Sequential Decision Making for Elevator Control

Emre Oner Tartan [1,*] and Cebrail Ciflikli [2]

[1] Vocational School of Technical Sciences, Baskent University, Ankara, Turkey
[2] Vocational School of Technical Sciences, Kayseri University, Kayseri, Turkey;
Email: cebrailciflikli@gmail.com (C.C.)
*Correspondence: onertartan@gmail.com (C.O.T.)

***Abstract*—In the last decade Reinforcement Learning (RL) has significantly changed the conventional control paradigm in many fields. RL approach is spreading with many applications such as autonomous driving and industry automation. Markov Decision Process (MDP) forms a mathematical idealized basis for RL if the explicit model is available. Dynamic programming allows to find an optimal policy for sequential decision making in a MDP. In this study we consider the elevator control as a sequential decision making problem, describe it as a MDP with finite state space and solve it using dynamic programming. At each decision making time step we aim to take the optimal action to minimize the total of hall call waiting times in the episodic task. We consider a sample 6-floor building and simulate the proposed method in comparison with the conventional Nearest Car Method (NCM).**

***Keywords*—elevator control, Markov decision process, dynamic programming, optimal policy, sequential decision making**

I. Introduction

Elevators have been considered as indispensable parts of urban life since the high-rise buildings became wide-spread. Meanwhile developing efficient control methods for elevators has been a research field. High-rise buildings require installation of multiple elevators and elevator group control systems. Today most common form of elevator control is collective control in which hall calls and car calls are registered by push-buttons and responded in floor sequence [1]. In collective control systems single buttons or directional two push-buttons are used at each floor for hall call registration. Collective control with directional two push-buttons are called full collective control so-called as directional collective control.

A major concern in elevator control is minimizing average waiting time of passengers to improve quality of service. To deal with this problem conventional rule based methods and stochastic optimization based methods were proposed. Classical control algorithms such as nearest car, fixed sectoring, bidirectional sectors, fixed sectoring, priority timed unidirectional sectors, and dynamic sectoring rely on rule-based approach [1]. Since each set of rules cannot perform generalization, in different cases, particular algorithms can be used in a hybrid way. Another approach is evolutionary optimization based elevator dispatching which aim to optimize a cost function [2–5]. In this approach metaheuristic algorithms such as Genetic Algorithm is used to minimize a metric, usually average passenger waiting time.

In the last decade a great jump has been witnessed on Machine Learning (ML). With the availability of big data, computing resources and new algorithms, training deeper neural networks became feasible and Deep Learning concept emerged. Today ML applications are widely used in daily life as well as in various engineering problems. ML approaches can be considered under three main groups as supervised, unsupervised and reinforcement learning. Supervised learning techniques can be applied in problems like classification or regression if a set of inputs and corresponding labels are available. On the other hand, if labels are not available and extracting insights on data is aimed such as finding clusters or dimensionality reduction, then unsupervised learning approach can be followed. Reinforcement learning differs from other approaches as learning is performed through interaction of an agent with environment. In Reinforcement Learning (RL) problems, the agent takes action and receives a response from the environment. The response is called as reward and the goal is to maximize the total of expected rewards. Markov Decision Process (MDP) frames this problem on a more solid, idealized basis. Particularly, MDP forms a mathematically idealized base for reinforcement learning problem.

In this study we regard the elevator control problem as a reinforcement learning problem and investigate it in a MDP framework specifically. Unlike dispatching approach, which allocates elevators to hall calls, our sequential decision making approach relies on choices of optimal actions based on the instant states. Previously, elevator system was represented as MDP model and dynamic programming was used to find optimal parking policy [6]. In another study decision making for elevator scheduling based on Markov Chains was presented [7]. However, an immediate assignment policy was assumed which prevents changing previous assignments and state space was limited to 84 states. We cover all possible actions including reversal of the car is possible if the car is empty. Consequently, size of state space reaches to 122,880 for a six-floor building with single elevator. Since the state space grow exponentially as the number of elevators increases, we considered a single elevator setting.

Manuscript received January 20, 2023; revised February 25, 2023; accepted March 31, 2023; published October 26, 2023.

