Multi-Agent Reinforcement Learning for Assessing False-Data Injection Attacks on Transportation Networks

Taha Eghtesad
Pennsylvania State University
University Park, PA, USA
tahaeghtesad@psu.edu

Sirui Li
Massachusetts Institute of Technology
Cambridge, MA, USA
siruil@mit.edu

Yevgeniy Vorobeychik
Washington University St. Louis
St. Louis, MO, USA
yvorobeychik@wustl.edu

Aron Laszka
Pennsylvania State University
University Park, PA, USA
alaszka@psu.edu

ABSTRACT

The increasing reliance of drivers on navigation applications has made transportation networks more susceptible to data-manipulation attacks by malicious actors. Adversaries may exploit vulnerabilities in the data collection or processing of navigation services to inject false information, and to thus interfere with the drivers' route selection. Such attacks can significantly increase traffic congestions, resulting in substantial waste of time and resources, and may even disrupt essential services that rely on road networks. To assess the threat posed by such attacks, we introduce a computational framework to find worst-case data-injection attacks against transportation networks. First, we devise an adversarial model with a threat actor who can manipulate drivers by increasing the travel times that they perceive on certain roads. Then, we employ hierarchical multi-agent reinforcement learning to find an approximate optimal adversarial strategy for data manipulation. We demonstrate the applicability of our approach through simulating attacks on the Sioux Falls, ND network topology.

KEYWORDS

Transportation networks, False-data injection, Navigation system, Cybersecurity, Deep reinforcement learning, Multi-agent reinforcement learning, Hierarchical reinforcement learning

ACM Reference Format:
Taha Eghtesad, Sirui Li, Yevgeniy Vorobeychik, and Aron Laszka. 2024. Multi-Agent Reinforcement Learning for Assessing False-Data Injection Attacks on Transportation Networks. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 8 pages.

1 INTRODUCTION

In today's digitally interconnected world, drivers rely on navigation applications and online information more than before. Furthermore, the availability of social media has accelerated the spread of misinformation. A malicious actor could manipulate the drivers directly by sending malicious information through SMS messaging [22], manipulating traffic signals [2, 4, 7, 8, 17], or physically changing the road signs [3] to interfere with drivers' route selection. With the availability of social media, the drivers can further spread this misinformation to their peers to snowball the effect of manipulation. Alternatively, the adversary can inject false information into the navigation application. For example, one can place phones in a cart and pull them on the street, tampering with the navigation application to result in marking the road with heavy traffic and rerouting the drivers [18].

Manipulating transit networks can lead to increased traffic congestion leading to devastating consequences. Modern societies heavily rely on road networks for accessing essential services such as education, healthcare, and emergency services. Moreover, road networks contribute to economic growth by enabling logistic movements of materials, goods, and products. Disruption of transportation networks can therefore lead to food insecurity, job losses, or even political disarray, such as the Fort Lee scandal [24].

Efforts have been made to measure the impact of false information injection on dynamic navigation applications [10], traffic congestion [22], and navigation applications [16]. However, finding an optimal attack is in general computationally challenging [22], complicating vulnerability analysis.

The injection of false data into navigation applications is a complex task that involves several actions and decisions over time, given the dynamic nature of traffic patterns. The use of Reinforcement Learning (RL) approaches provides an effective and versatile solution to tackle such sequential decision-making challenges.

When faced with decision-making problems of any size, Reinforcement Learning (RL) can be a powerful tool [14, 19]. However, the larger the problem, the more computation power is needed to train an RL model. For example, when dealing with a persistent adversary injecting false information into a city-wide transportation network of a major city, manipulating thousands of network links and millions of vehicles would make training of out-of-the-box RL strategies impractical. In such cases, a multi-agent and hierarchical RL framework becomes necessary. This framework should have local adversarial RL agents assigned to a subsection of the transportation network, observing their local information, and making local decisions. Additionally, these agents should cooperate to find the optimal sequence of false information to be injected. The agents should be coordinated with a global agent that assigns significance to each locality.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. 2024.

1.1 Contributions

Our goal is to examine the possible effects of false information injection on navigation apps. To achieve this, we created an adversarial threat model that outlines the relationships between vehicles, the navigation app, and a false information actor. Using a Markov Decision Process (MDP), we constructed this model to optimize the total travel time for all vehicles to arrive at their destinations.

We introduce a Hierarchical Multi-Agent (Deep) Reinforcement Learning (HMARL) system that consists of two levels. At the lower level, local agents observe the information of their immediate surroundings and collaborate to find the best local strategy to inject false information. The agents are constrained by an attack budget. At the higher level, a global agent coordinates the activities of the lower-level agents.

Finally, we evaluated our approach by comparing it with baseline RL algorithms and non-optimal heuristic approaches through an ablation study in Sioux Falls, ND.

1.2 Organization

The article is divided into several sections. In Section 2, we conduct a literature review on false information injection in navigation apps and scalable reinforcement learning techniques. In Section 3, we describe the interactions of the the adversarial threat actor and the transportation network. In Section 4, we provide the background information necessary to comprehend our approach. In Section 5, we explain our HMARL approach. In Section 6, we assess the HMARL approach on a benchmark transportation network, Sioux Falls, ND. Finally, in Section 7, we provide a higher-level analysis of the approach and suggest future directions.

2 RELATED WORK

The literature review discusses false information injection in navigation apps and hierarchical reinforcement learning.

