

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In *European Conference on Planning*, 169–181. Springer.

Ding, W.; Chen, B.; Li, B.; Eun, K. J.; and Zhao, D. 2021. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. *IEEE Robotics and Automation Letters* 6(2):1551–1558.

Du, P., and Driggs-Campbell, K. 2021. Adaptive failure search using critical states from domain experts. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 38–44. IEEE.

Fremont, D. J.; Kim, E.; Dreossi, T.; Ghosh, S.; Yue, X.; Sangiovanni-Vincentelli, A. L.; and Seshia, S. A. 2020. Scenic: A language for scenario specification and data generation. *arXiv preprint arXiv:2010.06580.*

Gangopadhyay, B.; Khastgir, S.; Dey, S.; Dasgupta, P.; Montana, G.; and Jennings, P. 2019. Identification of test cases for automated driving systems using bayesian optimization. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 1961–1967. IEEE.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Gelfond, M., and Lifschitz, V. 1989. The stable model semantics for logic programming. In *Proc. 5th International Conference and Symposium on Logic Programming*, 1070–1080.

Hauer, F.; Pretschner, A.; and Holzmüller, B. 2020. Reusing concrete test scenarios generally is a bad idea. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, 1305–1310. IEEE.

Hooke, R., and Jeeves, T. A. 1961. Direct search solution of numerical and statistical problems. *Journal of the ACM (JACM)* 8(2):212–229.

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

Kesting, A.; Treiber, M.; and Helbing, D. 2007. General lane-changing model mobil for car-following models. *Transportation Research Record* 1999(1):86–94.

Koren, M., and Kochenderfer, M. J. 2019. Efficient autonomy validation in simulation with adaptive stress testing. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 4178–4183. IEEE.

Koren, M.; Alsaif, S.; Lee, R.; and Kochenderfer, M. J. 2018. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, 1–7. IEEE.

Koren, M.; Nassar, A.; and Kochenderfer, M. J. 2021. Finding failures in high-fidelity simulation using adaptive stress testing and the backward algorithm. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5944–5949. IEEE.

Koschi, M.; Pek, C.; Maierhofer, S.; and Althoff, M. 2019. Computationally efficient safety falsification of adaptive cruise control systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2879–2886. IEEE.

Kothawade, S.; Khandelwal, V.; Basu, K.; Wang, H.; and Gupta, G. 2021. Auto-discern: Autonomous driving using common sense reasoning. *arXiv preprint arXiv:2110.13606.*

Kuutti, S.; Fallah, S.; and Bowden, R. 2020. Training adversarial agents to exploit weaknesses in deep control policies. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 108–114. IEEE.

Li, Y.; Tao, J.; and Wotawa, F. 2020. Ontology-based test generation for automated and autonomous driving functions. *Information and software technology* 117:106200.

Majumdar, R.; Mathur, A.; Pirron, M.; Stegner, L.; and Zufferey, D. 2019. Paracosm: A language and tool for testing autonomous driving systems. *arXiv preprint arXiv:1902.01084.*

Menzel, T.; Bagschik, G.; and Maurer, M. 2018. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, 1821–1827. IEEE.

Nguyen, V.; Obermeier, P.; Son, T. C.; Schaub, T.; and Yeoh, W. 2017. Generalized target assignment and path finding using answer set programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 1216–1223.

Queiroz, R.; Berger, T.; and Czarnecki, K. 2019. Geoscenario: An open dsl for autonomous driving scenario representation. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 287–294. IEEE.

Riedmaier, S.; Ponn, T.; Ludwig, D.; Schick, B.; and Diermeyer, F. 2020. Survey on scenario-based safety assessment of automated vehicles. *IEEE access* 8:87456–87477.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2017. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374.*

Subrahmanian, V., and Zaniolo, C. 1995. Relating stable models and ai planning domains. In *ICLP*, volume 95, 233–247. Citeseer.

Treiber, M.; Hennecke, A.; and Helbing, D. 2000. Congested traffic states in empirical observations and microscopic simulations. *Physical review E* 62(2):1805.

U.S. Department of Transportation Federal Highway Administration. 2016. Next Generation Simulation (NGSIM) Vehicle Trajectories and Supporting Data. [Dataset]. Accessed 2022-07-23.

Vallender, S. 1974. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications* 18(4):784–786.

Zhong, Z.; Kaiser, G.; and Ray, B. 2022. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering*.