Final Project Report

CSCI-SHU 375 Fall 2023 Reinforcement Learning

Elevator Group Control Systems Via Deep Reinforcement Learning

David Li,
Keith Wang,
Michael Yuan,
Zhihao Shan

supervised by
Mathieu Laurière

Contents

1 Introduction

Elevators are a common feature in our daily lives, particularly in academic buildings and apartment complexes. They are essential for efficiently transporting us to our desired floors. The introduction of multiple elevators is a response to the challenges posed by modern high-rise structures. These systems aim to transport passengers from the lobby to their respective floors, or between floors, in a reasonable time frame. While the addition of more elevators to a system doesn't eliminate waiting times, it does significantly reduce them compared to systems with a single elevator. It's generally believed that an optimal Elevator Group Control System (EGCS) should efficiently manage the demands of passengers by assigning elevators to floors while minimizing the Average Waiting Time (AWT). However, the complexity of the operation policy increases with the number of elevators and floors, making an ideal policy challenging to implement.

Over the past decades, various approaches have been explored to enhance the performance of EGCS, including zoning and search-based strategies. Yet, the quest for an optimal policy remains unsolved. Traditional methods are often effective under heavy traffic conditions, but there's a growing need for a more flexible policy, potentially one capable of self-learning, to determine the best course of action in real-time. Deep Reinforcement Learning (DRL) has been recognized as a promising method for self-learning and generating optimal policies in complex environments. Furthermore, Multi-Agent Deep Reinforcement Learning (MADRL) appears well-suited for addressing these challenges. In EGCS, each elevator can be represented as an agent, with a neural network to estimate its actions.

In our study, we first explored the application of DRL using an existing environment, RLevator [1], which, though simplified, still possesses key characteristics of real-world scenarios. We implemented two DRL algorithms: Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C). The results showed promising performance of DRL in a single-agent, multi-elevator context. Our primary goal was to employ MADRL to minimize the AWT in a more realistic environment. To this end, we used a sophisticated environment and implemented decentralized Deep Q Networks (DQN) and PPO with multiple agents. However, after several hours of simulation, the results did not support our hypothesis. We will further analyze these outcomes and the reasons for the discrepancy in the following sections.

2 Related Works

2.1 Elevator Group Control Systems

In the realm of elevator system optimization, Siikonen's works are great contributions. The "Elevator Traffic Simulation" [2] delves into the intricacies of modeling elevator group performance, shedding light on the interplay between elevator dynamics, control mechanisms, and passenger traffic patterns. The study challenges conventional assumptions about the direct correlation between interval and passenger waiting times, emphasizing the subtle influence of elevator configuration and performance capabilities. Siikonen's subsequent contribution, "Elevator Group Control with Artificial Intelligence" [3], introduces a novel control system leveraging fuzzy logic and artificial intelligence for optimized passenger service. By adapting to daily traffic variations through statistical forecasts, the approach effectively reduces average waiting times and enhances ride time distribution. Real-world implementations underscore the practical benefits of this innovative control strategy, exemplified by substantial improvements in landing call times, especially evident in the modernization of an outdated electronic control system in an office building. Together, these works advance our understanding of elevator dynamics and offer practical solutions for improved service levels in diverse settings.

2.2 Deep Reinforcement Learning

This section discussed several deep reinforcement algorithms and techniques we seek to solve our problem on the elevator group control (EGC). Mnih's "Asynchronous Methods for Deep Reinforcement Learning" [4] contributes a lightweight framework for deep reinforcement learning, employing asynchronous gradient descent (A2C) to optimize deep neural network controllers. The study's asynchronous actor-critic method outperforms previous existing methods, demonstrating efficiency in training time and success across diverse tasks, from continuous motor control problems to navigating 3D mazes with visual input.

In further extension of the policy-based method, Schulman's "Proximal Policy Optimization Algorithms" [5] introduces the Proximal Policy Optimization (PPO). PPO's alternating approach between sampling data and optimizing a surrogate objective function exhibits advantages over trust region policy optimization (TRPO), showcasing simplicity, generality, and improved sample complexity across benchmark tasks.