Even for a single elevator, planning problem in elevator control is a NP-hard problem [7]. We believe that solving this problem by framing it as MDP and using dynamic programming, can form basis for future studies on control of complex elevator group systems by means of more advanced RL methods.

The paper is structured as follows: First, we describe Markov Decision Process, then present elements of reinforcement learning. In the next part, we explain solving Bellman Optimality Equation using dynamic programming. Then, we model elevator control problem as MDP defining action / state spaces and rewards. Finally, we discuss simulation results and give conclusion.

II. Markov Decision Process

The Markov chain, which was introduced by A. Markov in the early 20th century, is a stochastic model that describes a sequence of possible events in which the probability of each event depends only on the previous state [8]. The Markov process, a broad term often used for Markov chains, is an extension of Markov chain, which also covers continuous state space. The history of the Markov process in control dates back to the 1950s. The term "optimal control" was first considered in the 1950s as a problem of designing a controller to minimize a system criterion for a dynamic system [9]. One of the approaches to this problem was developed in the mid-1950s by Bellman based on the theory of Hamilton and Jacobi in the 19th century [9]. This approach uses the concepts of the state of a dynamic system and a value function or "optimal return function" to define an equation known as the Bellman equation [10]. Bellman presented the stochastic version of the optimal control problem using the Bellman equation as the Markov Decision Process [10]. The methods used to solve Bellman equation in a recursive manner are also introduced by Bellman and named as dynamic programming [11, 12]. Ronald Howard designed the policy iteration method for the optimal solution in MDPs [13]. The resulting theoretical foundation has formed the basis for one of the approaches in the field of artificial intelligence called Reinforcement Learning. Many problems in RL can be addressed within the MDP framework [14].

Markov processes are processes in which the probabilities of future possible states can be predicted based on the current state, rather than states in the past. The property of this lack of memory or the need for past in modeling transitions between states in the system is called the Markov property. For the state s_t at time t, Eq. (1) gives the probability of transition to the next time step:

$$P[s_{t+1} \mid s_t] = P[s_{t+1} \mid s_1, \ldots, s_t] \tag{1}$$

In a discrete-time Markov process, at each time step, there is a probability distribution for the states that the stochastic process will progress to. This distribution gives the probabilities of transitions to different states depending on the choice of an action within the current state. A Markov process in which it is assumed that a reward is gained upon transition to the state according to the chosen action and in which the goal is to achieve the highest total reward throughout the process is referred to as a Markov Decision Process. MDPs provide a general framework for sequential decision making. Classic full collective elevator control systems are dynamic systems in which information about the destination floors of passengers is not available at the controller. The control of this dynamic system can be considered as a Reinforcement Learning problem and can be solved using dynamic programming in an MDP framework.

III. Elements of Reinforcement Learning

Reinforcement learning, in general, refers to the process of an agent learning which actions to take in which situations, in order to achieve a goal while interacting with its environment [9]. The fundamental elements of reinforcement learning are the agent, environment, states actions, reward signal, value function, policy, and the model.

A. Agent

Agent is the decision maker and has a goal in the reinforcement learning problem. Agent, interacts with the environment through its actions, can transition to different states in the environment as a result of its actions, and receives feedback from the environment based on its actions. Examples of agents include a robot with the goal of collecting trash and a player in the game of Go with the goal of defeating their opponent [15]. In elevator control problem, the agent is the controller of the elevator(s).

B. Environment

The environment is the medium in which the agent interacts. The agent cannot change the environment, but it can change the state within the environment through its actions. For example, in the elevator problem, the number of elevators and the number of floors in the building are elements related to the environment and cannot be changed. However, the elevator controller (agent) can change its current state by deciding to go down or up.

C. State

The state is the representation of the agent in the environment. For example, in the example of the trash collecting robot, the state could be represented by the robot's position and battery status. Different state definitions may be given depending on the information available to the agent and the definition of the environment. A state at time t, s_t, is an element of the set of possible states St in the problem.

D. Action

An action is a choice that the agent can make from the set of possible actions it can take in its current state, causing an interaction with the environment. A_t, the action at time t is selected from $A(s_t)$, the set of possible actions at state s_t. For example, in the Go game, every move the player can make is an action. In the considered elevator control problem actions can be moving up, moving down, waiting and picking up actions for up/down calls.

