Aggressiveness-regulated Multi-agent Stress Testing of Autonomous Vehicles

by

Xiaoliang Zhou

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

Abstract

The emerging era of autonomous vehicles (AVs) presents unprecedented potential for transforming global transportation. As these vehicles begin to permeate our streets, the challenge of ensuring their safety, especially in unprecedented scenarios, looms large, due to the infrequent occurrence of high-risk scenarios within an essentially infinite number of test cases. This Master's thesis explores the intricate challenge of stress testing autonomous vehicles in simulated environments. The study delves into the application of multi-agent reinforcement learning (MARL) as a tool for stress testing AVs. Although MARL demands higher computational resources, it demonstrates strong ability in uncovering complex accident scenarios. This marks a shift from the state-of-the-art which deploys single-agent reinforcement algorithms that encounter limitations both in the quality of the generated accident scenarios and in their ability to generate complex accident scenarios as the number of traffic participants increases. Central to our approach is the integration of constraints that regulate the level of aggressiveness of traffic participants to induce more realistic and insightful accident scenarios. The thesis also presents the *highway-attack-env*, an environment for black-box AV testing that allows the assessment of both single and multi-agent reinforcement learning algorithms. The contributions of this research include the introduction of the aforementioned environment and a comprehensive benchmark, as well as a comparative analysis of single-agent and MARL algorithms, underscoring the superiority of the proposed multi-agent, aggressiveness-regulated methodology for AV validation.

Acknowledgements

I would like to thank my supervisors Professor Mark Crowley and Professor Seyed Majid Zahedi for their invaluable mentorship throughout my master's journey. Their unwavering support was instrumental in the realization of my research.

I'm equally thankful to my parents and my life partner Xiaomeng Lei for all their love and support.

I also want to thank all members in the Multi-agent Systems Team and ECE Machine Learning Lab for their kindness.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of autonomous vehicles (AVs) stands at the forefront of groundbreaking innovations in transportation. As the combination of machine learning and robotics transforms our roads, the promise of safer, more efficient, and environmentally friendly transportation emerges. However, with any paradigm shift, challenges arise, and in the case of AVs, one of the most significant hurdles is validation.

Autonomous vehicle validation is not merely about ensuring that the vehicle operates correctly under typical conditions but about proving that it can handle the rarest, most dangerous scenarios that could arise on the road. To understand the full scope of this challenge, it is important to note that the risks of AV technologies remain largely uncharted. Fatal accidents involving AVs underscore the pressing need to determine if AVs can safely coexist with humans. Real-world testing, while the most direct method, is time-consuming and, at times, dangerous due to the infrequency of severe accidents.

Simulation-based testing has emerged as a more economical and efficient alternative to real-world testing. Much effort has been made both in industry and in academia to build realistic simulations [14] [40]. However, due to the substantial computational overhead associated with high-fidelity simulators, this Master's thesis employs a more simplified AV simulation environment for the experiments, which is adapted from *highway-env* [28]. Additionally, AV validation under normal driving conditions is insufficient for revealing the full spectrum of vulnerabilities within an AV system, given the rarity of these high-risk scenarios. In our thesis, we focus on the AV stress testing problem instead, in which we design adversarial agents tasked with the objective of compromising the target AV system, thereby generating failure scenarios that offer revealing insights into its weaknesses. Moreover, we acknowledge that not every accident involving the target AV system serves as a constructive input for its improvement. There exist unavoidable accidents that cannot be attributed to the target AV system. Additionally, given that the search space for failure scenarios grows exponentially with the number of traffic participants in the simulated environment, the search for realistic and insightful accidents becomes a challenging problem.

Reinforcement learning (RL), particularly deep RL, powered by neural networks, has shown remarkable efficacy across a wide array of complex tasks. Several studies have leveraged RL as an approach to tackle different tasks in autonomous driving. In particular, RL algorithms have been effectively applied to solve the AV motion planning task across a range of scenarios [24] [20] [36] [33]. On the challenge of AV stress testing, adaptive stress testing method has been utilized to find likely failure scenarios [25], where Trust Region Policy Optimization (TRPO) [37] is employed as a solver to handle the sequential decision making problem of finding failure scenarios. Following the naming convention of RL, we name the target AV system as the target agent, and the other traffic participants as the attacker agents. To the best of our knowledge, only single-agent RL algorithms are explored in the literature of AV stress testing, which typically suffers from two issues:

- Given the exponential growth of the search space, single-agent RL algorithms might fall short of detecting complex failure scenarios that involve multiple traffic participants.
- The identified failure scenarios are often unavoidable and unrealistic, offering limited utility for enhancing the AV system.

To address the issue of search space complexity, we argue that conceptualizing the AV stress testing problem within a multi-agent framework, where agents are independently controlled, could offer substantive advantages. In this context, this Master's thesis focuses on a more complex but arguably more natural scenario: defining the AV stress testing problem as a multi-agent MDP and leveraging multi-agent reinforcement learning (MARL) algorithms to stress test AVs. This approach allows each agent to optimize independently while guided by a centralized value estimation shared across the agents, which encourages coordinated behavior between the agents and leads to discovery of more interesting failure scenarios.

To improve realism, we introduce safety constraints into the learning process of the attacker agents. In many real-world applications, agents are frequently deterred from accessing certain states or actions labeled as "unsafe." Addressing these constraints while ensuring the learning agents achieve their goals is paramount. While this sentiment can be employed by the training of safer AVs, we contend that the same philosophy can be adapted to the context of AV stress testing. Specifically, these constraints can serve to regulate the actions of attacker agents, thereby encouraging the generation of accident scenarios in which attacker agents perform less unrealistic actions. Such accidents provide more insights into the vulnerabilities inherent in the AV system.

In essence, this thesis not only delves into the complexities of stress testing AVs in a simulated environment but also emphasizes generating insightful accidents by constraining attacker agents' behavior in a multi-agent setting. The ultimate objective is to create a more insightful and realistic stress testing framework for AVs, ensuring their safe integration into our daily lives.

1.1 Summary of Contributions

Our contributions are two-fold:

- We introduce a simulation environment *highway-attack-env* featuring black-box autonomous vehicle testing, which enables both single-agent and multi-agent reinforcement learning algorithms to be tested. Additionally, we provide a benchmark to evaluate the performance of different RL algorithms on the AV stress testing problem.
- We introduce multiagency and aggressiveness regulation into the AV stress testing problem to more effectively find useful accidents. We compare several single-agent and multi-agent reinforcement learning algorithms in the literature on our proposed environment and the benchmark to justify our proposal.

1.2 Organization of Thesis

In Chapter 2, we provide an overview of existing research related to black-box AV stress testing. Chapter 3 delves into the foundational concepts of single-agent and multi-agent reinforcement learning, the constrained Markov Decision Process, and the algorithms that we compared in this thesis. Chapter 4 offers a detailed description of our proposed simulation environment, specifically designed for black-box AV stress testing. Chapter 5 presents our method of introducing multiagency and aggressiveness regulation, and will provide an in-depth discussion of our experimental results. Finally, Chapter 6 summarizes our contributions and experimental findings, and outlines potential future research directions.

Chapter 2

Related Work

Stress testing for AVs is typically conducted in a black-box fashion, primarily due to intellectual property (IP) constraints preventing access to the target agent's model. Furthermore, even when such models are accessible, directly inspecting those based on deep neural networks rarely offers a thorough understanding of the model's behavior. In [9], Corso et al. provide a nice survey of algorithms for black-box AV validation problem. They classify the safety validation task into three categories:

- Finding the adversarial disturbances of the system that cause the target agent to fail (falsification).
- Finding the most-likely failure cases.
- Estimating the probability of failure of the system.

This thesis focuses on the quality of accidents generated, which falls into the first category. While many different approaches have been proposed on other categories [22] [34] [11], we focus our first part of literature review on the falsification methods. Additionally, reinforcement learning has emerged as a promising approach in AV stress testing, we dedicate another section to provide an overview of the usage of reinforcement learning in the stress testing of AVs.

2.1 AV Falsification Methods

We'd like to acknowledge that [21] offers a thorough examination of falsification methods. Some of the works we spotlight in this section have been elaborated upon in [21]. Abbas et al. [1] introduce an automated framework to identify dangerous scenarios for AVs. Their method involves a grid-search across a discretized state space that defines parameters such as the starting position of the autonomous vehicle, weather conditions, and the position and velocity of other vehicles. They choose Grand Theft Auto Five (GTA5) as their simulation environment and discuss the necessary simulator attributes to ensure that accidents identified in the virtual environment can be translated to real-world scenarios. While Abbas et al. invest significant effort into evaluating the perception algorithms, our study is solely centered on the motion planning module, assuming that the AV has accurate knowledge of the position and velocity of surrounding vehicles.

Koschi et al. [26] put forward the idea of utilizing rapidly-exploring random trees, supplemented by domain knowledge, to falsify an Adaptive Cruise Control (ACC) system. Their approach leverages domain knowledge to classify "unsafe" states, which are more prone to result in collisions. Their findings reveal that their technique can effectively falsify sophisticated ACC systems, achieving greater computational efficiency in the process. Li et al. [29] also employ domain knowledge in the search for scenarios likely to result in collisions. Furthering this, they present AV-FUZZER, an framework that employs genetic algorithms to randomly control traffic participants and discard states that are not safety-critical. Similarly, our study incorporates domain knowledge to assess the aggressiveness of traffic participants, encouraging the generation of more insightful accidents.

2.2 Reinforcement Learning Usage in Black-box AV Stress Testing

We want to underscore the success of employing reinforcement learning in AV falsification. Specifically, Koren et al. [25] propose to model the black-box AV stress testing problem as a Markov Decision Process and apply the Adaptive Stress Testing (AST) strategy to find the most-likely failure scenarios of the target agent, where reinforcement learning algorithms are adopted as solvers to the sequential decision making process. Even though their goal is to find the most likely failure scenarios of the AV, the adaptive stress testing strategy can be used in a broader scheme to search for adversarial disturbance that will cause accidents. Their work showed the successful construction of an adversarial policy against the black-box target agent, while whether the accidents generated are insightful or not is not specifically addressed. Corso et al. [6] also integrate Responsibility Sensitive Safety (RSS) and trajectory dissimilarity into the reward function, enabling AST to generate a wider range of accidents that are of greater interest. In contrast, our research takes a different approach by regulating the level of aggressiveness exhibited in the actions of the attacker agents.

Corso et al. [7] introduce an adaptive importance sampling approach to enhance reinforcement learning algorithms for rare event simulation, allowing for a more comprehensive exploration of potential failure scenarios. Ding et al. [13] [12] propose to combine reinforcement learning with generative model to facilitate the generation of risky scenarios. Notably, Feng et al. [15] propos Dense Deep Reinforcement Learning (D2RL) algorithm to address the curse of dimensionality and the curse of rarity in the AV stress testing problem by removing the non-critical states from the Markov chain with the help of domain knowledge. They show that D2RL was able to find accidents that were intractable for regular deep reinforcement learning algorithm to find. Our approach addresses this problem differently by introducing multi-agency, which helps in escaping local optima and avoiding premature convergence.