To address the sparsity of reward under certain situations, "Policy invariance under reward

transformations: Theory and application to reward shaping" [6] explores conditions preserving the optimality of a policy in a Markov decision process (MDP) during reward function modifications. Beyond linear transformations, the study emphasizes potential-based rewards in reinforcement learning, offering methods to address challenges and bugs in reward-shaping procedures, ultimately reducing learning time.

Moreover, it seeks to tackle the challenge of learning cooperative policies in complex, partially observable domains without explicit agent communication. "Cooperative multi-agent control using deep reinforcement learning" [7] showcases the efficacy of policy gradient methods over temporal-difference and actor-critic methods in scaling reinforcement learning algorithms for diverse tasks involving numerous cooperating agents, by extending single-agent deep reinforcement learning algorithms to cooperative multi-agent systems and employing a multi-agent variant of curriculum learning.

2.3 Deep Reinforcement Learning for Elevator Group Control

To leverage the power of reinforcement learning (RL) to the elevator group control(EGCS), multiple attempts have been made by previous researchers. Crites's work in "Improving Elevator Performance Using Reinforcement Learning" [8] applies reinforcement learning (RL) to the intricate problem of elevator dispatching, demonstrating superior performance over existing heuristic control algorithms. Their team of RL agents, each responsible for an individual elevator car, showcases the effectiveness of RL in large-scale stochastic dynamic optimization problems.

Building on RL advancements, Crites's subsequent paper with Barto, "Elevator Group Control Using Multiple Reinforcement Learning Agents" [9], explores collective RL algorithms for addressing complex control problems. The study demonstrates the potential of multi-agent RL in optimizing elevator group control, surpassing heuristic algorithms even with noise and incomplete state observations.

In a different vein, "Optimal Elevator Group Control via Deep Asynchronous Actor-Critic Learning" [10] introduces the asynchronous advantage actor-critic (A3C) method for optimal elevator group control. Employing deep convolutional and recurrent neural networks, this novel deep RL approach shows promise in reducing average waiting times within complex building environments, showcasing adaptability and efficiency in transporting passengers.

"Application of Deep Q Learning with Simulation Results for Elevator Optimization" [11] proposes a methodology that integrates programming and mathematics to optimize elevator wait

times. Using simulated user data and a canonical three-peak model, the authors create a naive model grounded in intuitive elevator logic and employ Deep Q Learning for further optimization, highlighting considerations such as capacity, acceleration, and maximum wait time thresholds. The study operates primarily within a Markov Decision Process (MDP) framework, later addressing the limitations of this assumption in characterizing the stochastic nature of Elevator Group Control Systems (EGCS).

3 Environment

Elevator scheduling is the process of efficiently managing the operation of elevators in a building to transport passengers between floors. It is crucial to optimize factors such as waiting time and power consumption. The goal is to make elevator systems responsive and adaptive to real-time demands, ensuring efficient and safe transportation within a building.

3.1 States

The state space is very large under the elevator group control setting. In our environment, there are direction buttons (hall call) for the queues, destination floor buttons (car call) inside the elevators, destinations of passengers, the number of passengers who pressed the destination buttons in the elevator, the number of passengers who pressed the hall call and are waiting in the queue, the waiting time and existence time of each passenger, the number of passengers who would like to join the queue but were rejected due to a large number of waiting passengers, the number of passengers who left the queue because the waiting time exceeded the passengers' maximum waiting time, the number of passengers in the elevator, the number of passengers who went toward their destination floors while in the elevator, the number of passengers who went away from their destination floors while in the elevator and the number of passengers who reached their destination floors. Additionally, the elevators are aware of their current floor position.

- **Observation Space**

  Not all the information in the states is available to the agent. The limited observation space is meant to mimic a realistic elevator environment, where the only data available to an elevator system are the direction button (hall call) for the queues and the destination floor buttons (car call) inside the elevators. Additionally, the elevators are aware of their current floor position. The three components are all Multi-binary types (0 or 1). The

true destinations of passengers in the queue and the number of passengers who pressed the destination buttons are not known. Numerically, if we have n floors and k elevators, then the observation space shape will be represented as follows:

- **Car Call:** $n \times k$

  Each number indicates whether the button of each floor in the elevator is pressed or not.