E. Reward and Return

A reward signal serves to representing the goal of a reinforcement learning problem. The environment provides feedback to the agent in the form of a numerical reward signal after each of its actions. This allows the agent to measure whether an action is good, bad, or neutral. A reward signal can be positive, zero or negative. For example, in the game of Super Mario, moving to the right is a positive reward, while falling into a pit is a negative reward [16]. The agent aims to maximize the total of the reward signals, from the time current time step and onwards. Negative reward can be referred to as punishment. In a game consisting only of punishments (negative rewards), the goal is to receive the least total punishment.

In applications with an end state, the agent-environment interaction is divided into sub-sections called episodes. These applications are called episodic tasks, while applications that cannot be divided into sub-sections are called continuous tasks [9]. The total amount of reward the agent will receive in tasks is called return and is given by Eq. (2).

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^{T-1} r_T = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \quad (2)$$

In Eq. (2), γ is the discount factor $0 \leq \gamma \leq 1$. If it is considered that importance of rewards decrease over time, γ is set to less than 1. A discount factor less than 1 can also guarantee convergence of G to a finite value in continuous tasks.

F. Policy

Policy represented by π is the probability matchings between the current state and the possible actions that can be taken in that state at each time step. In other words, it is the mechanism that determines which action will be chosen with what probability in the current state.

An agent and environment interact in a series of discrete time steps $t = 0, 1, 2, 3, \ldots$ At each time step t, the agent receives a representation of the state, $s_t \in S_t$. Based on its policy π, the agent selects an action at from the set of possible actions $A(s_t)$ with probability $\pi(a_t|s_t)$. As a result of this action, the agent receives a numerical reward $r_t \in R(s_t)$ and state transition to the next state s_{t+1} occurs. This process is shown in Fig. 1.

Under a stochastic π policy, the expected reward in the state s_t can be given by Eq. (3) if the rewards for state transitions are constant.

$$\mathbb{E}_\pi\left[r_{t+1} \mid s_t = s\right] = \sum_a \pi(a \mid s_t) \sum_{s'} p(s' \mid s, a) r(s', a, s) \quad (3)$$

The probability $p(s',r|s,a)$ is the probability of receiving the reward r in the possible s' state, after the action a is selected in the state s_t. $r(s',a,s)$ is the reward for transition from the state s_t to the state s.

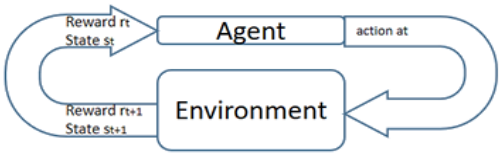

Figure 1. Agent—environment interaction in reinforcement learning.

G. Model

In reinforcement learning, the model imitates the behavior of the environment and includes probability distributions of actions, states, and rewards. A model enables making inferences about how the agent-environment interaction will go on. Using the model, given a state and action, the model can predict the next state and next reward. The model enables decision-making about actions by considering the possible future states before the actions referred to as planning are carried out [9]. Reinforcement learning methods without a model, are called model-free learning. In model free learning, exploration is needed through trial and error.

H. State and Action Value Functions

While the reward signal is an immediate feedback resulting from an action, the value function indicates the return that can be obtained from being in a particular state or from taking a particular action in a state while following a specific policy over consecutive time steps. The function that represents the value of the state is called the state value function, and the function that represents the value of the action is called the action value function. A particular action in a particular state may have a low instant reward, but it can have a high value if it leads to states with high rewards [9].

The state value function in the case of following policy π is shown as $v_\pi(s)$ and is defined by Eq. (4).

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid s_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \mid s_t = s\right] \quad (4)$$

The state value function in Eq. (4) indicates the expected return if a π policy is followed at any time t in the state s. The action value function under a π policy is shown as $q_\pi(s,a)$ and is expressed by Eq. (5).

