9-2024

Enhancing multi-agent system testing with diversity-guided exploration and adaptive critical state exploitation

Xuyan MA

Yawen WANG

Junjie WANG

Xiaofei XIE
Singapore Management University, xfxie@smu.edu.sg

Citation

MA, Xuyan; WANG, Yawen; WANG, Junjie; and XIE, Xiaofei. Enhancing multi-agent system testing with diversity-guided exploration and adaptive critical state exploitation. (2024). *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Vienna, Austria, 2024 September 16-20*. 1491-1503.

Available at: https://ink.library.smu.edu.sg/sis_research/9443

Enhancing Multi-agent System Testing with Diversity-Guided Exploration and Adaptive Critical State Exploitation

Xuyan Ma[*][†]
Institute of Software
Chinese Academy of Sciences
Beijing, China
maxuyan2021@iscas.ac.cn

Yawen Wang[*][†]
Institute of Software
Chinese Academy of Sciences
Beijing, China
yawen2018@iscas.ac.cn

Junjie Wang[*][‡]
Institute of Software
Chinese Academy of Sciences
Beijing, China
junjie@iscas.ac.cn

Xiaofei Xie
Singapore Management University
Singapore, Singapore
xfxie@smu.edu.sg

Boyu Wu[*]
Institute of Software
Chinese Academy of Sciences
Beijing, China
boyu_wu2021@163.com

Shoubin Li[*]
Institute of Software
Chinese Academy of Sciences
Beijing, China
shoubin@iscas.ac.cn

Fanjiang Xu[*]
Institute of Software
Chinese Academy of Sciences
Beijing, China
fanjiang@iscas.ac.cn

Qing Wang[*][‡]
Institute of Software
Chinese Academy of Sciences
Beijing, China
wq@iscas.ac.cn

Abstract

Multi-agent systems (MASs) have achieved remarkable success in multi-robot control, intelligent transportation, and multiplayer games, etc. Thorough testing for MAS is urgently needed to ensure its robustness in the face of constantly changing and unexpected scenarios. Existing methods mainly focus on single-agent system testing and cannot be directly applied to MAS testing due to the complexity of MAS. To our best knowledge, there are fewer studies on MAS testing. While several studies have focused on adversarial attacks on MASs, they primarily target failure detection from an attack perspective, i.e., discovering failure scenarios, while ignoring the diversity of scenarios. In this paper, to highlight a typical balance between exploration (diversifying behaviors) and exploitation (detecting failures), we propose an advanced testing framework for MAS called MASTest with diversity-guided exploration and adaptive critical state exploitation. It incorporates both individual diversity and team diversity, and designs an adaptive perturbation mechanism to perturb the action at the critical states, so as to trigger more and more diverse failure scenarios of the system. We evaluate MASTest on two popular MAS simulation environments: Coop Navi and StarCraft II. Results show that the average distance of the resulting failure scenarios is increased by 29.55%-103.57% and 74.07%-370.00% on two environments compared to the baselines. Also, the failure patterns found by MASTest are improved by 71.44%-300.00% and 50%-500.00% on two experimental environments compared to the baselines.

[*]Also With State Key Laboratory of Intelligent Game, Beijing, China;
Science & Technology on Integrated Information System Laboratory, Beijing, China;
University of Chinese Academy of Sciences, Beijing, China;

[†]These authors contributed equally to this work
[‡]Corresponding author

ISSTA '24, September 16–20, 2024, Vienna, Austria
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680376

CCS Concepts

• **Software and its engineering → Software testing and debugging**.

Keywords

Multi-agent System Testing, Diversity-guided Exploration, Adaptive Perturbation Exploitation

ACM Reference Format:
Xuyan Ma, Yawen Wang, Junjie Wang, Xiaofei Xie, Boyu Wu, Shoubin Li, Fanjiang Xu, and Qing Wang. 2024. Enhancing Multi-agent System Testing with Diversity-Guided Exploration and Adaptive Critical State Exploitation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3680376

1 Introduction

In recent decades, multi-agent systems (MASs) have received considerable attention due to their ability to solve complex problems that involve interaction between multiple agents [2], e.g., multi-robot control [6, 8], intelligent transportation [32, 39], smart grids [34], unmanned aerial vehicles [5, 36] and multiplayer games [4, 33], etc. However, the robustness of MAS is a well-known challenge, largely due to training complexities such as sparse rewards and credit assignment. These issues render MAS particularly vulnerable to the dynamic and unforeseen scenarios encountered in real-world applications. Thus, before deploying a multi-agent system in real-world environments, it must undergo rigorous testing to confirm its robustness, particularly in safety-critical scenarios such as air traffic control systems, military systems and industrial automation.

The primary goal of MAS testing is to uncover as many potential failure scenarios as possible, where the target MAS makes undesirable decisions and ultimately fails to complete its task. Identifying these diverse failure scenarios enables developers to understand the weaknesses and different root causes, further enhancing overall robustness. However, due to the unpredictable nature of open environments, such as changing weather conditions, varying terrain, and the behaviors of other participants, there could be an infinite number of scenarios. Consequently, a major challenge in MAS testing is effectively detecting these diverse failure scenarios.

Numerous studies [7, 16, 29, 42, 44] have been developed to test single-agent systems (SASs), such as autonomous driving systems. However, these methodologies are challenging to apply to MASs due to the complex interactions and cooperation between multiple agents. To the best of our knowledge, MAS testing is less explored. While several studies have focused on adversarial attacks on MASs by perturbing the observations or actions of agents within the team [10, 19, 22, 45], they primarily target failure detection from an attack perspective, often overlooking the diversity and comprehensiveness of failures. From a testing perspective, our objective is to identify diverse types of failures, thereby reducing redundant analysis on similar failures and uncovering a broader spectrum of robustness issues in the target MAS.

This paper aims to develop an effective MAS testing method capable of generating diverse failure scenarios to expose various robustness issues. There are two primary challenges: 1) Measuring the diversity of scenarios to ensure the generation of varied failures. A straightforward approach might involve comparing the trajectories of two scenarios; however, this method faces significant challenges in complex MAS environments. Differences such as trajectory lengths, the number of participants, agent interactions, and even minor environmental changes can significantly influence trajectory comparisons, affecting the measurement of the scenario diversity. 2) Effectively generating failures while balancing the need for scenario diversity against failure detection. Focusing solely on covering diverse behaviors could lead to many scenarios that do not necessarily cause failures, highlighting a typical balance between exploration (diversifying behaviors) and exploitation (detecting failures). Existing works [7, 12, 16, 25], which primarily perturb the initial state before the system runs, may not effectively achieve this balance, as modifications in the initial stage are difficult to control over their impact on the internal state of the MAS during operation.

To tackle these challenges, this paper introduces MASTest, an advanced testing framework for MAS that balances diversity-guided exploration with adaptive critical state exploitation to trigger diverse failure scenarios during testing. To address challenge ❶ for improving diversity, we have developed an abstraction-based method to measure the behaviors of multiple agents, considering both individual and team dynamics. Similar to [7], individual behavior is represented by the abstraction sequence of the agent's trajectory. For team behavior, our approach not only abstracts and aggregates state information among teammates but also examines their collaborative relationships through interaction strength. Both individual and team behaviors are used to guide the exploration, i.e., generating scenarios with different behaviors. To overcome challenge ❷ for enhancing failure detection, our framework includes an adaptive mechanism for identifying critical states that are likely to trigger diverse failures. We perform targeted action perturbations on these critical states to increase the likelihood of triggering various failures. To realize this, MASTest maintains a *state criticality table*, which records the perturbation potential of each state based on three aspects: diversity gain, failure gain, and selection frequency. A state with a high potential score is considered critical and should be prioritized for perturbation. This table is dynamically updated after each test run to reflect the latest diversity and failure feedback, thereby providing accurate guidance for subsequent testing.

We evaluate the effectiveness of MASTest in two distinct multi-agent environments: Coop Navi [25] (a cooperative task) and StarCraft II [27] (a competitive task). For each environment, we compare MASTest against three baseline methods. The results demonstrate that MASTest significantly outperforms the baselines in discovering diverse failures. Specifically, in the Coop Navi environment, MASTest increases the average distance of generated failure scenarios by 29.55% to 103.57%, and in StarCraft II, this increase ranges from 74.07% to 370.00%. Additionally, MASTest enhances the coverage of unique failure patterns by 71.44% to 300.00% in Coop Navi and by 50.00% to 500.00% in StarCraft II. Furthermore, an ablation study confirms that failure feedback, individual diversity, and team diversity each significantly contribute to the discovery of diverse failure scenarios. We also demonstrate the practical value of these diverse failure scenarios in enhancing system robustness. After applying fixes based on the failure scenarios generated by MASTest, we observed a 45.83% reduction in collisions and a 45.42% improvement in task completion rates.