- **Hall Call:** $2 \times n$

  Each number indicates whether the button (up or down) on the floor is pressed or not.

- **Elevator Position:** $n \times k$

  Each number indicates whether the elevator is on this floor or not.

The observation space length will be $n \times k + 2 \times n + n \times k = 2n(k + 1)$.

3.2 Actions

There are 6 actions for each elevator. In the implementation, a single agent is used to make decisions of actions for multiple elevators, thus the agent has a multi-discrete action space.

- Stay on the current floor without loading or unloading passengers
- Go up one floor, but stay on the same floor if on the top floor
- Go down one floor, but stay on the same floor if on the bottom floor
- Load passengers intended to go up within the elevator's maximum capacity and stay on the current floor
- Load passengers intended to go down within the elevator's maximum capacity and stay on the current floor
- Unload passengers whose destination is the current floor and stay on the current floor

3.3 State Transition Probability Function

The state transition probability function $p(s_{t+1}|s_t, a_t)$ is a joint distribution of the motion of elevators, new passengers generation, and new destinations of new passengers. The state transition probability only depends on the current state, but not on any previous states. In our setting, the motion of elevators is deterministic based on the actions taken, the arrival of new passengers

follows poison distributions based on floor number, and the destinations of new passengers are randomly decided. To emphasize, if an elevator is on the top floor but still chooses to go up, the elevator will remain on the top floor. Similarly, if an elevator is on the bottom floor but still chooses to go down, the elevator will remain on the bottom floor.

3.4 Reward Function

There are 7 terms in our reward $r(s_t, a_t)$, and all of them can be derived from the current state and action s_t, a_t.

Positive terms:

- Number of passengers reaching their destinations, denoted as b_t
- Number of passengers going toward their destinations while in the elevator, denoted as c_t

Negative terms:

- Number of passengers trying to join a full queue but are denied, denoted as d_t
- Number of passengers reaching their maximum waiting time and leaving the queue, denoted as e_t
- Number of passengers going away from their destination floors while in the elevator, denoted as f_t
- Number of passengers in the elevator, denoted as g_t
- Number of passengers waiting in the queue, denoted as h_t

  The reward function is defined as follows:

$$r_t = 30b_t + 5c_t - 10d_t - 10e_t - 3f_t - g_t - h_t$$

4 Algorithms

Traditional elevator scheduling algorithms are typically rule-based and rely on pre-defined heuristics. However, these approaches are often suboptimal and may not adapt well to changing conditions. In contrast, reinforcement learning (RL) offers a data-driven approach to elevator scheduling. By modeling the problem as a Markov Decision Process (MDP), RL algorithms can

learn optimal policies for elevator control through interactions with the environment. And since we have a multi-discrete action space, we first gave preference to policy-based methods like A2C, and PPO.

4.1 Deep Q-Network (DQN)

DQN is feasible since both observation space and action space are discrete and not too large. The DQN we implemented is not exactly the same DQN as in stable baseline and is twisted for multi-discrete action space. There are multiple Q local and Q target network pairs, one for each dimension of the action space. For example, each elevator has 6 possible actions, and if there are three elevators, then there would be three Q networks. Then the action generated would be a combination of each action from each Q network.