Chapter 3

Background

In this chapter, we provide the technical background of the algorithms that will be explored in the subsequent chapters of the thesis.

3.1 Reinforcement Learning Basics

In this section, we give a concise introduction to the fundamentals of reinforcement learning, including the Markov Decision Process and the general reinforcement learning paradigm.

3.1.1 Markov Decision Process

An MDP, or Markov Decision Process, is characterized by the tuple $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathcal{R}, \rho_0, \gamma)$:

- $\mathbf{S}$ denotes the state space, or all possible states. A state comprised all the information needed to make a decision of the next time step.
- $\mathbf{A}$ denotes the action space. An action is a choice of decision an agent can make when in a particular state.
- $\mathbf{P}(s'|s, a)$ is the state transition function, indicating the likelihood of transitioning from current state s to the next state s' after taking action a. The Markov property ensures that this probability is influenced solely by the current state-action pair and not any previous states or actions.

- $\mathcal{R} : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$ is the reward function, mapping a state-action pair to a real-valued reward.
- ρ_0 denotes the probability distribution of initial states.
- γ is the discount factor, determining the balance between immediate and future rewards. Furthermore, it ensures that the cumulative reward remains finite, even within an infinite horizon MDP.

3.1.2 Reinforcement Learning Algorithm

Reinforcement learning (RL) is a technique designed to solve a Markov Decision Process. In RL, two primary entities exist: the *agent* and the *environment*. The environment is responsible for managing state transitions and providing the agent with observations and rewards. Based on the observation, the agent then decides the subsequent action to take. Note that, the observation from the environment might differ from its state; when it does, the environment is considered to be *partially observable*, making the MDP a Partially Observable Markov Decision Process (POMDP). Here we define the *return* G, as the discounted sum of future rewards:

$$G = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)$$

The main objective of RL is to discover a *policy*, a rule for selecting actions, denoted as $\pi : \mathbf{S} \times \mathbf{A} \to [0, 1]$, that optimizes this return.

In the following, we outline the definitions of some essential functions that are used throughout this chapter:

- The value function for a given policy π is represented as:

$$V_\pi(s_t) = \mathbf{E}_{a_t \sim \pi(a_t|s_t), s_{t+1} \sim \mathbf{P}(s_{t+1}|s_t, a_t), \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l}) \right],$$

  denoting the expected discounted sum of reward for policy π starting at state s_t.

- The state-action value function associated with policy π is defined as:

$$Q_\pi(s_t, a_t) = \mathbf{E}_{s_{t+1} \sim \mathbf{P}(s_{t+1}|s_t, a_t), a_{t+1} \sim \pi(a_{t+1}|s_{t+1}), \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l}) \right],$$

denoting the expected discounted sum of reward for policy π starting at state s_t, given that the action taken at s_t is a_t .

- The advantage function for policy π is given by:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s).$$

3.2 Vanilla Policy Gradient and Trust Region Methods

Policy gradients (PG) are one of the fundamental methods used in reinforcement learning. Characterized as an on-policy learning technique, it relies on data acquired from the current policy. PG operates by calculating the gradient of a specified loss function to directly propose updates to the policy π. In this section, we present a brief overview of the basic Policy Gradient, alongside its extension, the actor-critic method. Subsequently, we delve into two of its notable variations: Trust Region Policy Optimization (TRPO) [37] and Proximal Policy Optimization (PPO) [39].

3.2.1 Vanilla Policy Gradient

The essence of the policy gradient technique is to update the policy π through the gradient of a designated loss function. We can define a trajectory τ spanning a length T as a finite sequence of (s, a, r) tuples: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{T-1}, a_{T-1}, r_{T-1})$. Herein, we use r to denote the output from the reward function, specified as: $r_t = \mathcal{R}(s_t, a_t)$. For an estimation of the expectation using a collection of N trajectories, we denote the set of trajectories as $\mathcal{D} = \{\tau_1, \tau_2, ..., \tau_N\}$.

Consider a policy π parameterized by θ. The optimization problem addressed by PG can be formulated as:

$$\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s_0 \sim \rho_0, a_t \sim \pi_\theta, s_t \sim \mathbf{P}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right] \tag{3.1}$$

The result gradient, denoted as g, for the aforementioned objective function can be derived as:

$$g = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{\infty} G_t \nabla \log \pi_\theta(a_t|s_t) \tag{3.2}$$

Subsequently, the policy parameter θ is updated by applying gradient ascent using the step size α.

$$\theta_{k+1} = \theta_k + \alpha g_k$$

3.2.2 Actor-Critic Method

Given the high variance when utilizing the return G_t, alternative functions often serve as replacements. In this chapter, the advantage function $A(s_t, a_t)$ is adopted, which is common in the actor-critic variety of policy gradient techniques [17]. For actor-critic methodologies, the term 'actor' denotes the policy π, while 'critic' denotes the assessment of the value of a given state, often represented by the value function V parameterized by ϕ. Subsequently, the advantage function is approximated using V_ϕ and the trajectories' rewards. Numerous techniques exist for approximating the advantage function, among which the Generalized Advantage Estimate (GAE) [38] is the most widely utilized option. In essence, actor-critic approaches employ a trained value function to approximate the advantage function, directing the update of the policy. The actor-critic algorithm is shown in Algorithm 1.

3.2.3 Surrogate Loss and Trust Region

The fundamental challenge with the Vanilla Policy Gradient technique is its sensitivity to the choice of the step size α. Given that PG directly updates the parameter space, minor modifications in parameters can induce significant shifts in the policy. This can lead to the undesired effect of forgetting prior learned behaviors. Hence, maintaining a small step size becomes crucial, which leads to suboptimal sample efficiency.

Trust Region Policy Optimization (TRPO) [37] introduces a surrogate loss, rooted in the policy improvement bound introduced by [23]. This loss takes into account the KL divergence between new and old policies. Rather than viewing the KL divergence as a penalization term in the objective function, TRPO treats it as a constraint within the optimization problem. This leads to larger step sizes and they named it as the trust region constraint. A trust region represents a sphere in the policy space wherein any policy updates are deemed reliable and safe. By ensuring policy updates remain within this defined region, TRPO offers theoretical assurances on the monotonic improvement of the objective function, typically characterized by expected cumulative rewards or a corresponding performance metric.

```
Algorithm 1: Actor-Critic Algorithm
  Input : policy network parameter $\theta_0$, value network parameter $\phi_0$, actor learning
          rate $\alpha_\theta$, number of training epochs $K$
1 for $k \leftarrow 0$ to $K-1$ do
2     Sample a set of trajectories $\mathcal{D}$ using the policy $\pi_k$.
3     Calculate the return $G$ using the sampled set of trajectories $\mathcal{D}$:
          $$G_t = \sum_{i=t}^{\infty} \gamma^i r_i.$$
4     Estimate advantage function $A(s_t, a_t)$ based on value function $V_{\phi_k}$ using GAE.
5     Calculate the gradient $g_k$:
          $$g_k = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{\infty} A(s_t, a_t) \nabla \log \pi_\theta(a_t|s_t).$$
6     Update the policy network parameter $\theta_{k+1}$:
          $$\theta_{k+1} = \theta_k + \alpha_\theta g_k.$$
7     Update the value network parameter $\phi_{k+1}$ by performing gradient descent on
        the mean squared error:
          $$\frac{1}{N} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{\infty} (V_\phi(s_t) - G_t)^2.$$
```

TRPO updates the current policy π_k by solving the following constrained optimization problem:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} \mathbb{E}_{s \sim d^{\pi_k}, a \sim \pi}[A^{\pi_k}(s,a)]$$
$$\text{s.t. } \overline{D}_{KL}(\pi, \pi_k) \leq \delta$$

Here, the notation Π_θ represents the policy space, parameterized by θ. The term

d^{π_k} is defined as the discounted future state distribution, mathematically expressed as: $(1-\gamma)\sum_{t=0}^{\infty}\mathbf{P}(s_t = s|\pi)$. Meanwhile, $\overline{D}_{KL}(\pi,\pi_k)$ denotes the expected KL divergence between policies π and π_k, averaged over state distributions obtained from the current policy π_k. Formally, it is described as: $\mathbb{E}_{s\sim\pi_k}[D_{KL}(\pi(.|s)||\pi_k(.|s)]$. δ is the boundary of the trust region, which is a hyperparameter of the algorithm.

Practically, this optimization problem is solved by approximating the KL divergence using its second order expansion and obtain the fisher information matrix H. Consequently, the update direction is given by $\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$. Note that, the term $H^{-1}g$ is approximated by the conjugate gradient algorithm [19], since calculating H^{-1} is expensive. Because the series of approximations may result in an excessively large update step, which may cause a catastrophic performance decrease, a line search is subsequently employed to ensure both the improvement of the objective and the satisfaction to the constraints.

Since TRPO requires complex second order method, Schulman et al. [39] propose Proximal Policy Optimization (PPO), which updates using a new surrogate loss with multiple steps of minibatch update:

$$min(\frac{\pi(a_t|s_t)}{\pi_k(a_t|s_t)}A^{\pi_k}(s_t,a_t), clip(\frac{\pi(a_t|s_t)}{\pi_k(a_t|s_t)}, 1-\epsilon, 1+\epsilon)A^{\pi_k}(s_t,a_t))$$

where ϵ is a hyperparameter that regulates the divergence between the new policy and the old policy. The PPO algorithm achieves comparable or superior performance to the TRPO algorithm across various benchmarks.

3.3 Constrained Policy Optimization

The conventional policy optimization methods, such as PPO [39] and TRPO[37], primarily focus on maximizing the expected cumulative reward without explicitly considering safety constraints. While these methods can achieve impressive performance in a wide range of tasks, they lack mechanisms to ensure that the learned policies satisfy specified constraints. In safety-critical applications, such as autonomous driving, healthcare, and finance, an agent's behavior must remain within predefined safety bounds to prevent dangerous or harmful actions.

To overcome this limitation, Constrained Policy Optimization [2] has emerged as a promising approach. CPO introduces explicit constraints on the policy parameters to prevent policy updates that violate safety requirements. By incorporating constraints into the optimization process, CPO aims to strike a balance between maximizing the reward and adhering to the specified safety boundaries. In the following sections, we first provide the definition of Constrained Markov Decision Process and then describe CPO in details.

