applied sciences

MDPI

*Article*

# Application of Reinforcement Learning in Decision Systems: Lift Control Case Study

**Mateusz Wojtulewicz [1] and Tomasz Szmuc [2,*]**

[1] Center of Excellence in Artificial Intelligence, AGH University of Krakow, 30-059 Krakow, Poland; mateusz.wojtulewicz@gmail.com
[2] Department of Applied Computer Science, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, AGH University of Krakow, 30-059 Krakow, Poland
* Correspondence: tsz@agh.edu.pl

**Abstract:** This study explores the application of reinforcement learning (RL) algorithms to optimize lift control strategies. By developing a versatile lift simulator enriched with real-world traffic data from an intelligent building system, we systematically compare RL-based strategies against well-established heuristic solutions. The research evaluates their performance using predefined metrics to improve our understanding of RL's effectiveness in solving complex decision problems, such as the lift control algorithm. The results of the experiments show that all trained agents developed strategies that outperform the heuristic algorithms in every metric. Furthermore, the study conducts a comprehensive exploration of three Experience Replay mechanisms, aiming to enhance the performance of the chosen RL algorithm, Deep Q-Learning.

**Keywords:** decision systems; artificial intelligence; reinforcement learning; lift control



**Citation:** Wojtulewicz, M.; Szmuc, T. Application of Reinforcement Learning in Decision Systems: Lift Control Case Study. *Appl. Sci.* **2024**, *14*, 569. https://doi.org/10.3390/app14020569

Academic Editor: Douglas O'Shaughnessy

Received: 29 November 2023
Revised: 27 December 2023
Accepted: 4 January 2024
Published: 9 January 2024



## 1. Introduction: Contribution of the Paper

In our fast-paced world, lifts are irreplaceable in multi-floor buildings, transporting hundreds of passengers daily. As stated in an IBM survey from 2010, the employees in New York collectively spent 16 years waiting for a lift and another 6 years inside the lift cabin during a 12-month period [1]. The time spent waiting to be served by the lift control system can lead to significant frustration and tangible losses. Addressing the challenge of optimizing lift control to minimize passenger waiting times in a predetermined and deterministic environment is a proven $\mathcal{NP}$-hard problem [2]. The real-world scenario introduces uncertainties that necessitate adaptive decision-making without complete knowledge of future events. Heuristic algorithms have traditionally tackled this issue [3], with recent exploration of solutions based on artificial intelligence algorithms [4–7].

The dissatisfaction of lift users, the difficulty of the problem, and the potential improvements achievable through reinforcement learning techniques were the motivations behind the studies and experiments presented in this article. While the task seems intuitive, finding an optimal algorithm is, as previously mentioned, complex, and this complexity grows in real-world scenarios.

This case study's primary objective and main contribution is to demonstrate a viable path and actions for leveraging reinforcement learning methods in addressing a real-world control problem, specifically lift control. To achieve this goal, a universal lift emulator was developed, incorporating real-world traffic distribution data from the intelligent building system of ASTOR sp. z o.o. Company, Krakow, Poland. The research involves a comparison between strategies developed using reinforcement learning algorithms and established heuristic solutions, evaluating their performance through predefined metrics. Additionally, a secondary aim of the study is to compare the effectiveness of three Experience Replay mechanisms. These mechanisms are designed to improve the performance of the Deep

Q-Learning algorithm, which was chosen as the reinforcement learning algorithm for this investigation.

The article begins by defining the lift control problem and introducing the concept of our proposed solution in Section 2. Traditional heuristic algorithms are enumerated and explained according to a rationale for choosing the Reinforcement Learning (RL) to tackle the problem. The RL methods, particularly Q-Learning algorithms and their variants with the Experience Replay mechanism, are described. Section 3 provides an overview of significant solutions found in the literature.

Moving on to Section 4, we delve into the steps taken in our solution, explaining the construction of a lift control system emulator characterized by real-world passenger distribution. We detail how this emulator serves as the environment for RL learning, the chosen RL algorithms, and the learning process itself. In Section 5, we describe the evaluation procedure, and showcase and analyse its results. Concluding the article in Section 6, we discuss the findings and outline potential ideas for future research.

## 2. Description of the Problem and Concept of the Solution

The lift control system consists of one or more lifts operating within a multi-story building. In the latter case, it is referred to as group control [8]. The system registers lift calls from the floors and cabins and processes them according to a chosen strategy. Calls from the floors are generated by physical buttons located on the floors, sometimes distinguishing between requests to travel upwards or downwards. Inside the lift cabin, there are stop buttons that trigger calls to specific floors. The objective of the lift control system is to serve its passengers optimally in the following aspects:

- The average and maximum wait times for passengers as they await the lift's arrival.
- The average and maximum travel times for passengers to reach their destinations.
- The distance traveled by each lift.

Finding an optimal strategy in this context is part of a broader category of real-time transportation and demand-related problems, often referred to as online-dial-a-ride [9].