Algorithm 1 DQN

Hyperparameters: replay buffer capacity N, reward discount factor γ, delayed steps C for target action-value function update, ϵ-greedy factor ϵ
Input: empty replay buffer D, initial parameters list $[\theta_1, \theta_2, ...]$ of action-value function list $[Q_1, Q_2, ...]$
Initialize target action-value function $[\hat{Q}_1, \hat{Q}_2, ...]$ with parameter $[\hat{\theta}_1, \hat{\theta}_2...] \leftarrow [\theta_1, \theta_2, ...]$
for episode $= 0, 1, 2, \ldots$ **do**
  Initialize environment and get observation O_0
  Initialize sequence $S_0 = \{O_0\}$ and preprocess sequence $\phi_0 = \phi(S_0)$
  for $t = 0, 1, 2, \ldots$ **do**
    for each Q-network Q_i in $[Q_1, Q_2, ...]$ **do**
      With probability ϵ select a random action A_t, otherwise select $A_{t,i} = \arg\max_a Q_i(\phi(S_t), a; \theta_i)$
    end for
    action $A_t = [A_{t,1}, A_{t,2}...]$
    Execute action A_t and observe O_{t+1} and reward R_t
    If the episode has ended, set $D_t = 1$. Otherwise, set $D_t = 0$
    Set $S_{t+1} = \{S_t, A_t, O_{t+1}\}$ and preprocess $\phi_{t+1} = \phi(S_{t+1})$
    Store transition $(\phi_t, A_t, R_t, D_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, A_j, R_j, D_j, \phi'_j)$ from D
    for each Q_i in $[Q_1, Q_2, ...]$ and $\hat{Q}_i$ in $[\hat{Q}_1, \hat{Q}_2, ...]$ **do**
      If $D_j = 0$, set $Y_j = R_j + \gamma \max_{a'} \hat{Q}_i(\phi'_j, a'; \hat{\theta}_i)$. Otherwise, set $Y_j = R_j$
      Perform a gradient descent step on $(Y_j - Q_i(\phi_j, A_j; \theta_i))^2$ with respect to θ_i
      Synchronize the target networks $\hat{Q}_i$ every C steps
    end for
    If the episode has ended, break the loop
  end for
end for

4.2 Advantage Actor Critic (A2C)

As we have multi-discrete action space, it might be better to use policy-based learning such as A2C. A2C can handle high-dimensional action spaces better than value-based methods like DQN, as they do not require the action-value function for each action-state pair.A2C a set of actors controlling how the agent behaves, and a critic measuring how good the actions taken. Moreover, the A2C also replaces the standard $R(\tau)$ with the advantage, which can be estimated by the TD error: $\hat{A}_t = R_t + \gamma V_\psi^{T\theta}(S_{t+1}) - V_\psi^{T\theta}(S_t)$. It measures how better taking the action is to further stabilize learning. The pseudo-code is provided below.

Algorithm 2 A2C

Master:
Hyperparameters: step size η_ψ and η_θ, set of workers W
Input: initial policy parameters θ_0, initial value function parameters ψ_0
Initialize $\theta = \theta_0$ and $\psi = \psi_0$
for $k = 0, 1, 2, \ldots$ **do**
  $(g_\psi, g_\theta) = 0$
  for worker in W **do**
    $(g_\psi, g_\theta) = (g_\psi, g_\theta) + \text{worker}(\psi^{T\theta}, \theta)$
  end for
  $\psi = \psi - \eta_\psi g_\psi$; $\theta = \theta + \eta_\theta g_\theta$.
end for

Worker:
Hyperparameters: reward discount factor γ, the length of trajectory L
Input: value function $V_\psi^{T\theta}$, policy $T\theta$
Run policy $T\theta$ for L time steps, collection $\{S_t, A_t, R_t, S_{t+1}\}$
Estimate advantages $\hat{A}_t = R_t + \gamma V_\psi^{T\theta}(S_{t+1}) - V_\psi^{T\theta}(S_t)$
$J(\theta) = \sum_t \log T\theta(A_t|S_t)\hat{A}_t$
$J_\psi^{T\theta}(\psi) = \sum_t \hat{A}_t^2$
$(g_\psi, g_\theta) = (\nabla_\psi J_\psi^{T\theta}(\psi), \nabla_\theta J(\theta))$
return (g_ψ, g_θ)

4.3 Proximal Policy Optimization (PPO)

For A2C, a small step in the parameter of the policy network might change the probability distribution space of the policy largely. Unlike A2C, PPO seeks to optimally update the policy without stepping too far in the probability distribution space of the policy that causes performance collapse. PPO addresses the problem and keeps the new policy close to the old through several different tricks like penalties or clipping. In this project, we implemented a clip version of PPO in stable-baseline. The pseudo-code is provided below.

