Master's Thesis

CALIBRATING TRAFFIC MODELS AND OPTIMIZING AGENTS IN ROBOCUP RESCUE SIMULATION USING REINFORCEMENT LEARNING

Jaebak Hwang

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

2022

CALIBRATING TRAFFIC MODELS AND OPTIMIZING AGENTS IN ROBOCUP RESCUE SIMULATION USING REINFORCEMENT LEARNING

Jaebak Hwang

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

CALIBRATING TRAFFIC MODELS AND OPTIMIZING AGENTS IN ROBOCUP RESCUE SIMULATION USING REINFORCEMENT LEARNING

A thesis/dissertation submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Master of Science

Jaebak Hwang

12.16.2021 of submission

Approved by

Advisor

Tsz-Chiu Au

CALIBRATING TRAFFIC MODELS AND OPTIMIZING AGENTS IN ROBOCUP RESCUE SIMULATION USING REINFORCEMENT LEARNING

Jaebak Hwang

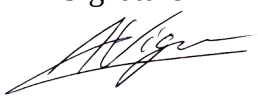
This certifies that the thesis/dissertation of Jaebak Hwang is approved.

12.16.2021 of submission

Signature

Advisor: Tsz-Chiu Au

Signature



Committee Member : Prof. Antoine Vigneron

Signature

Committee Member : Prof. Yuseok Jeon

Signature

Abstract

Simulators need calibration with real data to match the simulation results with reality. Based on the calibrated simulators, we learn artificial intelligence models to find the best way to achieve goals in simulation. This thesis has two parts: (1) how to calibrate a traffic model using real traffic data collected in real-time; and (2) how to train an agent to act optimally in a simulated environment that models the situation after a large-scale earthquake. In the first part, the widespread deployment of inductive-loop traffic detectors allows us to obtain a massive amount of traffic data in real-time. A key step of utilizing the data is to use the data to fit a traffic model. Simultaneous perturbation stochastic approximation (SPSA) and its variants are popular techniques for the calibration of dynamic traffic assignment (DTA) models by searching for Origin-Destination (OD) matrices that fit the data. However, the performance of SPSA cannot scale with modern multi-core CPU architectures due to its sequential nature. This paper proposes the use of the active covariance matrix adaptation evolution strategy (Active-CMA-ES) for optimizing OD matrices in microscopic traffic simulation. CMA-ES is a black-box optimization algorithm that was found highly effective in many domains. According to our case study in Ulsan, South Korea, Active-CMA-ES outperforms Restart-WSPSA, one of the best SPSA algorithms, in terms of the calibration error and the running time. Moreover, the running time of Active-CMA-ES decreases as the number of parallel simulation processes increases. In the second part, as a learning artificial intelligence model, we have been developing a deep reinforcement learning software package that finds optimal policy to extinguish the fire. Effective acting of agents in disaster situations is essential to rescuing citizens in danger, but the diversity of the initial environments and rapidly changing hazardous environments make the training of an agent difficult. Since it is not possible to directly test an agent in a real disaster environment, we opt for training an agent by reinforcement learning in a disaster simulator. We utilized a simulator called RoboCup Rescue Simulation(RCRS) that models how fire spreads in a city after an earthquake and used deep reinforcement learning with supervised learning and self imitation methods to train a team of agents to extinguish the fire. In the RCRS map of this study, the PPO algorithm with self-imitation and behavior cloning, which is the most effective methods what we tested, can extinguish all fire successfully in the case of fixed initial fired building location and can extinguish all fire with high probability in the case of the random initial fired building location.

Contents

List of Figures

I Introduction

These days, sophisticated simulators are being developed and used for various purposes. Simulators use real data to implement results similar to reality, and based on such simulators, they learn artificial intelligence models to find the best way to achieve goals in simulations. Our thesis has a total of two parts: a part that implements similar results using real data of a simulator and a part that learns an artificial intelligence model in a sophisticated simulator to find the best way.

Nowadays, we can obtain many traffic data in real-time from inductive loop sensors installed on roads. Data-driven approaches for analyzing traffic flows could potentially be more accurate than traditional macroscopic traffic flow modeling. The key step of utilizing the data is to use the data to fit a traffic model such as a microscopic traffic simulation model. Dynamic traffic assignment (DTA) models have been used to compute the equilibrium traffic flow in a traffic simulation. For example, PTV Vissim[1] uses a module called Dynamic Assignment to iteratively compute the traffic flow in equilibrium given a map and an OD matrix. To calibrate a DTA model, a calibration algorithm adjusts the OD matrix until the equilibrium traffic flow matches the collected data.

Simultaneous perturbation stochastic approximation (SPSA) is a popular approach for calibrating DTA models. Lu Lu et al. [1] proposed weighted SPSA (W-SPSA) for estimating OD matrices in dynamic settings in which the OD matrix is assumed to be non-static, and the DTA estimation algorithm has to estimate several OD matrices in a row for consecutive time intervals. Huan Yang et al. [2] presented a modified SPSA algorithm called Restart-SPSA, which improves SPSA by a restart strategy for dynamic origin-destination estimation. Both approaches showed that SPSA could be quite effective in calibrating DTA models. However, they are too slow for some applications in which we need to estimate the traffic flow as quickly as possible. Since both methods use gradient descent in which the search steps must be executed in sequential order, it is difficult to reduce their running times.

Recently, there have been more cases of solving problems by learning artificial intelligence models from various simulators. Silver et.al developed AlphaGo [3] that plays the game Go and won the human champion, and Vinyals et.al developed Alphastar [4], which plays StarCraft2 as Grandmaster Level. However, research on finding the best model to find the optimal policy of disaster agents in disaster simulators such as fires or earthquakes is not yet active. This is because disaster measures such as overpowering fires often become uncontrollably difficult due to a little mistake in the early stages and a small difference in behavior in early stages makes significantly different results. Thus, we have been developing a software package to find the optimal policy in the fire simulator.

The research statement of this thesis is:

[1] `https://www.ptvgroup.com/en/solutions/products/ptv-vissim`

What is the best way to (1) calibrate simulation models such as microscopic traffic simulation models or disaster simulation models in parallel, and then (2) train a team of agents such as autonomous vehicles and fire brigades to act optimally in the simulation by deep reinforcement learning?

This research has two major parts: (1) Calibrating Dynamic Traffic Assignment Models by Parallel Search; and (2) Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning. In the first part, we propose to replace SPSA by active covariance matrix adaptation evolution strategy (Active-CMA-ES) [5] in calibrating DTA models. CMA-ES, proposed by Hansen and Ostermeier, is an evolution strategy for black-box optimization [6]. CMA-ES is highly effective in many application domains, including robotics [7]. Like the classical evolution strategies, CMA-ES evolves a population of solutions in parallel, making it suitable for parallel processing in modern multi-core CPU architectures. Some previous works have already used evolution algorithms or genetic algorithms for calibrating traffic simulation [8, 9]. However, their work address slightly different calibration problems, and their focus are not on achieving parallel search. Moreover, CMA-ES could outperform these classical evolution strategies, as demonstrated in other application domains. Active-CMA-ES is a modified version of CMA-ES with new update rules for covariance matrices, making it more suitable for noisy, non-smooth, non-continuous, and non-convex black-box optimization problems, including our DTA calibration problem. In the second part, we developed a deep reinforcement learning software package that finds optimal policy to extinguish the fire using self-imitation and behavior cloning. It works on a famous disaster simulator, RoboCup Rescue Simulation(RCRS) [2] , and a toy simulator developed based on RCRS. There have been no cases of attempting to incorporate deep reinforcement learning into RCRS, and various algorithms have been tested to learn the model, while also incorporating self-limitation techniques and supervised learning.

[2] http://rescuesim.robocup.org/competitions/agent-simulation-competition/

II Related Works

This section introduces the related works about this thesis. This thesis has two major topics as the goal, (1) Calibrating Microscopic Traffic Simulations by Parallel Search; and (2) Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning. Therefore, this section also has two parts of the related works, the first one is related to traffic simulations and the second one is related to deep reinforcement learning.

2.1 Calibrating Microscopic Traffic Simulator

One of the early works on the calibration of traffic models was based on genetic algorithms (GA) [9, 10]. However, as the size of OD matrices increases, GA requires a large population per generation and a larger number of iterations to converge to a solution. Thus, these early attempts were later overshadowed by gradient-descent methods, notably simultaneous perturbation stochastic approximation (SPSA) [11–13]. SPSA is a derivative-free optimization algorithm that estimates gradients by sampling, making it suitable for the calibration of traffic models which have no obvious derivative functions. When compared with GA, SPSA requires less computation time per iteration and can converge much faster. Huan Yang et al. [2] presented Restart-SPSA, which applies a restart strategy to improve the performance of SPSA. Lu Lu et al. [1] proposed Weighted SPSA (W-SPSA), which utilizes the correlation between OD matrices and the ground truth data to generate a weight matrix that gives more weight to certain elements in the OD matrix during gradient descent.

One drawback of SPSA is that there is no obvious way to parallelize its search process. In this paper, we consider covariance matrix adaptation evolution strategy (CMA-ES) [6], a famous black-box optimization algorithm that has been quite successful in hundreds of applications [14]. Like GA, CMA-ES maintains a pool of solution candidates in each generation, and the evaluation of the fitness of the candidates can be done in parallel due to the lack of dependency among the candidates. Although CMA-ES takes more time to evolve one generation when compared with SPSA, CMA-ES is much easier than SPSA in escaping from local minima since the gradient are estimated by multiple sample points. Bojan Kostic et al. [15] compared SPSA, CMA-ES, and Nelder-Mead's Simplex algorithm (NMSIM) [16], and showed that both SPSA and CMA-ES are much faster than NMSIM when calibrating a macroscopic simulator based on traffic equilibrium for first-order dynamic traffic simulation models [17, 18]. In this paper, we calibrate a microscopic traffic simulator by Active-CMA-ES [5], an enhanced version of CMA-ES with faster convergence by using negative results to update the covariance matrix.

2.2 Finding Optimal Policy of Simulations Using Deep Reinforcement Learning

Many studies show that Deep Reinforcement Learning can solve complex problems such as a game. In the games, a deep reinforcement learning model finds the optimal policy of players to win the games. Silver et.al developed AlphaGo [3] that plays the game Go and won the human champion, and Vinyals et.al developed Alphastar [4], which plays StarCraft2 as Grandmaster Level. In this paper, we aim to find

the optimal policy of disaster agent in simulator RCRS instead of finding an optimal policy of players in the games, but it is not different technically and has similar complexity.

There are lots of studies using classical artificial intelligence algorithms for RCRS agents' effective actions, but only a few studies applied the machine learning model. In Robocup Rescue competitions, most teams used clustering methods to simplify the map and generate graphs based on the result of the k-clustering algorithm. Based on the simplified map, most teams, such as team AURA and MRL, use A*search-based algorithm, and a few teams, such as team RoboAKUT, use Dijasktra's shortest path algorithm to find the path for each agent but nobody tried to apply machine learning algorithms for it. To select the actions of agents, agents follow a rule-based algorithm instead of artificial intelligence. In this paper, we applied one of the machine learning algorithms, the deep reinforcement learning model to select the action of agents without a k-clustering algorithm or A*search algorithm.

III Technical Background

This section introduces the technical backgrounds related to this thesis. This thesis has two major topics as the goal, (1) Calibrating Microscopic Traffic Simulations by Parallel Search; and (2) Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning. Therefore, this section also has two parts of the technical backgrounds also.

3.1 Calibrating Microscopic Traffic Simulations

This subsection introduce technical backgrounds related with first goal, Calibrating Microscopic Traffic Simulations by Parallel Search.

VISSM

VISSIM is a widely used microscopic traffic simulator, with the Dynamic Assignment module for DTA calculation. Microscopic traffic simulator express a realistic and detailed overview about the status of the traffic flow and impacts, with the possibilities to define multiple what-if scenarios.

SPSA

SPSA is a gradient descent algorithm that starts from an initial point of the parameter vector and then updates the parameter by using gradient approximation until the error converges to zero. Let $\hat{X}_k$ be the estimated vector of parameters in the kth iteration. The iterative process of the SPSA algorithm is based on the following equation:

$$\hat{X}_{k+1} = \hat{X}_k - a_k \hat{g}_k(\hat{X}_k)$$

where $\hat{g}_k(\hat{X}_k)$ is the estimated gradient column vector at $\hat{X}_k$, and a_k is a small positive number as the gain factor of $\hat{g}_k(\hat{X}_k)$. In other words, a_k is the main factor of the step size per iteration. In each iteration, the gradient is estimated by:

$$\hat{g}_k(\hat{X}_k) = \frac{(Z(\hat{X}_k + c_k\Delta_k) - Z(\hat{X}_k - c_k\Delta_k))}{2c_k} \begin{bmatrix} \Delta_{k1}^{-1} \\ \Delta_{k2}^{-1} \\ \vdots \\ \Delta_{kN}^{-1} \end{bmatrix} \tag{1}$$

where 1) Δ_k is a $N \times 1$ vector those elements are generated by the Bernoulli distribution with the values of ± 1, 2) N is the dimension of $\hat{X}$, and 3) c_k is a positive number called the gain factor of Δ_k. Both a_k and c_k get smaller as k increases according to the following equations:

$$a_k = a/(A + k + 1)^{\alpha} \tag{2}$$

$$c_k = c/(k+1)^{\gamma} \tag{3}$$

where a, c, α, γ, and A are parameters that are tuned manually. According to [12], we can pick $\alpha = 0.602$ and $\gamma = 0.101$. If a is too large, the oscillation of the estimated parameter vector occurs, and

the algorithm cannot converge. If a is too small, the algorithm will converge slowly and may easily be trapped at local minima.

Genetic Algorithm

Genetic algorithm is a optimization algorithm based on natural selection. It uses a method to create a better next generation by creating offspring and mimicking natural behavior such as crossover or mutation.

Algorithm 1 Genetic Algorithm

1: Create population size λ offspring with uniform distribution.
2: Evaluate the objective function for all offspring and discard the lower-ranked half of them.
3: For each offspring, make a pair of them and shuffle their value with a probability of crossover rate.
4: For each offspring, change their value with a very small probability of mutation rate.
5: Generate new offspring with size $\lambda/2$.
6: Repeat 2 to 5 until terminate

Restart-SPSA

Restart Strategy: Restart-SPSA is an extension to SPSA with the following restart strategy. First, Restart-SPSA runs SPSA with a large value of a for a given number of iterations. Let $\hat{X}_k$ be the best parameter in the execution of SPSA. Then Restart-SPSA reruns SPSA from scratch with 1) a smaller value of a and 2) $\hat{X}_k$ as the initial search point. Let $\hat{X}'_k$ be the best parameter in the second execution of SPSA. Then Restart-SPSA returns $\hat{X}'_k$ as the solution. Typically, Restart-SPSA can find better solutions than SPSA.