### *2.1. Problem Definition*

The task presented to the lift system controller can be defined as follows: for $N$ passengers wishing to travel from the starting floor $s_i$ to the destination floor $d_i$, denoted as $(s_1, d_1), (s_2, d_2), \ldots, (s_N, d_N)$, the goal is to determine a sequence of visited floors $p_1, p_2, \ldots, p_k$ in such a way that, for each passenger, their starting floor $s_i$ precedes their destination floor $d_i$. Moreover, if the task includes finding a sequence of minimal length, the problem becomes $\mathcal{NP}$-complete due to a reduction from the problem of cyclic vertex cover, as demonstrated by Seckinger and Koehler [2] in 1999. Similarly, the problem addressed in this work may share similarities with variants of the Traveling Salesman Problem or the Routing Problem, but none of them fully models it. This is because the cost of travel depends not only on the order in which passengers are serviced but also on how they are grouped. Additionally, these aforementioned problems operate in static environments and assume full knowledge of them, while in lift control systems, passengers appear dynamically, and the complete system state is unknown. Hence, heuristic algorithms for real-time lift system control have been developed. Below, we outline those that can be used in a single-lift system, as described by Crites and Barto [3].

### *2.2. Traditional Heuristic Algorithms*

The Longest Queue First (LQF) algorithm utilizes a queue to determine which call to service at a given moment. When a passenger summons the lift from a floor or from inside the cabin, their request is added to the end of the queue. Requests are serviced in order of the longest waiting time. On each floor, there is typically only one button, so passengers cannot specify whether they want to travel up or down. Additionally, requests generated by pressing buttons inside the cabin take precedence over those generated on floors. This

prioritization ensures that passengers already inside are serviced before the lift responds to a new passenger's call.

The Collective Algorithm (CA) extends the Longest Queue First algorithm. It involves stopping at intermediate floors during the service of the longest-waiting request, provided that there are active cabin or floor requests for those intermediate floors. Moreover, the Last Mission (LM) variant of this algorithm only stops at a floor if the direction indicated by its request aligns with the current direction of lift movement. In this variation, each floor typically has two call buttons, allowing passengers to specify the direction of their travel.

Those heuristic approaches, although they solve the problem, have their limitations. They often rely on simple rules, like servicing the longest waiting queue, which may not optimize passenger waiting times or energy usage. These methods struggle to adapt to changing traffic patterns. As a result, more advanced techniques capable of adapting to passenger traffic patterns are being explored to address these limitations.

### 2.3. Reinforcement Learning

Reinforcement Learning (RL) offers a promising solution to the limitations of traditional heuristic approaches in lift control. This specific field of machine learning introduces a dynamic approach in which an agent learns through active interaction with its environment. RL finds widespread application in various fields, including robotics, finance, healthcare, and game playing [10,11]. In robotics, RL is widely employed to enable adaptive and autonomous behavior in robots, allowing them to navigate and perform tasks in unstructured and dynamic environments [12]. It offers the unique advantage of exploring complex environments to find intricate strategies that are often beyond the scope of simple rule-based traditional solutions. This adaptability makes RL well-suited for lift control systems, where the dynamic nature of traffic patterns and user demands requires an intelligent and evolving decision-making approach.

The environment can be represented as a Markov Decision Process (MDP). This MDP is composed of several components: a set of possible states $S$, a set of available actions $A$, and two functions—the transition function $T(s, a, s')$ and the reward function $R(s, a, s')$. The transition function captures the dynamics of the environment, determining the probability of transitioning from state $s$ to $s'$ when taking action $a$. On the other hand, the reward function relates to the rewards received in this process, quantifying the goodness of the action choice.

In each time step $t \in \{0, 1, 2 \ldots\}$ during the interaction loop, the agent observes a state of the environment $s_t \in S$ and selects an action $a_t \in A$. Consequently, he receives an immediate reward $r_t \in \mathbb{R}$ and observes a new state $s_{t+1}$ in accordance with the dynamics of the environment. The agent's choices are represented by a policy function denoted as $\pi$, defined as follows:

$$\pi(a, s) = Pr(a_t = a \mid s_t = s). \tag{1}$$

The policy function assigns a probability value to each possible state-action pair. It must be ensured that the sum of probabilities for all available actions in a given state equals 1.

Based on the policy, we can define an action-value function, denoted as $Q^\pi(s, a)$, which represents the expected cumulative reward when starting from state $s$, taking action $a$, and then following policy $\pi$ thereafter:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s,\ a_0 = a;\ \pi\right], \tag{2}$$

where $\gamma \in [0, 1]$ is the discount factor that trades off the importance of immediate and later rewards. The optimal value for this state-action pair is $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. Then, the optimal policy can be easily derived by assigning a probability of 1 to the highest valued action in each state while setting a probability of 0 to all other actions in that state.

#### 2.3.1. Q-Learning and Deep Q-Learning

Q-Learning is a form of temporal difference learning, where the goal is to estimate the optimal action values by learning a parameterized action-value function $Q(s, a; \theta_t)$ [13]. In a standard Q-Learning algorithm, after taking an action $a_t$ in state $s_t$ and observing the immediate reward $r_t$ and a new state $s_{t+1}$, the target action value is calculated using the Bellman's equation as follows:

$$Y_t^Q = r_t + \gamma \max_a Q(s_{t+1}, a; \theta). \tag{3}$$

The parameter is updated using the stochastic gradient descent method to modify the current value $Q(s_t, a_t; \theta)$ towards the target value $Y_t^Q$.