2.1 Attacks on Navigation Applications

Several studies have been conducted to assess the effects of manipulating drivers to disrupt the normal functioning of transportation systems. These attacks can be carried out through various means, such as physically tampering with traffic signals. Researchers [2, 4, 7, 8, 17] have all explored these approaches. For example, Laszka et al. [6, 7] demonstrated that vehicles may be susceptible to altered and tampered traffic signs that can mislead them into taking the wrong path.

In a recent study, Waniek et al. [22] surveyed approximately 3,300 participants to investigate the effects of direct manipulation of drivers through SMS notifications and invalid road signs. The results showed that fake traffic signals and SMS notifications can significantly alter predetermined travel routes. As a result, there can be up to 5,000 additional vehicles on major road networks in Chicago.

Through an experiment conducted by Simon Weckert [18], it was found that navigation apps can be misled by false information. Weckert pulled a wagon with 99 smartphones while using Google Maps, which resulted in the app marking the street as heavily congested and suggesting alternative routes to drivers. This highlights the potential vulnerability of navigation apps to false information injection.

Recent research has revealed that data manipulation can impact transportation networks and users. However, it has been difficult to determine the full extent of this impact due to computational constraints. To address this issue, a state-of-the-art hierarchical reinforcement learning algorithm is being developed, which promises to offer a feasible solution.

2.2 Hierarchical RL Approaches

Hierarchical Reinforcement Learning (HRL) has gained significant attention due to its applications and development. These methods have proven to be successful in tasks that require coordination between multiple agents, such as Unnamed Aerial Vehicles (UAVs) and autonomous vehicles, to complete objectives efficiently.

For instance, Yang et al. [26] devised a general framework for combining compound and basic tasks in robotics, such as navigation and motor functions, respectively. However, they limited the application to single-agent RL at both levels. Similarly, Chen et al. used attention networks to incorporate environmental data with steering functions of autonomous vehicles in a hierarchical RL manner so that the vehicle can safely and smoothly change lanes.

In the UAV applications, Zhang et al. [27] demonstrated the success of hierarchical RL in the coordination of wireless communication and data collection of UAVs.

Although our problem is in a different domain, the fundamental ideas of these works are applicable to us since we are dealing with cooperation and coordination between adversarial agents in finding an optimal manipulation strategy in navigation applications. The study results indicate that the use of Reinforcement Learning approaches accurately modeled the effects of false information injection on navigation apps.

3 SYSTEM MODEL

In this section, we devise and formalize our threat model with respect to a transportation network environment where the adversarial agent injects false traffic information with a restricted budget with the aim of increasing the total travel time of vehicles traveling in this network.

3.1 Environment

The traffic model is defined by a *road network* $G = (V, E)$, where V is a set of nodes representing road intersections, and E is a set of directed edges representing road segments between the intersections. Each edge $e \in E$ is associated with a tuple $e = \langle t_e, b_e, c_e, p_e \rangle$, where t_e is the free flow time of the edge, c_e is the capacity of the edge, and b_e and p_e are the parameters for the edge to calculate actual edge travel time $W_e(n_e)$ given the congestion of the network, where n_e is the number of vehicles currently traveling along the edge [20]. Specifically, we use the following function for $W_e(n_e)$:

$$W_e(n_e) = t_e \times \left(1 + b_e \left(\frac{n_e}{c_e}\right)^{p_e}\right) \tag{1}$$

The *set of vehicle trips* are given with R, where each trip $r \in R$ is a tuple $\langle o_r, d_r, s_r \rangle$, with $o_r \in V$ and $d_r \in V$ the origin and destination

of the trip, respectively, and s_r the number of vehicles traveling between the origin-destination pair $\langle o_r, d_r \rangle$.

3.2 State Transition

For each vehicle trip $r \in R$ at each time step $t \in \mathbb{N}$, *vehicle location* $l_r^t \in V \cup (E \times \mathbb{N})$ represents the location of vehicle r at the end of time step t, where the location is either a node in V or a tuple consisting of an edge in E and a number in $\mathbb{N}$, which represents the number of timesteps left to traverse the edge.

Each vehicle trip begins at its origin; hence $l_r^0 = o_r$. At each timestep $t \in \mathbb{N}$, for each vehicle trip r that $l_r^{t-1} \in V \setminus \{d_r\}$, i.e., the vehicle trip is at a node but has not reached its destination yet, let $\oslash_r^{t-1} = (l_r^{t-1}, e_1, v_1, e_2, v_2, \ldots, e_k, d_r)$ be a shortest path from l_r^{t-1} to d_r considering congested travel times $\boldsymbol{w}^t$ as edge weights. Then $l_r^t = \langle e_1, \lfloor w_e^{t-1} \rceil \rangle$, where the travel time of edge e is

$$w_e^t = W_e \left(\sum_{\{r \in R \mid l_r^{t-1} = \langle e, \cdot \rangle\}} s_r \right). \tag{2}$$

Thus, for a trip r with $l_r^{t-1} = \langle e, n \rangle$, i.e., the vehicle is traveling along an edge, if $n = 1$, that is, the vehicle is one time step from reaching the next intersection, $l_r^t = v_1$. Otherwise, $l_r^t = \langle e, n-1 \rangle$.

3.3 Attacker Model

At the high level, our attack model involves adversarial perturbations to *observed* (rather than actual) travel times along edges e, subject to a perturbation budget constraint $B \in \mathbb{R}$. Let $a_e^t \in \mathbb{R}$ denote the adversarial perturbation to the observed travel time over the edge e. The budget constraint is then modeled as $\|\boldsymbol{a}^t\|_1 \leq B$, where $\boldsymbol{a}^t$ combines all perturbations over individual edges into a vector. The observed travel time over an edge e is then

$$\hat{w}_e^t = w_e^t + a_e^t. \tag{3}$$

It is this observed travel time that is then used by the vehicles to calculate their shortest paths from their current positions in the traffic network to their respective destinations. Since we aim to develop a defense that is robust to informational assumptions about the adversary, we assume that the attacker completely observes the environment at each time step t, including the structure of the transit network G, all of the trips R, and the current state of each trip l_r.