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{T-1} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \quad (5)$$

The action value function $q_\pi(s,a)$ indicates the expected return if the action a is taken at any time t in the state s, and then policy π is followed. Using action value function, state value function $v_\pi(s)$ can be given in Eq. (6) as:

$$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s,a) \quad (6)$$

The action value function $q_\pi(s,a)$ can be represented using probability function p as follows:

$$q_\pi(s,a) = \sum_{s'} p(s' \mid s,a)[r(s',a,s) + \gamma v_\pi(s')] \quad (7)$$

IV. Optimal Policy and Bellman Optimality Equation

In a Markov Decision Process, a policy π equal to or better than policy π' in all states, indicates that policy π results in equal or more return than policy π'. There is at least one policy that is equal to or better than all other policies and this policy is called as the optimal policy. An optimal policy is represented by π* and there may be more than one optimal policy. All optimal policies have the same optimal state-value function [9] as given in Eq. (8).

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S, a \in A(s) \quad (8)$$

The optimal action value function is obtained by following the optimal policy after selecting an action *a* in a state *s* as follows:

$$q_*(s,a) = \mathbb{E}_\pi\left[r_{t+1} + \gamma v_*(s_{t+1}) \mid s_t = s, a_t = a\right] \quad (9)$$

It can be expressed in the form of Bellman's optimality equation in Eq. (10) by utilizing the recursive correlation relationship of the state value function.

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\left[r_{t+1} + \gamma v_*(s_{t+1}) \mid s_t = s, a_t = a\right] \\ &= \max_a \sum_{s'} p(s' \mid s,a)[r(s',a,s) + \gamma v_*(s')] \end{aligned} \quad (10)$$

Similarly, Bellman's optimality equation for the action value function is expressed as in Eq. (11).

$$q_*(s,a) = \sum_{s'} p(s' \mid s,a)[r(s',a,s) + \gamma v_*(s')] \quad (11)$$

Value iteration is a method in dynamic programming to evaluate optimal value functions and find optimal policy. In value iteration Bellman optimality equations are transformed to update rules as follows:

$$v_{k+1}(s) = \max_a \sum_{s'} p(s' \mid s,a)\left[r(s',a,s) + \gamma v_k(s')\right] \quad (12)$$

$$q_{k+1}(s,a) = \sum_{s'} p(s' \mid s,a)\left[r(s',a,s) + \max_{s'} \gamma v_k(s')\right] \quad (13)$$

For arbitrary v_0 and q_0, as $k \to \infty$ value functions converge to v^* and q^*. Using Eq. (13) state value function can be represented as in Eq. (14).

$$v_{k+1}(s) = \max_a q_{k+1}(s,a) \quad (14)$$

Pseudo code of value iteration is given as in Algorithm 1. As the termination condition either a threshold for maximum difference of state values between consecutive iterations or maximum number of iterations can be used. After value iteration, assuming that value function *v* converged to *v**, one more iteration over all states is needed to calculate *q** again to find and record optimal actions in π(s). In other words, one policy update is needed to extract deterministic optimal policy from the optimal value function as given in Algorithm 2. A Matlab implementation of this process for sample GridWorld example is given in [17].

Algorithm 1: Value Iteration Algorithm - Finding v ≈ v*

```
Initialize θ, V, S, A
Set Δ as θ
while Δ is greater than or equal to θ, do
    for each state s in the set S, do
        record last state value V(s) in v
        for each action a in the set A(s), do
            calculate q(s,a)
        end
        update V(s) as the maximum q value
        update Δ as the greater one of Δ and |v-V(s)|
    end
end
```

Algorithm 2: Policy Extraction Algorithm - Finding π ≈ π*

```
Initialize π
for each state s in S, do
    update v as V(s)
    for each action a in the set A(s), do
        calculate q(s,a)
    end
    update π(s) as the action a giving the maximum q value
end
```

V. State and Action Space in Elevator Control

In this study we consider single elevator control in a MDP framework. This is a finite MDP, as the sets of states, actions, and rewards all have a finite number of elements. In a simplified setting, environment includes building, elevator and arriving passengers based on the traffic. In conventional directional collective elevator control systems passengers register a hall call using direction push-buttons. After the passenger enters the car, a car call is registered by the passenger which indicates the destination floor. We define a state in MDP using car floor, car direction, up car calls, down car calls, up hall calls and down hall calls. For a 3-floor building, sample state representations are given in Table I. Car direction is represented as 1 if the direction is up, −1 if the directions is down and 0 otherwise. For a building with NF floors, a state will be represented by $4 \times (NF - 1) + 2$ vector.