In Deep Q-Learning, a multi-layered neural network is used to approximate the action-value function. For each state $s$, the network outputs a vector of action values $Q(s, \cdot; \theta)$ of length $card(A)$, where $\theta$ are the parameters of the network. To mitigate value estimation errors, Mnih et al. [14] introduced a dual-network approach. They utilized the online network for making updates and the target network to estimate the target value $Y_t^Q$.

#### 2.3.2. Experience Replay

Experience Replay technique has been applied to improve the stability and efficiency of the learning process. Its fundamental approach involves an Experience Replay buffer, as employed by Mnih et al. [14]. It is a fixed-size storage for experience, or (state, action, reward, next state) transitions, encountered by the agent during interactions with the environment. During training, a predefined number of such transitions are uniformly sampled from this buffer to perform parameter updates.

A more advanced variant, known as Prioritized Experience Replay, was introduced by Horgan et al. [15]. This technique redefines the sampling process to prioritize transitions based on their estimated importance calculated through the temporal differences, denoted as $Y_t^Q - Q(s_t, a_t)$. The experiences are sampled from a probability distribution that is proportional to the stored temporal differences. Consequently, transitions with a higher impact on the learning process are used more frequently, thereby improving the overall learning efficiency.

In certain environments, there may be states or groups of states that are encountered less frequently by the agent, leading to an imbalance in the experience replay buffers. This imbalance can result in the agent making suboptimal decisions in these less common states. To solve this challenge in the context of controlling traffic lights using reinforcement learning, Wei et al. [16] introduced the Memory Palace technique. In this approach, experiences are distributed across multiple memory palaces (essentially conventional Experience Replay buffers) based on the active light phase. During training, the same number of samples are drawn from all of the palaces, ensuring balance. This prevents more frequent phase states from dominating the training process, ultimately improving the accuracy of the final value function estimation.

## 3. Related Works

The solution proposed by Imasaki et al. [4] utilizes a fuzzy neural network to predict passenger wait times under various control parameters. The input state space is partitioned using fuzzy logic rules, with the controller making decisions based on the network's predictions, although the specific control algorithm is not explicitly defined.

Markon et al. [5] developed a system for training neural networks to perform immediate cabin allocations. Their training process involved three phases. The first phase used supervised learning with data collected from a system controlled by existing solutions from Fujitec. Selected network weights were frozen to represent the control system's principles. The remaining network was fine-tuned in the second phase to mimic the lift controller. In the third phase, the entire network was further trained in a system simulator to enhance its performance using a simplified form of reinforcement learning. The network's input

layer consisted of 25 neurons for each lift, and the output layer had one neuron per cabin. Passenger floor requests were assigned to the lift with the highest activation value of the corresponding output neuron. The system was tested in a 15-floor building simulator with six elevators, resulting in a minor improvement over the existing controller.

In later years, reinforcement learning saw its implementations in elevator control. Crites and Barto [6] applied the concept of Deep Reinforcement Learning (Deep RL) to a single elevator system in a ten-floor building during peak downward traffic. Passengers selected routes with the lowest floor as their destination, and they appeared based on a distribution tied to the simulation phase. The training process lasted four days, during which the algorithms spent over 60,000 h in a simulated environment. The trained algorithms demonstrated notably better performance compared to well-known heuristic algorithms.

One of the first reinforcement learning implementations for a lift control system was designed by Li [17]. The author utilized three RL algorithms, all from the family of Q-Value function estimation methods. The results show a decrease in the average waiting time metric for all trained strategies as compared to a reference heuristic policy.

A more recent solution was presented in 2020 by Wei et al. [7]. The researchers used a new Deep Reinforcement Learning method, called Asynchronous Advantage Actor-Critic, to train an agent capable of optimal dispatching of elevators in a group control setting. The learning process was conducted within an emulated system designed to mimic real-life building conditions. The results obtained during test simulations in various traffic patterns revealed that the trained agent successfully reduced average waiting times compared to traditional heuristic algorithms.

## 4. The Proposed Solution

In the proposed solution, the reinforcement learning approach was chosen to develop a control strategy capable of outperforming traditional heuristic approaches. To accomplish this, it was necessary to model the lift control system as a Markov Decision Process (MDP) to serve as an environment for training an agent.

### *4.1. Lift System Simulator*

For this purpose, a lift system simulator was created, representing a discrete-time system that allows the control of the lift in discrete time intervals. The simulator's parameters include the number of floors, denoted as $N_f \in \mathbb{N}_+$, and the capacity of the lift cabin, denoted as $C \in \mathbb{N}_+$. Passengers can be introduced into the system at any time at the end of a queue on any starting floor and are assigned a target floor different from their starting floor.