The attacker's goal is to maximize the total vehicle travel times, which we formalize as the following optimization problem:

$$\max_{\{\boldsymbol{a}^1, \boldsymbol{a}^2, \ldots\}: \forall t (||\boldsymbol{a}^t||_1 = B)} \sum_{t=0}^{\infty} \gamma^t \cdot \sum_{\{r \in R \mid l_r^t \neq d_r\}} s_r, \tag{4}$$

where $\gamma \in (0, 1)$ is a temporal discount factor.

4 BACKGROUND

In this section, we define the terminology and background to help better understand the solution approach.

4.1 Deep Reinforcement Learning

Let the tuple $\langle S, A, R, T \rangle$ define a *Markov Decision Process* (MDP) where S denotes the state space, A denotes the action space, $R(s^t, a^t) \mapsto r^t \in \mathbb{R}$ is the rewarding rule for transitioning from state $s^t \in S$ by taking action $a^t \in A$ at timestep t, and $T(s^t, a^t, s^{t+1}) \mapsto [0, 1]$ is the probability that taking action a^t in state s^t will lead to state $s^{t+1} \in S$ at the next timestep.

A *Deep Reinforcement Learning* (DRL) algorithm finds an approximately optimal action strategy $\pi(s^t) \mapsto a^t$ for a MDP such that it maximizes the discounted reward $\mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^t \cdot r^{t+\tau} \,\middle|\, \pi\right]$.

An *Action-Value* RL method learns the expected discounted value of taking an action in a state by training an approximated parameterized function Q^θ such that

$$Q^\theta(s^t, a^t) = \mathbb{E}\left[r^t + \gamma \max_{a'} Q^\theta(s^{t+1}, a') \right]. \tag{5}$$

Hence, making the approximately optimal action strategy $\pi(s^t) = \text{argmax}_{a'} Q^\theta(s^t, a')$. The training of Q^θ is based on fitting samples of *experiences* that minimize the squared Bellman loss to the *Temporal Difference* target

$$L^\theta = \mathbb{E}_{e \sim E}\left[\left(Q^\theta(s^t, a^t) - (r^t + \gamma \max_{a'} Q^\theta(s^{t+1}, a')) \right)^2 \right] \tag{6}$$

Each experience $e = \langle s^t, r^t, a^t, s^{t+1} \rangle \in E$ is a tuple of the state s^t that the agent was in at time t, the action it took a^t, the state it arrived at at the next timestep s^{t+1}, and the reward that it received as the result of the state transition.

With a discrete action space, one can enumerate all possible actions in the action-value function for calculating the policy (argmax Q^θ) and the target value (max Q^θ) in a process called *Deep-Q-Learning* (DQL) [12]. However, with continuous action spaces, one needs to find the best action and value with a gradient search on the Q function. This led to the emergence of actor-critic methods such as *Deep Deterministic Policy Gradients* (DDPG) [9]. In DDPG, a separate parameterized action function $\mu^{\theta'}(s^t) \mapsto a^t$ is used that is updated based on moving the parameters in the direction of increasing the Q function by gradient ascent. Specifically, the performance of policy (J) that needs to be maximized can be expressed as:

$$J^{\theta'} = \mathbb{E}_{e \sim E}\left[Q^\theta(s^t, \mu^{\theta'}(s^t)) \right] \tag{7}$$

leading to $\pi(s^t) = \mu^{\theta'}(s^t) = \text{argmax}_{a'} Q(s^t, a')$.

4.2 Multi-Agent Deep Reinforcement Learning

While both DQL and DDPG perform quite well for large state spaces, they lack scalability to large action spaces where the action gradient when updating the policy diminishes, making the policy function impossible to train. Another approach to scalability is to split the environment into K disjoint components and assign one DRL agent to find an approximate optimal policy for the particular component, given the observation from the component, while the agents receive separate rewards.

One such Multi-Agent Reinforcement Learning algorithm is the *Multi-Agent Deep Deterministic Policy Gradient* MADDPG [11] that follows a *centralized training decentralized execution* model where the training of the component's Q function requires access to global state information while execution of the policy function μ is done by only relying on local observations pertaining to the component. In MADDPG, each agent i has a $Q^{\theta_i}(\boldsymbol{s^t}, o_i^t, a_1^t, a_2^t, \cdots, a_k^t) \forall_{k \in K}$ where $\boldsymbol{s}$ that is a joint representation of the state of the system, o_i^t is the observation of agent i from its component, and a_k^t is the action taken

by each agent k. Q^{θ_i} predicts the estimated discounted rewards for agent i and can be updated by reducing the Bellman loss to the temporal difference target. The policy of agent i is a function approximator μ parameterized by θ_i' that can be trained by assuming a similar performance function $J^{\theta_i'}$ to DDPG (Equation 7):

$$J^{\theta_i'} = \mathbb{E}_{e\sim E}\left[Q^{\theta_i}\left(\boldsymbol{s^t}, o_i^t, a_1^t, a_2^t, \cdots, a_k^t, \mu^{\theta_i'}(o_i^t)\right)\right] \forall_{k\neq i} \quad (8)$$

5 HIERARCHICAL MULTI-AGENT REINFORCEMENT LEARNING

At each timestep of the game, the adversary needs to find the approximately optimal perturbations to all the edges in a city network G.