Algorithm 3 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $D_k = \{s_i, a_i, r_i, s'_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ using value network V_ϕ.

for each minibatch B in D_k **do**

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t),\ g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

via Adam optimizer

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

via Adam optimizer

end for

end for

5 Results

The training rewards of the three algorithms are provided. We plotted A2C and PPO on the right side of Figure 1 for comparison purposes. And DQN is shown on the left side of Figure 1. All three algorithms can improve the rewards through training. However, DQN converges much quicker than the other two since it uses a replay buffer and is thus more sample-efficient. Compared to PPO, the fluctuation in rewards of A2C is larger, which aligns with our assumption that some updates would step too far in the probability distribution space of policy.

5.1 Implement details

In our experiments, we configured the existing environment with specific parameters: the simulation environment termination steps for 1,000 steps, utilized 3 elevators that have the capacity of 10 passengers each, featured a building with 6 floors and each floor can only have a maximum of 20 passengers in queue. For three algorithms, we set the learning rate to 1e-3 for DQN and 3e-4 for A2C and PPO, and discounted factor gamma to 0.999. For DQN, we conducted 800 episodes, set the replay buffer size to 1e5, and the batch size to 256, updated the network every 4 steps, and cloned the network every 1,000 steps. In the case of A2C, the configuration included 1,024 steps per update, a Generalized Advantage Estimation (GAE) lambda of 0.98, an entropy

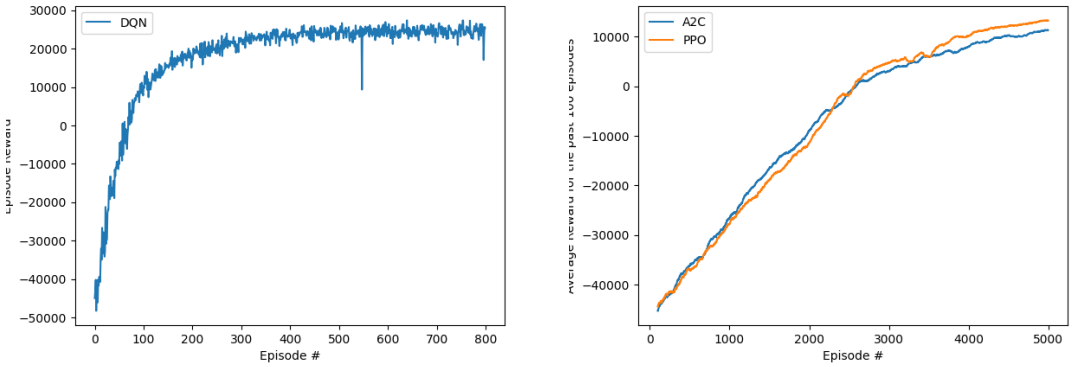

Figure 1: training rewards

coefficient of 0.01, and a total of 5 million timesteps. Similarly, PPO was set with 1,024 steps per update, a GAE lambda of 0.98, an entropy coefficient of 0.01, a total of 5 million timesteps, a batch size of 256, and 4 epochs per update cycle. These configurations were carefully selected to optimize the elevator system's performance after we tuned the parameters several times.

6 Discussion

After solving for a relatively simple environment, this project seeks to explore more environment settings that further align with the setting in Crites &. Barto's 1998 paper[9], as well as trying to extend it to a multi-agent Deep RL.

6.1 Environment

- **States**

  The state is defined with continuous state space, continuous time, discrete event, and non-stationary, and incomplete state information. The state space is continuous because it includes the elapsed times since any hall calls were registered, which are real-valued. But in practice, the real values are approximated as binary values. For the case of 10 floors with 4 agents, the observable state space is roughly about: $2^{18} * 2^{40} * 18^4 \approx 10^{22}$, with 2^{18} possible combinations of the 18 hall call buttons (up and down buttons at each landing except the top and bottom), 2^{40} possible combinations of the 40 car buttons, 18^4 possible combinations of the positions and directions of cars. Other parts of the states are not fully observable, for example, the exact arrival of each passenger, or the exact number of passengers waiting on each floor.