At any time-step the simulator accepts one of three available control actions, which are as follows:

(A) Move the lift one floor up (not possible on the highest floor).
(B) Move the lift one floor down (not possible on the lowest floor).
(C) Do not move the lift and serve the passengers in two steps:
    (i) All passengers inside the cabin whose target floor is the current floor exit the cabin.
    (ii) The first passengers from the current floor's queue enter the cabin, where the available free space in the cabin determines the number of passengers entering.

### *4.2. Passengers Distribution*

A raw simulation is not sufficient to create an environment model that represents the real lift control problem. What is also needed is the distribution of passengers' arrivals depending on their route and time. To approximate such a distribution, real-world data from the lift control system reading in the intelligent building of the ASTOR sp. z o.o. company were used. With a mature approach to data and knowledge of their increasing value over time, ASTOR began archiving sensor readings in the entire building (over

300,000 parameters) over a decade ago, including those from the lift control system using an advanced AVEVA Historian 2023 tool.

The considered building comprises seven floors (from floor −1 to 5) and one cabin with stop buttons for every floor. On each floor there are two buttons, one associated with the intention to travel to a higher floor and the other to a lower floor (except for the highest and lowest floors, which have only one button). The available parameters for further analysis included the current cabin position and the current state of all the buttons.

To count and classify the passengers, we used data ranging from August 2018 to August 2019. The assumption made for this purpose was that when a user enters a lift standing on floor $A$, he immediately selects and presses the button $Z_B$ inside the cabin corresponding to his destination floor $B$. Such an assumption allowed, by sequentially analyzing the recorded parameter changes in time, for the association of each change in the activity of the $Z_x$ stop button inside the cabin with a specific passenger travelling to floor $x$ from floor $y$, where the lift's last stop was registered.

In this way, 65,142 passengers were counted and categorized based on their travel route, considering both the starting and destination floors, the specific day of the week, and the travel time. The passenger counts were averaged over four-second intervals, aligning with the typical duration of lift travel between two floors. This process resulted in the calculation of the average number of passenger appearances for each route within specific time slots.

The derived distribution revealed that, on average, a passenger appeared once every 120 steps. To increase the dynamics of the environment and introduce a higher level of challenge for the learning agents, we reduced this time interval to 10 steps by multiplying the distribution values by 12. This adjusted distribution maintains real-world characteristics while providing a more demanding problem for the agents to learn a more interesting case for this study. The distribution, illustrated in Figure 1, indicates that almost all of the trips are the longer ones, either starting from the lower floors or concluding at the higher floors, represented by values in lower left and upper right corners. Additionally, the higher values are present in the upper right corner, meaning that a majority of trips start on either floor −1 or 0 and head up to one of floors 3, 4 or 5. This is a common pattern, as people are more likely to call and wait for the cabin when going up than when going down [18].

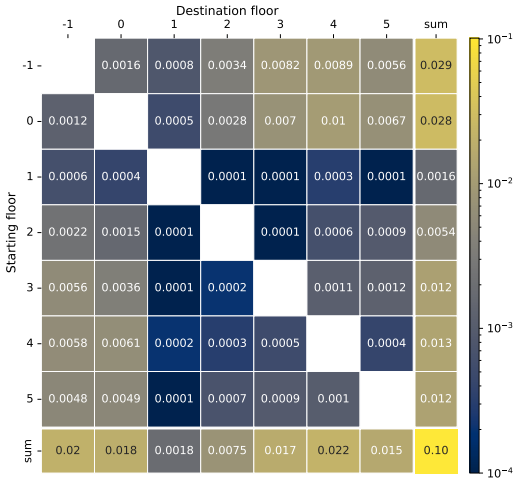


**Figure 1.** The distribution of the average number of passenger appearances for each possible route. Aggregated values for the starting and destination floors are presented in the last column and the last row, respectively. Regardless of the chosen route, the average number of passenger appearances at each step is indicated in the bottom-right corner. The values in the matrix are colour-coded according to a logarithmic colour map, with the colour bar located on the right side of the figure.

An analysis of the summed values along the destination floor axis, presented in the last 'sum' column, reveals that over half of the passengers start on either floor −1 or 0.

At the same time, the remaining part is mostly concentrated around floor 4. Regarding destination floors, a significant majority of passengers are headed to higher floors, with floor 4 being the most common choice. Interestingly, very few passengers either start or finish at floors 1 or 2 in this building.

### 4.3. State Representation

The state of the environment observed by the agent at time-step $t$ is represented by the environmental state vector $s_t$, determined based on the current state of the simulator. The specific components of this vector construction for the system adopted in this work are detailed below:

$$s_t = [p_t,\ Z_t^1, Z_t^2,\ \ldots,\ Z_t^N,\ D_t^2, D_t^3,\ \ldots,\ D_t^N,\ U_t^1, U_t^2,\ \ldots,\ U_t^{(N-1)}]^T, \tag{4}$$

where:

- $p_t \in \{1, 2,\ \ldots, N\}$ represents the current position of the lift cabin at time-step $t$.
- $Z_t^i \in \{0, 1\}, i = 1, 2,\ \ldots, N$ indicates the state of the lift cabin button for floor $i$ at time-step $t$. It is equal to 1 if there is at least one passenger inside the cabin with floor $i$ as their destination and 0 otherwise.
- $U_t^i \in \{0, 1\}, i = 1, 2,\ \ldots, (N-1)$ denotes the state of a call button on floor $i$ associated with the intention to go up. It is set to 1 if at least one passenger is on floor $i$ with a destination floor $j > i$, and 0 otherwise.
- $D_t^i \in \{0, 1\}, i = 2, 3,\ \ldots, N$ represents the state of a call button on floor $i$ associated with the intention to go down. It is valued at 1 if at least one passenger is on floor $i$ with a destination floor $j < i$, and 0 otherwise.