W-SPSA

Weighted-SPSA (W-SPSA) was introduced in [1]. The idea is to modify the estimated gradient in Eq. 1 by a weight matrix W such that the ith element in the gradient becomes larger (or smaller) if the ith element in $\hat{X}_k$ has a stronger (or weaker) influence to the convergence of $\hat{X}$. More precisely, W-SPSA modifies Eq. 1 by computing the ith element in $\hat{g}_k(\hat{X}_k)$ using the following equation instead:

$$\hat{g}_{ki}(\hat{X}_k) = \frac{\acute{Z}(\hat{X}_k + c_k\delta_k) - \acute{Z}(\hat{X}_k - c_k\delta_k)}{2c_k\delta_{ki}} W_i, \tag{4}$$

where $\acute{Z}(\theta)$ and W_i are computed by the following equations:

$$\acute{Z}(X_k) = \begin{bmatrix} w_1((Y(X_k) - \hat{Y}(X_k)) \bigotimes ((Y(X_k) - \hat{Y}(X_k)) \\ w_2((X_k - X_{k-1}) \bigotimes (X_k - X_{k-1})) \end{bmatrix} \tag{5}$$

$$W_{i,m} = \frac{d_{i,j}}{\sum_{j=1}^{M} d_{i,j}}, \tag{6}$$

where $Y(X_k)$ is the observed traffic measurements with OD matrix X_k, $\hat{Y}(X_k)$ is the simulated measurements, X_{k-1} is the prior value of parameters, w_1 and w_2 are constants, and $d_{i,j}$ is the increment or decrement of the jth observation when the ith element increases. In [1], the same weight matrix W is used in all time intervals. However, in this paper, we generated different W in each time interval, using the calibrated OD matrix in the previous time interval. We also apply the restart strategy to W-SPSA to form *Restart-WSPSA* as in [2].

CMA-ES

CMA-ES is a blackbox optimization algorithm based on evolution strategies. Active-CMA-ES is an enhanced version of CMA-ES [5]. CMA-ES generates λ offspring based on mutation strength σ and $n \times n$ covariance matrix C from the search point θ, and updates C using n-dimension search path vectors p_σ and p_c. The search path vectors accumulate the information of high-ranking offsprings. Active-CMA-ES also uses low-ranking offsprings for updating the covariance matrix C negatively. This negative update makes CMA-ES adaptation much faster. The search path vectors are initialized to zero, and the covariance matrix is initialized to a unity matrix. More precisely, Active-CMA-ES proceeds according to the following steps:

1. For N number of parameters, $C = BD(BD)^T$ where $n \times n$ matrix B is normalized eigenvectors of C, and D is a diagonal $n \times n$ matrix. The diagonal elements are the square roots of the eigenvalues of C.

2. Generate an offspring X based on the following equation:

$$X_i = \theta + \sigma BDz_i, \tag{7}$$

   where z_i is a mutation vector with n elements, generated from a normal distribution, for $1 \leq i \leq \lambda$.

3. Calculate the loss $Z(X_i)$ and sort the results by $z_{k;\lambda}$, where $k;\lambda$ means the index of the kth ranked result in the population. Compute the average of μ rank mutation vectors:

$$E^z = \frac{1}{\mu} \sum_{k=1}^{\mu} z_{k;\lambda} \tag{8}$$

   In this paper, we set $\mu = \lambda/2$.

4. Update the search point:

$$\theta = \theta + \sigma BDE^z$$

5. Update the search paths:

$$p_c = (1 - c_C)p_c + \sqrt{\mu c_C(2 - c_C)}BDE^z \tag{9}$$

$$p_\sigma = (1 - c_\sigma)p_\sigma + \sqrt{\mu c_\sigma(2 - c_\sigma)}BE^z \tag{10}$$

   where $c_C = c_\sigma = \frac{4}{n+4}$.

6. Update the covariance matrix:

$$C = (1 - c_{cov})C + c_{cov} p_c p_C^T + \beta E, \tag{11}$$

where

$$E = BD\left(\frac{1}{\mu}\sum_{k=1}^{\mu} z_{k;\lambda} z_{k;\lambda}^T - \frac{1}{\mu}\sum_{k=\lambda-\mu+1}^{\lambda} z_{k;\lambda} z_{k;\lambda}^T\right) BD^T \tag{12}$$

and

$$c_{cov} = \frac{2}{(n+\sqrt{2})^2},$$

where β is set according to the function type.

7. Update the mutation strength:

$$\sigma = \sigma exp(\frac{\|p_\sigma\| - X_n}{d_\sigma X_n}), \tag{13}$$

where $X_n = \sqrt{n}(1 - 1/(4n) + 1/(21n^2))$ is an approximation of the expectation value of the length of a random vector with a normal distribution and $d_\sigma = 1 + 1/c_\sigma$ is the damping factor.

In Eq. 12, the term $\frac{1}{\mu}\sum_{k=1}^{\mu} z_{k;\lambda} z_{k;\lambda}^T$ denotes the positive adaptation based on the high-ranking offsprings from rank 1 to μ, and the term $-\frac{1}{\mu}\sum_{k=\lambda-\mu+1}^{\lambda} z_{k;\lambda} z_{k;\lambda}^T$ denotes the negative adaptation based on the low-ranking offsprings from $\lambda - \mu + 1$ to λ.

CMA-ES and Active-CMA-ES have many parameters, but most of them are set by some recommended equations. In most cases, we only need to tune σ. For more information about how to set the parameters, please refer to [19].

3.2 Finding Optimal Policy of Simulations using Deep Reinforcement Learning

This subsection introduce technical backgrounds related with seconds goal, Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning.

Reinforcement Learning

Reinforcement learning is one of machine learning area about how intelligent agent act for specific environment. Reinforcement learning express model as Markov decision process: A set of environment and agent states S, A set of action of agents A, a probability $P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ that transition from state s to state s' at time t under action a, and immediate reward $R_a(s,s')$ after transition from state s to state s'. Probability of actions for specific state called policy π. Agent need to explore environment for reward of each action and update policy or value function, which is expectation of total reward from a state.

$$\pi(s,a) = P[a|s] \tag{14}$$

$$V^\pi(s) = E_\pi\{R_\pi | s_t = s\} \tag{15}$$

Learning policy or value function properly is an important task of deep learning.

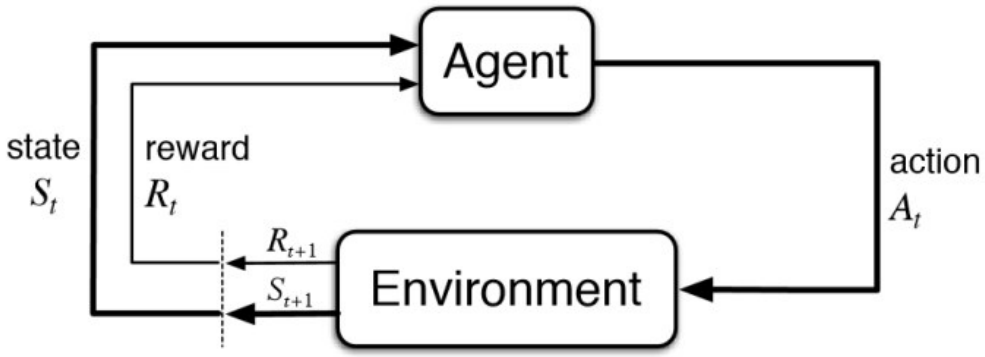


Figure 1: Agent selects action based on policy and environment return changed state and reward.

Deep Reinforcement Learning

In Reinforcement learning, learning policy or value function properly is an important task but it's hard for huge or complex environmental problems. In Deep Reinforcement Learning, policy and value functions are represented as a deep neural network, and these neural networks make deep learning can solve complex and huge problems. Since deep reinforcement learning still follows the reinforcement learning format, the researcher should set environment structure and rewards for model learning.

Actor-Critic Algorithm

The actor-Critic algorithm uses two deep neural networks for policy and value function. Actor-network return probability distribution of actions for a state as policy, and critic network return expected reward of each action for a state as Q function. Q function is a value-action function which is the expectation of total reward from a state with an action. Because two deep neural networks update separately, the Actor-Critic algorithm reduces the variance of the value function and makes convergence speed of value function faster.

$$Q^{\pi}(s,a) = E_{\pi}\{R_{\pi}|s_t = s, a_t = a\} \tag{16}$$

The RoboCup Rescue Simulation (RCRS)

The RoboCup Rescue Simulation (RCRS) is one of the famous disaster simulators, which is used in RoboCup competition. It is a large-scale multi-agent system that aims to study disaster response management, such as earthquake or fire, and it used a target disaster simulator to find the optimal policy of disaster agents in this study. In the RCRS, first, the simulator simulates the earthquake occurrence and generates blockades on the roads that agents or civilians can not pass. After the earthquake, certain buildings get fires and the simulations start. RCRS simulator progresses a unit of time called time step and as time step increases, fire spreads to neighbor buildings. The fire brigade can extinguish the fire and the police force can remove blockades on the roads to pass those roads. The ambulance team can transport civilians who can not move themselves the cause of injures.

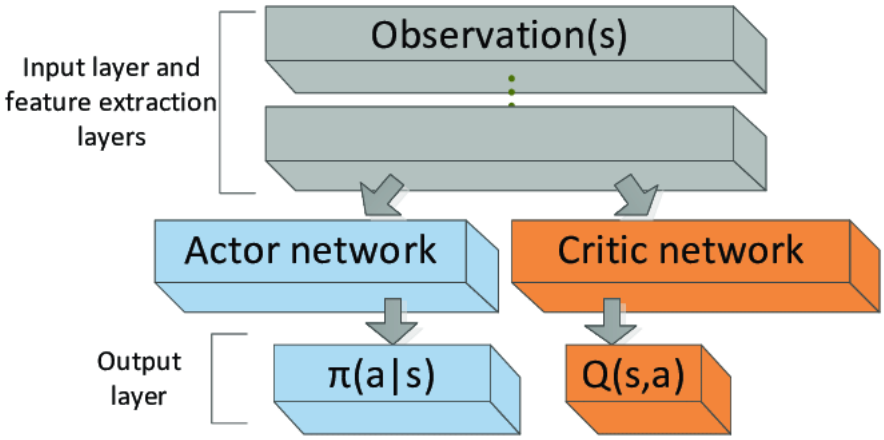


Figure 2: Actor-Critic algorithm use two neural networks for policy and value function.

We can choose and generate the city where the earthquake and fire will occur from the Open Street Map (OSM) using the convert program that RCRS supplied. The program converts maps on the OSM into simulation maps and we can manage more detailed scenarios such as the number of civilians and agents, additional earthquakes, and fired buildings at the start of the simulation.

In this study, we aim to extinguish the fire, so there is only fire disaster without earthquakes and civilians. RCRS map consists of roads, buildings, hydrants, and fire brigades and fire starts at certain buildings. The fire causes injury to fire brigades, and it can make them die. As the time step increase, the fire spreads, and the fire brigade can move and extinguish the fire. The fire brigade consumes water from the water tank to extinguish the fire, and if the water tank is empty, fire brigades can refill the water from hydrants. This study aims to find the optimal policy for the fire brigade to extinguish the fire.

Proximal Policy Optimization Algorithms (PPO)

Schulman, et.al propose proximal policy optimization algorithm [20]. In this algorithm, the advantage function A^{π}, which is express how a better reward could be obtained by the agent by taking that particular action than the expected reward of the state, is used. Advantage function can be expressed as Q function (Q^{π}) - Value function (V^{π}).

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{17}$$

PPO algorithm update advantage estimation for time steps T at once with T samples.

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{18}$$

where t is the time index in [0, T] within a given length-T trajectory segment.
With generalizing, advantage estimation can be expressed as:

$$\hat{A}_t = \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t+1}\delta_{T-1} \tag{19}$$

$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{20}$$

Because PPO algorithm is one kind of actor-critic algorithm, PPO algorithm has actor-network for policy and critic network for the value function and PPO algorithm use loss function which combines two-term, surrogate objective function for policy term and value function error for value function term. The surrogate objective function gives a penalty if the update of advantage estimation is too large to ensure monotonous improvement. Surrogate objective function follows below equation:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)] \tag{21}$$

Clip function clipping the updates of advantage estimation if the update ratio is outside of the range [1-ε,1+ε], where epsilon is a hyperparameter that controls the clipping range.

Value function error term is square error loss with target value function.

$$L^{VF} = (V_\theta(s_t) - V^{targ}(s_t))^2 \tag{22}$$

Thus, total loss function of PPO algorithm is :

$$L^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t{}^{CLIP}(\theta) - c_1 L_t{}^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{23}$$

where c_1, c_2 are coefficient and S denotes an entropy bonus. For N actors with time steps T, PPO

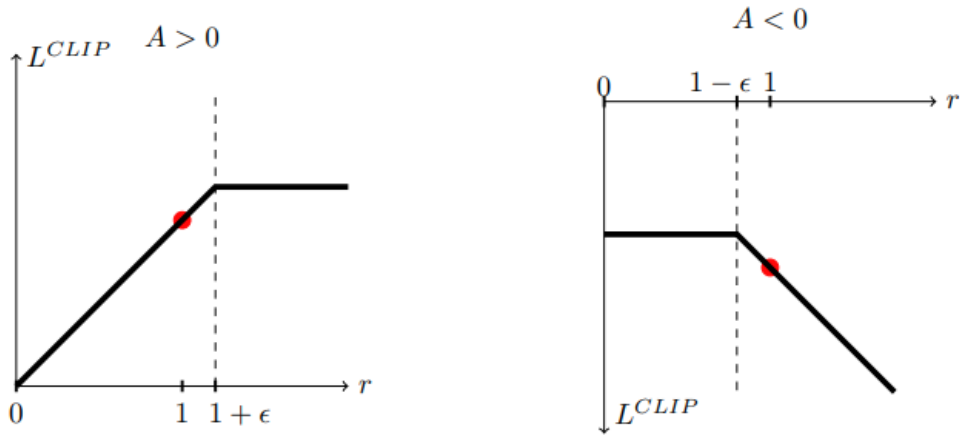


Figure 3: Clip function of surrogate objective function clipping updates of advantage estimation if the update ratio is outside of the range[1-ε,1+ε]

algorithm updates parameter θ for each iterations and repeat K epoch. For more information about how to set the parameters, please refer to [20].

Self Imitation

The self-imitation learning algorithm is purposed by Oh, Junhyuk [21]. The purpose of the Self-Imitation algorithm is to allow models to imitate their own past good experiences. It updates the parameters of the network only when the experience is better than the past, otherwise, the model learns from the experience.

Note that loss $L_{policy}{}^{sil}$ is used as policy gradient.