TABLE I. STATE REPRESENTATION

Car floor	Car dir.	Up Car Calls		Down Car Calls		Up Hall Calls $S^{\blacktriangle}$		Down Hall Calls $S^{\blacktriangledown}$	
		F2	F3	F1	F2	F1	F2	F2	F3
1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

An up hall call cannot be registered at the highest floor and a down hall call cannot be registered at the lowest (entrance) floor. Therefore, in Table I, there is no column for F3 under up hall calls and there is no down column for F1 under down hall calls. Similarly, lowest floor (F1) cannot be an up destination floor, since there is no floor below it. Similarly, highest floor (F3) cannot be a down destination floor because there is no floor above it. Hence, there is no column for F1 under up car calls and there is no column for F3 under down car calls. Besides car direction cannot be 0 if there is at least one car call.

Other constraints taken into account in creating state and action space, are based on the following common rules in directional elevator control [18]:

- A car may not pass a floor at which a passenger wishes to exit.
- The car calls are sequentially served in accordance with the car trip direction.
- A car carrying passengers cannot change the trip direction if at least one passenger is inside.

According to the rules above, there cannot be car call below the car floor, if the car is moving up, and there cannot be car call above the car floor, if the car is moving down. Considering these constraints, we create state space using four groups as $S_{\square}$, S_{Δ}, $S_{\triangledown}$, and S_{end}. Here, car direction is represented as Δ if the direction is up, $\triangledown$ if the direction is down and $\square$ otherwise. S_{end} includes states at terminal floors (first and last floor). Hall call direction is represented as ▲ if the direction is up and ▼ if the direction is down.

A. $S_{\square}$ State Sub-space and $A_{\square}$ Action Sub-space

$S_{\square}$ state sub-space defined as in Eq. (15) includes states in which elevator does not have a direction, $s^{\text{car\_floor}} \neq 1$ and $s^{\text{car\_floor}} \neq \text{NF}$.

$$S_{\square} = \begin{Bmatrix} S_{\square}^{no\_hall\_call} = \{ s^{dir} = 0,\ s^{\blacktriangle}[s^{car\_floor}] = 0,\ s^{\blacktriangledown}[s^{car\_floor} - 1] = 0 \} \\ S_{\square}^{\blacktriangledown hall\_call} = \{ s^{dir} = 0,\ s^{\blacktriangle}[s^{car\_floor}] = 0,\ s^{\blacktriangledown}[s^{car\_floor} - 1] = 1 \} \\ S_{\square}^{\blacktriangle hall\_call} = \{ s^{dir} = 0,\ s^{\blacktriangle}[s^{car\_floor}] = 1,\ s^{\blacktriangledown}[s^{car\_floor} - 1] = 0 \} \\ S_{\square}^{\blacktriangle\blacktriangledown hall\_call} = \{ s^{dir} = 0,\ s^{\blacktriangle}[s^{car\_floor}] = 1,\ s^{\blacktriangledown}[s^{car\_floor} - 1] = 1 \} \end{Bmatrix} \quad (15)$$

Corresponding action space $A_{\square}$ is given follows:

$$A_{\square} = \begin{cases} A_{\square}^{no\_hall\_call} = \{\uparrow, \downarrow, \Uparrow, \bullet\} \\ A_{\square}^{\blacktriangledown hall\_call} = \{\uparrow, \downarrow, \Downarrow, \bullet\} \\ A_{\square}^{\blacktriangle hall\_call} = \{\uparrow, \downarrow, \Uparrow, \bullet\} \\ A_{\square}^{\blacktriangle\blacktriangledown hall\_call} = \{\uparrow, \downarrow, \Uparrow, \Downarrow, \bullet\} \end{cases} \quad (16)$$

Actions are represented by symbols in Eq. (16). Action symbols and meanings are as follows: ↑ move up, ↓ move down, ⇧ pick up up-passenger, ⇩ pick up down-passenger and • wait. State transition probabilities due to the actions depend on traffic components and destination probability distributions [19]. For instance, the action moving up results in deterministic state transition which will change the elevator floor increasing it by 1. However, state transition due to the picking up passenger is stochastic in many cases and depend on the traffic components. For example, in pure incoming traffic, passenger can intend to travel to one of above floors according to uniform probability. This is shown in Fig. 2 for the sample case. Car direction is 1(up) and the incoming passenger at F1 is picked up. However, car call in the next state can be F2 or F3 for uniform destination distribution. On the other hand, in pure outgoing traffic, picking up a passenger will result in a car call for exit floor F1, which is deterministic state transition.