### 4.4. Reward Function

The construction of the reward function is a critical aspect of creating the environment. It defines the optimal strategy that maximizes the cumulative reward over time, serving as the sole quality metric for an agent in the context of an unknown and abstract goal set by the environment. Careless selection of the reward function can lead to unintended agent behaviour.

The reward is calculated following the agent's chosen action and depends on both the action and the resulting state of the simulator. Its value can also be negative and is often interpreted as a penalty. In this work, the reward was computed as a linear combination of two individual components, each aimed at capturing distinct aspects of the target strategy:

$$R(t) = -R_{buttons}(t) - 20 \times R_{illegal}(t). \tag{5}$$

The first component, denoted as $R_{buttons}(t) \in \mathbb{N}$, represents the number of active buttons in the system. Minimizing this component is associated with reducing the number of waiting passengers. The second component is termed as $R_{illegal}(t) \in \{0, 1\}$ and represents the penalty for an illegal move, such as choosing to go up on the highest floor or to go down on the lowest floor. Penalizing the agent for an illegal action is an easy solution to the issue of the dependency of available actions on the environment's state. The scaling factor of 20 was chosen apriori to ensure that selecting an illegal action is always penalized more than choosing every other legal action. In extreme scenarios where all 19 buttons are active, selecting a legal action results in a penalty of at most $-19$, while choosing an illegal action incurs a higher penalty of $-20$.

The simulator with passengers distribution, together with the state representation and reward function, formed the Markov Decision Process (MDP) model for the lift control system. It encompasses all its components, making it a suitable environment for the reinforcement learning process.

### *4.5. Learning Process*

We utilized a dual-network Deep Q-Learning algorithm with the Experience Replay method and an $\varepsilon$-greedy policy for training. The training process spanned 1000 episodes, each consisting of 20,000 time-steps. Both the online and target neural networks featured three fully connected layers, with 20 neurons (matching the state vector length), 128 neurons, and 3 neurons (representing the number of actions), respectively.

To further investigate the impact of Experience Replay techniques on learning dynamics, we conducted three experiments using all variants discussed in Section 2.3.2: basic Replay buffer, Prioritized Replay buffer, and Memory Palace. In the Memory Palace variant, the rooms corresponded to different floors where the lift cabin was currently located. This design aimed to distribute the sampled experience among specific contexts, preventing the domination of more frequent phase states and enhancing the overall learning process.

Both the basic Replay Buffer and the Prioritized Replay Buffer had a capacity of $10^6$, while the Memory Palace consisted of seven rooms, each being a Replay Buffer with a capacity of $10^5$. During training, every four time-steps, a mini-batch of 32 transitions was sampled from the experience replay for weight updates. When using the Memory Palace, the mini-batch comprised thirty-five elements, with five from each room.

To enhance learning stability, network weights were synchronized between the online and target networks every 40,000 time-steps. To enhance the extensive environment exploration, the exploration factor, $\varepsilon$, started at 1 and gradually decreased by 0.02 after each episode until reaching the final value of 0.1. The discount factor, $\gamma$, was set to 0.99 to significantly prioritize future rewards. In the learning process, we applied the Huber Loss function, a robust alternative to Mean Squared Error, particularly suitable when dealing with errors in reinforcement learning that might exhibit outlier values [19]. Additionally, we employed the Adam optimizer, a popular optimization algorithm known for its efficiency in adapting learning rates for individual parameters. All the hyperparameter's values were inspired by the work of Mnih et al. [14].

### *4.6. Learning Dynamics and Evaluation Metrics*

In supervised learning, monitoring a model's progress is easily achieved by evaluating its performance on training and validation datasets. However, evaluating the performance and progress of agents in reinforcement learning presents more significant challenges [20]. To address this, we tracked multiple metrics during the learning process. These included the average Huber Loss value over all updates in the episode, the average reward obtained by the agent, the average waiting time of served passengers and the number of illegal actions the agent took.

Although the average reward for an episode is the most obvious metric to evaluate the agents' progress, it tends to be noisy due to the changing distribution of the encountered states during $Q$ function updates. The three plots in Figure 2 illustrate the average reward (light blue line) and its smoothed trajectory (blue line) for learning variants with Replay Buffer, Prioritized Replay Buffer and Memory Palace, respectively. Despite the noise, there is a notable improvement in the initial episodes from values around $-0.6$ to values hovering around $-0.12$.