In summary, this paper makes the following contributions:

- This work is, to the best of our knowledge, the first to specifically address the testing of multi-agent systems (MASs), highlighting the importance of this emerging type of artificial intelligence system to the research community.
- We introduce MASTest, a novel framework that employs diversity-guided exploration and adaptive critical state exploitation. It assesses both individual and team diversity among agents and adaptively identifies critical states for action perturbation, aiming to trigger increasingly diverse failure scenarios.
- We conduct extensive evaluations on two categories of MASs and compare with six baselines, demonstrating promising results.
- We release the source code of MASTest and detailed experiment results to facilitate the replication and further research[1].

2 Background

2.1 Multi-Agent System

A multi-agent system consists of a group of interacting agents, each of which communicates, cooperates with each other to accomplish a large number of complex tasks that cannot be accomplished by a single agent. In MAS, a complex task is divided into multiple smaller tasks, each assigned to a different agent. The actions of an agent affect not only its own state, but also those of its neighbors.

[1]More details can be found on our website: https://github.com/issta24/ISSTA24_MAT

This requires each agent to consider the actions of the other agents when deciding on the best goal-directed action. Based on the task type, the applications of MAS can be classified into two categories: cooperative task (e.g., Coop Navi) and competitive task (e.g., StarCraft II). Cooperative task is to finish the given goal together, such as reaching the destination, while competitive task is to defeat their opponents together.

The decision-making process of a MAS is often modeled as a Markov game. Specifically, a Markov game for m agents is defined as $(\mathcal{N}, \{\mathcal{S}^i\}_{i\in\mathcal{N}}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, p, \gamma)$, where $\mathcal{N}$ is a set of m agents, $\mathcal{S}^i$ and $\mathcal{A}^i$ are the state space, action space of agent i, respectively. $\gamma \in [0, 1)$ is the discounting factor. $\mathcal{S} = \mathcal{S}^1 \times \cdots \mathcal{S}^m$ is the joint state space. $\mathcal{A} = \mathcal{A}^1 \times \cdots \mathcal{A}^m$ is the joint action space. The state transition $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is controlled by the current state and joint action, where $\Delta(\mathcal{S})$ represents the set of all probability distributions over the joint state space $\mathcal{S}$. Each agent i obtains reward r^i as a function of the state and agent's action: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At time t, agent i chooses its action a_t^i according to a policy π^i: $\mathcal{S}^i \to \Delta(\mathcal{A}^i)$. The agents' joint policy is $\pi = \prod_{i\in\mathcal{N}} \pi^i : \mathcal{S} \to \Delta(\mathcal{A})$. Each agent i aims to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where γ is the discounting factor and T is the time horizon.

2.2 Scenario Description

Scenario. A scenario is a temporal sequence of scenes, where each scene is a snapshot of the environment including the static and dynamic objects. Therefore, a scenario can be described by a series of configurable static and dynamic attributes. Static attributes set the static objects of the scenario, such as the map, destination, obstacle, etc. Dynamic attributes define the state (e.g., position, orientation, etc.), trajectory, and behavior (e.g., move, speed up, etc.) of dynamic objects, such as Non-Player Character (NPC) (i.e., other agents which are not target agents). To find diverse and critical scenarios, we can manipulate the static and dynamic objects in the environment by adjusting their configurable attributes, i.e., the trajectory described by a set of waypoints.

Scenario observation is a sequence of global state of all objects at each time step in the simulation environment, including the target agents, NPCs and other environment elements. We can also obtain the team trajectory of the target MAS to focus only on the behavior of the target system. Specifically, the team trajectory τ of the target MAS consists of the individual trajectory of each agent τ_i, where i is the id of agent. τ_i contains the local state of the target agent, such as its position, speed, and health value (the aspects are task-specific, and vary in different MASs). Given a k-dimensional state space R^k, the individual trajectory τ_i can be represented as $(s_0^i, \cdots, s_k^i)$, where s_j^i refers to the j-th dimension state.

Failure Scenario. The failure scenario refers to the scenarios where the target MAS makes the undesirable decisions and ultimately fails the task. The failure has different definitions in different test environments. For example, in Coop Navi, the failure refers to whether the target MAS collides with obstacles and whether all the agents in the system reach their destinations. While in StarCraft II, it is defined as whether the target MAS are defeated or tied by the opponents. Users can configure MASTest with other failure definitions, as long as such undesirable behaviors can induce reasonably low rewards.

3 Approach

3.1 Overview

Fig. 1 illustrates the overview of MASTest. Essentially, MASTest modifies the participants (e.g., NPCs or opponents) within the environment to evaluate the robustness of the MAS in completing tasks. Specifically, MASTest captures a concrete state (s) at each time step. If this state is the ending state, a trajectory is obtained and it checks if the task has been successfully completed. If not, a failure is detected. We then collect feedback from the trajectory, which includes individual and team diversity feedback, failure feedback (encompassing both the degree and frequency of failures), and the selection frequency of each state. This feedback is used to update the state criticality table by adjusting the potential scores of the perturbed states within the test scenario. For states that are not ending, MASTest checks their scores in the table to determine their criticality; the higher the score, the more critical the state is considered. Action perturbations are applied only on critical states, while original actions are maintained for non-critical states. The action is dispatched to the environment to obtain the next state for further exploration and testing. The scores in the table reflect various feedback, guiding the generation of diverse failures. Note that after reaching the ending state, we will reinitialize from the initial state and continue retesting until the test budget is exhausted.

3.2 State Abstraction

Due to the complexity of the state and the infinite number of possible states in the environments, comparing behaviors and maintaining the state criticality table is challenging. Following existing methods [17, 18], we employ grid-based abstraction to cluster similar states into abstract states, thereby reducing complexity.

For an agent A in the target MAS, as detailed in Sec. 2.2, the concrete trajectory of A denotes as $\tau_A : (s_0^A, \cdots, s_k^A)$. Each state is a multi-dimensional vector, and each dimension is divided into N equal intervals based on its range $[l, u]$, where l and u are the lower and upper bounds respectively. This division transforms the concrete state space R^k into N^k discrete abstract states. Hence, each concrete state s is mapped to a grid (denoted as the abstract state $\hat{s} = g(s)$), where g represents the grid-based abstraction function. Thus, multiple similar concrete states may map to the same abstract state. From this abstraction, we derive the abstract trajectory from the concrete trajectory τ_A, i.e., $\hat{\tau_A} = (g(s_0^A), \cdots, g(s_k^A))$.

In Fig. 2, we illustrate this concept using a 2-dimensional concrete state space. The concrete states are abstracted into abstract states, represented by grid IDs. For example, the abstract trajectory of Trajectory 1 can be represented as $\{\cdots, 15, 11, 7, 13, 18, \cdots\}$ and that of Trajectory 2 as $\{\cdots, 5, 1, 7, 18, 14, \cdots\}$.

Above, we explained the abstraction of an individual agent's state. Similarly, we can also calculate the abstraction of the global state. The global state encompasses the entire environment, including the state information of all agents, NPCs, and others.

3.3 Diversity Feedback

To generate diverse failures, it is necessary to assess the behaviors of multi-agents within a scenario, including incidents such as collisions with other agents (either partners or NPC agents), obstacles,

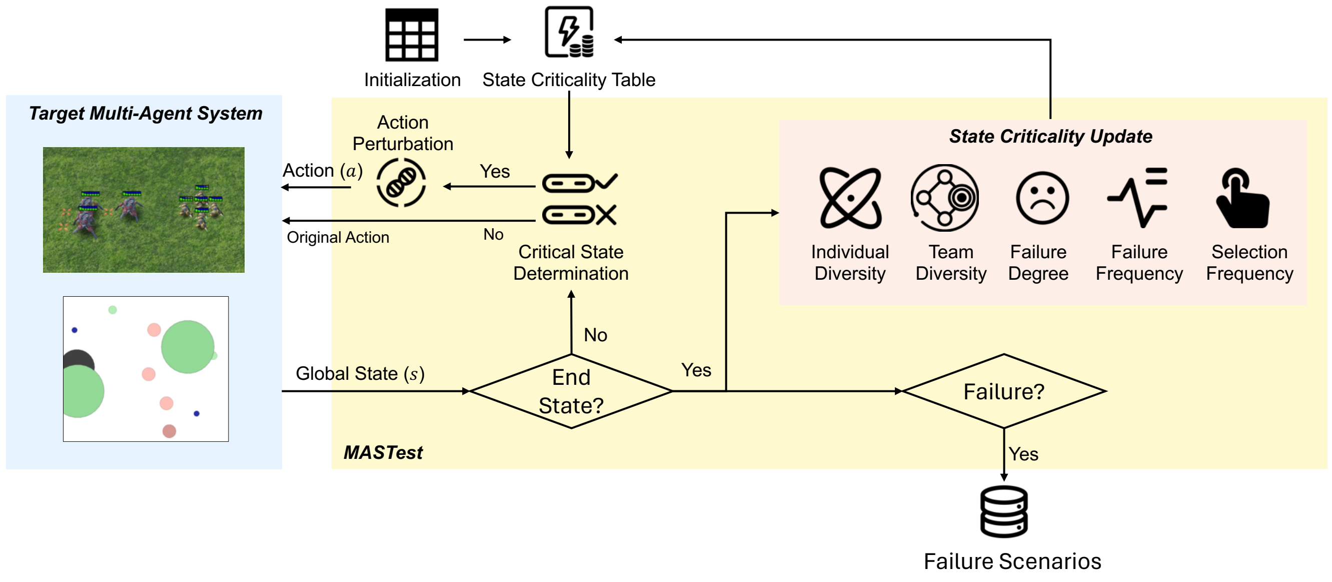


Figure 1: Overview of MASTest.