The action space for the low-level agent is $|E|$-dimensional. Given a moderate-sized city such as Anaheim, CA or Chicago, IL, that has 914 and 2950 road links, respectively [20], it is infeasible for a Single-Agent RL algorithm to learn the optimal budget allocation strategy.

This requires that the transit network be broken down into components. Then, an RL agent will be responsible for the edges in the component, observing the information pertaining to the component and only finding the optimal perturbations for that component.

Approaches such as MADDPG will fail in this scenario as the agents will compete over the budget, making the MDP difficult to learn. This makes the need to devise a two-level hierarchical multi-agent reinforcement learning algorithm where the purpose of the high-level agent is to allocate the budget to the components, eliminating the competition over budget, and the purpose of the low-level agent, which itself is comprised of component agents, is to further allocate the perturbation budget between the edges in their component constrained to the allocated budget to the component by the high-level agent.

5.1 K-Means Node Clustering

First, these components can be formed by applying a K-Means clustering algorithm, assuming the distance between two nodes is the shortest path distance given edge weights $w_e = t_e$. Then, each edge $e = uv$ is assigned to the component of its source node u. Algorithm 1 shows a pseudocode for the K-means clustering algorithm. Figure 2 shows the decomposition of the Sioux Falls, ND transportation network with K-Means clustering into four components.

Algorithm 1 K-Means Graph Clustering

Require: A road network graph $G = (V, E)$
  Calculate all-pairs shortest path distance $d_{u,v} : \forall_{u,v\in V}$.
  Select $|K|$ initial nodes as component centers arbitrarily and call them $c_k \in K$.
  for *n_iterations* **do**
    $c_u \leftarrow \text{argmin}_v\, d_{u,v} : \forall_{v\in C}$.
    $c_k \leftarrow \text{argmin}_{c_{k'}}\, \text{argmax}_u\, d_{u,c_{k'}}$ such that $c_u = c_k$
  end for
  $c_e = \langle uv\rangle \leftarrow c_u \forall e \in E$
    return c_e for all $e \in E$ the centroid node for all edges.

5.2 High and Low Level DRL Agents

We assume that the adversarial agent has access to all features of the transportation network G and all rider information l_r^t at all time steps. Thus, it can summarize the information into features that can be used to train the high and low-level agents. Figure 1 summarizes our HMARL architecture.

5.2.1 Low-Level Multi-Agent MADDPG. When graph G is broken down into $|K|$ components, the agent supervising component $k = \hat{G}(\hat{V}_k, \hat{E}_k) \subset G(V, E)$ observes a feature vector of $\hat{\boldsymbol{o}}_k^t = \langle\langle s_e, n_e, \hat{s}_e, m_e, \tilde{s}_e\rangle : \forall_{e\in\hat{E}_k}\rangle$, where $s_e = \sum\{s_r | l_r^t \in V \wedge e \in \varnothing_r^t : \forall_{r\in R}\}$ is the number of vehicles that are currently at a node with an unperturbed shortest path to the destination passing through e, $\hat{s}_e = \sum\{s_r | l_r^t \in V \wedge e = \varnothing_r^t(e_1) : \forall_{r\in R}\}$ is the number of such vehicles that will immediately take e, $m_e = \sum\{s_r \cdot n | l_r^t = \langle e', n\rangle \wedge e' = e : \forall_{r\in R}\}$ is the sum of required timesteps for vehicles traveling e to arrive at its endpoint, and $\hat{s}_e = \sum\{s_r | e \in \varnothing_r^{t-1} : \forall_{r\in R}\}$ is the number of all vehicles taking e as their shortest path at some timestep assuming the perceived travel times to remain unchanged. The agent then outputs a vector of perturbations $\boldsymbol{a}_k^t = \langle a_e^t | e \in \hat{E}_k\rangle$ to perturb all the components' edges. This agent would receive a reward $r_k^t = \sum\{s_r | l_r^t \in \hat{G}(\hat{V}_k\hat{E}_k) : \forall_{r\in R}\}$ as the number of vehicles in its component.

As the low-level agents participate in a cooperative setting with our hierarchical approach, they do not need to see other agents' actions to train their critics. The Q function for each agent can be constructed with $Q^k(\hat{\boldsymbol{o}}_k^t, \hat{\boldsymbol{b}}_k^t, \boldsymbol{a}_k^t)$ with a *Multi-Layer Perceptron* (MLP) such that its output is activated with a *Rectified Linear Unit* (ReLU) as the reward for each component is non-negative, i.e., the number of vehicles in the component. The agent's action function $\mu_k(\hat{\boldsymbol{o}}_k^t, \hat{\boldsymbol{b}}_k^t) \mapsto \boldsymbol{a}_k^t$ can be constructed using an MLP. As the output of the actor function of k-th low-level agent needs to sum to $\hat{b}_k$ to satisfy the budget and allocation constraint, it needs a normalizing function that can be either a Softmax function or 1-norm normalizer. The final perturbations can then be drawn by multiplying budget of the component to its action output $\boldsymbol{a} = \langle \boldsymbol{a_1}\times\hat{b}_1, \boldsymbol{a_2}\times\hat{b}_2, \cdots \boldsymbol{a_k}\times\hat{b}_k\rangle$. The training of μ_k and Q_k functions can be performed according to the MADDPG algorithm (Section 4.2).