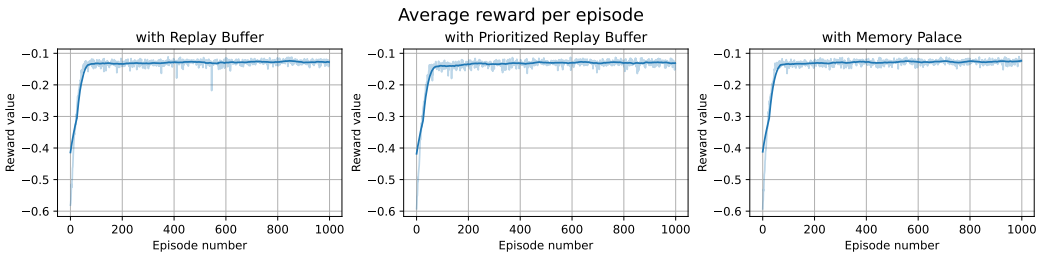


**Figure 2.** Average reward in subsequent episodes across three Experience Replay learning variants.

The number of illegal moves provides insight into the progress of $Q$ values evaluation accuracy in the subset of state space where the lift is on the lowest or highest floor. As shown in Figure 3, each learning variant's number of illegal moves sharply decreased from almost 2000 in episode 1 to below 250 from episode 50 onward. The consistent nonzero values result from the $\varepsilon$-greedy policy during learning, promoting exploration through random action choices.

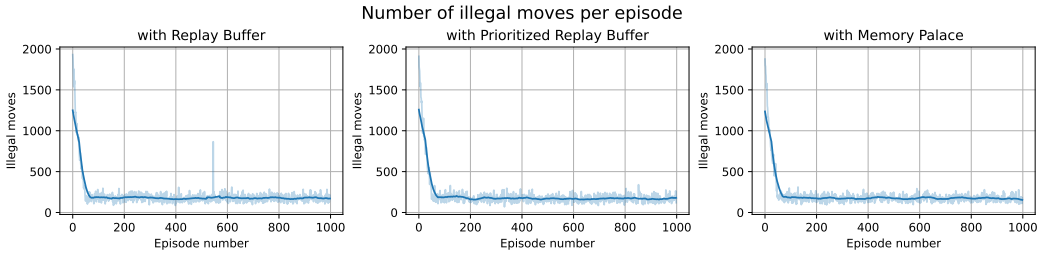


**Figure 3.** Number of illegal moves in subsequent episodes across three Experience Replay learning variants.

Although average waiting time is not explicitly stated in the agent's reward, the reward construction anticipates a decrease in average waiting time as a byproduct of increasing the reward. Figure 4 confirms this expectation, with each Experience Replay variant witnessing a visible decrease in average waiting time from over 100 steps to values oscillating around 10 steps. This alignment indicates that the increase in reward function values aligns with the general goal of reducing the average passenger's waiting time.

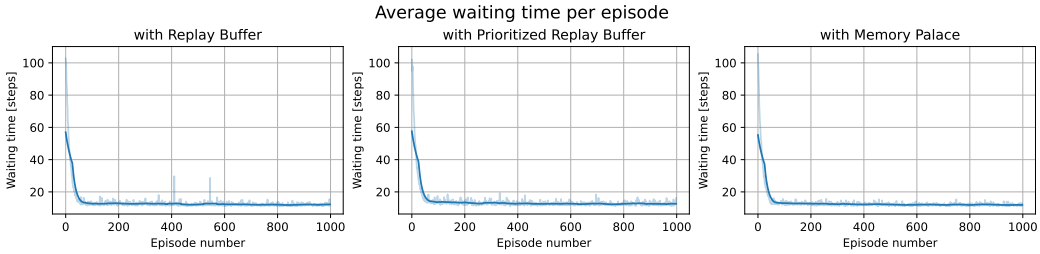


**Figure 4.** Average waiting time of all served passengers in subsequent episodes across three Experience Replay learning variants.

### *4.7. Implementation Details*

The lift simulator, the data processing pipeline and the employed learning algorithms were implemented in Python 3.10, with PyTorch serving as the primary library for the deep learning components, and libraries such as NumPy 1.26 and Pandas 2.1 delivering data processing functionalities [21–23]. The computations were performed on a machine equipped with an AMD Ryzen 5 4600H @ 3.0 GHz CPU, an NVIDIA GeForce GTX 1650 GPU, and 32 GB of RAM. The metrics were tracked using the Tensorboard utility developed by Google [24].

## 5. Tests and Their Interpretation

### *5.1. Evaluation Procedure*

To comprehensively assess and compare the performance of the trained agents and heuristic lift control strategies, we prepared 30 test simulations, each spanning 1000 time-steps. All heuristic algorithms, including Longest Queue First (LQF), Collective Algorithm (CA), and Last Mission Algorithm (LM) described in Crites and Barto [3], as well as strategies developed by the agents, were used to control the lift in each simulation. This approach allowed us to calculate mean and standard deviations for each performance metric, providing a more detailed understanding of their effectiveness.

During each test simulation, the trained agents utilized the strategy $\pi$ derived from their learned $Q$ function values:

$$\pi(s) = \arg\max_a Q(s, a). \quad (6)$$

This strategy implies that the agent always chooses action that, based on its gathered knowledge, results in the highest expected cumulative reward, prioritizing environment exploitation over exploration.