3.3.1 Constrained Markov Decision Process

A Constrained Markov Decision Process (CMDP) [3] can be thought of as an extension of the standard MDP, augmented with a set of cost functions $C_1, ..., C_m$. Each individual cost function, denoted as $C_i : S \times A \times S \to \mathbb{R}$, outputs the associated cost from a transition (s, a, s'). Correspondingly, each of these cost functions has an associated set of cost limits $d_1, ..., d_m$. Typically, constraints within CMDPs are categorized into two types: average and peak constraints. For the average constraints, the requirement is that the expected discounted sum of the policy's cost should not exceed the designated threshold d_i. Conversely, peak constraints enforce that none of the transitions should exceed the designated threshold d_i. In the following subsections, we focus on the Constrained Policy Optimization algorithm that ensures satisfaction of the average constraints.

3.3.2 Constrained Policy Optimization Algorithm

To ensure the satisfaction of the constraints while achieving the maximum reward, Constrained Policy Optimization (CPO) [2] has been proposed as a trust region method that both enjoy the monotonic policy improvement like TRPO and strike a balance between reward maximization and constraint satisfaction.

The optimization problem that CPO initially tries to solve is

$$\begin{aligned} \pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} & \, J(\pi) \\ \text{s.t.} \;\; & J_{C_i}(\pi) \le d_i, \quad i = 1, \dots, m \\ & D(\pi, \pi_k) \le \delta \end{aligned}$$

where $J(\pi)$ is the expected return of the policy π.

CPO derives a new policy improvement bound for the safety constraints and connects it to KL-divergence based on [23] and [37].

$$J_{C_i}(\pi') - J_{C_i}(\pi) \le \frac{1}{1-\gamma} \mathbf{E}_{s \sim d^\pi, a \sim \pi'} [A^\pi_{C_i}(s,a) + \frac{2\gamma \epsilon^{\pi'}_{C_i}}{1-\gamma} \sqrt{\frac{1}{2} \mathbf{E}_{s \sim d^\pi} [D_{KL}(\pi'(.|s) || \pi(.|s))]}$$

Combining with the trust region method, the optimization problem becomes

$$\begin{aligned}
&\pi_{k+1} = \arg\max_{\pi \in \Pi_\theta} \mathbf{E}_{s\sim d^{\pi_k}, a\sim\pi}[A^{\pi_k}(s,a)] \\
&\text{s.t. } J_{C_i}(\pi_k) + \frac{1}{1-\gamma}\mathbf{E}_{s\sim d^{\pi_k}, a\sim\pi}[A^{\pi_k}_{C_i}(s,a)] \leq d_i, \quad \forall i \\
&\quad\ \overline{D}_{KL}(\pi, \pi_k) \leq \delta
\end{aligned}$$

The CPO paper [2] also provides a theoretical guarantee of monotonic improvement of performance improvement of approximate satisfaction of constraints, which makes CPO a reliable method in solving the CMDP problem.

To solve the optimization problem and calculate the update step practically, similar to TRPO, CPO expands the KL-divergence constraint through its second-order Taylor expansion and linearizes the constraints on cost functions using their first-order Taylor expansions:

$$\begin{aligned}
&\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k) \\
&\text{s.t. } c_i + b_i^T(\theta - \theta_k) \leq 0 \quad i = 1, \ldots, m \\
&\quad\ \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta
\end{aligned}$$

where H is the Hessian of the KL-divergence, b_i is the gradient of $J_{C_i}(\pi_k)$, and $c_i = J_{C_i}(\pi_k) - d_i$. Subsequently, the optimization problem is solved through the application of duality.

It should be highlighted that the CPO paper offers an analytical solution through the utilization of the primal-dual method when there is only one cost function, i.e. $m = 1$.

Like TRPO, a line search is followed after computing the update step direction to ensure the constraints satisfaction and objective improvement.

3.4 Multi-agent Reinforcement Learning Methods

In the previous subsections, we provide a brief introduction to certain single-agent reinforcement learning algorithms. Nonetheless, the complexities inherent in numerous real-world situations demand interactions amongst multiple agents, which gives rises to the paradigm of Multi-Agent Reinforcement Learning (MARL) [43]. In the subsections that follow, we first define the Multi-agent Markov Decision Process and then outline the challenges associated with MARL. Subsequently, we present the multi-agent counterparts of the single-agent reinforcement learning algorithms discussed in the prior sections.

3.4.1 Multi-agent Markov Decision Process and Markov Games

The Multi-agent Markov Decision Process of n agents applies the following modifications to the MDP:

- The action space $\mathbf{A}$ is transformed into a set of action spaces $\{\mathbf{A}_i, i = 1, \ldots, n\}$.
- The domain of the transition function $\mathbf{P}$and the reward function $\mathcal{R}$ are modified to the incorporate the joint action space $\mathbf{A}_1 \times \ldots \times \mathbf{A}_n$.

A Markov game, also known as a stochastic game, is a type of game that is played in a sequence of stages. The players select actions based on policies and they receive a payoff that depends on the current stage of the game and the actions selected. The current stage and the actions selected also determine the probability of the next stage that the game will transit to following a transition function. The stage sequence can either be infinite or finite, and the total payoff of a player is usually calculated as the discounted sum of the payoff received at each stage.

3.4.2 Multi-agent Reinforcement Learning Challenges

Transitioning from single-agent reinforcement learning problem to multi-agent reinforcement learning problem is not trivial. One main issue people are facing is the non-stationary environment. From the point of view of one learning agent, the other learning agents' policy is evolving throughout the learning process. This results in non-stationarity in the environment for this particular learning agent, which makes it harder to converge to stable policies. Another challenge is the equilibrium selection problem, in multi agent games, multiple equilibria, especially in competitive scenarios, might exist, and determining which equilibrium the system will converge to, or should converge to, becomes a critical challenge. When agents engage in cooperative learning with a focus on optimizing a global reward, an additional challenge is the multi-agent credit assignment problem: Determining which agent is responsible for which part of a global reward or outcome can be non-trivial.

3.4.3 Multi-agent Trust Region Methods

For the policy gradient methods PPO and TRPO, considerable efforts have been made to adapt these methodologies to multi-agent contexts. One simple attempt leverages the effectiveness of independent learning to let the agents run PPO and TRPO independently, oblivious to the presence of other agents. Specifically, de Witt et al. [10] compare the performance of Independent PPO (IPPO) with QMIX [35] and independent Q learning on the StarCraft Multi-Agent Challenge (SMAC) benchmark. Their findings suggest that IPPO performed surprisingly better than the baselines. Another simple approach involves sharing the critic network among the agents, enabling them to learn a global value function. Yu et al. [42] designate this technique as Multi-Agent PPO (MAPPO), and illustrate that PPO-based methods (MAPPO, IPPO) perform comparable or superior to other methods including QMIX and MADDPG [32] on various benchmarks. However, both IPPO and MAPPO methods lose the theoretical assurance of monotonic policy improvement while adapting to multi-agent environment. In contrast, Kuba et al. [27] propose a novel multi-agent trust region optimization algorithm that extends the monotonic objective improvement property to multi-agent settings, and the resultant algorithms are named Heterogeneous-Agent TRPO (HATRPO) and Heterogeneous-Agent PPO (HAPPO). They also verified that the HATRPO and HAPPO methods outperform IPPO, MAPPO and MADDPG on many benchmarks. Firstly, the simple advantage function in Section 3.1.2 is expanded to handle multiple agents. The authors propose a multi-agent advantage decomposition formula for cooperative Markov games as follows

$$A_\pi^{i_{1:m}}(s, \mathbf{a}^{i_{1:m}}) = \sum_{j=1}^{m} A_\pi^{i_j}(s, \mathbf{a}^{i_{1:j-1}}, a^{i_j})$$

where $i_{1:m}$ can be any subset of agents. This formula suggests a way to ensure performance improvement: given n agents following an arbitrary update sequence $i_{1:n}$, and under the condition that the first agent takes an action a^{i_1} that ensures $A^{i_1}(s, a^{i_1}) > 0$, and every succeeding agent m ensures $A^{i_m}(s, \mathbf{a}_{i_{1:m-1}}, a^{i_m}) > 0$, by applying the above formula, $A_\pi(s, \mathbf{a}) > 0$ is guaranteed.

The authors extend this formula to the multi-agent TRPO and PPO algorithms. In other words, similar to MATRPO or MAPPO, a centralized value network is shared across all agents. After the collection of a sample batch, a global advantage function $A(s, \mathbf{a})$ is calculated. During each update of the algorithm, agents follow a randomized permutation sequence for updating. The first agent i_1 employs $A(s, \mathbf{a})$ to calculate the gradient g^{i_1} in HATRPO or the clip objective in HAPPO. After an agent i_m finishes updating the current policy parameter $\theta_k^{i_m}$ to $\theta_{k+1}^{i_m}$, the advantage term is multiplied by the factor

$$\frac{\pi^{i_m}_{\theta^{i_m}_{k+1}}(a^{i_m}|o^{i_m})}{\pi^{i_m}_{\theta^{i_m}_{k}}(a^{i_m}|o^{i_m})},$$

where a^{i_m} denotes the action executed by agent i_m given the observation o^{i_m} during the sample collection phase. The succeeding agent i_{m+1} will then employ this modified advantage term to continue the update. The details of the HATRPO algorithm [27] is shown in Algorithm 2.

3.4.4 Multi-agent Constrained Policy Optimization

To extend the CPO algorithm to the multi-agent domain, Gu et al. [18] first define a constrained Markov game, which is the Multi-agent MDP augmented with cost functions and corresponding cost limits. Subsequently, they introduce the Multi-Agent Constrained Policy Optimization (MACPO) algorithm. This approach combines the CPO algorithm with the HATRPO update paradigm. Furthermore, they have proved that the resultant MACPO algorithm attains both the monotonic objective improvement guarantee and constraints satisfaction guarantee.

Put simply, combining the HATRPO and CPO algorithms is straightforward. One merely needs to replace the TRPO update within the HATRPO algorithm with the CPO update.