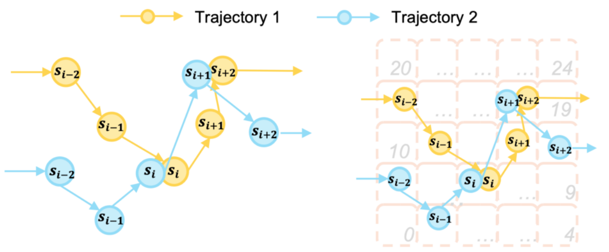


Figure 2: Grid-based state abstraction.

or failures to reach a designated destination. We measure diversity by comparing currently explored behaviors against historical behaviors. Specifically, the diversity of behaviors in MAS is evaluated from two distinct perspectives: the actions of individual agents (***individual diversity***) and the coordinated actions among team members (***team diversity***).

3.3.1 Individual Diversity. Individual diversity evaluates the diversity of individual agents behavior in the MAS, i.e., individual trajectories generated by each agent. After the ending state, we can obtain the individual trajectory of each agent A τ_A and its corresponding abstract trajectory $\hat{\tau_A}$. We use $\mathcal{T}_A$ to represent the historical abstract trajectories of the agent A explored in previous testing. The diversity of the behavior of agent A is calculated as the differences between its current abstract trajecotry and the explored abstract trajectories:

$$d^A_{\tau_A} = \min_{\hat{\tau_A}' \in \mathcal{T}_A} dis(\hat{\tau_A}, \hat{\tau_A}')$$

The abstract trajectories have different lengths due to different execution time. Therefore, for the distance calculation dis, the existing time sequence distance metrics, such as Dynamic Time Warping [3], may not work well in our situation. To this end, we adopt the normalized Hamming distance to compute the distance between two given traces t and t' (i.e., abstract trajectories in our context), which is specified as follows:

$$dis(t, t') = \frac{Hamming(t, t') + |len(t) - len(t')|}{max(len(t), len(t'))} \quad (1)$$

where $Hamming(t, t')$ computes the Hamming distance [11] between the segments of common length in t and t', i.e., the forehand states in each trace. The denominator is the maximum length of two traces, which takes into account the length of the trace and normalizes the distance to the interval $[0, 1]$.

The final individual diversity within the current scenario is calculated by averaging the diversity scores of each agent:

$$D^I = avg(\{d^A_{\tau_A} | A \in AG\})$$

where I is short for *Individual* and AG denotes the set of all agents. A higher D^I indicates greater diversity in the behaviors of individual agents within the new test scenario.

3.3.2 Team Diversity. Differences in team strategies tend to affect the behavior of the whole team, e.g., a decentralized strategy on the formation exhibits a more spread out stand of agents and a larger distance between agents, while a compact strategy exhibits a smaller distance between agents. In order to discover different team behaviors, MASTest does the following four steps: 1) **Time Step Sample**, 2) **Graph-based Abstraction**, 3) **Graph Embedding**, 4) **Team Diversity Measurement**.

1) **Time Step Sample**: Analyzing the team states at every time step in a scenario can be time-consuming, given that states at neighboring time steps often exhibit similarity. Therefore, we sample a selected number of time steps to representatively capture the essence of the entire scenario.

Previous work has proved that rewards in reinforcement learning often guide the behavior of the agents and can reflect the importance

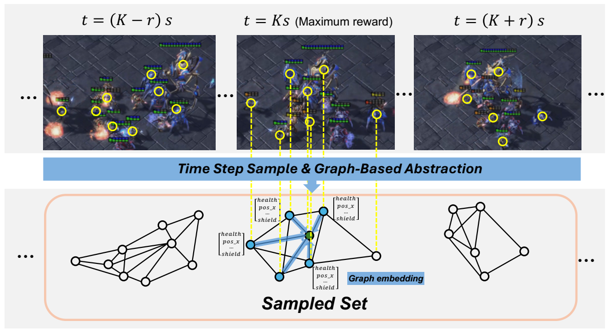


Figure 3: Illustration of time step sample, graph-based abstraction and graph embedding.

of the states [1]. To this end, we record the reward of MAS at each time step, and take a window centered on the time step with the highest reward and r as intervals to sample a time step set. As shown in Fig. 3, assuming the window size is 3, and the center is the K-th time step, then the set of sampled time step is: $\{K - r, K, K + r\}$, which is used for subsequent team diversity measurement. We will present detailed discussion on the determination of window size and window center in Sec. 6.

2) **Graph-based Abstraction**: The actions taken by an agent may potentially change the relations between agents and thus change the graph. To measure the diversity of behavior embodied by the MAS team, we abstract the team and the interactions between teammates as an undirected graph. The state information and collaborative relations of the target MAS can be well characterized by the structure, the vertices' attributes, and edges' weights of the graph. In this way, MASTest can measure the team diversity by measuring graph similarity.

As Fig. 3 shows, each agent in the team is a vertex, and each vertex contains a set of attributes of corresponding agent, i.e., position, speed, health value, shield. The edge between two vertices and its weight quantifies interaction between the two agents and the situation of interaction. We use the distance between the two agents obtained from the environment to represent the weight, indicating the extent to which the two agents can interact with each other.

3) **Graph Embedding**: It is difficult to compare the similarity between two graphs directly because the features of the graph are hard to extract and quantify [20]. Meanwhile, vector operations provide an easier and faster way to do this [9]. Therefore, we take the idea of embedding and represent the graph as vectors based on vertex attributes and their interactions with neighbors for subsequent team diversity measurement. Based on this, MASTest generates graph embedding for the abstract graph at each sampled time step, as shown below:

First, MASTest initializes the embedding representation of each vertex according to its own attribute. Then, MASTest aggregates the information contained in the neighboring vertices through a weight-based aggregation function. For a vertex v, $\mathcal{N}(v)$ is the neighbor vertices of v and h_v^k is the embedding representation of vertex v after the k-th iteration. $w(u, v)$ is the weight of the edge between vertices u and v, which is defined in Graph-based Abstraction. Then the embedding of v at $(k+1)$-th iteration h_v^{k+1} is the sum of h_v^k and the weighted and averaged neighbor embedding, which is specified as follows:

$$h_v^{k+1} \leftarrow Sum(h_v^k, Mean(\frac{1}{w(u,v)} * h_u^k), u \in \mathcal{N}(v)) \tag{2}$$

We can obtain the embedding representation of each vertex in the graph as h_v. The embedding of the graph G is represented as $emd(G) = (h_{v_0}, \cdots, h_{v_m})$, where $v_0, \cdots, v_m$ are all vertexes in the graph G.

4) **Team Diversity Measurement**: Similar to the individual diversity measurement, the team diversity is measured by comparing the current team behavior and historical team behaviors explored in the previous testing. Specifically, we use the graph embedding to represent the team behavior and the team diversity in the current scenario is calculated as:

$$D^T = \min_{G' \in \mathcal{G}} graph\_dis(emd(G), emd(G'))$$

where the superscript T is short for *Team*, $\mathcal{G}$ represents the set of historical graphs, *graph_dis* calculates the distance between two sampled graph set by the Graph Wasserstein distance [30], which is widely used to measure the similarity between pairwise graphs. The larger the D^T, the more diverse the team behaviors in the scenario.

3.4 Failure Feedback

Except for the behavior diversity, the primary goal of MAS testing is to search for scenarios where target agents fail. Hence, we also evaluate the failure degree of the target MAS in given scenarios. We define the failure degree of a test scenario from *task completion* (i.e., the gap to the goal), and the *cost consumption* (i.e., the loss in the process of completing the task).

$$F = f_{goal} + f_{cost} \tag{3}$$

Task Completion refers to the gap to the goal, which is defined:

$$f_{goal} = \sum_{i=1}^{N} D(p_{a_i}, p_{goal}) \tag{4}$$

where a_i represents the i-th agent in the MAS, $D(p_{a_i}, p_{goal})$ indicates the distance of agent a_i from the goal. The specific calculation will be slightly different in different environments. For example, the gap to the goal can be the distance from the agent to the destination in cooperative navigation task, or the remaining health of the opponent agents in competitive game task.