### *5.2. Results and Analysis*

During test simulations, various metrics describing the performance of each strategy were collected. The metrics included average reward per simulation step, average waiting time of served passengers, number of served passengers, and number of illegal moves. The results, presented in Table 1, include both mean and standard deviation of each of the metric values over all 30 prepared simulations.

**Table 1.** Performance metrics during 30 test simulations of 1000 time steps for various lift control strategies. For each metric, its mean ($\mu$) and standard deviation ($\sigma$) are presented. The best mean value in each metric is bolded.

| **Strategy** | **Average Reward** | | **Average Waiting Time** | | **Served Passengers** | | **Illegal Moves** | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| LQF | −0.1101 | 0.0137 | 11.8963 | 1.0981 | 84.1333 | 6.2020 | 0.0 | 0.0 |
| CA | −0.0968 | 0.0097 | 10.0738 | 0.5569 | 88.3667 | 6.7338 | 0.0 | 0.0 |
| LM | −0.0954 | 0.0094 | 9.9724 | 0.5191 | 87.7667 | 6.7705 | 0.0 | 0.0 |
| DQL_MP | −0.0914 | 0.0082 | 9.5548 | 0.5229 | 89.1000 | 6.9746 | 0.0 | 0.0 |
| DQL_RB | −0.0909 | 0.0086 | 9.5050 | 0.4727 | 89.2667 | 6.9378 | 0.0 | 0.0 |
| DQL_PRB | **−0.0906** | **0.0090** | **9.4452** | **0.5751** | **89.6333** | **6.9107** | 0.0 | 0.0 |

The initial observation indicates that the LM strategy is the best among the heuristic algorithms. However, it notably underperforms compared to each trained agent across all metrics. The trained agents not only consistently served more passengers, but also served them in a shorter time. Particularly, the agent trained with Prioritized Replay Buffer achieved the shortest average waiting time at 9.44, around 5% better than the LM strategy and over 20% better than the LQF strategy.

Additionally, even stronger relative improvement is visible in the average reward, where trained agents achieved around 5% better scoresthan the LM strategy without any overlap of the confidence intervals, indicating statistically significant differences in their performance. However, the average reward is more similar among the trained agents than the average waiting time. This occurred because the agent was trained solely to maximize reward, a method that was not constructed based on passenger waiting times but with an idea to minimize that time as a byproduct of maximizing the reward. Furthermore, the agent lacks information about how long passengers have been waiting, as the state representation only includes pressed buttons. In this setting, the environment is not overly complex yet delivers crucial information.

The absence of illegal move choices by all agents underscores the accurate $Q$ function estimations for states in which the lift was positioned on either the highest or lowest floors. This outcome demonstrates the accuracy of the $Q$ function estimations and suggests that penalizing the agent for incorrect actions is an effective approach for handling state-dependent actions. This is particularly relevant when such actions impact only a limited portion of the state space.

For a more extensive exploration and analysis of the experimental results, please refer to the Engineering Thesis authored by Wojtulewicz [25]. The thesis explores diverse environments with different complexities, providing valuable insights into the utility of Reinforcement Learning in addressing decision problems like the lift algorithm. Additionally,

it shows the crucial role of the reward function in shaping environments for Reinforcement Learning applications.

## 6. Conclusions and Future Works

In conclusion, the study successfully demonstrates a viable approach for utilizing reinforcement learning (RL) methods to address a complex real-world control problem, specifically lift control. The development of a universal lift emulator, enriched with real-world traffic distribution, modelled using data from ASTOR Company's intelligent building system, allowed for a comprehensive evaluation of RL-based strategies against conventional heuristic solutions. The research showcases that strategies devised by agents, trained using the advanced Deep Q-Learning algorithm, consistently outperform traditional heuristic solutions regarding average waiting time, served passengers, and overall reward.

Furthermore, the exploration of three Experience Replay mechanisms, namely basic and Prioritized Replay Buffers and Memory Palace, demonstrated their effectiveness in the Deep Q-Learning algorithm. The learning process for each variant of DQL was smooth and stable, displaying rapid improvements in the average episode reward metric at the initial stages of training. While all agents, incorporating distinct Experience Replay mechanisms, formulated strategies surpassing traditional solutions, the agent trained with the Prioritized Replay Buffer mechanism achieved the highest average reward and the lowest average passenger waiting time in test simulations. Nevertheless, the discrepancies in metric values among the agents are marginal, making it challenging to declare the Prioritized Replay Buffer as the superior technique definitively.

While this study provides valuable insights, there are several avenues for future research in this domain. Firstly, the exploration of different RL algorithms and architectures could yield further improvements in lift control optimization. Investigating the impact of varying state representations and reward functions on the learning process may enhance the adaptability of RL agents to diverse scenarios.

Additionally, expanding the study to consider more complex lift systems, such as those with multiple cabins or buildings with interconnected elevators, could provide a more comprehensive understanding of reinforcement learning versatility. The integration of dynamic factors, such as varying passenger arrival rates depending on the system hour, could provide a more complex environment that can assess the adaptability of reinforcement learning methods.