Algorithm 2: HATRPO Algorithm

```
Input : number of agent $n$, policy network parameters $\{\theta_0^i, \forall i \in n\}$, global value network
        parameters $\phi_0$, replay buffer $\mathcal{B}$, actor learning rate $\alpha_\theta$, number of training epochs $K$,
        batch size $B$, episode length $T$, maximum number of line search steps $L$, KL divergence
        threshold $\delta$
1  for $k \leftarrow 0$ to $K-1$ do
2      Sample a set of trajectories $\mathcal{D}$ using the joint policy $\boldsymbol{\pi_k} = (\pi^1_{\theta^1_k}, \ldots, \pi^n_{\theta^n_k})$.
3      Push transitions $\{ s_t^i, a_t^i, s_{t+1}^i, r_t^i \}$ into $\mathcal{B}$.
4      Sample a random minibatch of $B$ transitions from $\mathcal{B}$.
5      Estimate global advantage function $A(s, \mathbf{a})$ based on $\mathcal{D}$ and the gloabl value function $V_\phi$.
         using GAE.
6      Calculate the return $G$.
7      Draw a random permutation of agents $i_{1:n}$.
8      Set $M^{i_1}(s, \mathbf{a}) = A_{\phi_k}(s, \mathbf{a})$.
9      for agent $i_m = i_1, \ldots, i_n$ do
10         Calculate the gradient $g_k^{i_m}$:
               $$g_k^{i_m} = \frac{1}{B} \sum_{b=1}^{B} \sum_{t=0}^{T} M^{i_{1:m}}(s_t, \mathbf{a}_t) \nabla_{\theta_k^{i_m}} \pi^{i_m}_{\theta_k^{i_m}}(a_t^{i_m} | s_t^{i_m}).$$
11         Use the conjugate gradient algorithm to compute $x_k^{i_m} = (H_k^{i_m})^{-1} g_k^{i_m}$, where $H_k^{i_m}$ is the
             average KL divergence:
               $$H_k^{i_m} = \frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T} D_{KL}(\pi^{i_m}_{\theta_k^{i_m}}(\cdot | s_t^{i_m}) | \pi^{i_m}_{\theta^{i_m}}(\cdot | s_t^{i_m})).$$
12         Calulate the step size $\beta_k^{i_m}$:
               $$\beta_k^{i_m} = \sqrt{\frac{2\delta}{(x_k^{i_m})^T H_k^{i_m} x_k^{i_m}}}.$$
             Update agent $i_m$'s policy parameter $\theta_{k+1}^{i_m} = \theta_k^{i_m} + \alpha_\theta^j \beta_k^{i_m} x_k^{i_m}$, where $j \in \{0, 1, \ldots, L\}$ is
             the smallest such $j$ which improves the sample loss, found by the backtracking line
             search.
13         Compute $M^{i_{1:m+1}}(s, \mathbf{a}) = \frac{\pi^{i_m}_{\theta_k^{i_m}}(a^{i_m} | s^{i_m})}{\pi^{i_m}_{\theta_{k+1}^{i_m}}(a^{i_m} | s^{i_m})} M^{i_{1:m}}(s, \mathbf{a})$ /*Unless m = n*
14     Update the global value network parameter $\phi_{k+1}$ by performing gradient descent on the
         mean squared error:
               $$\frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T} (V_\phi(s_t) - G_t)^2$$
```

Chapter 4

Simulation Environment

This chapter describes the simulation environment used for autonomous vehicle evaluation. The simulation environment used in reinforcement learning typically follows a similar API model as *gym* [4] published by OpenAI. To the best of our knowledge, there doesn't exist simulation environment specifically designed to attack a black-box autonomous vehicle under test, but there are some open-sourced gym-liked environment to train the autonomous vehicle to learn how to drive. We design a new simulation environment *highway-attack-env* on top of one of the minimalist environments for decision making in autonomous driving, *highway-env* [28]. In the following subsections, we describes the overarching framework for autonomous vehicle stress testing, termination conditions, observation space, action space, reward structure and cost structure. Additionally, it's essential to emphasize that our devised environment have the flexibility to be configured to be either multi-agent or single-agent environment.

4.1 Framework for Autonomous Vehicle Stress Testing

Within the context of autonomous vehicle stress testing, it comprises two types of agents: the attacker agent and the target agent. The attacker agents, under the control of reinforcement learning algorithms, learn behaviors aimed at instigating accidents involving the target agent. While the target agent is treated as a black box which receives observations of the current state in the environment and generates corresponding actions as outputs. The experiment setup employed in our study contains a single target agent, while the

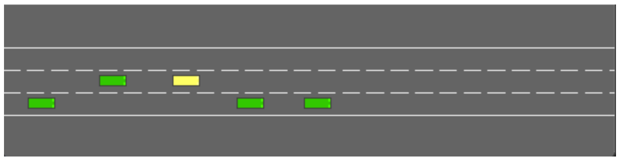

Figure 4.1: Example environment starting positions of the agents, where the target agent is highlighted in yellow, while the attacker agents are colored in green.

number of attacker agents is not limited and could be multiple. In this thesis, we limit the number of attacker agents to four, based on the reasoning that it's uncommon for more than four attacker agents to be involved in a single accident with the target agent. On the initialization of the environment, both the attacker agents and the target agent start from some starting positions. Starting positions can be set as either fixed or randomized. The environment we employed is a two-dimensional three-lane highway environment of infinite length. An example of this configuration is depicted in 4.1.

4.2 Termination Conditions and Absorbing State

At the end of an episode, the environment sends a termination signal. In our designed environment for single-agent reinforcement learning algorithms, there are two termination conditions:

- The target agent is involved in an accident.
- The time limit of the episode is reached and the target agent remains collision-free throughout the episode.

Note that, while the attacker agents may collide with each other during the episode, such collisions do not trigger the termination condition for the single-agent environment. Nevertheless, these collisions will result in the involved attacker agents being immobilized at the collision position.

In our designed multi-agent environment, the termination signal is issued agent-wise, such that when an attacker agent receives the termination signal, it should stop learning. Therefore, other than the two termination conditions in single-agent environment, which will send termination signal to all attacker agents, the termination signal will also be sent to the corresponding attacker agents when they collide with each other. To summarize, the termination conditions for multi-agent environment are as follows:

- The attacker agent is involved in any accident.
- The time limit of the episode reached and the target agent remains collision-free throughout the episode.

Similarly, when attacker agents collide with each other, they will enter an absorbing state, where the subsequent actions will no longer take an effect on these attacker agents.

$$\mathbf{P}(\hat{s}|\hat{s}, a) = 1, \forall a \in \mathbf{A}$$

where $\hat{s}$ denotes the absorbing state.

4.3 Observation Space

For the observation space, we adopt the same features used in *highway-env* [28]. The state for the ith attacker agent s_i and target agent s_{target} contains five features: $[\mathcal{I}, \mathbf{x}, \mathbf{y}, \mathbf{v_x}, \mathbf{v_y}]$ where

- $\mathcal{I}$ is a Boolean value indicates whether the agent exist in the environment or not.
- $\mathbf{x}$ and $\mathbf{y}$ describes the x and y coordinates of the location of the agent in the environment.
- $\mathbf{v_x}$ and $\mathbf{v_y}$ describes the velocity of the agent on x and y axis.

In the single-agent environment, the state vectors of the attacker agents are centered around the target agent. And the observation contains information about all agents in the environment: $s = [s_{target}, s_1, s_2, ..., s_n]$.

While in the multi-agent environment, every attacker agent receive different observations. When using a simple design where each agent is assigned a constant index in the observation, every agent invariably receives identical observations. This leads to challenges in self-identification for the agent, making it difficult to adapt and optimize based on its specific needs. Instead, We rearrange the sequence of the agents' states in the observation, so that the first state vector is always the ith attacker agent itself and the second is always the state vector of the target agent. Additionally, the values are centered around the ith attacker agent itself. This measure guarantees unique and tailored observations for each individual attacker agents, which is crucial to the learning of disparate policies.

4.4 Action Space

For the action space, we adopt the same *DescreteMeta* action space as in *highway-env* [28] for simplicity. Also, since the target agent is trained with *DescreteMeta* action space, it's fair to use the actions from the same action space to attack it. There are five available actions in the *DescreteMeta* action space, as depicted in 4.1

Table 4.1: DiscreteMeta action space

Action	**Behavior**
LaneLeft	If the vehicle is not on the leftmost lane, it will change lane to the left following a predefined speed profile and trajectory.
Keep	The vehicle will not change lane and maintain its current velocity.
LaneRight	If the vehicle is not on the rightmost lane, it will change lane to to the right following a predefined speed profile and trajectory.
Faster	The vehicle will accelerate if its current speed is less than the predefined maximum speed.
Slower	The vehicle will decelerate if its current speed is greater than the predefined minimum speed.

4.5 Reward

This section describes the reward structure in our designed environment. The reward calculation is performed on an agent-wise basis in both multi-agent and single-agent environments. However, in the single-agent environment the sum of all attacker agents' reward is returned as a single total reward.

The primary objective behind the reward design is to cause accidents involving the target agent, and to maintain a higher number of controllable attacker agents, so the attacker agents are trained to avoid colliding with each other. To achieve this, a positive reward of 2.5 is issued for every attacker agent when the target agent is involved in a collision, making the total reward received on target agent's collision 10, because all attacker agent could contribute to the collision of target agent, even though they are not the one directly collides with it. A negative reward of -2.5 is assigned to each collided attacker agent when they collide with each other. Note that, we choose not to incorporate an additional time penalty, as we consider accidents discovered at various time steps to be equally useful.

A crucial aspect worth emphasizing is that since we want to exploit the full potential of reinforcement learning in exploring diverse ways of causing accidents with the target agent, we did not explicitly specify how accidents should be generated in the reward function. Instead, only a sparse positive reward is assigned when the accident involves the target agent. By adopting this approach, the attacker agents are encouraged to discover innovative and effective strategies through exploration, leading to a more comprehensive understanding of the environment and fostering diverse accident-causing behaviors throughout the learning process.

4.6 Cost

To regulate the aggressiveness of the attacker agents when generating accidents, we use the cost to place a penalty on aggressive or unrealistic behaviors. We define six behaviors that assign a cost for the attacker agent, as depicted in Table 4.2

Table 4.2: Aggressive or unrealistic behavior that will issue cost

Behavior	**Cost**
If the vehicle is on the leftmost lane, but it is going to perform a LaneLeft action.	*invalid action cost*
If the vehicle is on the rightmost lane, but it is going to perform a LaneRight action.	*invalid action cost*
If there is an obstacle (could be other vehicles) in front of the vehicle within 12 meters, and it performs a Faster action.	*close vehicle cost*
If there is an obstacle (could be other vehicles) behind the vehicle within 12 meters, and it performs a Slower action.	*close vehicle cost*
When the vehicle is going to perform a LaneLeft action, but there is another vehicle on the lane it is going to change to or there is another vehicle in front of it, and their absolute distance is within 12 meters.	*close vehicle cost*
When the vehicle is going to perform a LaneRight action, but there is another vehicle on the lane it is going to change to or there is another vehicle in front of it, and their absolute distance is within 12 meters.	*close vehicle cost*

Similar to the reward computation, the cost calculation is performed on an agent-wise basis in both the multi-agent and single-agent environment. And the sum of all individual attacker agents' costs is returned in the single-agent environment.

Notably, the cost structure defined above are fairly simple, but the last four behaviors capture some unsafe driving behavior, such as changing lanes without checking the blind-spot or mirror, deliberately rear-end the front vehicle . . ., so we can use the occurrence of these events to measure the aggressiveness of the attacker agents. The actions in the first two behaviors are characterized as invalid actions, and the simulator will perform a Keep action instead if an invalid action is received. By assigning cost on these invalid actions, we discourage the attacker agents from performing these actions in the simulation.