Algorithm 2 Proximal policy optimization algorithms

for iteration=1, 2, $\cdots$ **do**
  for actor=1, 2, $\cdots$, N **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, \cdots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and mini batch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

Algorithm 3 Self-Imitation Learning Algorithm

Initialize parameter θ
Initialize replay buffer D
Initialize episode buffer ε
for each iteration **do**
  # COLLECT on-policy samples
  for each step **do**
    Execute an action $s_t, a_t, r_t, s_{t+1} \sim \pi_\theta(a_t, s_t)$
    Store transition $\varepsilon \leftarrow \varepsilon \cup \{(s_t, a_t, r_t\}$
  end for
  if s_{t+1} is terminal **then**
    # Update replay buffer
    Compute returns $R_t = \sum_k^\infty r^{k-t} r_k$ for all t in ε
    $D \leftarrow D \cup \{(s_t, a_t, R_t)\}$ for all t in ε
    Clear episode buffer $\varepsilon \leftarrow \emptyset$
  end if
  # Perform actor-critic using on-policy samples
  $\theta \leftarrow \theta - \eta \nabla_\theta L^{a2c}$
  Perform self-imitation learning
  for $m = 1$ to M **do**
    Sample a mini-batch {(s,a,R)} from D
    $\theta \leftarrow \theta - \eta \nabla_\theta L^{a2c}$
  end for
end for

Behavior Cloning

Behavior cloning is one kind of supervised learning that pre-train a deep neural network of deep reinforcement learning models using collected data from humans or other algorithms. In this thesis, behavior cloning collected data from a greedy algorithm that makes the fire brigades extinguish the nearest fired

building.

IV Calibrating Dynamic Traffic Assignment Models by Parallel Search using Active-CMA-ES

This section is part of the first major goal, Calibrating Dynamic Traffic Assignment Models by Parallel Search. Nowadays, we can obtain many traffic data in real-time from inductive loop sensors installed on roads and data-driven approaches for analyzing traffic flows could potentially be more accurate than traditional macroscopic traffic flow modeling. The key step of utilizing the data is to use the data to fit a traffic model such as a microscopic traffic simulation model. Dynamic traffic assignment models have been used to compute the equilibrium traffic flow in a traffic simulation. One of the famous microscopic traffic simulators, PTV Vissim[3] uses a module called Dynamic Assignment to iteratively compute the traffic flow in equilibrium given a map and an OD matrix. To calibrate a DTA model, a calibration algorithm adjusts the OD matrix until the equilibrium traffic flow matches the collected data. In this part, we studied which algorithm is effective to calibrate a DTA model.

4.1 Definition of Problem

We define the calibration problem for DTA models as in [2]. The calibration process is comprised of a sequence of iterations, each of which aims to compute an estimated OD matrix that minimizes an objective function as defined below. Suppose there are M lane groups with inductive loop sensors in a traffic network (e.g., the sensors in Fig. 4). Let $y_i(\tau)$ be the number of vehicles detected by the inductive loop sensors in the lane group i in the time interval τ. Then $Y(\tau) = [y_1(\tau), y_2(\tau), \ldots, y_M(\tau)]^T$ is the ground truth for the calibration. Suppose there are N OD pairs in the OD matrix. Let $X_k(\tau) = [x_1(\tau), x_2(\tau), \ldots, x_N(\tau)]^T$ be the estimated OD matrix in the k iteration, where $x_j^k(\tau)$ is the traffic demand of the jth OD pair in τ. Let $\hat{Y}_k(\tau) = [\hat{y}_1^k(\tau), \hat{y}_2^k(\tau), \ldots, \hat{y}_M^k(\tau)]^T$ be a vector of the estimated numbers of vehicles in the lane groups in time interval τ according to $X_k(\tau)$. The objective function $Z(\cdot)$ in the k iteration is:

$$\min_{X_{k+1}(\tau)\geq 0} Z(X_{k+1}(\tau)) = \omega_1 E(X_{k+1}(\tau), X_k(\tau)) + \omega_2 E(\hat{Y}_{k+1}(\tau), Y(\tau)) \tag{24}$$

where $E(V_1, V_2) = ||V_1 - V_2||$ is the root mean square error between any two vectors V_1 and V_2, and $\omega_1 + \omega_2 = 1$ for $\omega_1, \omega_2 \geq 0$. Eq. 24 is a weighted sum of two terms. The first term is used to reduce the change of the estimated OD matrix after each iteration, and the second term is used to minimize the error between the estimated traffic flows of each lane group and the ground truth. Typically, ω_1 is much smaller than ω_2 so that the accuracy of the estimated traffic flows has a higher weight. In our experiments, $\omega_1 = 0.1$ and $\omega_2 = 0.9$.

[3] https://www.ptvgroup.com/en/solutions/products/ptv-vissim

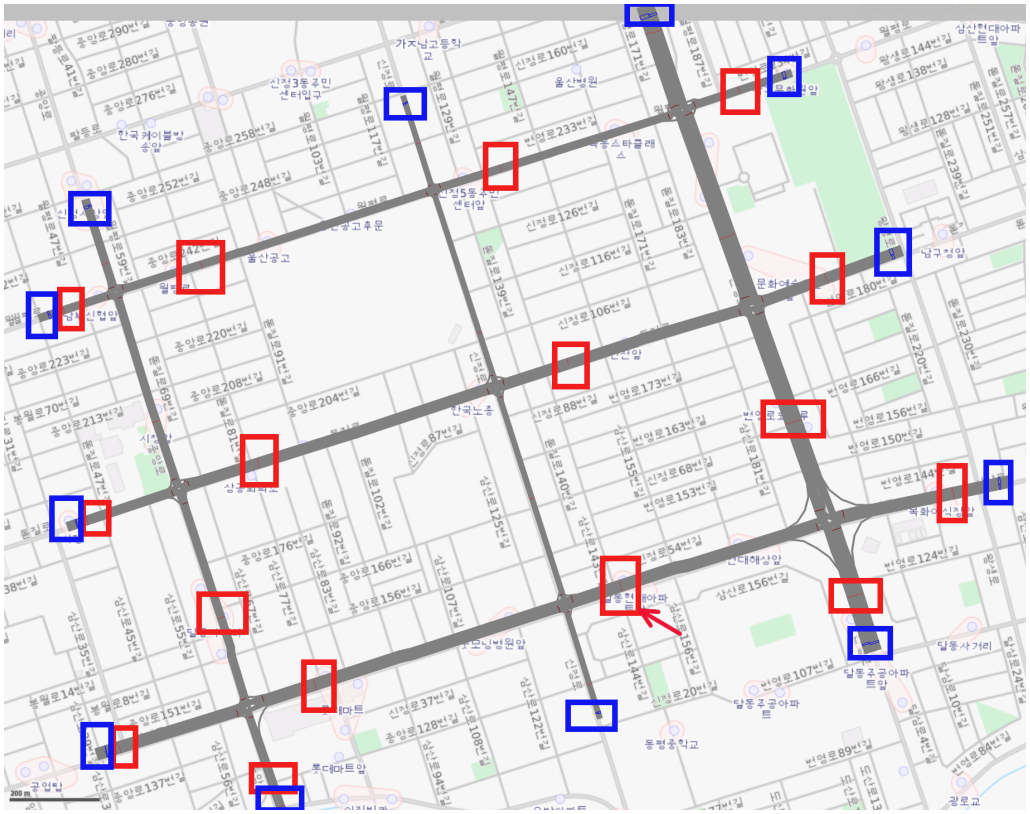


Figure 4: The map of a region in the downtown of Ulsan in South Korea. The grey roads are the road segments that traffic was simulated in PTV Vissim in our experiments. The blue boxes are the entries and the exits to the region. The red boxes indicate the physical locations of the inductive loop sensors. All but one red box contain two inductive loop sensors, one for each direction of the road. The red box pointed by an arrow contains one inductive loop sensor only.

4.2 Active CMA-ES and Restart-WSPSA

In the traffic simulator, the OD matrix's values represent the vehicle number from origin to destination, and the numbers are integer. However, most algorithms use float values from 0 to 1. Thus, we need to translate the algorithm's values as an integer values, and we translate the float value from 0 to max integer in this thesis. The max integer values are get from the historical value from ground truth data. So, inputs of simulation are values of the algorithms from 0 to 1, and the values are translated from 0 to maximum values. After that, the simulator simulate based on the OD matrix, and inductive loop sensors collect the passed vehicle number for each road and calculate objective function as output. Each output is used to calculate the next iteration for each algorithm. The CMA-ES generates 16 off-springs as sample inputs, and each off-springs has 132 elements for the OD matrix. After the sample inputs are simulated, CMA-ES lines the samples up to rank according to the objective function value, throws away half the low rank, and updates a search point to generate next iteration sample inputs and repeat. The SPSA generates 2 off-springs as sample inputs based on a search point and simulates the sample points. Unlike CMA-ES, the SPSA algorithm used objective function values to generate the next search

point. After finishing the one-time interval, the calibrated values are used for the first search point of the next time intervals. We defined the calibration problem to minimize the target loss functions, but we need to consider not only minimizing the target loss function but also calibrating the simulations in parallel. So, we also compared the total running time depending on the number of simulations in parallel. When we don't know which algorithm is better than other algorithms to calibrate simulation in parallel, we tested a much smaller map to see the simple and brief result, such as 2 intersection maps or 4 intersection maps. Also, we developed a toy simulator based on VISSIM and see the experiment result in 9 intersections maps. The gap between CMA-ES and other algorithms is too big than we expected, so we combined restart strategies to the WSPSA algorithm, which is named the Restart-WSPSA algorithm. It gets a higher score than the WSPA algorithm or restart-SPSA algorithm. Because CMA-ES is worse than Restart-WSPSA, we upgrade CMA-ES to Active-CMA-ES, which is an enhanced version of CMA-ES to converge speed. The detailed algorithms are explained in section 3.1. In both small map or toy simulations case, CMA-ES shows better result than SPSA algorithm and WSPSA algorithm, so we tested map figure 4 in the experiment setting section. To compare the performance of calibrations as equation 24, we compared Calibration Errors vs. the Number of Simulations in section 4.3, and Calibration Errors vs. the Total Running Time in section 4.4. The most important advantage of CMA-ES is that more simulations can run in parallel. Therefore, we estimated the total running time of the algorithms as the number of CPU cores used by the algorithms in parallel increases in section 4.4. Because SPSA based algorithm can run 2 simulation in parallel as maximum number of parallel running, we run CMA-ES 2 as default and increase the number of simulation in parallel to 4, 8 and 16. Becasue VISSIM support 4 simulations in parallel and the running times of other computations were negligible, we can estimate the total running time by greedily allocating simulations in each generation to the 8 and 16 CPU cores. Also, we compared calibration errors of individual roads in section 4.5 to check exactly how well the traffic flow at each inductive loop sensor readings is calibrated.

4.3 Experimental Settings

Our case study focuses on a region in the downtown of Ulsan, a city in South Korea. As shown in Fig. 4, the 1.5 km $\times$ 1.1 km region has 9 intersections and 24 road segments. All intersections are managed by traffic signals. We obtained the traffic signals data from Korean National Police Agency[4]. We used PTV Vissim (version 11), a widely used microscopic traffic simulator, with the Dynamic Assignment module for DTA calculation. There are 31 inductive loop sensors in the data. We collected the inductive loop data (i.e., the number of vehicles passing the inductive loops) of the four 15-minute time intervals from 7:00AM to 8:00AM on December 12, 2015 from the Road Traffic Authority of Ulsan[5].

The OD matrix consists of 12×12 entries since there are 12 entries and 12 exits in the region. The calibration algorithms have to tune 12×11 parameters in the OD matrix. In the first time interval, the initial value of each element in the OD matrix is half of the maximum value of each element, which is determined manually based on historical traffic records. In the other time interval, the initial OD matrix

[4] https://www.police.go.kr/eng/main.do

[5] https://utrhub.its.ulsan.kr

is the calibrated OD matrix in the previous time interval.

The experiments were conducted on a computer with an Intel Core i7 8700K CPU with 32GB RAM. The CPU has six cores, and thus it can run six traffic simulations in parallel. Nonetheless, PTV Vissim can utilize at most four cores in parallel due to license issues. Both Restart-WSPSA and Active-CMA-ES were implemented in Python 3, and they start and control the simulations in PTV Vissim by the COM interface. We used pycma[6] to implement Active-CMA-ES. The population size of Active-CMA-ES is 16.

To increase the performance of Restart-WSPSA, we optimized the W-matrix in each time interval by using the calibrated OD matrix in the previous time interval. According to [12], we set $\alpha = 0.602$ and $\gamma = 0.101$ in WSPSA. In addition, we restarted WSPSA every 150 generations. We also estimated the extra loss measurement for each iteration by an additional simulation and then rejected the iteration if the extra loss measurement is larger than the previous iteration loss measurement by 20%. According to [13], the rejection can speed up the convergence of Restart-WSPSA. Hence, each iteration in Restart-WSPSA has three simulations, two for calculating the gradient for SPSA and one for loss measurement. Note that the third simulation must be run after the first two simulations.

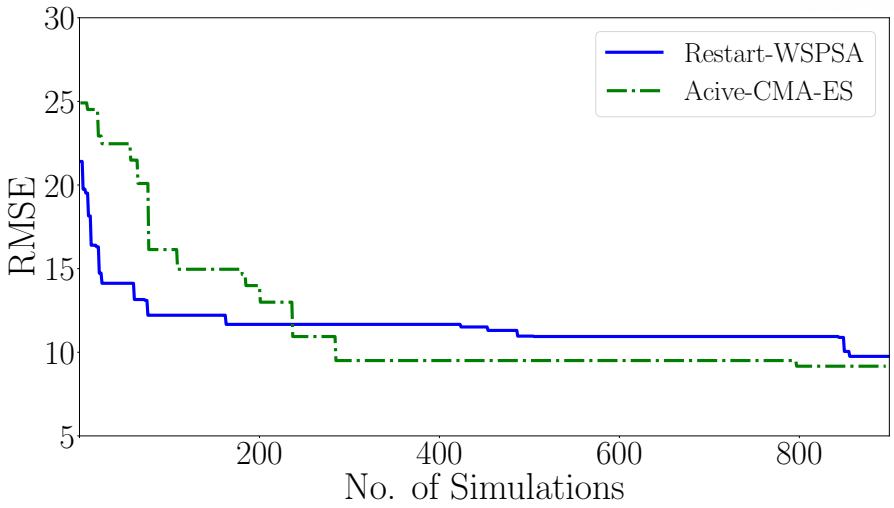
4.4 Calibration Errors vs. the Number of Simulations

First, we looked at the RMSE of the traffic volume on the road segments with inductive loop sensors. In our experiment, we gave both Restart-WSPSA and Active-CMA-ES a budget of 900 simulations they can run using PTV Vissim. In each simulation, PTV Vissim used the Dynamic Assignment module to iteratively estimate the traffic flow given an OD matrix until it reaches an equilibrium. Then we calculated the RMSE based on the traffic flow at the equilibrium and the ground truth. We kept tracking the change of RMSE as the algorithms used more and more simulations to refine the OD matrix. The results are shown in Figures 5(a), 5(c), 5(e), and 5(g). Although Restart-WSPSA ran two simulations in parallel and Active-CMA-ES ran four simulations in parallel, these figures report the running time as if the simulations were run sequentially one by one.