Cost Consumption refers to the cost consumed in completing the task, which is denoted as f_{cost}. For example, in cooperative navigation, the cost comes from collisions with obstacles; while in competitive game, the cost comes from the consumption of its own combat power.

3.5 *State Criticality Table* Initialization & Update

We maintain a *state criticality table* in MASTest to determine the criticality of each abstract state and guide subsequent action perturbations. Note that the abstract state represents the abstraction of the global state, rather than the abstract state of an individual agent. Initially, the *state criticality table* is set up such that all abstract states are assigned a uniform criticality of 1. After each testing iteration, the criticality of each perturbed state is updated based on three key factors: 1) the emergence of new *diverse* scenarios relative to existing ones, 2) the frequency of failures observed, and 3) how often the state is selected (selection frequency).

1) *Diverse Failures Gain.* We assess whether perturbing at the state could produce different failure scenarios compared with existing failure scenarios. We consider the failure degree (Sec. 3.4), individual diversity (Sec. 3.3.1), and team diversity (Sec. 3.3.2) to calculate the criticality update, which is specified as follows:

$$\Delta E_{dfg} = \frac{F}{1 - \frac{D^I + D^T}{2} + \gamma} \tag{5}$$

where γ is a small value (e.g., 10^{-5}) to avoid zero denominator. Note that, ΔE_{dfg} is measured from the entire scenario, e.g., from team and agent trajectories. We use this score to update the criticality of those abstract states within the scenario that have been perturbed. Eq. 5 indicates that if a test scenario generated by perturbing a specific state results in both a higher degree of failure and greater diversity, a higher bonus will be added to the criticality of that state.

2) *Failure Frequency*: In order not to make too many ineffective attempts on a state that is difficult to trigger a failure of the target MAS, we count the frequency each state has triggered the failure. For each state, we record the number of failure scenarios or non-failure scenarios caused by perturbation on it (denoted as $\#F$ and $\#NF$). A smaller weight (0.1) used in the benign case is to avoid deducting too much criticality score. The criticality update brought by failure frequency is computed as follows:

$$\Delta E_{ff} = \begin{cases} \frac{\#F}{\#F + \#NF}, & \text{Current scenario is a failure.} \\ -\frac{0.1 \cdot \#NF}{\#F + \#NF}, & \text{Current scenario is a benign case.} \end{cases} \tag{6}$$

3) *Selection Frequency*: To avoid MASTest selecting a particular critical state multiple times, which may result in local convergence, we reduce the criticality of the state when selecting it. In this paper, we choose a fixed value of criticality decay, i.e. $\Delta E_{sf} = -0.05$.

According to the above three aspects, the criticality update value of the perturbed state is:

$$\Delta E_s = w_1 \cdot \Delta E_{dfg} + w_2 \cdot \Delta E_{ff} + w_3 \cdot \Delta E_{sf} \tag{7}$$

where w_1, w_2 and w_3 are three hyperparameters to adjust the weights of the three factors. For each perturbed state, we update the criticality based on the calculated update value ΔE_s.

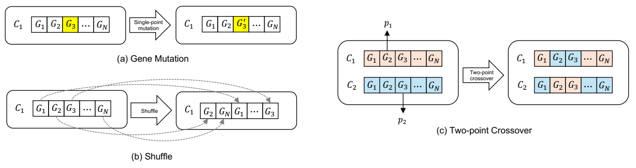
Critical State Determination. MASTest determines whether a state is critical to add a perturbation to, based on the criticality of each state. For each state s_i, the probability of being considered as the critical state is:

$$p_{s_i} = \frac{max(E_{s_i}, 0)}{\sum_{s \in \mathcal{S}} E_s} \tag{8}$$

where E_{s_i} is the criticality of state s_i, and the denominator is the sum of the criticality of all states. The higher the criticality of the state, the higher the probability of being a critical state. In each scenario execution, MASTest selects the top 10% states as the critical states to guide the action perturbation

3.6 Action Perturbation

At each time step, if the current state is determined as a critical state, MASTest will add perturbation on the action of NPCs. Note that all the dynamic objects except the target MAS which can affect the environment can be considered as NPCs. MASTest employs three types of perturbation operators: gene mutation, two-point crossover and shuffle. At each time step, analogising the action that each NPC originally took to a chromosome, we give an introduction to three perturbation operators, followed by a description of specific usage for different cases.



C_1
G_1 G_2 G_3 … G_N
Single-point mutation
C_1
G_1 G_2 G_3' … G_N
(a) Gene Mutation
C_1
G_1 G_2 G_3 … G_N
Shuffle
C_1
G_2 G_N G_1 … G_3
(b) Shuffle
p_1
C_1
G_1 G_2 G_3 … G_N
C_2
G_1 G_2 G_3 … G_N
p_2
Two-point crossover
C_1
G_1 G_2 G_3 … G_N
C_2
G_1 G_2 G_3 … G_N
(c) Two-point Crossover

Figure 4: Perturbation operators.

(a) Gene Mutation: Modifying one gene in a chromosome as shown in Fig. 4 (a). The original gene (G_3) is replaced by randomly generating parameter values (G_3'), such as the speed of an NPC at a given moment. (b) Shuffle: As Fig. 4 (b) shows, shuffle randomly changes the order of genes on the chromosome. (c) Two-point crossover: Selecting two random points (i.e., p_1 and p_2 in Fig. 4 (c)) on two chromosomes, and swapping the genes between them.

Due to the different dimensions of the action, the specific perturbation operations that can be employed are slightly different. When the dimension of the action is 1, the action is a finite and discrete value from a predefined set of actions that can only be perturbed with (a) and (c). When the dimension of the action is greater than 1, the action is a vector, so all perturbation operators can be adopted.

4 Experiment Design

4.1 Research Questions

We consider the following research questions:

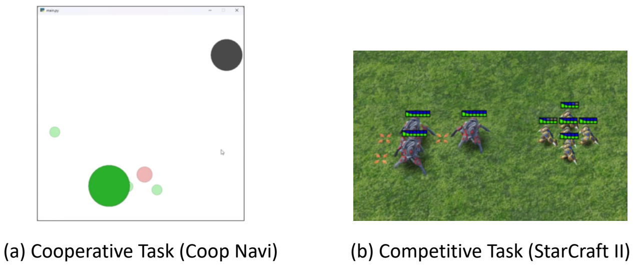
RQ1: (Effectiveness) Can MASTest effectively find diverse failure scenarios of the target MAS compared to the baselines?

RQ2: (Ablation Study) How do the failure feedback, individual diversity and team diversity contribute to the performance of MASTest?

RQ3: (Usefulness) How useful is the failure scenarios discovered by MASTest to repair the target MAS?

4.2 Target Models and Environments

Based on the task type, the application of MAS can be classified into two categories: cooperative task and competitive task. As Fig. 5 shows, the cooperative task (e.g., Coop Navi) requires the target MAS (small green) to cooperate to reach the destination (big green) without collisions with teammates, obstacles (black), or NPCs (pink).



(a) Cooperative Task (Coop Navi)
(b) Competitive Task (StarCraft II)

Figure 5: Two task types of MAS.

Whereas the competitive task (e.g., StarCraft II) requires the target MAS (right & blue) to defeat its opponents, called adversarial agents (left & red).

For completeness and generalization of experiment, we verify the effectiveness of MASTest on both cooperative and competitive tasks. We choose two commonly-used simulation environment: Multi-Agent Particle Environment (MPE) [21] which is a two-dimensional multi-agent environment and StarCraft II [27] which is a popular real-time strategy game. For MPE, we use Coop Navi [25] which requires the MAS to cooperate to reach their destinations without collision. The target model of MAS is officially provided by OpenAI [24] which is trained by the MADDPG algorithm [21]. For StarCraft II, we conduct experiments on the map $1c3s5z$, where two-sided teams each have three types with a total of nine soldiers and try to defeat their opponents. We use the official model provided by DeepMind as the target MAS, which is trained by off-policy actor-critic and experience replay techniques.

4.3 Baselines and Evaluation Metrics

4.3.1 Baselines. Different categories of tasks (i.e., cooperative and competitive) have different characteristics, and existing techniques tend to focus on a specific category. Besides, existing testing techniques proposed for a specific category of tasks can not easily or even impossible to be used in another category, e.g., one can not modify the initial state in the competitive tasks like StarCraft II, thus the baselines MDPFuzz and GMT cannot take effect in these tasks. Therefore, we choose the corresponding baselines in terms of the category of tasks.

For cooperative tasks, we compare MASTest with three commonly-used and SOTA techniques.