Chapter 5

Methodology and Experiments

This chapter initially outlines the design of the target agents, serving to demonstrate the efficacy of the algorithms. Following this, we enumerate the algorithms selected for comparative analysis and articulate the research questions we aim to address. Subsequently, we provide a detailed description of the experiments conducted and analysed the results.

5.1 Design of the Target Agent

To verify the effectiveness of an autonomous vehicle stress testing system, it is imperative to employ complex target vehicles to enhance the challenge in simulating realistic accidents. Given the time constraints and the scope of this thesis, we opted not to design a normal autonomous driving system with advanced obstacle avoidance capabilities within the *highway-env*. Instead, to provide a baseline for comparison that will be challenge to find weaknesses in, we proposed a *perfect target* for our *highway-attack-env*. We reveal the forthcoming actions of the attacker agent to the target agent. This allows the target agent to simulate an additional step and select an action that will avert an accident based on the attacker agents' next move. In scenarios where no actions by the target result in a collision-free outcome, the target agent defaults to the *Keep* action. Conversely, when multiple actions by the target ensure a collision-free outcome, we instruct the target agent to adopt a cautious lane-following strategy, taking into account the position and speed of both preceding and following attacker agents in the same lane.

Because the *perfect target* is too good at avoiding collisions with attacker agents, we introduced designed vulnerabilities to make better comparisons of different algorithms. We incorporated two different vulnerabilities, leading to the creation of *target0* and *target1*. To maintain the degree of challenge in identifying realistic accidents, both *target0* and *target1* will default to the action chosen by the *perfect target* when the vulnerability is not triggered. Detailed descriptions of these vulnerabilities are as follows:

- target0: The first vulnerability we introduced necessitates the coordination of two attacker agents. For the vulnerability to be triggered, both attacker agents must drive adjacent to *target0* at a close proximity. Upon detection of this configuration, *target0* will override the action determined by the *perfect target* and execute the *LaneLeft* action, resulting in a collision with the attacker agent to its left.

- target1: The second vulnerability we introduced necessitates the coordination of three attacker agents and builds upon the configuration of the first vulnerability. To activate this vulnerability, apart from the two attacker agents driving adjacent to the target at a close proximity, a third attacker agent must be positioned directly in front of the target agent in the same lane, also within a proximate distance. When this configuration is detected by *target1*, it will override the action determined by the *perfect target* and execute the *LaneLeft* action, resulting in a collision with the attacker agent to its left.

The purposefully engineered target agents afford us the ability to conduct experiments in a more controlled fashion. Understanding the known vulnerabilities of these agents allows for definitive verification when such weaknesses are detected. In the absence of this controlled design, we would be restricted to statistical arguments regarding the likelihood of uncovering potential vulnerabilities.

5.2 Comparison of algorithms

Similar to the framework presented in the “Adaptive Stress Testing of Autonomous Vehicle” paper[25], the autonomous vehicle stress testing problem can be characterized as a Markov Decision Process. Drawing from the notion that individual optimization by each attacker agent can promote coordinated behavior among them to unveil more complicated vulnerabilities in the target system, we propose to formulate the autonomous vehicle stress testing problem as a multi-agent Markov Decision Process given in Section 3.4.1.

To foster the generation of more insightful accidents, we do not want the attacker agents to take actions that are too aggressive, so we propose to regulate the aggressiveness of the attacker agents' actions. A straightforward strategy to regulate aggressiveness is to remove aggressive actions from the action set at each time step. However, we argue that such a method enforces an overall aggressiveness level of zero in the whole episode. It's important to consider that accidents with a non-zero value of aggressiveness might offer valuable insights as well, particularly when the cost structure exhibits greater complexity. As an alternative, we propose to frame the autonomous vehicle stress testing problem as a constrained Markov game, as detailed in [18] and discussed in Section 3.4.4. Within this framework, we employ cost functions to evaluate the aggressiveness of the actions. Note that within a CMDP framework, constraints can be managed in two different manners:

- Peak Constraint: Ensures the cost associated with any individual transaction does not exceed the threshold d.
- Average Constraint: Ensures the mean cost across an episode remains below the threshold d.

Typically, the peak constraint is integrated within the reinforcement learning algorithm by deducting the cost value of each step from its corresponding reward—a technique that can be seamlessly extended to multi-agent reinforcement learning paradigms. On the other hand, the state-of-the-art approach to addressing the average constraint is through the Constraint Policy Optimization (CPO) [2] method. Additionally, a multi-agent version, Multi-Agent Constrained Policy Optimizaiton (MACPO) [18], has also been proposed in recent literature. We have given a short introduction to both CPO and MACPO in Chapter 3. It's worth mentioning that the presence of multiple cost functions introduces extra layers of complexity in solving the optimization problem in CPO and MACPO. Given this difficulty, our approach aggregates the cost values incurred by a singular transition, thereby reducing it to a single unified cost function for simplification. Consequently, the analytical solution derived in [2] becomes applicable.

We conduct a series of experiments to address the subsequent research questions:

- Do MARL algorithms outperform single-agent RL algorithms in autonomous vehicle stress testing?
- Does the integration of cost values contribute to the generation of more realistic accidents?
- In the context of producing realistic accidents, is the CPO algorithm that optimizes average constraints better than treating peak constraints as a penalty term in the reward function?

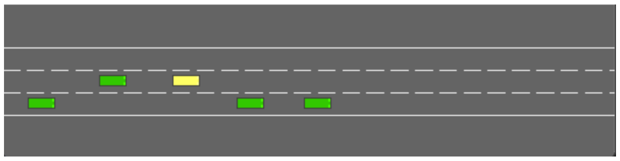

Figure 5.1: The fixed starting positions of the agents, where the target agent is highlighted in yellow, while the attacker agents are colored in green.

- Will the magnitude of the cost values affect the results in this process?

In the following subsections we describe each experiment in detail.

5.3 Experiments

Since all the algorithms under consideration employ the actor-critic framework, we standardiz the network architecture across them. Specifically, both the actor and critic networks consist of a single hidden layer of 128 neurons, with the tanh function serving as the activation function. The critic network yields a singular output, representing the value of the given state. While the actor network outputs the logits for all available actions the agent might take. Note that for single-agent algorithms, due to the presence of only one policy network, the actor network outputs the logits for every action across all learning agents.

In the subsequent experiments, we perform training using TRPO, CPO, HATRPO, and MACPO algorithms under varying conditions to answer the above research questions.

5.3.1 Experiments on a Fixed Starting Position

In this subsection, we train the algorithms within a fixed starting scenario. This specific scenario has been validated to permit the triggering of both vulnerabilities that we introduced. The hyperparameters we use in all our experiments are summarized in Table 5.1. Additionaly, we have selected the *close vehicle cost* value to be 20. The rationale behind this choice is elaborated upon in Section 5.3.2. This approach facilitates a clearer illustration of an algorithm's learning trajectory when training on a specific starting scenario. A representation of this starting scenario can be viewed in Figure 5.1.

Table 5.1: List of key hyperparameters

Hyperparameter	Value
Number of training episodes	100000
Batch size	2000
Number of minibatches	4
Policy network update epochs	4
Value network update epochs	10
Discount factor	0.99
Maximum KL divergence	0.2
Maximum number of line search steps	10
line search fraction	0.5
Value network learning rate	2.5e-4

To elucidate the impact of incorporating cost values into the AV stress testing problem, we compare the experimental outcomes generated both with and without integrating cost in the learning process. We start our training with the TRPO and HATRPO algorithms within the aforementioned fixed starting scenario. Note that these two algorithms omit the cost received from the environment, relying solely on the return as their training signal. The results of this experiment are depicted in Figure 5.2 and Figure 5.3. It's important to highlight that, although the cost is excluded in the algorithm, we still plot it in the visualization. Additionally, we plot the difference between the reward and the cost, offering insights into the aggressiveness of the resulting accidents. Notably, a value of 10 in this plot suggests an accident that involves the target agent, but bears no cost. From the data, it's clear that both the TRPO and HATRPO algorithms converge to a stable policy. Both algorithms attain a maximum return of 10, indicating that the policies can learn an accident involving the target agent. However, the cost associated with these accidents is not constrained and substantially exceeds 0. This suggests that the accidents generated by these policies lack realism, which is intuitive as the algorithms maximize the return without consideration of the cost.

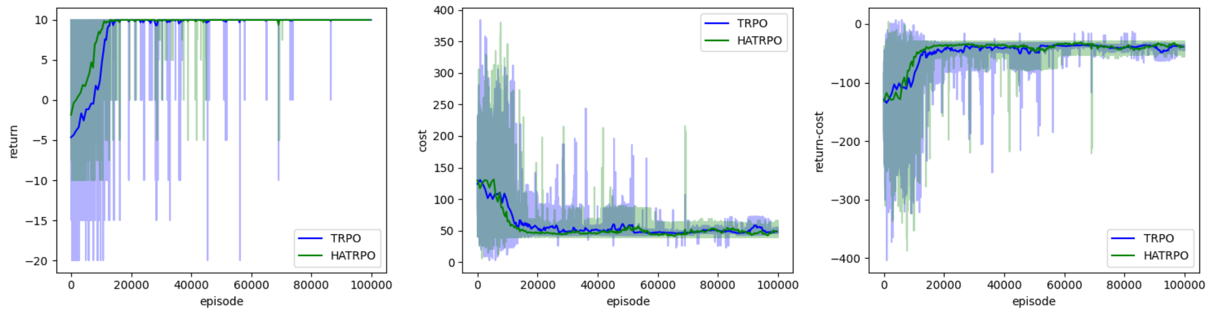

Figure 5.2: The training result of TRPO and HATRPO on the fixed starting scenario using *target0*.

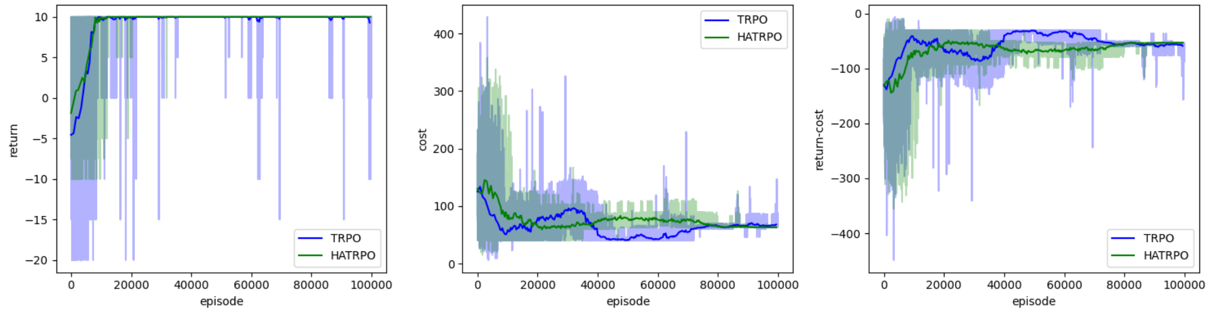

Figure 5.3: The training result of TRPO and HATRPO on the fixed starting scenario using *target1* .