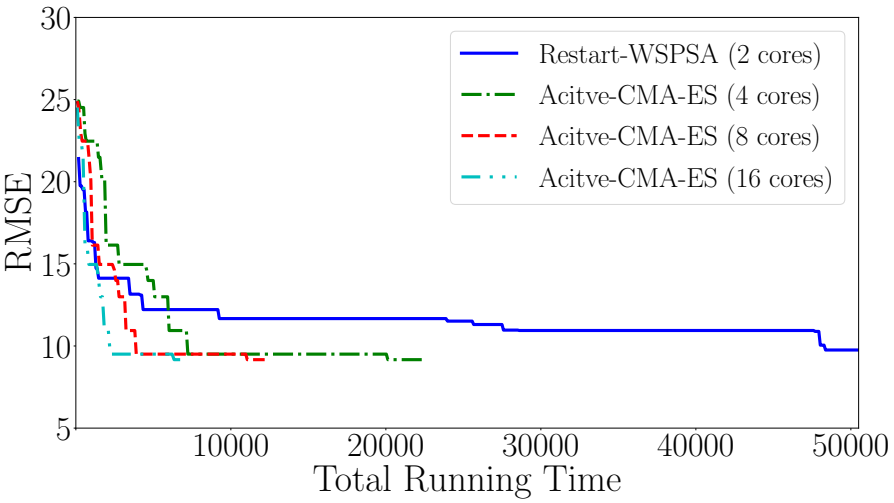
According to the four graphs on the left of Fig. 5, Active-CMA-ES had a higher RMSE at the beginning. But Active-CMA-ES's performance increased rapidly and eventually outperformed Restart-WSPSA in all time intervals. The reason for the poor initial performance is that Active-CMA-ES requires 16 simulations for each generation while Restart-WSPSA requires 3 simulations for each iteration. Hence, Active-CMA-ES needs more simulations to make progress initially. However, Active-CMA-ES eventually outperformed Restart-WSPSA since Active-CMA-ES is more capable of escaping the local minima.

Table 1 summaries the RMSE for each interval of Restart-WSPSA and Active-CMA-ES after running 900 simulations. According to the table, the performance gap between Restart-WSPSA and Active-CMA-ES increases as the time interval increases. Active-CMA-ES's superior performance is more noticeable in the higher time intervals because the error in a time interval can carry over to the next time

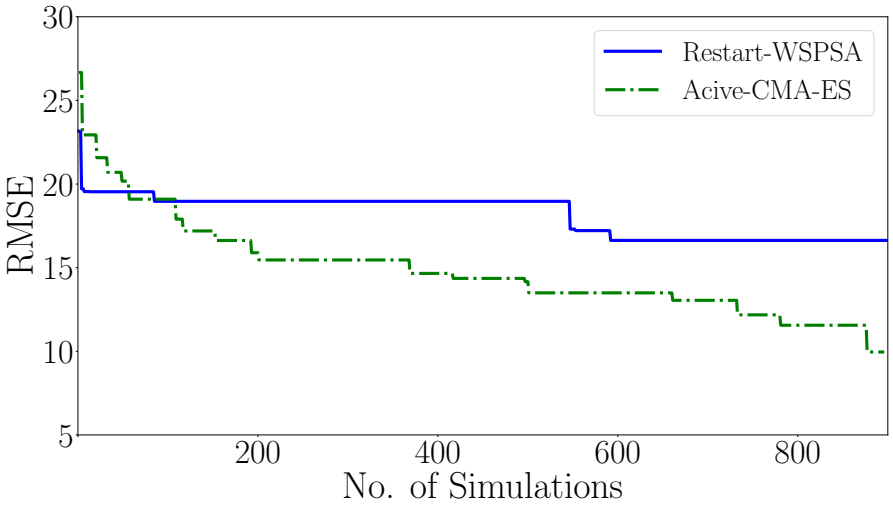
[6] `https://github.com/CMA-ES/pycma`



(a) RMSE vs. the number of simulations in the first time interval.

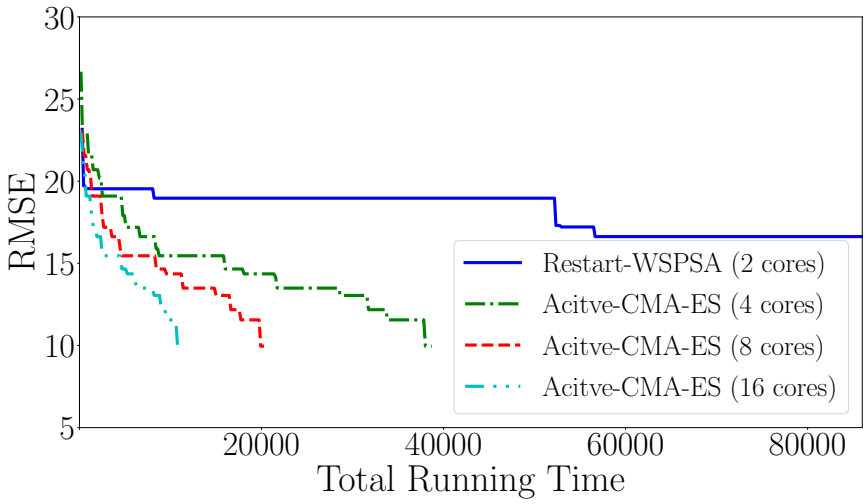


(b) RMSE vs. the total running time in the first time interval.

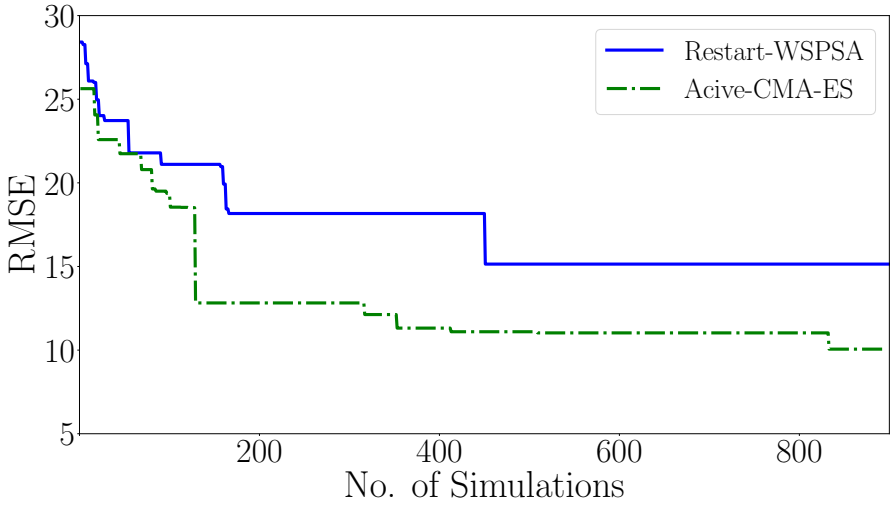


(c) RMSE vs. the number of simulations in the second time interval.

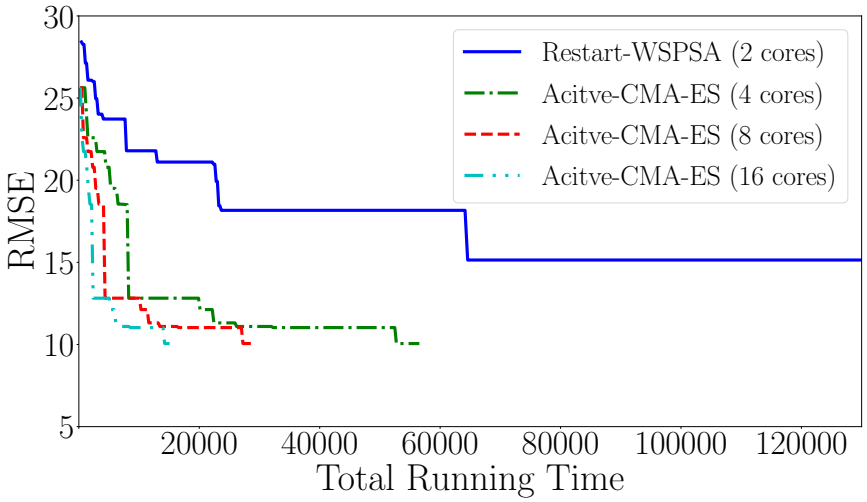
interval.



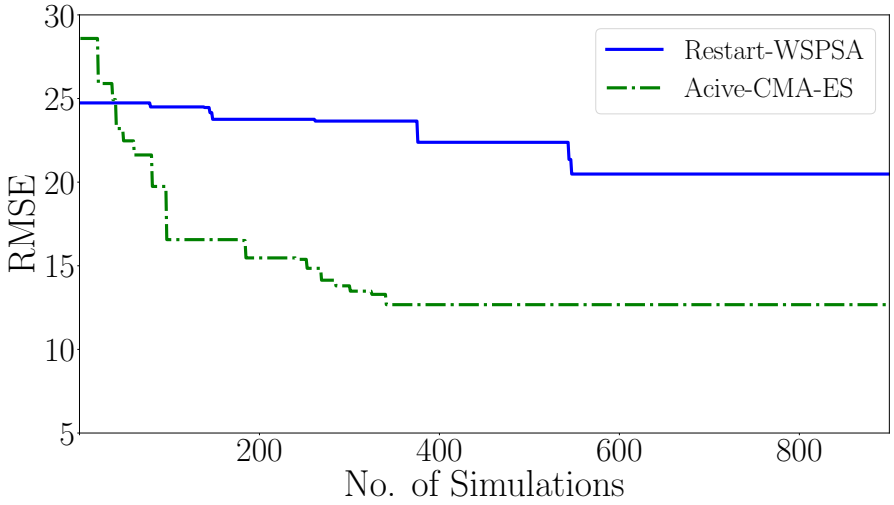
(d) RMSE vs. the total running time in the second time interval.



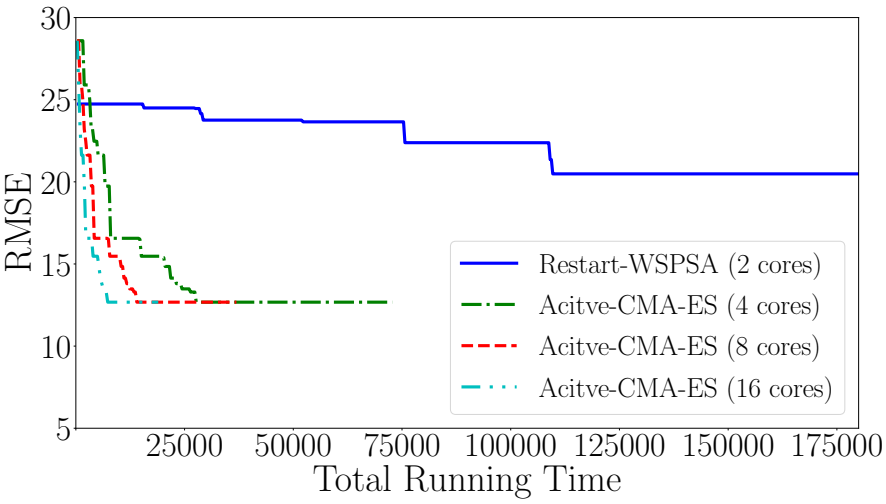
(e) RMSE vs. the number of simulations in the third time interval.



(f) RMSE vs. the total running time in the third time interval.



(g) RMSE vs. the number of simulations in the fourth time interval.



(h) RMSE vs. the total running time in the fourth time interval.

Figure 5: The performance of Restart-WSPSA and Active-CMS-ES in each of the four-time intervals when the traffic simulations ran sequentially and in parallel.

Table 1: The RMSE for each interval of Restart-WSPSA and Active-CMA-ES after running 900 simulations.

Interval	1st	2nd	3rd	4th
Restart-WSPSA	9.755	16.629	15.142	20.484
Active-CMA-ES	9.167	9.958	10.058	12.675

4.5 Calibration Errors vs. the Total Running Time

The advantage of Active-CMA-ES is prominent when more simulations can run in parallel. Therefore, we analyzed the data we collected in the experiment to estimate the total running time of the algorithms as the number of CPU cores used by the algorithms in parallel increases. We recorded the running time of every simulation in Active-CMA-ES. Then we estimated the total running time when there are more than 4 CPU cores by overlapping some of the simulations. Note that the running time of both Active-CMA-ES and Restart-WSPSA were dominated by the simulation times, and the running times of other computations were negligible. Hence, we can estimate the total running time by greedily allocating simulations in each generation to the CPU cores. We implemented a discrete event simulation to simulate the execution of Active-CMA-ES when the number of cores is 8 and 16. The results are shown in Figures 5(b), 5(d), 5(f), and 5(h). According to these figures, as the number of CPU cores increases, the total running time of Active-CMA-ES drops tremendously, and the initial poor performance of Active-CMA-ES disappears. Therefore, Active-CMA-ES can be an effective parallel search algorithm for calibrating DTA models.

4.6 Calibration Errors of Individual Roads

RMSE measures the error of the calibration of the entire model at once. However, RMSE cannot tell exactly how well the traffic flow at each inductive loop sensor readings is calibrated. Therefore, we examined the observed vehicle counts of the roads at each inductive loop sensor at every interval and compared them with the predicted values generated by Restart-WSPSA and Active-CMA-ES. Fig. 6 is a scatter plot of the vehicle counts of the roads at which the inductive loop sensors are located in the fourth time interval. There are 31 red dots and 31 blue dots. The x-axis is the observed vehicle count (i.e., the ground truth), and the y-axis is the predicted vehicle count (i.e., the simulated count) after calibration. Clearly, the observed counts of both Restart-WSPSA and Active-CMA-ES are the same. The difference between an observed count of a sensor and the corresponding simulated count is the calibration error of the traffic flow at the sensor.

As can be seen, most of the calibration errors are quite small. There are a few outliners, but they appear to occur randomly on different roads. In general, there is a strong linear relationship between the observed counts and the simulated counts for both Restart-WSPSA and Active-CMA-ES. However, Active-CMA-ES performed slightly better as the red points are closer to the diagonal.