5.2.2 High-Level DDPG Agent. The high-level agent H observes an aggregated observation of the components at time t, specifically the number of vehicles in the component and number of vehicles that are making a decision in that component $\boldsymbol{o}_H^t = \langle\langle \sum_e^{\hat{E}_k} n_e, \sum_e^{\hat{E}_k} \hat{s}_e\rangle : \forall_{k\in K}\rangle$, and outputs $\hat{\boldsymbol{b}} \in [0, B]^{|K|}$ such that $\|\hat{\boldsymbol{b}}^t\|_1 = B$ the portion of the budget allocated to each component. The high-level agent is rewarded by the total number of the vehicles in the network $r_H^t = \sum_{\{r\in R | l_r^t \neq d_r\}} s_r$.

Similar to the low-level agent actors, the output of the high-level actor is normalized with either Softmax or 1-norm and then multiplied by the total budget B to allocate each budget to the component. The training of the high level μ_H and Q_H can be performed according to the DDPG algorithm (Section 4.1).

6 EXPERIMENTS

We simulated the framework using benchmark data from [20] and evaluated the effectiveness of HMARL for finding an optimal strategy for false information injection on the Sioux Falls, ND testbed.

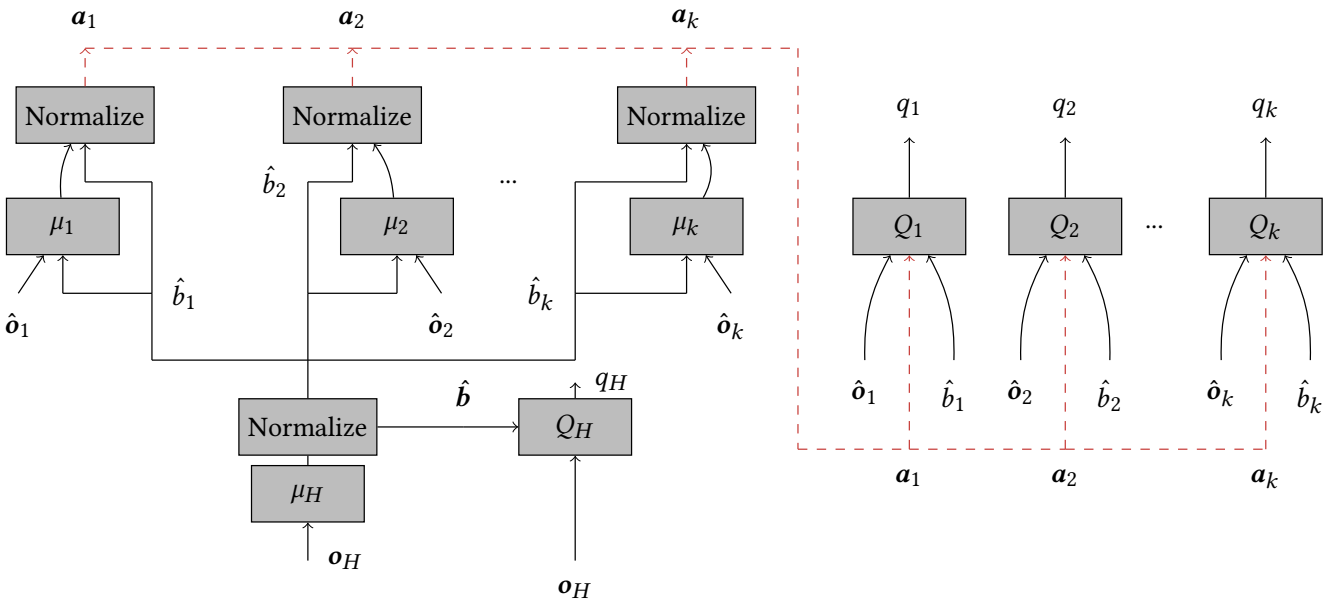


Figure 1: Hierarchical Multi-Agent Deep Reinforcement Learning Architecture. μ_H and Q_H are the high-level agent's actor and critic function approximators, respectively. μ_k and Q_k are the actor and critic function approximators of low-level agent k, respectively. $\boldsymbol{a} = \langle \boldsymbol{a_1} \times \hat{b}_1, \boldsymbol{a_2} \times \hat{b}_2, \cdots \boldsymbol{a_k} \times \hat{b}_k \rangle$ is the perturbations of all edges of the transit graph G where a_k is the perturbations of edges in component k. $\boldsymbol{o_k}$ and $\hat{b}_k$ are the observation of the k-th agent from its component and the proportion of budget allocated to it, respectively. The *Normalize* layer can be constructed using the *Softmax* function or the 1-norm normalization of ReLU-activated actor outputs.