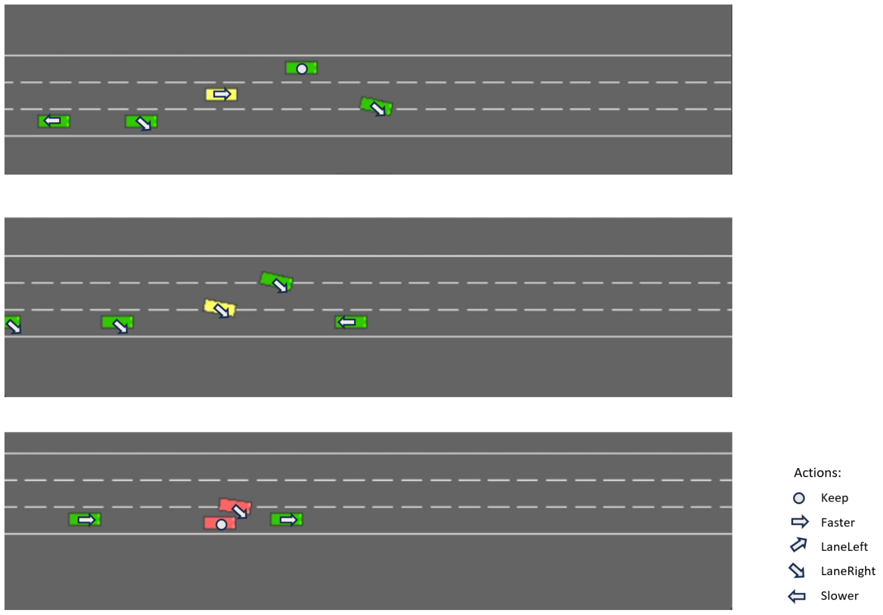

Figure 5.4: The unrealistic accident found by TRPO algorithm on *target0*

Upon evaluating the converged policies, it is also evident that the accidents found by these two algorithms are not insightful. Specifically, the attacker agents appear to straightforwardly drive towards the target agent. An illustrative example of this behavior can be seen in Figure 5.4, which shows an accident detected by the single-agent TRPO algorithm trained on *target0*. Such accidents are of limited utility, as the blame doesn't lie with the target agent. Consequently, they provide less opportunities for enhancing the target agent's algorithm.

In the next set of experiments, we integrate the cost component into the algorithms. For TRPO and HATRPO, we deducted the cost from the reward of each step, resulting in the newly defined algorithms TRPO-penalty and HATRPO-penalty, in which only the reward signal is modified. For both CPO and MACPO, we set the cost limit as zero, aligning our preference towards zero-cost accidents. It is worth mentioning that with the CPO and MACPO approaches, there exist the flexibility to adjust the cost limit to different values, should one wish to search for accidents with a specific value of aggressiveness. We train TRPO-penalty, HATRPO-penalty, CPO, and MACPO algorithms using the same fixed starting scenario and the two vulnerable target agents we previously defined. The training proceeds in accordance with the same set of hyperparameters outlined in Table 5.1. The *close vehicle cost* value is selected to be 20 as well. The results of this set of experiments are illustrated in Figure 5.5 and Figure 5.6. Note that, TRPO-penalty and HATRPO-penalty methods optimize the return-minus-cost value, while CPO and MACPO methods optimize return and cost separately. We present the progression of all three attributes for better comparison.

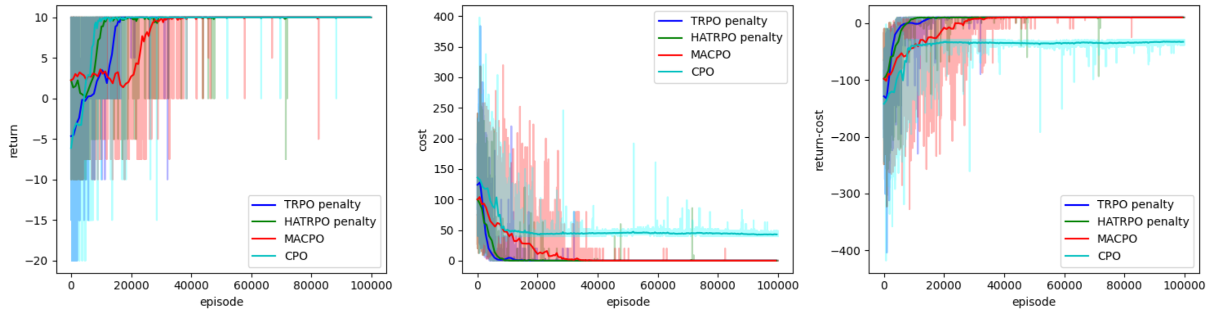

Figure 5.5: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on fixed starting scenario using *target0*.

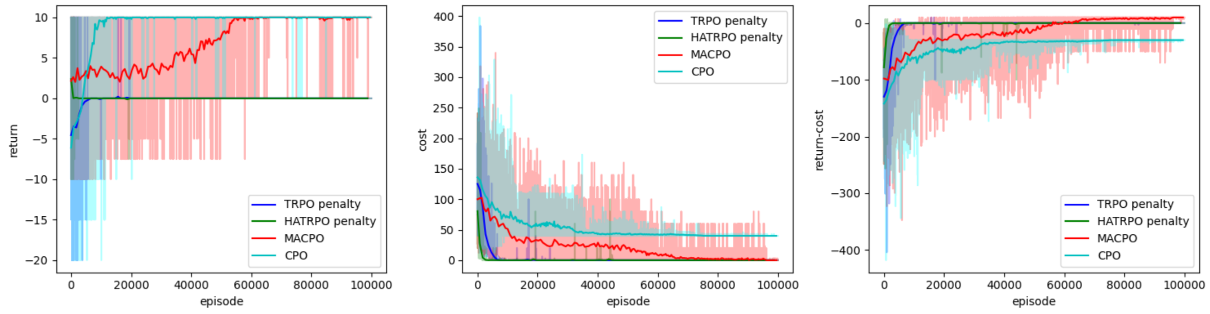

Figure 5.6: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on fixed starting scenario using *target1*.

For *target0*, all the algorithms, except for CPO, converge to similar sequences of actions that result in a collision with the target agent with no cost. The resulting accident can be viewed in Figure 5.7. In contrast, when considering *target1*, only the MACPO algorithm discovere a zero-cost approach to cause accidents involving the target agent. The other algorithms settle on policies that neither incurred cost nor resulted in accidents, with the only exception being the CPO algorithm, which converge to high cost accidents. Regarding the rate of convergence measured by the number of episodes sampled, we observe that for the less challenging target, *target0*, HATRPO-penalty reach convergence most rapidly, whereas the other three algorithms exhibit comparable rate of convergence, with MACPO being slower. For the more challenging target, *target1*, a similar pattern is observed. However, an exception is the MACPO algorithm, which has the slowest convergence rate but ultimately converg to the optimal accidents. When comparing the actual clock time required for each update, constrained policy optimization methods consume more time compared to the penalty methods because of the separate step of cost optimization. Additionally, multi-agent methods are more time-consuming than single-agent methods due to the necessity to optimize more networks. However, we would like to underscore that multi-agent methods offer the potential of distributed computing across different computing units. This capability has the potential to offset the efficiencies of single-agent approaches in terms of update speed.

We argue that the observed performance difference could stem from the inherent nature of constrained policy optimization methods (CPO and MACPO) undertaking multi-objective optimization. These methods explicitly compute the gradients of both the return and the cost. Even though this can sometimes make convergence more challenging because of possible conflicts between the two directions, it helps in avoiding premature convergence. Conversely, the penalty methods engage in single-objective optimization, focusing solely on the gradient of the net value of return-minus-cost. This approach can be more susceptible to convergences to local optima, particularly in scenarios where the increase in return mitigate the reduction in cost.

Furthermore, we want to emphasize that multi-agency plays an important role in finding the vulnerabilities we introduced. When individual agents optimize independently yet are guided by a joint value function, coordinated behaviors are more likely to emerge. Its effectiveness is evident in the performance difference between CPO and MACPO methods: While the CPO method quickly learned to cause accidents, it struggled to reduce costs. On the other hand, MACPO might take longer to learn how to cause accidents, but it managed to lower the cost of episodes.

The accident found by MACPO is depicted in Figure 5.8. Both of the accidents found exploit the vulnerabilities we introduced, which shows the effectiveness of our suggested

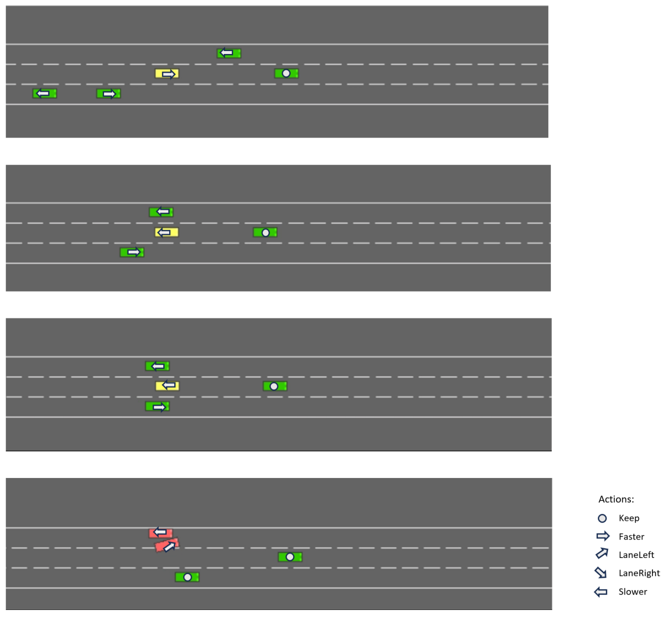

Figure 5.7: The zero cost accident involving *target0* found by all the algorithms, except for CPO.