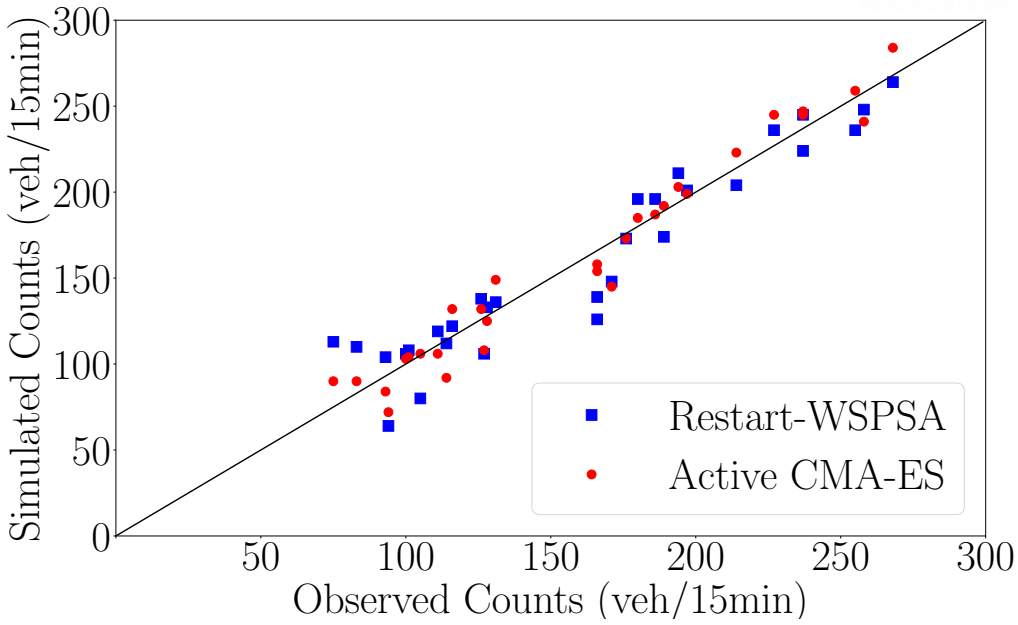


Figure 6: The relationship between the vehicle counts collected by the inductive loop sensors in the simulation and the vehicle counts in the observed data collected by the inductive loop sensors in the fourth time interval.

4.7 Conclusions

To utilize real-time traffic data to calibrate a traffic model as quickly as possible, we propose to use Active-CMA-ES, which can utilize modern multi-core computer architecture to conduct parallel search. Our experiments showed that Active-CMA-ES outperforms Restart-WSPSA, the best SPSA algorithm in the literature. We have also demonstrated the scalability of Active-CMA-ES as the number of CPU cores increases. Although Active-CMA-ES outperforms the existing calibration algorithms, it has not yet achieved the *real-time* calibration of traffic models. If one can calibrate a traffic model in real time, some important applications such as real-time traffic rerouting for congestion prevention can be more effective. In the future, we intend to extend Active-CMA-ES for incremental calibration of traffic models over time.

V Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning

This section is part of the second major goal, Finding Optimal Policy to Extinguish Fire using Deep Reinforcement Learning. Recently, there have been more cases of solving problems by learning artificial intelligence models from various simulators such as AlphaGo [3] and Alphastar [4]. However, research on finding the best model to find the optimal policy of disaster agents in disaster simulators such as fires or earthquakes is not yet active. This is because disaster measures such as overpowering fires often become uncontrollably difficult due to a little mistake in the early stages and a small difference in behavior in early stages makes significantly different results. Thus, we have been developing a software package by finding an optimal policy to extinguish the fire using Deep Reinforcement Learning.

5.1 Definition of Problem

In reinforcement learning, a performance is measured as the value of the reward function at the end of episode per learned episode, and it is up to the researcher how to set the reward function. Thus, the problem of the section V is get highest score of reward function at the end of episode. In this study, we set the reward function as followed:

$$R' = R + F \tag{25}$$

where R' is shaped reward for each time step return and R' is the sum of true reward R and shaping return F. True reward R is a reward at the end of the episode, and otherwise, it returns 0. Shaping returns F is the return for each time step and returns 0 at the end of the episode.

$$F = \gamma \cdot \phi(s') - \phi(s) \tag{26}$$

where γ is decay rate and ϕ is function that calculate return of each state. Thus, F is difference between return of new state s' and s. In this experiment, *gamma* is considered 1.0, 0.99, 0.9.

$$\begin{aligned} \phi : -\sum(z(fieryness))\, where\, z(fieryness) : \\ if\ fieryness = 4 : return\, 11 \\ if\ fieryness = 3 : return\, 10 \\ if\ fieryness = 2 : return\, 8 \\ if\ fieryness = 1 : return\, 5 \\ if\ fieryness = 0\, or\, fieryness >= 5 : return\, 1 \end{aligned} \tag{27}$$

ϕ calculate each fieryness of burning buildings and change it as a return. If there are bigger fires in the map, ϕ will return more negative returns.

$$\begin{aligned} R = 0.5 \cdot B - \sum(Fire_n) + 0.1 \cdot (MaxTimeStep - FinishedTimeStep), \\ n = number\, of\, burning\, building\, if\, s\, is\, the\, terminal\, state\, , 0\, otherwise. \end{aligned} \tag{28}$$

$$B :\sim (fire_1 \cap fire_2 \cap \cdots \cap fire_n), \tag{29}$$

where $fire_n$ is Boolean value if the building is on fire or not and n = the number of buildings. True return R has three parts. Fire part $0.5 \cdot B$ is a bonus value if agents extinguish all fire. It is 0.5 if there is no fire at all and 0 if there is at least one fire. Second part $-\sum(Fire_n)$ expresses how many fires remain at the end of episodes as minus return. It will give more negative value if there is more fire remaining. Third part $0.1 \cdot (MaxTimeStep - FinishedTimeStep)$ is bonus value if agents extinguish all fire faster. For example, if the agent extinguishes all fire 50 times steps earlier than the maximum time step, it will give a 0.5 bonus return.

5.2 Self-Imitation and Behavior Cloning

To find the optimal policy to extinguish the fire, we should define what is optimal. The optimal policy should extinguish the fire effectively, and it means it takes only a small time step to extinguish all fire. Thus, we set bonus value $0.1 \cdot (MaxTimeStep - FinishedTimeStep)$ in 28. Thus, we can say the optimal policy is the policy gets the highest value of the reward function at the end of the episode. Also, the optimal policy should extinguish the fire in various scenarios. So, we aim to find the methods that get the highest return value at the end of training with various scenarios. In the most of sections, we compared trained episode number VS return reward at the end of episodes and changed the experiment set to control initial fire condition or map size. We tested fixed initial fired building in section 5.5 and random initial fired building in section 5.6 and changed map size in section 5.9. During the experiments, we applied a self-imitation algorithm to speed up the model learning. The purpose of the self-imitation algorithm is to allow models to imitate their own past good experiences. It updates the parameters of the network only when the experience is better than the past, otherwise, the model learns from the experience. Through the self-imitation algorithm, we can speed up the model learning. Also, the PPO algorithm can not extinguish all fire sometimes, especially in the case that the initial fire location is random. We assume that only the initial steps of extinguishing a fire can prevent spreading fire, but the model can not find proper actions of initial steps because the model can not understand which part of time steps are more important than others. Thus, we collect data of greedy algorithm and do behavior cloning methods to model to expect that model can mimic the initial steps of greedy algorithm. The detailed algorithms are in section 3.2.

5.3 PPO Algorithm to RCRS

To learn the deep reinforcement learning model, we need to express the RCRS map states as observation space. In my study, there are three parts for observation space. Fire condition of buildings, condition of agents, condition of colleague agent. The fire condition part has two values. Fire Boolean value to check the building is burned or not. 0 means that the building is not on the fire and 1 means that the building is on the fire. Fieryness is the number to express how the fire is big, and how the damage of fire is big. 0 means that there is no fire on the building, and 1 4 means the level of fire. 1 is minimum fire, and 4 is maximum fire. 5 8 means the damaged level of the building after the building is extinguished. 5 means

a little damage to the building and 8 means the building is destroyed totally. The extinguished building also can be re-burn. (i.e. fieryness 5 can be 1.)

The second part is a condition of Agent and there is the water level and x,y coordinate value. Water level represents how much water the agent has. The water level is decreased when agents try to put the fire out (ex, 0.8->0.6) and if the water level is 0, agents can not try to put the fire out. x,y coordinate value represents x,y coordinate value of agents.

The third part is a condition of colleague agent and there is the water level of colleagues, x,y coordinate value, and busy Boolean value. The water level is the same as the water condition of agents, so it represents how many waters the colleague agents have. x,y coordinate value represents x,y coordinate value of colleague agents. The busy Boolean value is a Boolean value that expresses the colleague is busy or not. If the colleague is busy, the colleague has a target already. 0 means that the colleague is not busy and 1 means that the colleague is busy.

After the model gets states of RCRS as environment, the model needs to choose the action of agents. In the case of fire brigades, agents should choose which buildings to check and extinguish the fire. Thus, if the model chooses the id of each building, the agent will go to the building and check whether the building is on fire or not, and if the building is on fire, the agent will try to extinguish the building until the water tank is empty or the fire is extinguished. On our map, there are 57 buildings and there are 57 ids to choose from. Also, the agent can refill water on the nearest hydrant. It is also one of action to choose. Thus, including choices that do nothing, there are 59 actions that the model can choose.

After the model chooses an action for agents, we need to return the reward of each choice. The reward function is described in section 5.1.

Thus, the total flow of deep reinforcement learning is, get observation space as input states and give it to the model. The model selects an action of agents as output and we calculate the reward as reward functions and return the reward value to the model, then the model updates its parameters based on the reward values. Every deep reinforcement learning algorithms in this thesis (e.g, PPO algorithm or TD3 algorithm) follows that flows.

5.4 Experiment Setting

In the RCRS map, I used 57 buildings with 3 hydrants and 2 fire brigades. The map region is mimic the parts of Ulsan, South Korea. In the map, each building is expressed as rectangles in the dark grey and yellow rectangles mean that the building is on the fire. Based on the level of fire, the color of buildings is changed to red. Bright grey rectangles are express roads and red objects in the roads are hydrants. The red dots express fire brigades.

The RCRS uses a temperature sharing model for fire spreading. Each neighbor's buildings share their temperature and if the temperature is higher than the ignition point, the building will be ignited. In the RCRS, each building has its hp and it can be damaged by fire. If damage is enough, the building will be destroyed and can not extinguish fire anymore. The fire on each building has a fire level, which is called fiery, and fiery has 4 steps and if fiery is higher, the damage it gives to the building is high.

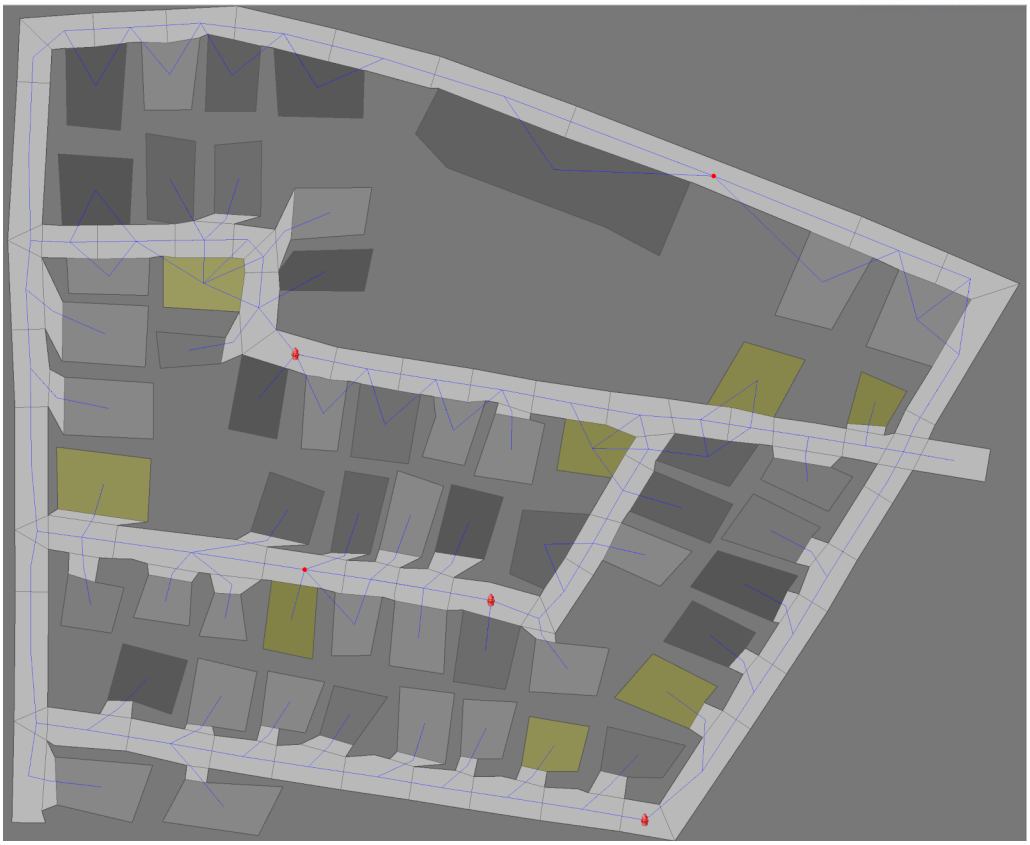


Figure 7: The map of a region in the downtown of Ulsan in South Korea. The rectangle. The boxes in the dark grey are buildings. The yellow boxes are on the fire and the color of each building can be changed if the fire level is changed. The other bright grey boxes are roads and red objects on the roads represent hydrants. Red dots represent the location of fire brigades.

When the fire brigade tried to extinguish a fire, the fiery decrease, and finally it is 0 when the fire is extinguished perfectly. The fire brigade has a water tank and uses water to extinguish the fire. If the agent uses all water in the water tank, the agent needs to refill water from a hydrant. I used the default value supplied by RCRS competition sample maps.

5.5 Fixed Location of Initial Fire

For the first experiment, we fix the location of the initial fire and compare each algorithm with a greedy algorithm. We tested three greedy algorithms, that extinguish the nearest fired building, the biggest fire level building, the smallest fire level building, and extinguish nearest fired building is the most effective. Thus, in the thesis, a greedy algorithm means that agents choose the nearest fired building. First, we tested 8 initial fired with fixed position. In the case of 8 initial fires, greedy methods never extinguish all fire. Each episode has 100-time steps as maximum.