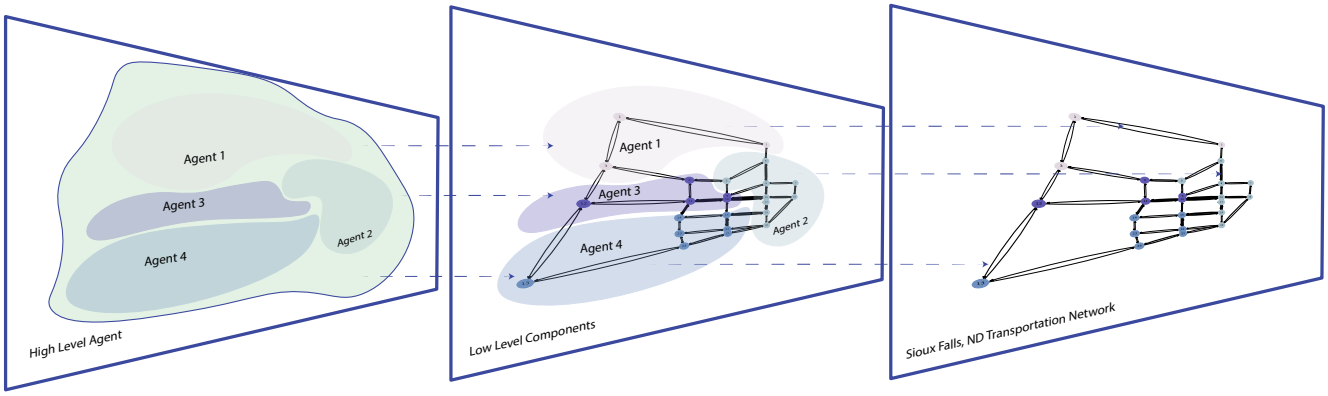


Figure 2: Decomposition of Sioux Falls, ND transportation network into four components, where one low-level agent is responsible for adding perturbation to edges in each component, and one high-level agent is responsible for allocating budget B to each low-level agent. Edge width represents the density of vehicles moving over the edge without any attacker perturbation added.

6.1 Experimental Setup

To make the environment non-deterministic, we randomly increased or decreased r_s by 5% at each training episode's beginning. We simulate the environment by following a vehicle-based simulation based on the state transition rules of Section 3.2.

6.1.1 Hardware and Software Stack. The experiments, including the neural network operations, are done on an Apple MacBook Pro 2021 with an M1 Pro SoC with eight processing cores and 16GB of RAM. None of the experiments, including the neural operations, have been done on the Metal Performance Shaders. The simulation of the environment has been implemented using Python. For neural network operations, we used PyTorch [15]. We used NumPy [5] as our scientific computing library. The source code is available at [1].

6.1.2 Seeds and Hyperparameters. To make sure that the results presented in this article are reproducible, we initialized the random seeds of Numpy, PyTorch, and Python to zero. The hyperparameters used for the simulation and the training of high and low-level agents are presented in Table 1 and the neural network architectures are presented in Table 2.

6.2 Heuristics

We used a *Greedy* heuristic as our baseline strategy. In the greedy approach, the adversarial agent counts the number of vehicles s_e passing through each edge e as their unperturbed shortest path to their destination. Then its applied perturbation will be

$$\boldsymbol{a} = \frac{\langle s_e : \forall e \in E \rangle}{\sum_{e \in E} s_e} \times B.$$

When running the ablation study (see Figure 3) and testing the high-level and low-level agents separately, we replaced the high-level with a *proportional allocation*, meaning that each component agent gets a proportion of the budget relative to the number of vehicles making a decision in that component. Further, the low-level agent can be replaced with a local greedy that perturbs the edges in its component relative to the number of vehicles passing through the edges:

$$a_k = \frac{\langle s_e : \forall e \in E_k \rangle}{\sum_{e \in E} s_e} \times \hat{b}_k.$$

6.3 Numerical Results

After the initialization of the environment, as the HMARL is off-policy, it can draw experiences of states, actions, next states, and rewards from the environment by taking either random actions or by taking Ornstein-Uhlenbeck [25] noise added to actions outputted by the low-level agent. Using these experiences, all actors and critics can be updated simultaneously. Algorithm 2 shows the training workflow of the HMARL.

When agents are trained simultaneously, the low-level agent should have lower learning rates as it needs the high-level agent to learn its behavior but should account for more steps in the future with a higher discount factor γ.

Figure 3 shows the result of the training with an ablation study on the Sioux Falls, ND transportation network [20]. This network has 24 nodes and 76 edge links. We ran HMARL with different attack budgets. As expected, the HMARL performs better by 10-50% depending on the budget, making it a viable solution to the scalability of Deep Reinforcement Algorithms.

Table 1: List of Hyperparameters

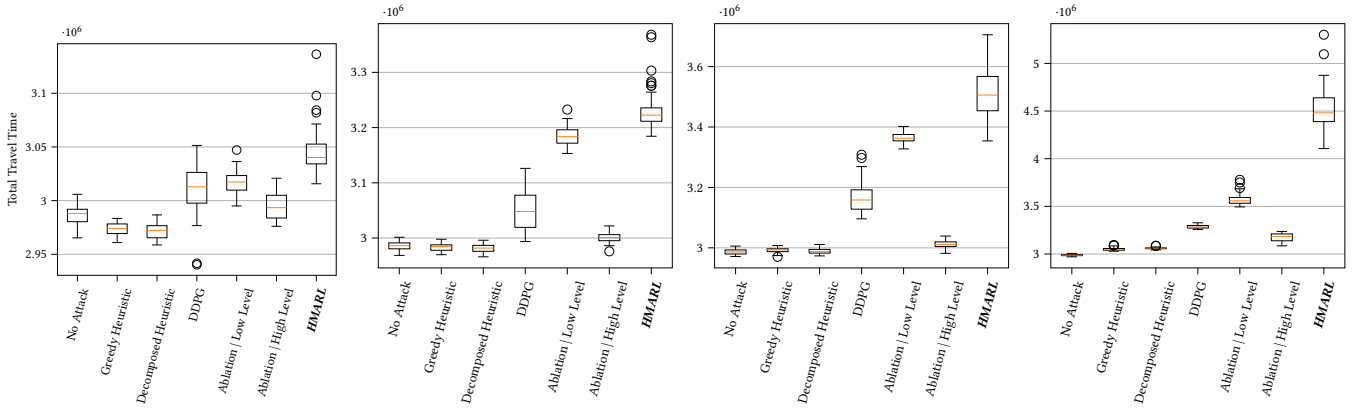
Hyperparameter	Value
Environment	
Training Horizon	400
Evaluation Horizon	50
$\|K\|$ Number of Components	4
Total Training Steps	200,000
Randomizing Factor of Number of Vehicles	0.05
Common	
τ_H, τ_k target network transfer rate	0.001
Training Batch Size	64
Experience Replay Buffer Size	50,000
Stand Alone High Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_H	0.99
τ_H	0.001
Noise Decay Steps	10,000
Stand Alone Low Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_k	0.99
Noise Decay Steps	30,000
Hierarchical	
Low-Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_k	0.9
Noise Decay Steps	10,000
High-Level	
μ_H Learning Rate	0.00001
Q_H Learning Rate	0.001
γ_H	0.99
τ_H	0.001
Noise Decay Steps	30,000
Standalone DDPG	
μ Learning Rate	0.00001
Q Learning Rate	0.001
γ	0.99
Noise Decay Steps	30,000