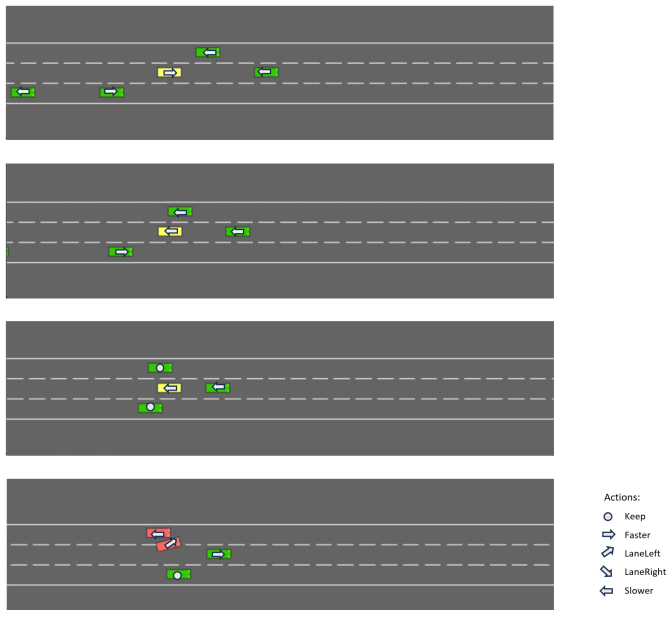

Figure 5.8: The zero cost accident involving *target1* found by MACPO.

target agent design.

5.3.2 Experiments on Random Starting Positions

In previous experiments, our focus is on various algorithms' ability in producing insightful accidents from a fixed starting position. In this section, we introduce randomness to the starting position by allowing attacker agents to initiate from random lanes, while the target agent always starts sandwiched between the attacker agents in the central lane. Given that each attacker agent can choose from three lanes and there are four attacker agents, this results in a total of $3^4 = 81$ unique starting scenarios.

We train the algorithms with these conditions to encourage the development of more general policies. Furthermore, we also perform a grid search on the magnitude of the *close vehicle cost* value within the range of $[20, 15, 10, 5]$. We maintain the *invalid action cost* value at a constant value of 3, since the *close vehicle cost* value has a more direct impact on the accident's associated cost. The training results are shown in Figure 5.9 to Figure 5.16

For a clearer depiction of performance difference, we test the trained policies within the environment. When an episode yields both zero cost and full reward outcomes for a particular starting scenario, that scenario is considered “solved”. We evaluate the trained policies with 2000 episodes, and once a starting scenario is “solved”, it will not be generated again. We count the number of “solved” scenarios for each algorithm after 2000 episode, and the results of this analysis can be found in Table 5.2.

Table 5.2: Number of “solved” starting scenarios after 2000 testing episodes

	Target0				*Target1*			
	cost 5	cost 10	cost 15	cost 20	cost 5	cost 10	cost 15	cost 20
MACPO	43	27	26	**54**	12	28	27	**33**
HATRPO-penalty	21	46	53	27	27	27	27	27
CPO	0	0	0	0	0	0	0	0
TRPO-penalty	27	0	0	0	27	27	0	0

From our collected data, MACPO stands out as the most successful algorithm in resolving the most starting scenarios for both *target0* and *target1*, particularly when trained using the maximum *close vehicle cost* value set at 20. Specifically for the relatively simpler *target0*, MACPO and HATRPO-penalty perform quite similarly. However, when dealing with the more challenging *target1*, MACPO surpasses HATRPO-penalty by successfully solving six additional starting scenarios. This aligns with our discussion in the previous section. Additionally, despite only doubling the training steps from the fixed starting scenario, the multi-agent algorithms learn many more starting scenarios. This indicates the capability of multi-agent algorithms in efficiently learning more generalized policies compared to their single-agent counterparts.

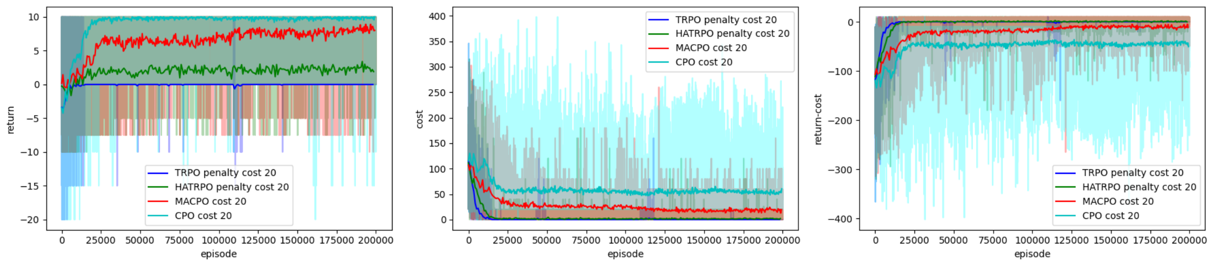

Figure 5.9: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target0* with *close vehicle cost* equals 20.

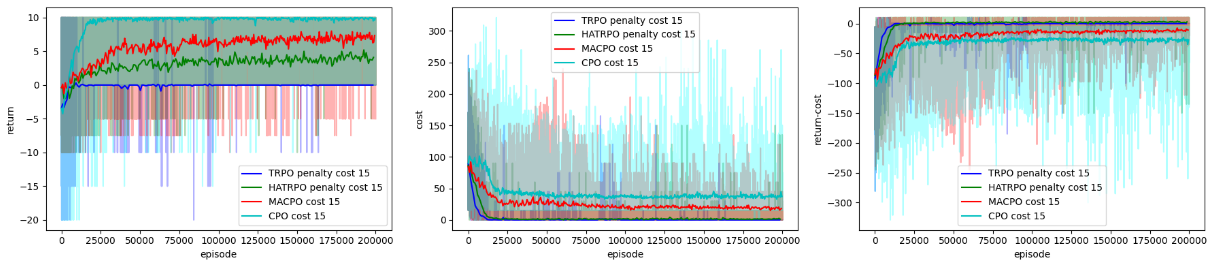

Figure 5.10: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target0* with *close vehicle cost* equals 15.

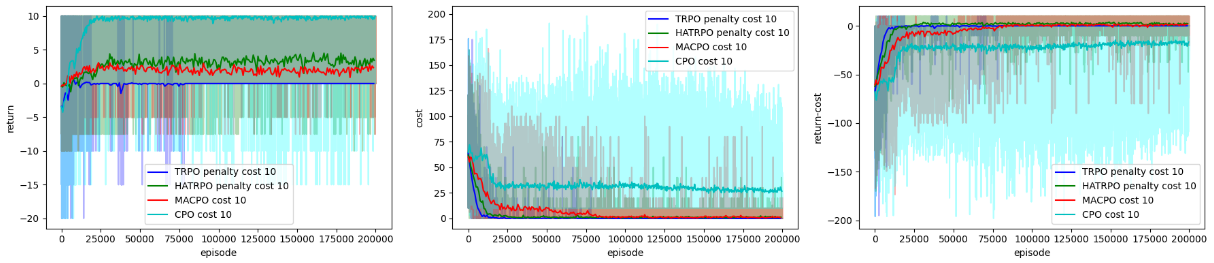

Figure 5.11: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target0* with *close vehicle cost* equals 10.

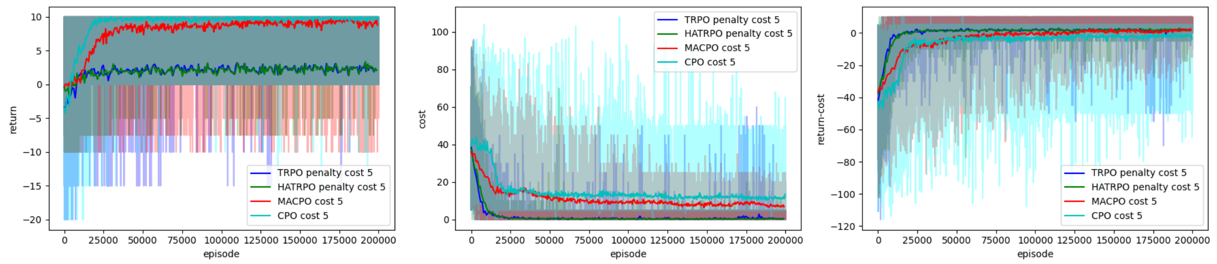

Figure 5.12: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target0* with *close vehicle cost* equals 5.

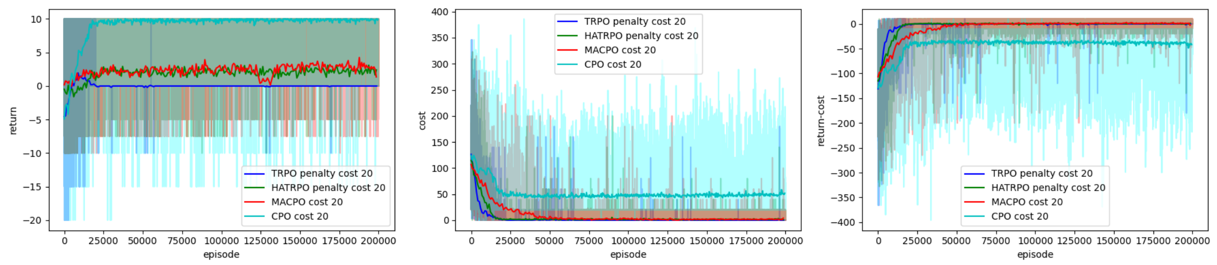

Figure 5.13: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target1* with *close vehicle cost* equals 20.

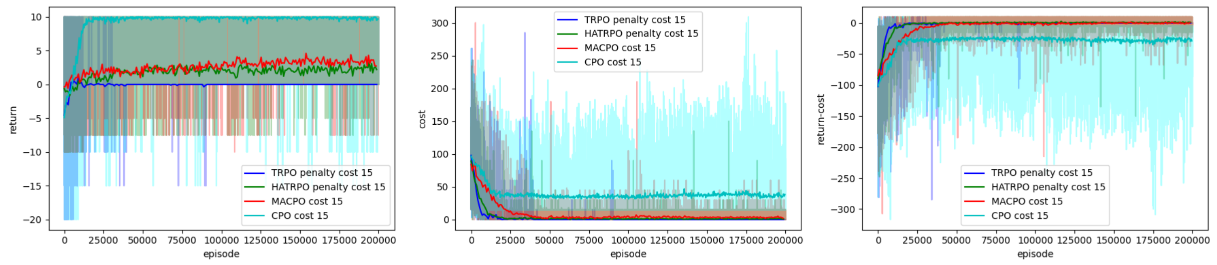

Figure 5.14: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target1* with *close vehicle cost* equals 15.

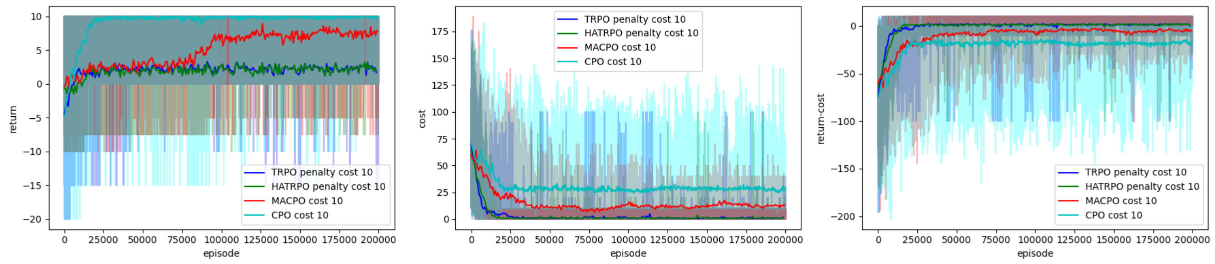

Figure 5.15: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target1* with *close vehicle cost* equals 10.