- **Random** randomly selects one NPC and perturbs its action at a random time step without any feedback.
- **MDPFuzz** [25] is a popular fuzzing testing framework particularly for models solving Markov decision process problems. It generates initial states as test cases that can lead to failure-triggering trajectories.
- **GMT** [17] is the SOTA testing framework for decision-making models. It uses generative diffusion models as test case generators and novelty-based guidance to diversify agent behaviors.

For competitive tasks, we compare MASTest with three commonly-used and state-of-the-art (SOTA) techniques.

- **Random** randomly selects one NPC which is trained by QMIX and perturbs its action at a random time step without any feedback.
- **QMIX** [26] is a commonly-used deep multi-agent RL algorithm. We use it to train a testing MAS to reveal failure scenarios of target MAS via adversarial interactions. We use the implementation in PyMARL2 [13] for this baseline.
- **Wuji** [43] is the SOTA testing framework for competitive tasks by generating diverse behavioral agents. It can generate a population of adversarial MAS trained by QMIX with diverse strategies to reveal the diverse failure scenarios of the target MAS. We use the implementation provided by the authors.

4.3.2 Evaluation Metrics. Following previous works [12, 17, 25, 43], we evaluate the diversity of generated failure scenarios from state coverage (%Coverage) and average distance (#Distance). Besides, we use the number of failure scenarios (#Failure) and the rate of failures (%Failure) to evaluate the efficiency of finding defects. We also summarize the failure patterns and define a metric (%Pattern) based on manual analysis, which reflects the proportion of found failure patterns to all patterns.

- **%Coverage**: The ratio of covered states in terms of the whole global state space. Following existing studies [17, 25], we divide each dimension of the state space into 5 parts, and the whole state space is composed of dozens of units. During the testing, we obtain the global state at each time step, and put it in the specific unit. %Coverage is then calculated with the ratio of covered units in terms of all units.
- **#Distance**: The average distance between the failure scenarios generated by MASTest, where the distance between each pair of two failure scenarios is calculated by Eq. 1.
- **#Failure**: The number of failure scenarios of the target MAS.
- **%Failure**: The ratio of the number of failure scenarios of the target MAS to the total number of test scenarios.
- **%Pattern**: The coverage of failure patterns generated by each method to all failure patterns.

4.4 Experimental Setup

The Overall Setup. In all experiments, we use MASTest and baselines to respectively test the target MAS within the same test duration of two hours following the common setup [12, 43]. For each method, testing is repeated three times to avoid randomness, and the average performance is presented.

We conducted preliminary experiments in other two settings of the experimental environment, i.e., $3m$ in StarCraft II and Coop Spread in MPE, and determine the value of parameters based on the experimental results as shown below. The weights w_1, w_2 and w_3 in Eq. 7 are set to 0.5, 0.5, and 1. The window size in Sec. 3.3.2 is set to 5. The interval of time step sampling r is set to 2. We implement MASTest with PyTorch. All experiments are launched on a server equipped with an NVIDIA TITAN RTX GPU, Intel Xeon Silver CPU, 64GB RAM, running on Ubuntu 20.04 OS.

For **RQ2**, we design an ablation study to investigate the effectiveness of failure feedback, individual diversity and team diversity. The detailed settings of variants are as follows:

- w/o Diversity Feedback: The *state criticality table* is only updated by failure feedback.
- w/o Failure Feedback: The *state criticality table* is only updated by diversity feedback (individual diversity and team diversity).
- w/o Team Diversity: The *state criticality table* is only updated by failure feedback and individual diversity.
- w/o Individual Diversity: The *state criticality table* is only updated by failure feedback and team diversity.

For **RQ3**, to verify the usefulness of the failure scenarios, we repair the target MAS used in Coop Navi with the failure scenarios uncovered by MASTest and other baselines. Here, "repair" represents constructing a dataset to fine tune the model, which contains data on the environmental settings that make the target MAS fail, i.e., trajectories of NPCs, locations of obstacles, etc. In the process of repairing, we take the failure scenario as the input data to retrain the target agents. These failure scenarios are generated by different methods, including random generation. In the process of retesting, we randomly generate scenarios as test data and test the performance of the target agents. Since the target MAS in StarCraft II are black-boxed and inaccessible, we cannot obtain the model structure and training code of the target MAS. Therefore, we cannot repair them by ourselves, but we can help the model repair by submitting the failure scenarios to the developer.

5 Results and Analysis

5.1 RQ1: Effectiveness of MASTest

We prove the effectiveness of MASTest from two aspects: automated evaluation and manual evaluation.

5.1.1 Automated Evaluation of Failure Scenario. Table 1 presents the results of %Coverage, #Distance, #Failure and %Failure of target MAS in column 2 - 5. For cooperative task, MASTest achieves satisfying performance on finding diverse failure scenarios. Compared with the best baseline (GMT), MASTest improves 14.40%, 29.55%, 14.49% and 14.77% on the four metrics, respectively. The main reason is that MASTest uses the diversity and failure degree to guide the generation of failure scenarios. While in GMT, the novelty of the termination state is used as a guide for generating test cases, which can not reasonably characterize the target MAS's strategy. In addition, Gaussian random noise is used to mutate the test cases, which is inefficient.

For competitive task, MASTest outperforms other baselines in terms of %Coverage and #Distance. The goal of the adversary trained by QMIX is only to win, so #Failure and %Failure is the highest. In order to explore diversity, Wuji and Random testing employs mutation which is random and without guidance. Therefore, the failure scenarios produced by these two methods have lower performance than MASTest on all four metrics. Compared with the best baseline (Wuji), MASTest improves 26.14%, 74.07%, 17.15% and 24.42% on the four metrics, respectively.

Due to space constraints, we also present the trends of %Coverage, #Distance and #Failure over time on our website. For example, over time, the %Coverage gradually increases and reaches a plateau, and at each moment, MASTest outperforms other baselines. #Distance tends to increase and then decrease or keep decreasing over time, with MASTest still outperforming other baselines.

Table 1: Effectiveness of MASTest and baselines.

Method \ Metric	%Coverage	#Distance	#Failure	%Failure	%Pattern
Cooperative Task					
Random	43.30%	0.28	841.6	40.50%	25.00%
MDPFuzz	57.60%	0.33	1373.6	67.90%	41.67%
GMT	61.80%	0.44	1469.3	73.10%	58.33%
MASTest	**70.70%**	**0.57**	**1682.2**	**83.90%**	**100.00%**
Competitive Task					
Random	26.53%	0.12	546.7	49.70%	16.67%
QMIX	31.60%	0.1	**1040.6**	**96.80%**	33.33%
Wuji	36.15%	0.27	772.7	72.90%	66.67%
MASTest	**45.60%**	**0.47**	905.2	90.70%	**100.00%**

5.1.2 Manual Evaluation of Failure Scenario. We manually analyze failure scenarios generated by MASTest and baselines to provide a high-level view of the failure patterns. Specifically, we use DBSCAN [38] to cluster all the failure scenarios generated by MASTest and the baselines. The parameter cluster radius in DBSCAN, indicating the maximum distance between failure scenarios in the same cluster, can influence the number of clusters generated. A small value would generate too much clusters and the failure scenarios in different clusters might correspond to the same pattern, while a big value would generate few clusters and the failure scenarios in a cluster can demonstrate different patterns, both of which is undesirable. Through pilot experiments, we set the parameter as 0.7 and obtain 12 clusters. We sample 10 failure scenarios from each cluster. Then, the first two authors work together to summarize the failure pattern of the scenarios for each cluster. When a discrepancy arises, the third author is involved and they discuss together and reach an agreement. In order to further confirm the reliability of manual analysis, after the manual analysis, we additionally recruited two testers with rich testing experience to analyze the failure patterns. Specifically, two testers each randomly selected 10 failure scenarios from each of the 12 clusters. Then they compare each failure scenario with the 12 patterns we summarized. If the match is successful, it is considered consistent; otherwise, it is inconsistent. The average consistency rate of the 12 clusters is 98% and no new failure patterns are found.

Table 2 shows the coverage of our approach and the baselines in revealing these failure patterns, where F1-1 to F1-8 and F2-1 to F2-4 are failure patterns of cooperative task, and F3-1 to F3-4 and F4-1 to F4-2 are failure patterns of competitive task. Fig. 6 demonstrates the discovered failure patterns for cooperative task, and the detailed failure pattern descriptions are presented below. The patterns and detailed descriptions for competitive task is shown on our website due to space limit. Based on the crash definition of the experimental environment, the failure patterns of cooperative task compose of two categories collision (F1) and failing to reach the destination (F2).

Collision (F1). Fig. 6 shows the failure patterns related with collision, which contains the following 12 unique failure patterns. **F1-1**: The target agents gather around the destination, while the NPC also moves towards the destination, and engage in a round-robin chase. The target agents do not choose to leave, resulting in multiple collisions.