7 DISCUSSION

First, we conducted a hyperparameter optimization with a simple grid search and reported only the hyperparameters that work best. Further evaluation of hyperparameters with more sophisticated search mechanisms, such as Bayesian Search [23], is required.

Based on the experiments conducted, it is important to note that the shortest path routing approach for traffic does not always result in an optimal network flow solution due to network congestion. It is possible for a non-optimal attack to actually reduce the total travel time of the vehicles by decreasing congestion on regular congested paths. For example, see Figure 3a with no attack ("No Attack") and Greedy ("Greedy Heuristic", "Decomposed Heuristic") strategies.

Further, the starting nodes for the k-means clustering can impact the training process and need to be thoroughly examined. There should be a correlation between certain metrics, such as the maximum diameter of components, the number of nodes in components, the balance of nodes and edges between the components of the outputted clusters, and the performance of HMARL, which has not yet been analyzed. Additionally, other graph decomposition methods have not been evaluated. It is worth considering a different decomposition algorithm that tends to produce more balanced components, as this may lead to a more stable training process and better performance of the HMARL.



(a) $B = 5$ **(b)** $B = 10$ **(c)** $B = 15$ **(d)** $B = 30$

Figure 3: Ablation study of HMARL on the Sioux Falls network. *"No Attack"* pertains to no attack on the network. *"Greedy Heuristic"* is a network greedy (see Section 6.2) attack. *"DDPG"* applies the general-purpose DDPG algorithm network-wide. In the remaining columns, the network is divided into four components. In *"Decomposed Heuristic,"* the low-level actors are low-level greedy agents, with the high-level being a proportional allocation to the number of vehicles in each component. In *"Ablation | Low Level,"* the high-level agent is the proportional allocation heuristic, while its low-level is the MADDPG approach. In *"Ablation | High Level,"* the low-level is the greedy heuristic, while the high-level is a DDPG allocator RL agent. *"HMARL"* is our HMARL approach. Here, the low-level MADDPG and high-level DDPG components have been trained simultaneously.

Table 2: Neural Network Architecture

Hyperparameter	Value
High-Level	
Actor μ_H	
Number of hidden layers	2
Sizes of layers	[256, 128]
Activation function	ReLU
Optimizer	Adam
Critic Q_H	
Number of hidden layers	2
Sizes of hidden layers	[128, 128]
Activation function	ReLU
Optimizer	Adam
Low-Level	
Actor μ_k	
Number of hidden layers	2
Sizes of hidden layers	[512, 512]
Activation function	[ReLU, ReLU, Sigmoid]
Optimizer	Adam
Critic Q_k	
Number of hidden layers	2
Sizes of hidden layers	[128, 128]
Activation function	ReLU
Optimizer	Adam

There is a trade-off between the number of components, the performance of high-level, low-level, and the hierarchical approach. With too many components ($|K| \rightarrow |V|$), or with too few components ($|K| \rightarrow 1$), the hierarchical approach will be equivalent to a single-agent RL. The best number of components should be extracted experimentally.

Algorithm 2 Hierarchical Multi-Agent Reinforcement Learning

Require: A road network graph $G = (V, E)$; Set of Riders R;
  Initialize environment *env*
  Initialize Replay Buffer E.
  Run K-Means Clustering to acquire components
  $s \leftarrow env.reset()$
  for step $\Rightarrow$ total steps **do**
    $\hat{\boldsymbol{b}} \leftarrow \mu_H(\hat{s})$
    $\boldsymbol{a} \leftarrow \odot\mu_k(\hat{b}, s) + \mathcal{N}$
    Normalize $\boldsymbol{a}$
    $s', r \leftarrow env.step(\boldsymbol{a})$
    $E \leftarrow E \cup \langle s, s', \boldsymbol{a}, r\rangle$
    $s \leftarrow s'$
    if $env.done()$ **then**
      $s \leftarrow env.reset()$
    end if
    Sample $\hat{E} \sim E$
    Update $Q_H, \mu_H, Q_k, \mu_k \forall_{k \in K}$ with $\hat{E}$
  end for

When analyzing heuristics and HMARL, it is essential to consider the sparsity of vehicles. The rider data in Sioux Falls is dense, with 100-500 vehicles traveling between each pair of nodes. On the contrary, the rider data in Eastern Massachusetts is sparse, with no vehicles present for over 75% of pairs of nodes.

Currently, the feature extractor of the state representation is a Multi-Layer Perceptron. As the state is inherently a graph, Graph Convolutional Networks [13] or Graph Attention Networks [21] can be incorporated to improve the accuracy of the Q and μ functions.