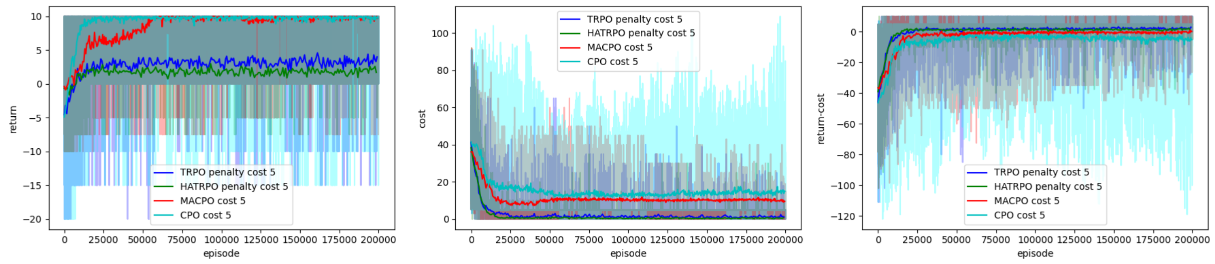

Figure 5.16: The training result of TRPO-penalty, HATRPO-penalty, CPO, MACPO on random starting scenario using *target1* with *close vehicle cost* equals 5.

We want to point out the existence of two local optima that lead to zero “solved” starting scenarios. Agents might easily become trapped in these optima:

- Local Optimum 1: the agent neither incurs cost nor causes accidents with the target agent.
- Local Optimum 2: the agent execute one high-cost action, leading to collisions with the target agent.

Analyzing the TRPO-penalty result specifically, at cost values of 15 and 20, Local Optimum 1 is more advantageous than Local Optimum 2 in terms of the return-minus-cost value. The TRPO-penalty agent cannot escape from Local Optimum 1, resulting in zero ”solved” initial scenarios. Conversely, for a cost value of 5, Local Optimum 2 is more advantageous than Local Optimum 1. We argue that Local Optimum 2 is easier to escape, eventually leading towards better local optima, where some zero-cost accidents can be found. This argument is supported by the data in Table 5.2: with a cost of 5, the TRPO-penalty approach resolved 27 starting scenarios when attacking either target. When the cost stands at 10, the distinction between both optima blurs, offering the same return-minus-cost value. As shown in Table 5.2 for attacking target 0 at cost 10, TRPO-penalty solves none of the scenarios, while for attacking target 1 at the same cost, it solves 27 scenarios.

It is evident in Table 5.2 that HATRPO-penalty can always solve a considerable amount of starting scenarios, which indicates escaping of these local optima, unlike TRPO-penalty. This shows that it’s hard for a single-agent algorithm to decompose the gradient and find the appropriate update direction for each agent, leading the algorithm to get trapped in a local optimum. The introduction of multi-agency helps in escaping local optima and finding the vulnerabilities in AVs. This advantage may also stem from the increased computational workload undertaken by multi-agent algorithms as compared to their single-agent counterparts.

Moerover, the CPO algorithm always converges to policies that generate accidents with non-zero cost, indicating an inability to sufficiently reduce the cost in CPO. This can also be attributed to the difficulty in single-agent algorithms to find the appropriate update direction for each agent.

TRPO-penalty’s data in Table 5.2 shows that it is sensitive to the choice of the penalty value, which aligns with the statement in [2]. However, we couldn’t determine the precise influence of the *close vehicle cost* value on other algorithms. Nevertheless, we can observe that the MACPO method benefits from the highest *close vehicle cost* value. We recommend conducting a grid search on these cost values for future usage of this benchmark.

Note that when the objective is to search for insightful accidents in a specific starting scenario, the reinforcement learning algorithm can be considered as a guided search algorithm. The training of the algorithm can be halted when an accident with desired cost is sampled. This removes the necessity for the RL algorithms to reach convergence, thus saves AV stress testing time.

Chapter 6

Conclusions and Future Works

In the last chapter of this thesis, we summarize our contributions and outline what we learned from the experiments. Additionally, we also discuss the potential future work for this thesis.

6.1 Conclusions

In this thesis, we propose a gym-like simulation environment *highway-attack-env* featuring black-box AV stress testing on top of the open-sourced *highway-env* [28], which opens the possibility for both single-agent and multi-agent reinforcement learning algorithms to be tested. We advocates multiagency and regulation on attacker agents' aggressiveness to be introduced to the AV stress testing problem. In our experiments, we showed that the integration of the cost value leads to more insightful accidents being generated. Additionally, the superiority of the MACPO algorithm over the CPO algorithm and the HATRPO-penalty algorithms concludes that introducing multi-agency and regulating the aggressiveness of attacker agents with constrained policy optimization method are crucial in AV stress testing problem. Since we were not able to conclude on the effect of the magnitude of the cost values, we recommend conducting a grid search to identify the optimal set of cost values.

6.2 Future Directions

The cost and reward structure we employ is simple and extendable, and breaks out a trade-off between the target outcome as a positive reward and the naturalness criteria as a cost. Additional cost or reward components could be added to extend this. For example, incorporating vehicle speed into the cost functions could provide a more accurate assessment of attacker agents' aggressiveness. We also acknowledge that the accidents with non-zero cost may also be insightful given a more comprehensive cost structure. Specifically, the cost limit can be set to an arbitrary value aiming to generate accidents where the attacker agents exhibit a corresponding level of aggressiveness. Therefore, one possible future work direction is to design better cost hierarchy and control the aggressiveness of the accidents generated.

Our proposed *highway-attack-env* environment is relatively simple, but fast, efficient and extendable. During our study, we have also considered several more advanced AV simulation environments, including CARLA [14], SMARTS [44] and SUMO [31]. However, these environment are either too costly for training or divert from our focus of black-box AV stress testing. Another future work direction is to extend our work to other simulation environments to incorporate more complex road structures and traffic scenarios.

We hope our proposed simulation environment and the benchmark can inspire novel algorithms designed to generate insightful accidents.

References

[1] Houssam Abbas, Matthew O'Kelly, Alena Rodionova, and Rahul Mangharam. Safe at any speed: A simulation-based test harness for autonomous vehicles. In *Cyber Physical Systems. Design, Modeling, and Evaluation: 7th International Workshop, CyPhy 2017, Seoul, South Korea, October 15-20, 2017, Revised Selected Papers 7*, pages 94–106. Springer, 2019.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[3] Eitan Altman. *Constrained Markov Decision Processes.* Routledge, 2021.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[6] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Adaptive stress testing with reward augmentation for autonomous vehicle validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 163–168. IEEE, 2019.

[7] Anthony Corso, Kyu-Young Kim, Shubh Gupta, Grace Gao, and Mykel J Kochenderfer. A deep reinforcement learning approach to rare event estimation. *arXiv preprint arXiv:2211.12470*, 2022.

[8] Anthony Corso, Ritchie Lee, and Mykel J Kochenderfer. Scalable autonomous vehicle safety validation through dynamic programming and scene decomposition. In *2020*

IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pages 1–6. IEEE, 2020.

[9] Anthony Corso, Robert Moss, Mark Koren, Ritchie Lee, and Mykel Kochenderfer. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72:377–428, 2021.

[10] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[11] Jyotirmoy Deshmukh, Marko Horvat, Xiaoqing Jin, Rupak Majumdar, and Vinayak S Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–18, 2017.

[12] Wenhao Ding, Baiming Chen, Bo Li, Kim Ji Eun, and Ding Zhao. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. *IEEE Robotics and Automation Letters*, 6(2):1551–1558, 2021.

[13] Wenhao Ding, Baiming Chen, Minjun Xu, and Ding Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2243–2250. IEEE, 2020.

[14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[15] Shuo Feng, Haowei Sun, Xintao Yan, Haojie Zhu, Zhengxia Zou, Shengyin Shen, and Henry X Liu. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615(7953):620–627, 2023.

[16] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[17] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[18] Shangding Gu, Jakub Grudzien Kuba, Munning Wen, Ruiqing Chen, Ziyan Wang, Zheng Tian, Jun Wang, Alois Knoll, and Yaodong Yang. Multi-agent constrained policy optimisation. *arXiv preprint arXiv:2110.02793*, 2021.

[19] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[20] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2034–2039. IEEE, 2018.

[21] Maximilian Kahn. Dynamic-occlusion-aware risk identification for autonomous vehicles using hypergames. 2021.

[22] Maximilian Kahn, Atrisha Sarkar, and Krzysztof Czarnecki. I know you can't see me: Dynamic occlusion-aware safety validation of strategic planners for autonomous vehicles using hypergames. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11202–11208. IEEE, 2022.

[23] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[24] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, 2019.

[25] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

[26] Markus Koschi, Christian Pek, Sebastian Maierhofer, and Matthias Althoff. Computationally efficient safety falsification of adaptive cruise control systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2879–2886, 2019.

[27] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[28] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[29] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. Av-fuzzer: Finding safety violations in autonomous driving systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–36, 2020.

[30] Chenyi Liu, Nan Geng, Vaneet Aggarwal, Tian Lan, Yuan Yang, and Mingwei Xu. Cmix: Deep multi-agent reinforcement learning with peak and average constraints. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*, pages 157–173. Springer, 2021.

[31] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[32] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Abbeel Pieter, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[33] Daniel Chi Kit Ngai and Nelson Hon Ching Yung. A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):509–522, 2011.

[34] Justin Norden, Matthew O'Kelly, and Aman Sinha. Efficient black-box assessment of autonomous vehicle safety. *arXiv preprint arXiv:1912.03618*, 2019.

[35] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[36] Kasra Rezaee, Peyman Yadmellat, and Simon Chamorro. Motion planning for autonomous vehicles in the presence of uncertainty using reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3506–3511. IEEE, 2021.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[38] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[40] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018.

[41] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[42] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

[43] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[44] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, Aurora Chongxi Huang, Ying Wen, Kimia Hassanzadeh, Daniel Graves, Dong Chen, Zhengbang Zhu, Nhat Nguyen, Mohamed Elsayed, Kun Shao, Sanjeevan Ahilan, Baokuan Zhang, Jiannan Wu, Zhengang Fu, Kasra Rezaee, Peyman Yadmellat, Mohsen Rohani, Nicolas Perez Nieves, Yihan Ni, Seyedershad Banijamali, Alexander Cowen Rivers, Zheng Tian, Daniel Palenicek, Haitham bou Ammar, Hongbo Zhang, Wulong Liu, Jianye Hao, and Jun Wang. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving, 11 2020.