As we can see in the fire 8 10, the deep reinforcement learning model can extinguish all fire very well and every deep reinforcement learning model shows better results than the greedy algorithm. Also, the PPO algorithm and PPO algorithm with self imitation show the best result. Because it's the case that

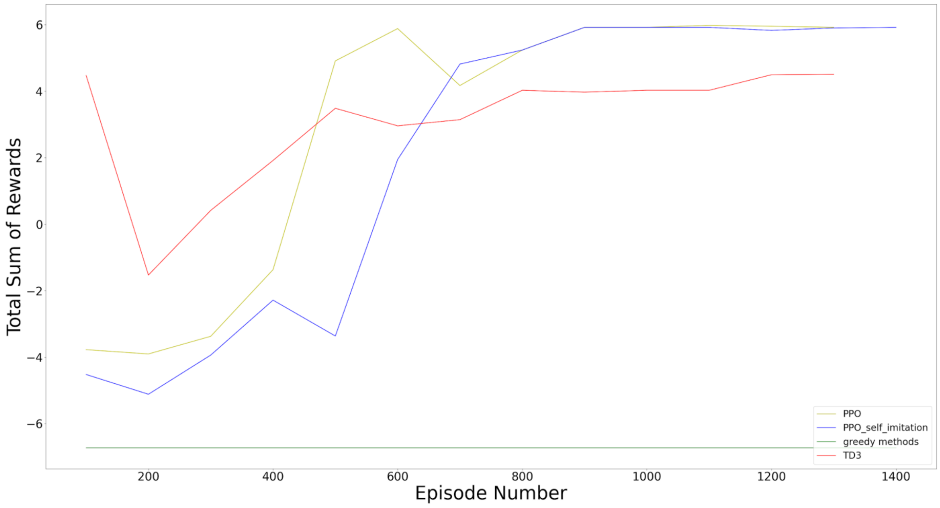


Figure 8: Total return at the end of an episode in the case of 8 initial fired with fixed location. In this case, the greedy algorithm never extinguishes all fire.

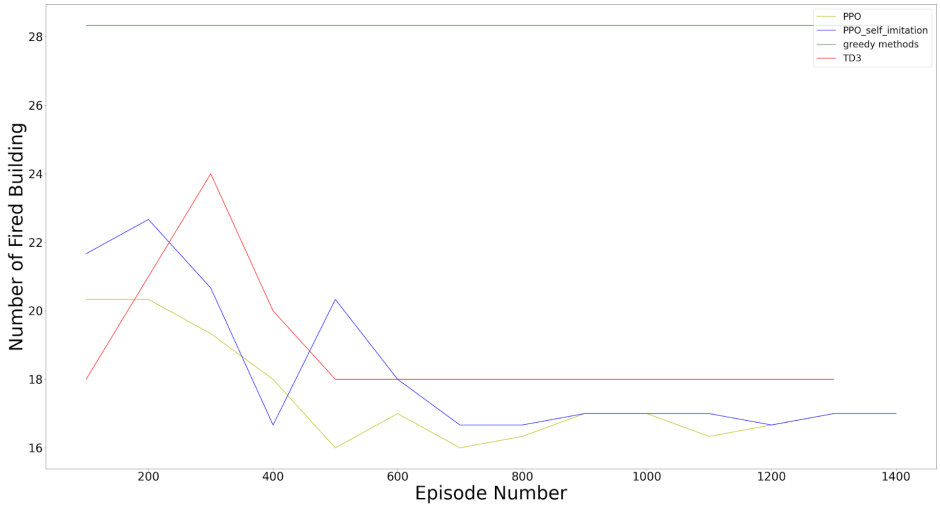


Figure 9: The number of remaining fired buildings at the end of an episode in the case of 8 initial fired with fixed location.

impossible to extinguish all fire, the deep reinforcement learning model finds the best way to control spreading fire and get the highest return.

In a second experiment, we tested 6 initial fired with fixed position. In this case, greedy methods always can extinguish all fire before spreading fire.

In the case of the agent can extinguish all fire, deep reinforcement learning model can not find the way to extinguish all fire, but PPO algorithm with self imitation shows the best result than other algorithm and it remains much less fired building than others.

To test more complex cases, we changed the initial fire location as random in the next section.

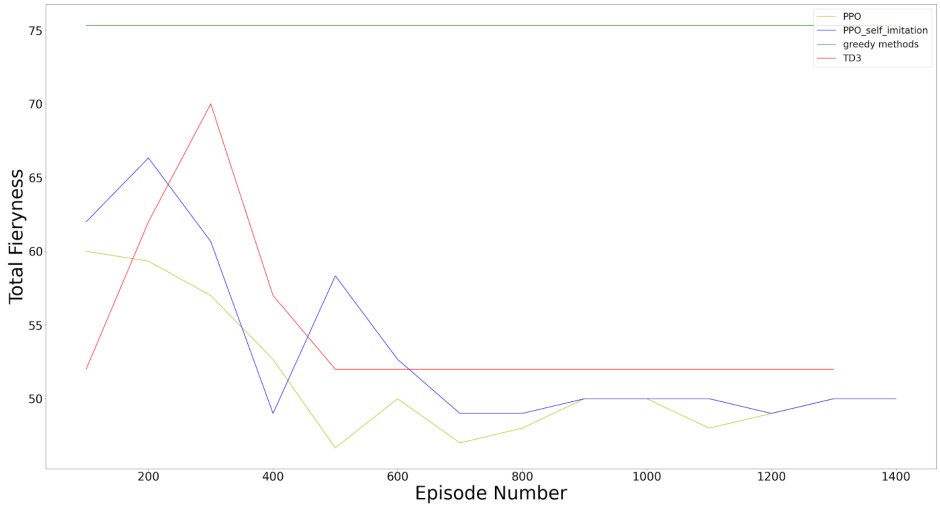


Figure 10: The sum of fieryness of remained fire at the end of an episode in the case of 8 initial fired with fixed location.

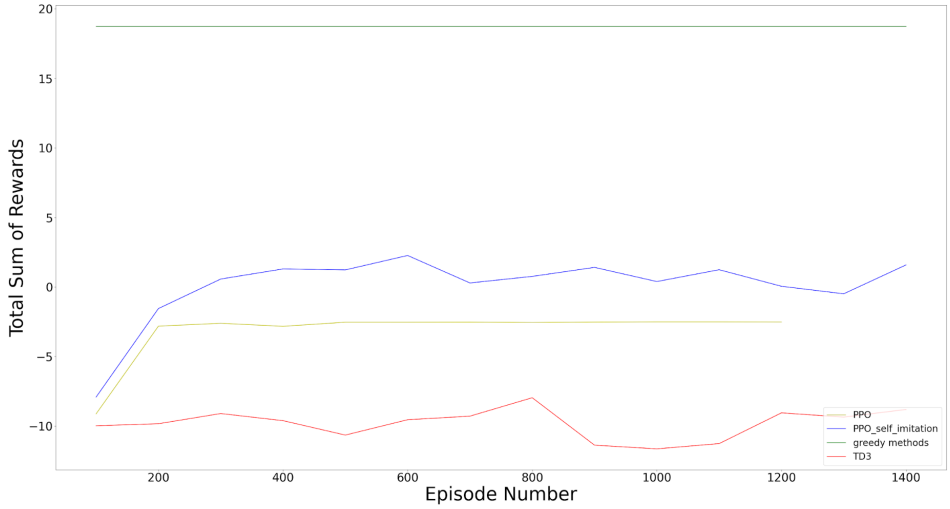


Figure 11: Total return at the end of the episode in the case of 6 initial fired with fixed location. In this case, the greedy algorithm always extinguishes all fire.

5.6 Random Location of Initial Fire

For the third experiment, we changed the location of the initial fire as random and tried to find the algorithm that can beat the greedy algorithm always. First, we tested 4 initial fires with random positions. In the case of 4 initial fires, the greedy algorithm always extinguish all fire. Each episode has 100-time steps as maximum.

In the random initial fire case, the PPO algorithm with self imitation can not extinguish all fire again and have a big variance during the learning. It seems that only the initial steps of extinguishing the fire can prevent spreading fire, but the model can not find proper actions of initial steps because the model

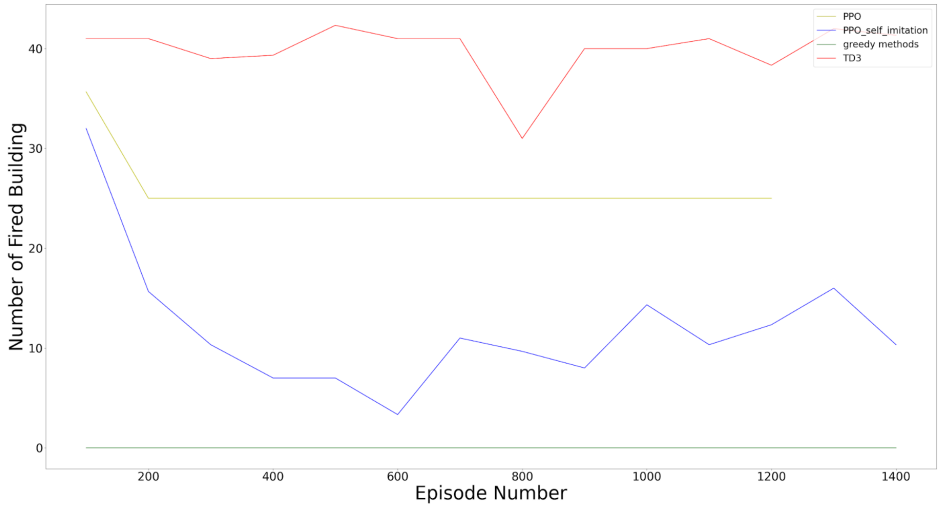


Figure 12: The number of remaining fired buildings at the end of an episode in the case of 6 initial fired with fixed location.

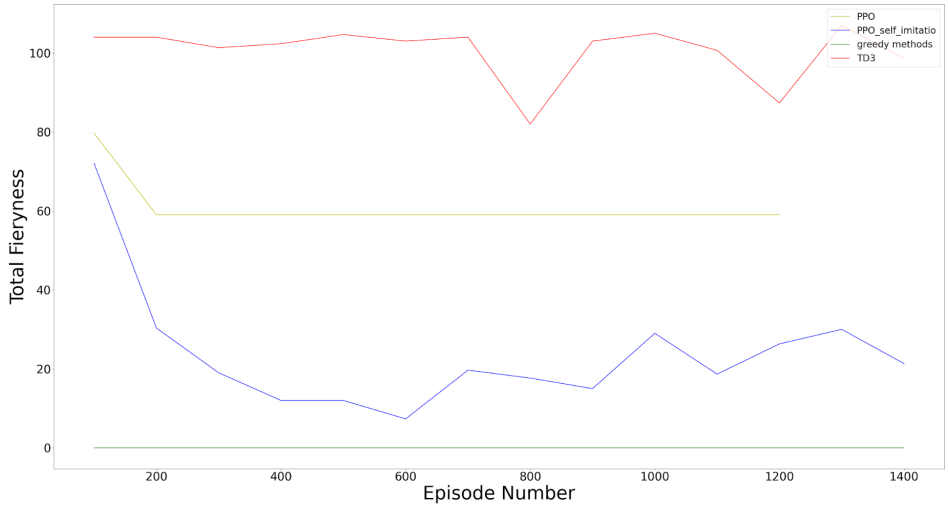


Figure 13: The sum of fieryness of remained fire at the end of an episode in the case of 6 initial fired with fixed location.

can not understand which part of time steps are more important than others. Thus, we collect data of greedy algorithm and do behavior cloning methods to model to expect that model can mimic the initial steps of greedy algorithm and finally get a better result than the greedy algorithm.

As we can see in figure 15, the PPO algorithm with self-imitation can extinguish all fire if there is behavior cloning. However, the greedy algorithm and PPO algorithm with behavior cloning always get a maximum score of 1, we need to check much more complex cases.

Thus in the fourth experiment, we tested several initial fired buildings with a random location.

In figure 16, we can see how the PPO algorithm with self-imitation and behavior cloning works

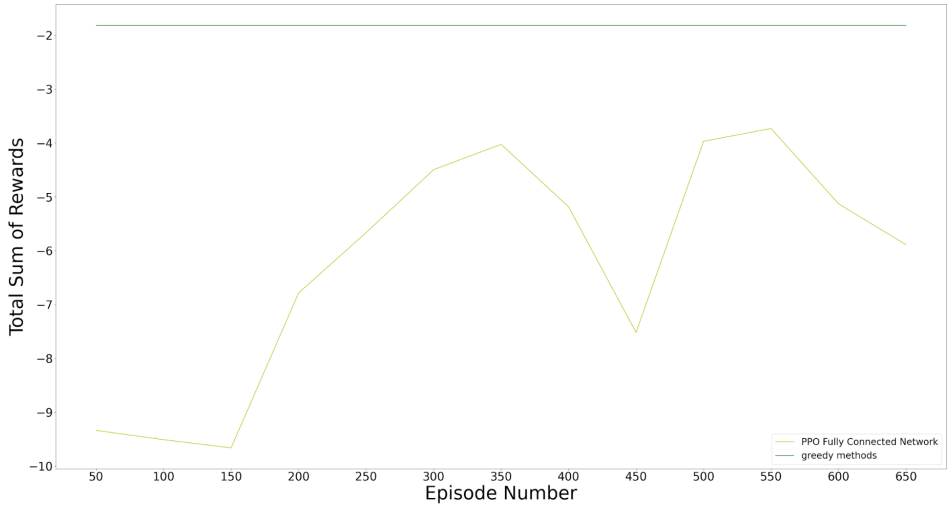


Figure 14: Total return at the end of an episode in the case of 4 initial fired with a random location. In this case, the greedy algorithm always extinguishes all fire. In the graph, we can see that the PPO algorithm with self-imitation can not extinguish all fire.

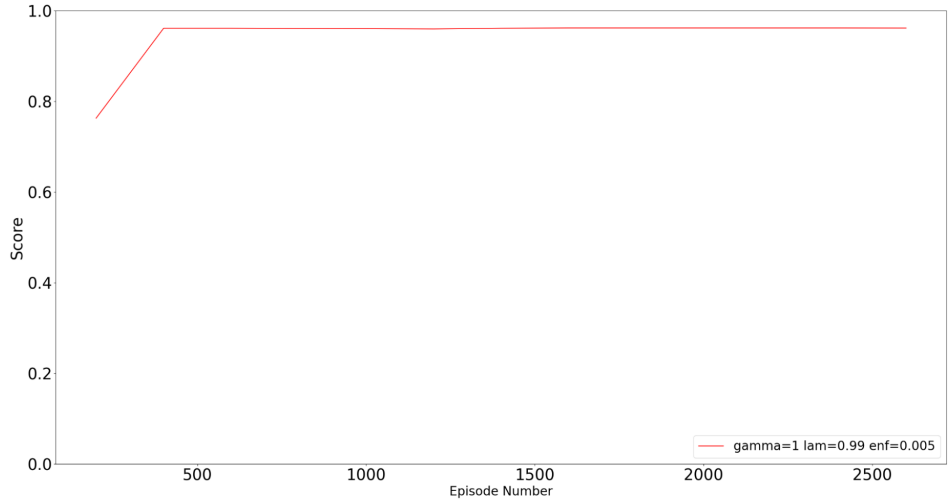


Figure 15: Total return at the end of an episode in the case of 4 initial fired with a random location. In this case, the greedy algorithm always extinguishes all fire. In the graph, we can see that the PPO algorithm with self-imitation and behavior cloning can extinguish all fire.

in several initial fire cases with a random location. As the number of initial fire increase, the return value decrease but still it's larger than 0, which means that the agent can extinguish all fire in most case. Because the greedy algorithm can not extinguish all fire over 6 initial fire cases, we can see that the PPO algorithm with self-imitation and behavior cloning is much better than the greedy algorithm.