7.1 Conclusion

Our research focused on the impact of adversarial influence on transportation networks. We investigated how drivers could be manipulated by injecting false information into navigation apps through a computational approach. We developed an adversarial model that included a threat actor capable of manipulating drivers by increasing their perceived travel times, leading them to take suboptimal and longer routes. To accomplish this, we created a computational framework based on Hierarchical Multi-Agent Deep Reinforcement Learning (HMARL) to determine an optimal strategy for data manipulation. Our simulation of the Sioux Falls, ND transportation network showed that the adversary could increase the total travel time of all drivers by 50%.

7.2 Future Direction

To effectively combat data attacks, it is crucial to have a robust defense mechanism. One of the most significant hurdles is precisely identifying data manipulation attacks. Our research has yielded positive outcomes, and we aim to tackle the problem of detecting such false-data injection attacks by utilizing sophisticated machine-learning methods and competitive multi-agent reinforcement learning algorithms.

ACKNOWLEDGMENTS

This material is based upon work sponsored by the National Science Foundation under Grants CNS-1952011, IIS-1905558, IIS-1903207, and IIS-2214141 and by the Army Research Office under Grant W911NF1810208. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and of the Army Research Office.

REFERENCES

[1] 2023. Transportation Security Research. https://github.com/tahaeghtesad/TransportAttack/releases/tag/v1.0. [Online; accessed 11-February-2024].

[2] Qi Alfred Chen, Yucheng Yin, Yiheng Feng, Z Morley Mao, and Henry X Liu. 2018. Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control. In *Proceedings 2018 Network and Distributed System Security Symposium (NDSS 2018)*.

[3] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1625–1634.

[4] Yiheng Feng, Shihong Huang, Qi Alfred Chen, Henry X Liu, and Z Morley Mao. 2018. Vulnerability of traffic control system under cyberattacks with falsified data. *Transportation Research Record* 2672, 1 (2018), 1–11.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[6] Aron Laszka, Waseem Abbas, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. 2019. Detection and Mitigation of Attacks on Transportation Networks as a Multi-Stage Security Game. *Computers & Security* 87 (November 2019), 101576.

[7] Aron Laszka, Bradley Potteiger, Yevgeniy Vorobeychik, Saurabh Amin, and Xenofon Koutsoukos. 2016. Vulnerability of transportation networks to traffic-signal tampering. In *Proceedings of the 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 1–10.

[8] C Levy-Bencheton and E Darra. 2015. Cyber security and resilience of intelligent public transport: good practices and recommendations. (2015). https://doi.org/10.2824/778225

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[10] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, and Linqiang Ge. 2018. Data integrity attacks against dynamic route guidance in transportation-based cyber-physical systems: Modeling, analysis, and defense. *IEEE Transactions on Vehicular Technology* 67, 9 (2018), 8738–8753.

[11] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems* 30 (2017).

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[13] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2021. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. arXiv:1810.02244 [cs.LG]

[14] Thanh Thi Nguyen and Vijay Janapa Reddi. 2021. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems* 34 (2021), 3779–3795. Issue 8.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[16] Simone Raponi, Savio Sciancalepore, Gabriele Oligeri, and Roberto Di Pietro. 2021. Road Traffic Poisoning of Navigation Apps: Threats and Countermeasures. *IEEE Security & Privacy* 20, 3 (2021), 71–79.

[17] Jack Reilly, Sébastien Martin, Mathias Payer, and Alexandre M Bayen. 2016. Creating complex congestion patterns via multi-objective optimal freeway traffic control with application to cyber-security. *Transportation Research Part B: Methodological* 91 (2016), 366–382.

[18] Ben Schoon. 2020. Google Maps 'hack' uses 99 smartphones to create virtual traffic jams. https://9to5google.com/2020/02/04/google-maps-hack-virtual-traffic-jam/ Accessed 14-August-2023.

[19] Liang Tong, Aron Laszka, Chao Yan, Ning Zhang, and Yevgeniy Vorobeychik. 2020. Finding Needles in a Moving Haystack: Prioritizing Alerts with Adversarial Reinforcement Learning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. 946–953.

[20] Transportation Networks for Research Core Team. 2020. Transportation Networks for Research. https://github.com/bstabler/TransportationNetworks/. Online; accessed 20 July 2023.

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[22] Marcin Waniek, Gururaghav Raman, Bedoor AlShebli, Jimmy Chih-Hsien Peng, and Talal Rahwan. 2021. Traffic networks are vulnerable to disinformation attacks. *Scientific Reports* 11, 1 (2021), 5329.

[23] Wikipedia contributors. 2023. Bayesian search theory — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Bayesian_search_theory&oldid=1154910859 [Online; accessed 10-October-2023].

[24] Wikipedia contributors. 2023. Fort Lee lane closure scandal — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Fort_Lee_lane_closure_scandal&oldid=1161650250 [Online; accessed 15-August-2023].

[25] Wikipedia contributors. 2023. Ornstein–Uhlenbeck process — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Ornstein%E2%80%93Uhlenbeck_process&oldid=1175439055 [Online; accessed 10-October-2023].

[26] Zhaoyang Yang, Kathryn Merrick, Lianwen Jin, and Hussein A Abbass. 2018. Hierarchical deep reinforcement learning for continuous action control. *IEEE transactions on Neural Networks and Learning Systems* 29, 11 (2018), 5174–5184.

[27] Yu Zhang, Zhiyu Mou, Feifei Gao, Ling Xing, Jing Jiang, and Zhu Han. 2020. Hierarchical deep reinforcement learning for backscattering data collection with multiple UAVs. *IEEE Internet of Things Journal* 8, 5 (2020), 3786–3800.