- **Actions**

  Actions are defined based on the status of an elevator with several constraints. An elevator can be either idle or in motion. If idle, the actions are either "move up" or "move down"; If in motion, actions are either "stop at the next floor" or "continue past the next floor" Constraints:

  - cannot pass a floor if a passenger wants to get off there
  - cannot turn until it has serviced all the car buttons in its present direction
  - cannot stop on a floor unless someone wants to get on or off there
  - cannot stop to pick up passengers on a floor if another car is already stopped

- **State Transition Probability Function**

  The state transition probability function $p(s_{t+1}|s_t, a_t)$ is a joint distribution of the motion of elevators, new passengers generation, and new destinations of new passengers. The state transition probability only depends on the current state, but not on any previous states. In our setting, the motion of elevators is deterministic based on the actions taken, the arrival of new passengers follows poison distributions based on floor number, and the destinations of new passengers are randomly decided.

- **Reward Function**

  Elevator systems can be modeled as discrete event systems[12], where significant events (such as passenger arrivals) occur discretely, but the amount of time between events is a real-valued variable. Since the task is to minimize the wait time of the passenger, the reward in this setting is converted to a cost that measures the sum of the squared average waiting time within the time interval of two significant events. The cost between two significant events(happened at t_x, t_y) is given as:

$$\int_{t_y}^{t_x} \sum_p e^{-\beta(\tau - t_x)} (\tau - t_x + w_p)^2 d\tau$$

  .

6.2 Problem

Our implementation of both DQN and PPO in this complex setting resulted in bad learning performance. We conclude the following reasons that could probably lead to this unsatisfied

result:

- State Space v.s. Observation Space

  In our implementation, the observation each agent gets is only part of the current state. The lack of information, like the exact arrival time of each passenger, can limit the agents' learning, especially when the reward depends on some of the missing information.

- Too complicated action space

  In this new setting of the environment, we have a larger action space than our previous one. Moreover, since current legal action depends on the current state that is determined by previous action, previous actions taken could have further influence on later decisions and rewards, making it more difficult for the agents to learn.

- Complexity of multi-agent setting

7 Conclusion

This report presented a study on the application of Deep Reinforcement Learning algorithms to optimize Elevator Group Control Systems. Initial tests with DQN, A2C, and PPO in an OpenAI gym-based environment were successful. However, when we later tried to implement multi-agent Reinforcement Learning algorithms in a more complex, real-life scenario, it did not produce the expected improvements in minimizing passenger waiting time. The challenges faced point to the limitations of our implementations of multi-agent DRL methods when dealing with the high-dimension action space and partially observable states of real-world EGCS. In our future work, we hope to solve these problems.

References

[1] M. Burke, "Rlevator: A farama gymnasium environment for elevator control," 2023. [Online]. Available: https://mwburke.github.io/RLevator

[2] M.-L. Siikonen, "Elevator traffic simulation," *SIMULATION*, vol. 61, no. 4, pp. 257–267, 1993.

[3] ——, "Elevator group control with artificial intelligence," *Citeseer*, 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:14627167

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[6] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99. Citeseer, 1999, pp. 278–287.

[7] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.

[8] R. Crites and A. Barto, "Improving elevator performance using reinforcement learning," *Advances in neural information processing systems*, vol. 8, 1995.

[9] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine learning*, vol. 33, pp. 235–262, 1998.

[10] Q. Wei, L. Wang, Y. Liu, and M. M. Polycarpou, "Optimal elevator group control via deep asynchronous actor-critic learning," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 12, pp. 5245–5256, 2020.

[11] Z. Cao, R. Guo, C. M. Tuguinay, M. Pock, J. Gao, and Z. Wang, "Application of deep q learning with simulation results for elevator optimization," 2022.

[12] C. Cassandra, *Discrete Event Systems: Modeling and Performance Analysis*. Homewood, IL: Aksen Associates, 1993.