Figure 17 is the average score compared with behavior cloning and without behavior cloning in 6 initial fire cases with random locations. In the graph, we can see that the PPO algorithm with self-

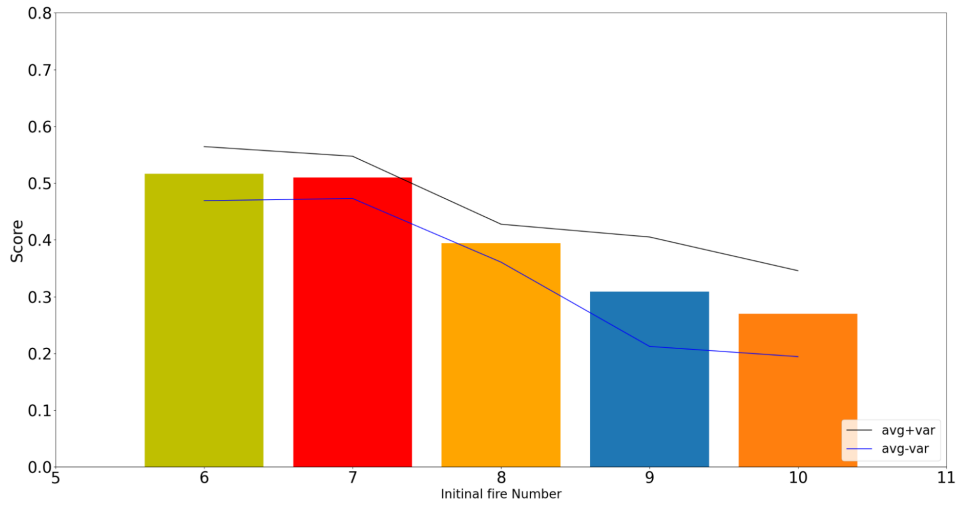


Figure 16: Total return at the end of an episode in the case of several initial fired with a random location. In this case, the greedy algorithm can not extinguish all fire always. In the graph, we can see that the PPO algorithm with self-imitation and behavior cloning can extinguish all fire because the average return value is larger than 0, which means that model can extinguish all fire in most case.

Figure 17: Total return at the end of an episode in the case of 6 initial fired with a random location. In this case, the greedy algorithm can extinguish all fire sometimes. In the graph, we can see that the PPO algorithm with self-imitation and behavior cloning can extinguish all fire in most cases but the PPO algorithm with self-imitation but without behavior cloning can not extinguish all fire.

imitation and behavior cloning can extinguish all fire in most cases but the PPO algorithm with self-imitation but without behavior cloning can not extinguish all fire.

In the next experiment, we want to test a much more complex case by increasing map size.

5.7 Toy Simulator

During the first to fourth experiment, it took more than a day to test the performance of an algorithm. To reduce the experiment time, we developed a toy simulator based on RCRS. Toy simulators follow the same methods to spread fire and the same methods to express fire level. It just remove other parts such as the police fore or ambulance team in RCRS and simplified map information as rectangles to dots. In figure 18 20, blue triangles are fire brigades, circles are roads, and boxes are buildings. Red circles are roads with hydrants and red boxes are buildings with fire. Gray line shows which roads and buildings are connected, and agents can only move to connected roads and can only extinguish connected fired buildings.

5.8 Experiment Setups

In the toy simulator, we upgrade observation spaces more effectively. There are three parts for observation space now. Fire condition of buildings, condition of agents, but the condition of colleague agent part is removed and the distance of each building part replaces it. The fire condition part has two values. Fire Boolean value is combined with the fieryness value and the fieryness is the number to express how the fire is big. 0 means that there is no fire on the building, and 1 4 means the level of fire. 1 is minimum fire, and 4 is maximum fire. 5 8 means the damaged level of the building after the building is extinguished. 5 means a burnout of the building. If the building is burned out, there is no more fire anymore. Targeting value expresses how many agents are targeting the building to extinguish the fire. 0 means any agents are not trying to extinguish the building, and 2 means two agents are trying to extinguish the building.

The second part is the condition of the Agent and there is the water level. Water level represents how much water the agent has. The water level is decreased when agents try to put the fire out (ex, 0.8->0.6) and if the water level is 0, agents can not try to put the fire out. x,y coordinate values are removed because the distance to each building part is added.

The third part is the distance of each building. It expresses the distance of the shortest path to each building.

The action space of the toy simulator is the same. Thus, if the model chooses the id of each building, the agent will go to the building and check whether the building is on fire or not, and if the building is on fire, the agent will try to extinguish the building until the water tank is empty or the fire is extinguished. Also, the agent can refill water on the nearest hydrant. It is also one of action to choose. Thus, including choices that do nothing, there are (the building number + 2) actions that the model can choose. The reward function also isn't changed.

In this toy simulator, we changed map size and test how the PPO self imitation algorithm works. We have three maps for the toy simulator, as figure 18 20. In the small map, There are 19 buildings with 3 initial random fires and a hydrant. In the middle map, there are 56 buildings with 4 initial random fires and three hydrants. On the big map, there are 78 buildings with 4 initial random fires and four hydrants. The time steps of every map are 100 as maximum.

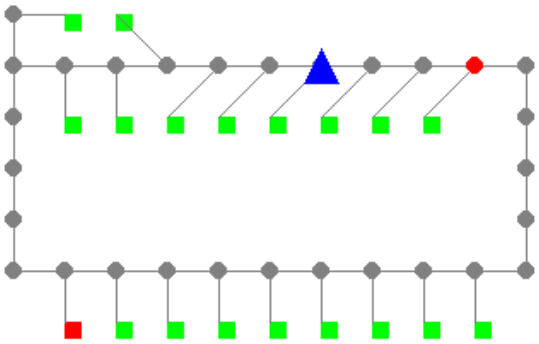


Figure 18: This is a small map for the toy simulator. There are 19 buildings with 3 initial random fires, and 1 hydrant

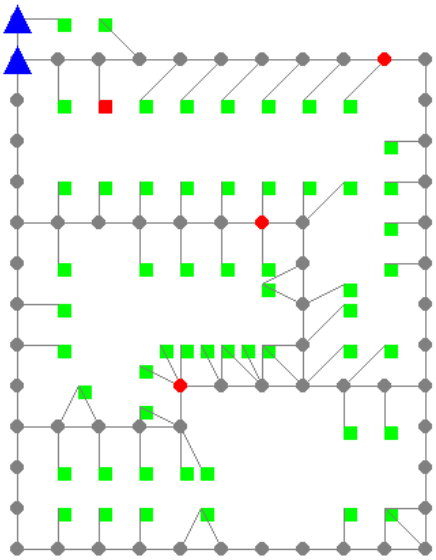


Figure 19: This is a middle map for the toy simulator. There are 56 buildings with 4 initial random fires, and 3 hydrants

5.9 Random Location of Initial Fire

As the first experiment of a toy simulator, we tested a small map with PPO self-imitation. As we can see in figure 21, agents can extinguish all fire even behavior cloning algorithms are not applied. It's a very

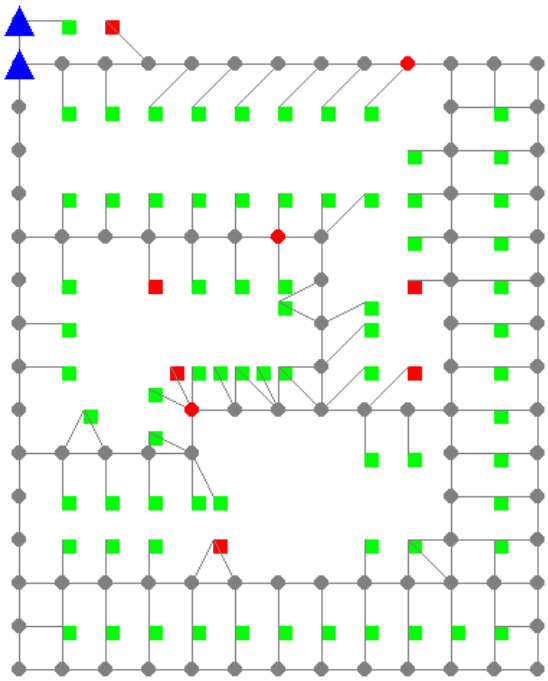


Figure 20: This is a big map for the toy simulator. There are 78 buildings with 4 initial random fires, and 3 hydrants

different result with the RCRS simulator with the random initial fired case.

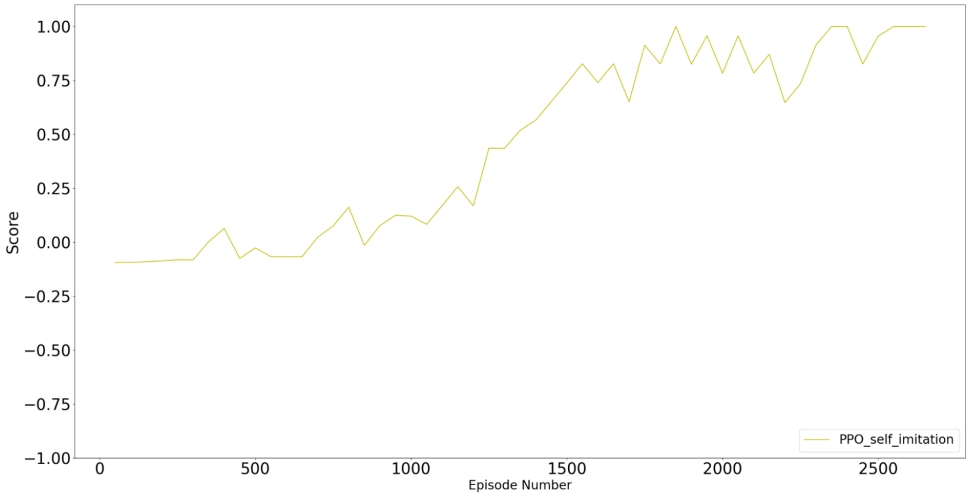


Figure 21: This is the small map result of the PPO algorithm with self-imitation. Even behavior cloning algorithms are not applied, the model can find how to extinguish all fire.

As the second experiment of the toy simulator, we tested the middle map with PPO self-imitation. As we can see in figure 22, agents can not extinguish all fire when behavior cloning algorithms are not applied. However, as figure 23, agents can extinguish all fires always when the behavior cloning

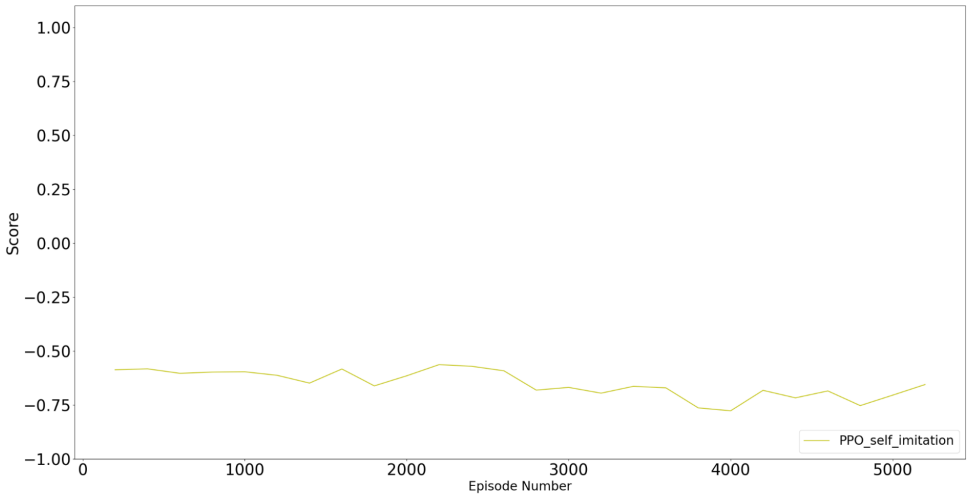


Figure 22: This is the middle map result of the PPO algorithm with self-imitation. Behavior cloning algorithms are not applied, and the model can not find how to extinguish all fire.

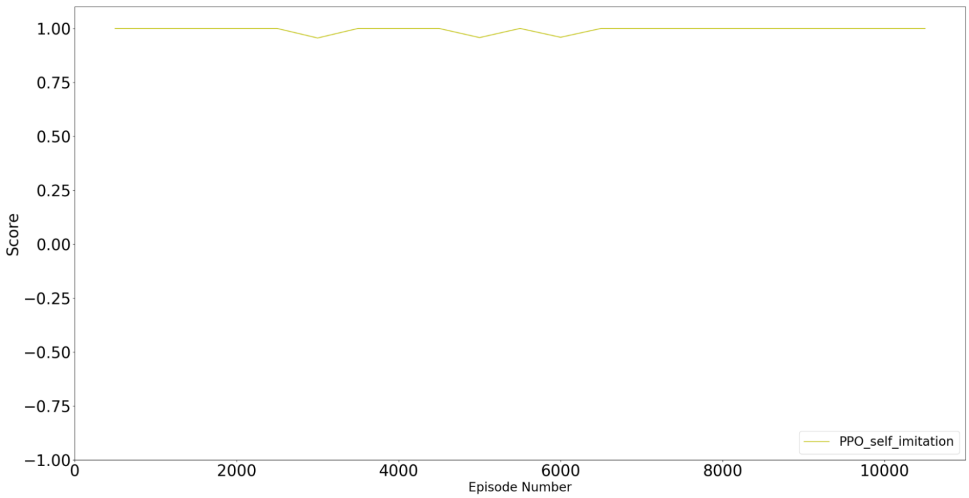


Figure 23: This is the middle map result of the PPO algorithm with self-imitation. When behavior cloning algorithms are applied, the model can find how to extinguish all fire.