Table 2: The coverage of failure patterns.

	Method	F1	F2	Sum	Details
Cooperative	**Random**	3	0	3	F1-4, F1-7, F1-8
	MDPFuzz	4	1	5	F1-1, F1-4, F1-7, F1-8, F2-1
	GMT	5	2	7	F1-1, F1-4, F1-6, F1-7, F1-8, F2-1, F2-4
	MASTest	**8**	**4**	**12**	F1-1 to F1-8, F2-1 to F2-4
	Method	**F3**	**F4**	**Sum**	**Details**
Competitive	**Random**	1	0	1	F3-1
	QMIX	2	0	2	F3-1, F3-2
	Wuji	3	1	4	F3-1, F3-2, F3-4, F4-2
	MASTest	**4**	**2**	**6**	F3-1 to F3-4, F4-1 to F4-2

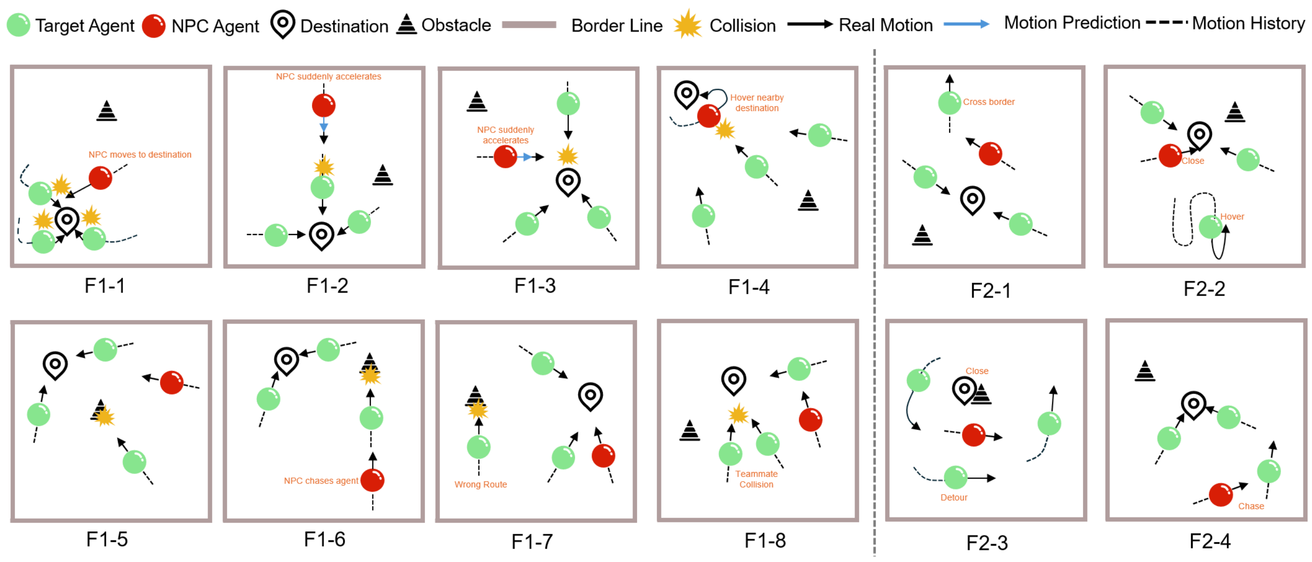


Figure 6: Failure patterns on Coop Navi task. The green globules are target agents, and the red globule is NPC.

F1-2: The NPC suddenly accelerates behind the target agent, and the target agent cannot escape in time, resulting in a collision.
F1-3: The NPC's moving direction is perpendicular to the target agent's moving direction. The NPC suddenly accelerates, and the target agent does not adjust its speed in time, resulting in a collision.
F1-4: The NPC hovers near the destination, but the target agent chooses to collide with the NPC in order to reach the destination.
F1-5: The obstacle, target agent, and destination are on the same line, and the target agent does not choose a detour and collides with the obstacle in order to reach the destination.
F1-6: During the chase between the NPC and the target agent, the target agent causes a collision with an obstacle in order to escape.
F1-7: A target agent makes a decision error: it should move to the top right, but chooses to move upwards, resulting in a collision with the obstacle.
F1-8: The target agents move to the same location because they have the same destination. As they move forward, their proximity results in a collision.

Failing to reach the destination (F2). MASTest finds four failure patterns related with not reaching the destination, shown in Fig. 6.
F2-1: A target agent chooses a wrong direction and leaves the setup area (i.e., crosses the border line), eventually failing to reach the destination.
F2-2: The NPC is close to the destination and the target agent chooses not to move up in order to avoid the NPC. In this way, the target agent does not reach the destination.
F2-3: In this case, the obstacle is very close to the destination and thus the target agent judges that it cannot move up and finally does not reach the destination.
F2-4: The target agent is constantly being chased by the NPC, resulting in a focus on escaping rather than going to the destination, and ultimately not reaching the destination.

As shown in Table 1 and 2, we present %Pattern and specific failure patterns discovered by each method. The results show that MASTest can discover the most diversified failure scenarios and cover all the failure patterns discovered by baselines.

Then, we carry out further qualitative analysis of the results. Specifically, we analyze why our approach can generate some failure scenarios and the baselines can not. This is due to the fact that our approach can recognize the critical state based on the feedback mechanism. It perturbs the NPCs at the critical state, which can trigger the failure scenario more effectively. In contrast, the baselines cannot recognize critical state, which makes many perturbation operations ineffective. Fig. 7 shows a concrete example and we provide more detailed analysis on the website due to space limit.

As shown in Fig. 7(a), our approach recognizes the current state as the critical state, i.e., the target agent is moving towards the destination, and the NPC is at a suitable distance from the topmost target agent. The NPC is perturbed to turn and run towards the target agent, and finally collides with it. In the contrast, baselines select a non-critical state for perturbation, as shown in Fig. 7(b). Due to missing the critical opportunity, after the perturbation, the NPC

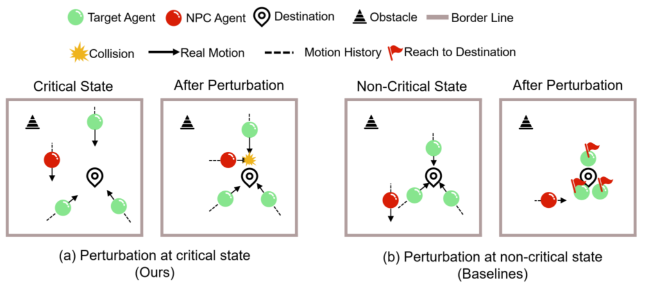


Figure 7: Illustration of qualitative analysis.

cannot interfere with the target agent in time, which eventually causes the target agent to reach the destination.

Answering to RQ1: Compared with the commonly-used and SOTA techniques, MASTest can discover diverse failure scenarios during the same testing period.

5.2 RQ2: Ablation Study

Table 3 shows the results of %Coverage, #Distance, #Failure and %Failure of the failure scenarios found by MASTest and four variants (w/o Failure Feedback, w/o Diversity Feedback, w/o Team Diversity and w/o Individual Diversity).

We first focus on the impact of failure feedback and diversity feedback on the results, i.e., w/o Failure Feedback, w/o Diversity Feedback. When removing Failure Feedback, %Coverage decreases, #Distance increases, #Failure and %Failure greatly decrease. The reason for this phenomenon is as follows: MASTest w/o Failure Feedback wastes a lot of energy on the states that is hard to trigger failure scenarios. As a result, #Failure and %Failure are reduced. As the total number of failure scenarios is reduced, the state space covered is correspondingly reduced, so %Coverage is reduced. However, although the number of failure scenarios is low, the generated failure scenarios are different from each other, so #Distance increases.

In addition, for the failure scenarios generated by w/o Diversity Feedback, %Coverage decreases, #Distance decreases, #Failure and %Failure increase. When Diversity Feedback is removed, MASTest focuses on perturbing those states that are prone to trigger target MAS failure and ignores the diversity of resulting failure scenarios. In this way, #Failure and %Failure increases. Although the number of failure scenarios is high, resulting failure scenarios are similar, so %Coverage and #Distance are still lower than MASTest.

To assess the effect of team diversity and individual diversity in finding diverse failure scenarios, we report the performance impact of removing these two parts separately. As Table 3 shows, removing one of team diversity and individual diversity results in a decrease in %Coverage and #Distance, with little effect on #Failure and %Failure. This is because removing one part of diversity feedback reduces the diversity of failure scenarios, but does not affect the function of the failure feedback. And the existence of the other part of diversity feedback can still maintain the balance with the failure feedback, so #Failure and %Failure do not change significantly. Moreover, removing team diversity leads to greater influence than removing individual diversity, which indicates that team diversity better represents the diversity of the entire team behaves in MAS, thus finding more diversified failure scenarios.