**Author Contributions:** Conceptualization, T.S.; Methodology, M.W.; Software, M.W.; Formal analysis, M.W.; Data curation, M.W.; Writing—original draft, M.W.; Writing—review & editing, T.S.; Supervision, T.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** The research was supported by the AGH University of Krakow, KIS Department Research Grant No 16.16.120.7998: Informatics in theory and practice.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy constraints.

**Acknowledgments:** The authors would like to thank the ASTOR sp. z o.o. company for providing access to their lift system in order to collect the data. Many thanks to the reviewers for their valuable remarks helping improve the paper.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. IBM. *The Smarter Buildings Survey*; IBM: Armonk, NY, USA, 2010.
2. Seckinger, B.; Koehler, J. Online synthesis of elevator controls as a planning problem. In Proceedings of the Thirteenth Workshop on Planning and Configuration, Department of Computer Science, University of Wuerzburg, Würzburg, Germany, 3–5 March 1999; Technical Report.

3. Crites, R.H.; Barto, A.G. Elevator Group Control Using Multiple Reinforcement Learning Agents. *Mach. Learn.* **1998**, *33*, 235–262. [CrossRef]
4. Imasaki, N.; Kubo, S.; Nakai, S.; Yoshitsugu, T.; Kiji, J.I.; Endo, T. Elevator group control system tuned by a fuzzy neural network applied method. In Proceedings of the 1995 IEEE International Conference on Fuzzy Systems, Yokohama, Japan, 20–24 March 1995; Volume 4, pp. 1735–1740. [CrossRef]
5. Markon, S.; Kita, H.; Nishikawa, Y. Adaptive Optimal Elevator Group Control by Use of Neural Networks. *Trans. Inst. Syst. Control. Inf. Eng.* **1994**, *7*, 487–497. [CrossRef]
6. Crites, R.; Barto, A. Improving Elevator Performance Using Reinforcement Learning. In *Advances in Neural Information Processing Systems*; Touretzky, D., Mozer, M.C., Hasselmo, M., Eds.; MIT Press: Cambridge, MA, USA, 1996; Volume 8.
7. Wei, Q.; Wang, L.; Liu, Y.; Polycarpou, M.M. Optimal Elevator Group Control via Deep Asynchronous Actor–Critic Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 5245–5256. [CrossRef] [PubMed]
8. Siikonen, M.L. Planning and Control Models for Elevators in High-Rise Buildings. Ph.D Thesis, Helsinki University of Technology, Espoo, Finland, 1997.
9. Lois, A.; Ziliaskopoulos, A. Online algorithm for dynamic dial a ride problem and its metrics. *Transp. Res. Procedia* **2017**, *24*, 377–384. [CrossRef]
10. Li, S.E. *Reinforcement Learning for Sequential Decision and Optimal Control*; Springer: Berlin/Heidelberg, Germany, 2023.
11. Wu, C.; Yao, W.; Luo, W.; Pan, W.; Sun, G.; Xie, H.; Wu, L. A Secure Robot Learning Framework for Cyber Attack Scheduling and Countermeasure. *IEEE Trans. Robot.* **2023**, *39*, 3722–3738. [CrossRef]
12. Singh, B.; Kumar, R.; Singh, V.P. Reinforcement learning in robotic applications: A comprehensive survey. *Artif. Intell. Rev.* **2022**, *55*, 945–990. [CrossRef]
13. Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D Thesis, King's College, Oxford, UK, 1989.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.A.; Fidjeland, A.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
15. Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; van Hasselt, H.; Silver, D. Distributed Prioritized Experience Replay. *arXiv* **2018**, arXiv:1803.00933.
16. Wei, H.; Zheng, G.; Yao, H.; Li, Z. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK, 19–23 August 2018; pp. 2496–2505. [CrossRef]
17. Li, H. The implementation of reinforcement learning algorithms on the elevator control system. In Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 8–11 September 2015; pp. 1–4. [CrossRef]
18. Liang, C.J.M.; Tang, J.; Zhang, L.; Zhao, F.; Munir, S.; Stankovic, J.A. On Human Behavioral Patterns in Elevator Usages. In Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings, Rome, Italy, 13–14 November 2013; pp. 1–2. [CrossRef]
19. Patterson, A.; Liao, V.; White, M. Robust Losses for Learning Value Functions. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *45*, 6157–6167. [CrossRef] [PubMed]
20. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602
21. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
22. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
23. The Pandas Development Team. Pandas-Dev/Pandas: Pandas. 2020. Available online: https://zenodo.org/records/10426137 (accessed on 1 November 2023) .
24. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: https://www.tensorflow.org (accessed on 1 November 2023).
25. Wojtulewicz, M. Elevator Behaviour Optimization Using Artificial Intelligence Algorithms (In Polish, Supervisor Szmuc, T.). Bachelor of Engineering Thesis, AGH University of Krakow, Kraków, Poland, 2022. Available online: https://github.com/mwojtulewicz/engineering-thesis (accessed on 1 November 2023).