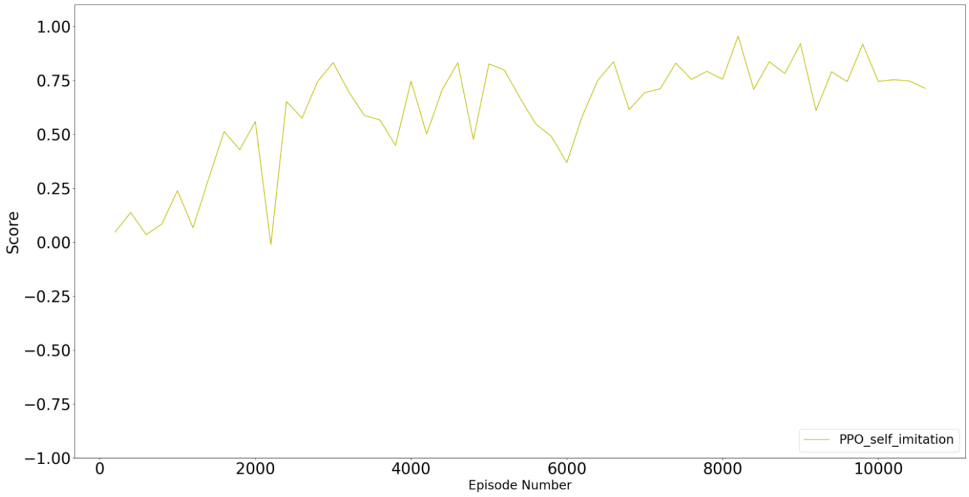


Figure 24: This is the middle map result of the PPO algorithm with self-imitation. In this graph, we reduced the pre-training epoch of the behavior cloning algorithm and we can see that model can learn how to extinguish all fire.

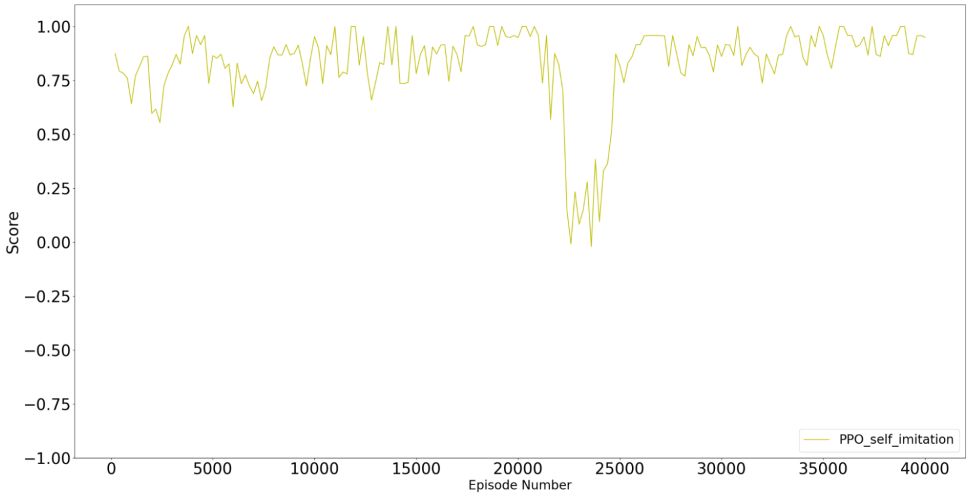


Figure 25: This is the bigger map result of the PPO algorithm with self-imitation and behavior cloning algorithm. The model can extinguish all fire but has a big variance and is unstable.

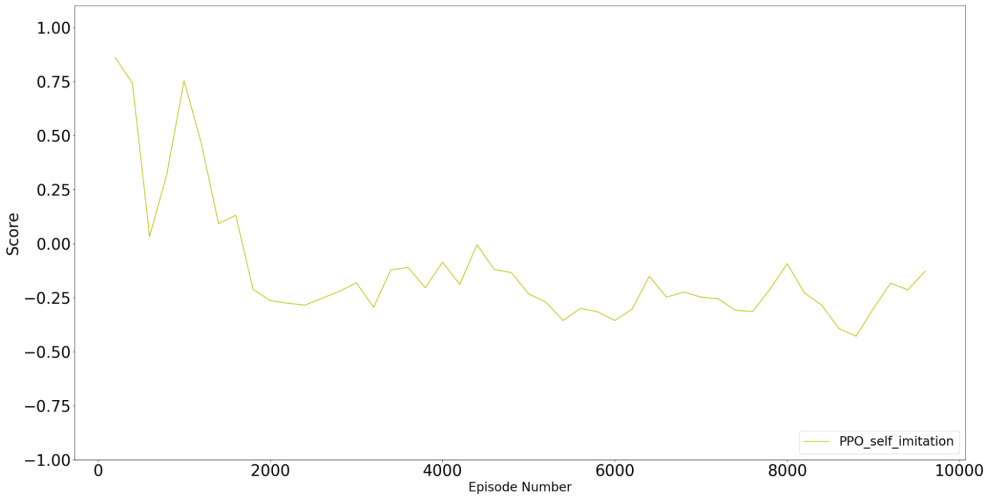


Figure 26: This is the big map result of the PPO algorithm with self-imitation and behavior cloning algorithm. In this case, we reduced the number of pra-train epoch and the model can not learn how to extinguish all fire.

algorithm is applied. The problem of figure 23 is we can see that model can extinguish all fire always but can not check the model learn correctly or not because the start point of return is already 1 after behavior cloning. Thus, we reduced the pre-train epoch of behavior cloning to decrease the start point of return as figure 24, and we can see that model learn correctly how to extinguish all fire. It's a very similar result with the RCRS simulator with a random initial fire case.

As the third experiment of the toy simulator, we tested a big map with PPO self-imitation with behavior cloning. As we can see in figure 25, the agent can extinguish all fire in most cases but has a big variance and is unstable. Also, its start point of return is 1, so we can not check the model learns correctly or not. Thus, we reduced the pre-train epoch of the behavior cloning algorithm again and we see that model can not learn how to extinguish all fire as figure 26.

5.10 Conclusions

In the RCRS simulator, we can see that the PPO algorithm with self-imitation can learn how to extinguish all fire correctly in the case of fixed initial fire location as figures 8 13. There are no reaming fired buildings at the end of a learning curve and get a similar score or better score than the greedy algorithm. However, in the random initial fire case, the model can not learn how to extinguish all fire as figure 14. We assume that only the initial steps of extinguishing a fire can prevent spreading fire, but the model can not find proper actions of initial steps because the model can not understand which part of time steps are more important than others. Thus, we collect data of greedy algorithm and do behavior cloning methods to model to expect that model can mimic the initial steps of greedy algorithm and finally get a better result than greedy algorithm as figure 15. In figure 16, we can see that the PPO algorithm with self imitation and behavior cloning extinguish all final in the case of 6 to 10 random initial fire. Also,

we can see that when the initial fire number increase, the model failed to extinguish all fire sometimes. Thus, in the RCRS map of this study, PPO algorithm with self-imitation and behavior cloning, which is the most effective methods what we tested, can extinguish all fire successfully in the case of fixed initial fired building location and can extinguish all fire with high probability in the case of the random initial fired building location.

When we changed map size in the toy simulator, the model can learn how to extinguish all fire in the small map even behavior cloning algorithm is not applied as figure 21, and the model can not learn how to extinguish all fire in the middle map with behavior cloning as figure 23 and 24. However model can not extinguish all fire in the bigger map in the case that the behavior cloning algorithm is applied with a small amount of epoch as figure 26. It assumes that when map size is increased, observation space size increase multiplies and is required to explore increase explosively, and that's why the model can not find the proper way in the small amount of epoch of pre-training data. Thus, in the toy simulator, we can see that the PPO algorithm with self-imitation and behavior cloning can extinguish all fire but can not learn how to extinguish all fire in the case that behavior cloning algorithm is applied with the small amount of epoch when map size increase.

VI Conclusion and Future Works

In this thesis, we studied what is the best way to (1) calibrate simulation models such as microscopic traffic simulation models or disaster simulation models in parallel, and then (2) train a team of agents such as autonomous vehicles and fire brigades to act optimally in the simulation by deep reinforcement learning? In the part of calibrating microscopic simulators, we used VISSIM, which is one of the most popular microscopic and dynamic traffic assignment simulations and calibrating simulations. We collected the real data of inductive loop sensors and traffic signals of Ulsan city and tested major algorithms to calibrate Root Mean Square Error(RMSE) between generated data based on real data and simulated data as target loss function as the equation 24 in the simulations. Restart-SPSA, WSPSA, and Active-CMA algorithms are tested, and to compare those algorithms, we tested simulation number versus target loss function and simulation time versus target loss function. In both comparisons, Active-CMA-ES shows the best performance as the figure 5. Especially, Restart-SPSA or WSPSA can only run two simulations inparallel, but Active-CMA-ES can run populasion number of simulations in parallel. Thus, as we can see in 5(b), 5(d), 5(f), and 5(h), Active-CMA-ES shows getting better performance as the number of CPU cores increases for the parallel running of simulations. Parallel algorithms are essential to speed up the calibration of traffic simulators, especially if we want to calibrate a traffic model in real-time given the online access to real-time traffic data. None of the existing popular calibration algorithms can achieve real-time calibration. Even though Active-CMA-ES is still not sufficiently fast enough for real-time calibration, it can scale up with the number of CPU cores of modern computer architecture. But we believe Active-CMA-ES is the first algorithm that can significantly speed up the calibration by parallel search and further improvement of Active-CMA-ES is possible by having better caching.

In the part of the train of a team of agents to act optimally, we used RCRS, which is one of the most popular disaster simulations, and a toy simulator based on RCRS. We set observation spaces, action spaces, and reward functions for each simulation and find effective algorithms, and compare them with greedy algorithms in the different scenarios. In the fixed initial fire location scenario, the PPO algorithm with self-imitation can extinguish all fire effectively as the figure 8,9,10, and in the random initial fire location scenario, deep reinforcement learning models required the behavior cloning algorithm to extinguish all fire effective as the figure 16,17. We assume that only the initial steps of extinguishing a fire can prevent spreading fire, but behavior cloning make model can find proper actions of initial steps. Also, We tested the most effective algorithm in different size maps using toy simulator and showed that the PPO algorithm with self-imitation and behavior cloning can extinguish all fire effectively in the most case as the figure 21,23,25. However, deep reinforcement learning model with behavior cloning is hard to learn how to extinguish all fire effectively with small amount of pre-training epoch when map size is big as the figure 26. We assume that when map size is increased, observation space size increase multiplies and is required to explore increase explosively, and the model can not find the proper way in the small amount of epoch of pre-training data. As a conclusion, in the toy simulator, we can see that the PPO algorithm with self-imitation and behavior cloning is the best algorithm than the TD3 algorithm or greedy algorithm, and the algorithm can extinguish all fire with different condition

of initial fire location and map size. However, the algorithm can not learn how to extinguish all fire if the behavior cloning algorithm is applied with a small amount of epoch and map size is big. Disaster measures such as overpowering fires are directly related to human lives but too complex and sensitive to find proper actions in the real world. Deep reinforcement learning can solve complex and sensitive problems now but it is not active in the disaster management field yet. In this study, we tested for a very part of disaster management using RCRS, and PPO algorithm with self-imitation and behavior cloning can extinguish all fire effectively. Although our method did not find the optimal policy for all parts of disaster simulations, we will study to find the optimal policy by solving complex problems that are closer to reality.

As a plan to solve the problem that model can not learn how to extinguish fire with small epoch of pre-training, we will apply a more advanced version of behavior cloning such as (1) the Human to Agent algorithm or (2) Long-Short-Term-Memory algorithm. (1) Human to agent algorithm collects data of human algorithm or greedy algorithm and do pre-training using the data like behavior cloning, but also use the data during model learning. It makes the model follow collected data with high probability in the early steps of learning, but reduces the probability as lower as learning progresses. This algorithm makes the model utilize the collected data and make the model avoid changing too much from pre-training data in the early step of learning, so we expect that model will make the model avoid losing its pre-training data such as figure 26. (2) Long-Short-Term-Memory saves each step of the model and gives a low decay rate to prevent losing effective actions. It can make the model learn which time steps of actions are important. For example, we expect that it makes the model to learn initial steps of fire simulation is much important than the last steps.

Right now, we only aim to extinguish the fire without earthquakes, civilians, and other agents such as police force or ambulance team. In future work, we will apply these things in RCRS, and find the optimal policy for each disaster agent in complex disasters. For the different types of agents, we will develop a specific observation space for each agent and learn each model separately first, and then combine the models in one main model. We believe that such an attempt can find the optimal way in a complex disaster model.

References

[1] L. Lu, Y. Xu, C. Antoniou, and M. Ben-Akiva, "An enhanced spsa algorithm for the calibration of dynamic traffic assignment models," *Transportation Research Part C: Emerging Technologies*, vol. 51, pp. 149–166, 2015.

[2] H. Yang, Y. Wang, and D. Wang, "Dynamic origin-destination estimation without historical origin-destination matrices for microscopic simulation platform in urban network," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2994–2999.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] G. A. Jastrebski and D. V. Arnold, "Improving evolution strategies through active covariance matrix adaptation," in *IEEE International Conference on Evolutionary Computation*, 2006, pp. 2814–2821.

[6] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 312–317.

[7] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone, "Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, July 2012.

[8] H. K. Lo and W. Y. Szeto, "A cell-based dynamic traffic assignment model: formulation and properties," *Mathematical and Computer Modelling*, vol. 35, no. 7-8, pp. 849–865, 2002.

[9] I. Yun and B. Park, "Estimation of dynamic origin destination matrix: A genetic algorithm approach," in *IEEE Intelligent Transportation Systems*, 2005, pp. 522–527.

[10] S. Baek, H. Kim, and Y. Lim, "Multiple-vehicle origin–destination matrix estimation from traffic counts using genetic algorithm," *Journal of Transportation Engineering*, vol. 130, no. 3, pp. 339–347, 2004.

[11] J. C. Spall *et al.*, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992.

[12] J. C. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 817–823, 1998.

[13] ——, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins Apl Technical Digest*, vol. 19, no. 4, pp. 482–492, 1998.

[14] P. I. C. Ryan *et al.*, "References to cma-es applications," *Strategies*, vol. 4527, no. 467, 2007.

[15] B. Kostic, L. Meschini, and G. Gentile, "Calibration of the demand structure for dynamic traffic assignment using flow and speed data: exploiting the advantage of distributed computing in derivative-free optimization algorithms," *Transportation Research Procedia*, vol. 27, pp. 993–1000, 2017.

[16] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[17] G. Gentile and L. Meschini, "Using dynamic assignment models for real-time traffic forecast on large urban networks," in *Proceedings of the 2nd International Conference on Models and Technologies for Intelligent Transportation Systems, Leuven, Belgium*, 2011.

[18] G. Gentile, L. Meschini, and N. Papola, "Spillback congestion in dynamic traffic assignment: a macroscopic flow model with time-varying bottlenecks," *Transportation Research Part B: Methodological*, vol. 41, no. 10, pp. 1114–1138, 2007.

[19] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

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

[21] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3878–3887.

Acknowledgements

Thank you very much for instruction of Professor Tsz-Chiu Au, Electrical and Computer Engineering at Ulsan National Institute of Science and Technology. Professor Tsz-Chiu Au help me to find the direction of study when I lost the way of study and writing.