Table 3: Ablation study on failure and diversity feedback.

Method \ Metric	%Coverage	#Distance	#Failure	%Failure
Cooperative Task				
w/o Failure Feedback	65%	**0.61**	1273.0	63.30%
w/o Diversity Feedback	50.10%	0.28	**1773.5**	**88.15%**
w/o Team Diversity	60.90%	0.42	1690.4	83.7%
w/o Individual Diversity	65.6%	0.49	1689.5	84.00%
MASTest	**70.70%**	0.57	1682.2	83.90%
Competitive Task				
w/o Failure Feedback	36.90%	**0.5**	513.0	50.20%
w/o Diversity Feedback	29.80%	0.21	**983.0**	**96%**
w/o Team Diversity	35.20%	0.33	910.5	90.70%
w/o Individual Diversity	38.10%	0.37	912.3	90.60%
MASTest	**45.60%**	0.47	905.2	90.70%

Table 4: Model repair by MASTest and baselines.

Method	#Collision	%Completion
Before Repair	386.4	51.3%
Repair by Random	375	51.8%
Repair by MDPFuzz	350.9	60.8%
Repair by GMT	320.1	64.7%
Repair by MASTest	**209.3**	**74.6%**

Answering to RQ2: Both failure feedback and diversity feedback (team diversity & individual diversity) are useful for finding diverse failure scenarios. In addition, team diversity is more helpful than individual diversity.

5.3 RQ3: Usefulness of MASTest

In RQ3, we repair the model used in Coop Navi with the failure scenarios uncovered by MASTest and other baselines. Table 4 reports the performance of the model before and after repair. We use the number of collisions as well as the percentage of task completion (i.e., the number of completed tasks out of the total number of tasks) to reflect the usefulness of failure scenarios for repairing models. The results show that the number of collisions occurred after the target MAS was repaired using the failure scenarios generated by MASTest was lower than that before the repair (209.3 vs 386.4), and the percentage of task completion is higher than that before the repair (74.6% vs 51.3%). Moreover, MASTest brings a higher enhancement to the target MAS than other baselines.

Answering to RQ3: The performance of the target MAS is effectively improved by repairing with the failure scenarios discovered by the MASTest, outperforming other baselines.

6 Discussion

6.1 Selection of Window Center

As mentioned in Sec. 3.3.2, we use a window to sample several time steps to represent the team's strategy throughout the test scenario. In this section, we evaluate the impact of window center on the performance. Specifically, we randomly set the window center (denoted as random center) or choose the time of maximum reward as the window center (denoted as reward center), and compare the diversity of failure scenarios resulting from both schemes.

Table 5: Performance comparison of different window center.

Setting \ Metric	%Coverage	#Distance	#Failure	%Failure
Cooperative Task				
random center	66.40%	0.52	1618.1	80.70%
reward center (MASTest)	**70.70%**	**0.57**	**1682.2**	**83.90%**
Competitive Task				
random center	40.70%	0.41	867.4	86.90%
reward center (MASTest)	**45.60%**	**0.47**	**905.2**	**90.70%**

Table 5 shows the results with different settings of window center. Results show that using the maximum reward as the basis for selecting the window center produces better results than randomly selecting the window center.

6.2 Threats to Validity

The first threat concerns about the generality of the proposed approach. Although we only experiment with StarCraft II and Coop Navi, the two environments are different and involve different types of tasks and agents, and we repeat the experiment multiple times. This could alleviate the threat to some extent.

The second threat is that sampling time step is based on the related work [1]. The performance of MASTest is superior to other baselines, and based on simple comparative experiments, the use of the maximum reward as the basis has a better effect than the random scheme. And in the future we would conduct more detailed experiments to study the impact of different settings.

The third threat is that the granularity of state abstraction follows existing works. Despite following existing work, we have achieved better performance than baselines under the current setup. Furthermore, this is a configurable parameter, and it would not be difficult for users to tune it based on a small scale validation dataset or directly apply the value we used considering its good performance in several popular experimental environments.

The fourth threat is the patterns of the failure scenarios are identified and analyzed with human efforts. Both MPE and StarCraft II are multi-agent environments widely used for academic research. Our in-depth analysis of the environment and the collaborative efforts of multiple authors to identify the patterns of the failure can effectively mitigate the threat.

7 Related Work

There are some studies about agent testing focusing on the diversity of policy failures. They collected the trajectories of agents and compared the similarity between trajectories to quantify the behavioral diversity of agents [7, 15, 35]. BehAVExplor [7] measured the potential of test cases to find multiple violations by comparing the distance of the trajectories of the vehicles under test. Li et al. [15] compared the mutual information of the agents' trajectories in order to motivate each agent to produce different behaviors. SeqDivFuzz [35] was based on the siamese network model to compare the similarity of state sequences to determine whether to continue to execute test cases, thereby improving test efficiency. EMOGI [28] used the duration and moving distance to represent different strategies of the agent, resulting in agents with different behaviors. Mazouni et al. [23] introduced quality diversity for policy testing, which describes how a solution actually solves the task. Depending on quality diversity, they optimized the test input to find different faults in reinforcement learning. However, these methods only focused on the behavioral diversity of the single agent and cannot measure the behavioral diversity of a multi-agent team well. By characterizing the structure and interactions of multi-agent teams, MASTest can discover diverse failure scenarios involving diverse team strategies.

Some techniques designed for testing decision-making models can also be applied to MAS testing [17, 25]. Pang et al. [25] proposed a black-box fuzzy testing framework MDPFuzz to test intelligent software in several Markov decision process scenarios. Li et al. [17] proposed a testing framework for decision-making policies based on novelty guidance and diffusion model to diversify agent behaviors. Although they can work on some MASs, they do not consider the characteristics of MASs, so they do not perform as well as our MASTest.

Adversarial attack is widely used to evaluate the robustness of MAS from the attack perspective [10, 14, 19, 22, 31, 45], including modifying observations, actions, communications, etc. The robustness is reflected by the failure probability of the target MAS after the attack. Ilahi et al. [14] considered attacks that rely on perturbing the state space, the reward space, the action space, and the model space, where one can perturb the model's learned parameters. Ma and Li [22] proposed a grey box attack method for MARL communication and evaluate the performance of MARL. Tu et al. [31] proposed a method to attack the communication of multi-agent systems by generating adversarial messages. However, these methods only focused on successful attack on MAS but ignore the diversity of the discovered failure scenarios. In addition, some general-purpose deep learning testing methods [37, 40, 41] have been proposed to evaluate model robustness. However, these methods tend to be ineffective in evaluating the robustness of MAS due to their lack of consideration for action sequences in MASs.

8 Conclusion

Multi-agent systems (MASs) draw considerable attention for solving complex problems involving interactions between multiple agents. Before deploying MAS into the real world, it should be adequately tested to ensure its robustness under diverse scenarios. However, to our best knowledge, MAS testing has not been investigated. In this paper, we propose a testing framework MASTest with diversity-guided exploration and adaptive critical state exploitation. It measures both individual diversity and team diversity among multiple agents, and adaptively selects critical states for action perturbation during operation, thus triggering more diversified failure scenarios of MAS. Experimental results show that MASTest is able to find more diverse failure scenarios compared to baselines in both automated and manual evaluation. Furthermore, we demonstrate failure scenarios generated by MASTest can repair the target MAS more effectively than other baselines.

Acknowledgments

This work was partially supported by the Youth Innovation Promotion Association Chinese Academy of Sciences, National Natural Science Foundation of China Grant No.62232016 and No.62072442, Basic Research Program of ISCAS Grant No. ISCAS-JCZD-202304, Major Program of ISCAS Grant No. ISCAS-ZD-202302, the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

[1] Dan Amir and Ofra Amir. 2018. HIGHLIGHTS: Summarizing Agent Behavior to People. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar (Eds.). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 1168–1176.

[2] Parasumanna Gokulan Balaji and Dipti Srinivasan. 2010. An introduction to multi-agent systems. *Innovations in multi-agent systems and applications-1* (2010), 1–27.

[3] Donald J. Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, USA, July 1994. Technical Report WS-94-03*, Usama M. Fayyad and Ramasamy Uthurusamy (Eds.). AAAI Press, 359–370.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *CoRR* abs/1912.06680 (2019).

[5] Zhiliang Bi, Xiwang Guo, Jiacun Wang, Shujin Qin, and Guanjun Liu. 2023. Deep Reinforcement Learning for Truck-Drone Delivery Problem. *Drones* 7 (07 2023), 445. https://doi.org/10.3390/drones7070445

[6] Cecilia E. Garcia Cena, Pedro F. Cárdenas, Roque Saltarén Pazmiño, Lisandro Puglisi, and Rafael Aracil Santonja. 2013. A cooperative multi-agent robotics system: Design and modelling. *Expert Syst. Appl.* 40, 12 (2013), 4737–4748. https://doi.org/10.1016/J.ESWA.2013.01.048

[7] Mingfei Cheng, Yuan Zhou, and Xiaofei Xie. 2023. BehAVExplor: Behavior Diversity Guided Testing for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, René Just and Gordon Fraser (Eds.). ACM, 488–500. https://doi.org/10.1145/3597926.3598072

[8] Yong Duan, Baoxia Cui, and Xinhe Xu. 2012. A multi-agent reinforcement learning approach to robot soccer. *Artif. Intell. Rev.* 38, 3 (2012), 193–211. https://doi.org/10.1007/S10462-011-9244-8

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 855–864. https://doi.org/10.1145/2939672.2939754

[10] Jun Guo, Yonghong Chen, Yihang Hao, Zixin Yin, Yin Yu, and Simin Li. 2022. Towards Comprehensive Testing on the Robustness of Cooperative Multi-agent Reinforcement Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2022, New Orleans, LA, USA, June 19-20, 2022.* IEEE, 114–121. https://doi.org/10.1109/CVPRW56347.2022.00022

[11] Richard Wesley Hamming. 1986. *Coding and information theory (2. ed.).* Prentice Hall.

[12] Fitash Ul Haq, Donghwan Shin, and Lionel C. Briand. 2023. Many-Objective Reinforcement Learning for Online Testing of DNN-Enabled Systems. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE, 1814–1826. https://doi.org/10.1109/ICSE48619.2023.00155

[13] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. 2021. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479* (2021). https://doi.org/10.48550/arXiv.2102.03479

[14] Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala I. Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. 2022. Challenges and Countermeasures for Adversarial Attacks on Deep Reinforcement Learning. *IEEE Trans. Artif. Intell.* 3, 2 (2022), 90–109. https://doi.org/10.1109/TAI.2021.3111139

[15] Chenghao Li, Tonghan Wang, Chengjie Wu, Qianchuan Zhao, Jun Yang, and Chongjie Zhang. 2021. Celebrating Diversity in Shared Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 3991–4002.

[16] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael B. Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng (Eds.). IEEE, 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[17] Zhuo Li, Xiongfei Wu, Derui Zhu, Mingfei Cheng, Siyuan Chen, Fuyuan Zhang, Xiaofei Xie, Lei Ma, and Jianjun Zhao. 2023. Generative Model-Based Testing on Decision-Making Policies. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023.* IEEE, 243–254. https://doi.org/10.1109/ASE56229.2023.00153

[18] Zhuo Li, Derui Zhu, Yujing Hu, Xiaofei Xie, Lei Ma, Yan Zheng, Yan Song, Yingfeng Chen, and Jianjun Zhao. 2023. Neural Episodic Control with State Abstraction. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.* OpenReview.net. https://doi.org/10.48550/ARXIV.2301.11490

[19] Jieyu Lin, Kristina Dzeparoska, Sai Qian Zhang, Alberto Leon-Garcia, and Nicolas Papernot. 2020. On the Robustness of Cooperative Multi-Agent Reinforcement Learning. In *2020 IEEE Security and Privacy Workshops, SP Workshops, San Francisco, CA, USA, May 21, 2020.* IEEE, 62–68. https://doi.org/10.1109/SPW50608.2020.00027

[20] Yayun Liu and Kuangfeng Ning. 2024. Improved graph representation learning based on neighborhood aggregation and interaction fusion. *J. Intell. Fuzzy Syst.* 46, 1 (2024), 1287–1314. https://doi.org/10.3233/JIFS-234086

[21] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA.* 6379–6390.

[22] Xiao Ma and Wu-Jun Li. 2023. Grey-box Adversarial Attack on Communication in Multi-agent Reinforcement Learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh (Eds.). ACM, 2448–2450. https://doi.org/10.5555/3545946.3598963

[23] Quentin Mazouni, Helge Spieker, Arnaud Gotlieb, and Mathieu Acher. 2024. Testing for Fault Diversity in Reinforcement Learning. *CoRR* abs/2403.15065 (2024). https://doi.org/10.1145/3644032.3644458

[24] OpenAI. 2020. multiagent-particle-envs. https://github.com/openai/multiagent-particle-envs/tree/master

[25] Qi Pang, Yuanyuan Yuan, and Shuai Wang. 2022. MDPFuzz: testing models solving Markov decision processes. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, Sukyoung Ryu and Yannis Smaragdakis (Eds.). ACM, 378–390. https://doi.org/10.1145/3533767.3534388

[26] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 4292–4301.

[27] Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019.* International Foundation for Autonomous Agents and Multiagent Systems, 2186–2188.

[28] Ruimin Shen, Yan Zheng, Jianye Hao, Zhaopeng Meng, Yingfeng Chen, Changjie Fan, and Yang Liu. 2020. Generating Behavior-Diverse Game AIs with Evolutionary Multi-Objective Deep Reinforcement Learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 3371–3377. https://doi.org/10.24963/IJCAI.2020/466

[29] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022.* ACM, 94–106. https://doi.org/10.1145/3540250.3549100

[30] Matteo Togninalli, M. Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten M. Borgwardt. 2019. Wasserstein Weisfeiler-Lehman Graph Kernels. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 6436–6446.

[31] James Tu, Tsun-Hsuan Wang, Jingkang Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. 2021. Adversarial Attacks On Multi-Agent Communication. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021.* IEEE, 7748–7757. https://doi.org/10.1109/ICCV48922.2021.00767

[32] Matthew Veres and Medhat Moussa. 2020. Deep Learning for Intelligent Transportation Systems: A Survey of Emerging Trends. *IEEE Trans. Intell. Transp. Syst.* 21, 8 (2020), 3152–3168. https://doi.org/10.1109/TITS.2019.2929020

[33] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha

Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.* 575, 7782 (2019), 350–354. https://doi.org/10.1038/S41586-019-1724-Z

[34] Perukrishnen Vytelingum, Thomas Voice, Sarvapali D. Ramchurn, Alex Rogers, and Nicholas R. Jennings. 2010. Agent-based micro-storage management for the Smart Grid. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen (Eds.). IFAAMAS, 39–46.

[35] Kairui Wang, Yawen Wang, Junjie Wang, and Qing Wang. 2023. Fuzzing with Sequence Diversity Inference for Sequential Decision-making Model Testing. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023, Florence, Italy, October 9-12, 2023.* IEEE, 706–717. https://doi.org/10.1109/ISSRE59848.2023.00041

[36] Liang Wang, Kezhi Wang, Cunhua Pan, Wei Xu, Nauman Aslam, and Lajos Hanzo. 2021. Multi-Agent Deep Reinforcement Learning-Based Trajectory Planning for Multi-UAV Assisted Mobile Edge Computing. *IEEE Trans. Cogn. Commun. Netw.* 7, 1 (2021), 73–84. https://doi.org/10.1109/TCCN.2020.3027695

[37] Longtian Wang, Xiaofei Xie, Xiaoning Du, Meng Tian, Qing Guo, Zheng Yang, and Chao Shen. 2023. DistXplore: Distribution-guided testing for evaluating and enhancing deep learning systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 68–80.

[38] Wei-Tung Wang, Yi-Leh Wu, Cheng-Yuan Tang, and Maw-Kae Hor. 2015. Adaptive density-based spatial clustering of applications with noise (DBSCAN) according to data. In *2015 International Conference on Machine Learning and Cybernetics, ICMLC 2015, Guangzhou, China, July 12-15, 2015.* IEEE, 445–451. https://doi.org/10.1109/ICMLC.2015.7340962

[39] Tong Wu, Pan Zhou, Kai Liu, Yali Yuan, Xiumin Wang, Huawei Huang, and Dapeng Oliver Wu. 2020. Multi-Agent Deep Reinforcement Learning for Urban Traffic Light Control in Vehicular Networks. *IEEE Trans. Veh. Technol.* 69, 8 (2020), 8243–8256. https://doi.org/10.1109/TVT.2020.2997896

[40] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. Npc: N euron p ath c overage via characterizing decision logic of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27.

[41] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*. 146–157.

[42] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 132–142. https://doi.org/10.1145/3238147.3238187

[43] Yan Zheng, Changjie Fan, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, and Yingfeng Chen. 2019. Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 772–784. https://doi.org/10.1109/ASE.2019.00077

[44] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: systematic physical-world testing of autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 347–358. https://doi.org/10.1145/3377811.3380422

[45] Ziyuan Zhou and Guanjun Liu. 2023. Robustness Testing for Multi-Agent Reinforcement Learning: State Perturbations on Critical Agents. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023) (Frontiers in Artificial Intelligence and Applications, Vol. 372)*, Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu (Eds.). IOS Press, 3131–3139. https://doi.org/10.3233/FAIA230632

Received 2024-04-12; accepted 2024-07-03