Current state st

Car floor	Car dir.	Up Car Calls		Down Car Calls		Up Hall Calls $s^{\blacktriangle}$		Down Hall Calls $s^{\blacktriangledown}$	
		F2	F3	F1	F2	F1	F2	F2	F3
1	0	0	0	0	0	1	0	0	0

Action : Pick up – up passenger ⇧

Possible next state $p(s_t, s_{t+1}|⇧) = 0.5$

Car floor	Car dir.	Up Car Calls		Down Car Calls		Up Hall Calls $s^{\blacktriangle}$		Down Hall Calls $s^{\blacktriangledown}$	
		F2	F3	F1	F2	F1	F2	F2	F3
1	1	1	0	0	0	0	0	0	0

Possible next state $p(s_t, s_{t+1}|⇧) = 0.5$

Car floor	Car dir.	Up Car Calls		Down Car Calls		Up Hall Calls $s^{\blacktriangle}$		Down Hall Calls $s^{\blacktriangledown}$	
		F2	F3	F1	F2	F1	F2	F2	F3
1	1	0	1	0	0	0	0	0	0

Figure 2. A possible state transition due to the action ⇧.

B. S_{Δ} State Sub-space and A_{Δ} Action Sub-space

S_{Δ} state sub-space defined as in Eq. (17) includes states where $s^{\text{car\_floor}} \neq \text{NF}$ and elevator direction is up. Corresponding action space A_{Δ} is given in Eq. (18).

$$S_{\Delta} = \begin{Bmatrix} S_{\Delta}^{no\_\blacktriangle hall\_call} = \{ s^{dir} = 1,\ s^{\blacktriangle}[s^{car\_floor}] = 0 \} \\ S_{\Delta}^{\blacktriangle hall\_call} = \{ s^{dir} = 1,\ s^{\blacktriangle}[s^{car\_floor}] = 1 \} \end{Bmatrix} \quad (17)$$

$$A_{\Delta} = \begin{Bmatrix} A_{\Delta}^{no\_\blacktriangle hall\_call} = \{\uparrow\} \\ A_{\Delta}^{\blacktriangle hall\_call} = \{\uparrow, \Uparrow\} \end{Bmatrix} \quad (18)$$

C. $S_{\triangledown}$ State Sub-space and $A_{\triangledown}$ Action Sub-space

$S_{\triangledown}$ state sub-space defined as in Eq. (19) includes states where $s^{car\_floor} \neq 1$ and elevator direction is down. Eq. (20) gives the corresponding action space $A_{\triangledown}$.

$$S_{\triangledown} = \begin{Bmatrix} S_{\triangledown}^{no\_\blacktriangledown hall\_call} = \{s^{dir} = -1, s^{\blacktriangledown}[s^{car\_floor} - 1] = 0\} \\ S_{\triangledown}^{\blacktriangledown hall\_call} = \{s^{dir} = -1, s^{\blacktriangledown}[s^{car\_floor} - 1] = 1\} \end{Bmatrix} \quad (19)$$

$$A_{\triangledown} = \begin{Bmatrix} A_{\triangledown}^{no\_\blacktriangledown hall\_call} = \{\downarrow\} \\ A_{\triangledown}^{\blacktriangledown hall\_call} = \{\downarrow, \Downarrow\} \end{Bmatrix} \quad (20)$$

D. S_{end} State Sub-space and A_{end} Action Sub-space

S_{end} state sub-space defined as in Eq. (21) includes states where $s^{car\_floor}=1$ or $s^{car\_floor}=NF$.

$$S_{end} = \begin{Bmatrix} S_{first\_floor}^{no\_\blacktriangle hall\_call} = \{s^{\blacktriangle}[s^{car\_floor}] = 0, \quad s^{car\_floor} = 1\} \\ S_{first\_floor}^{\blacktriangle hall\_call} = \{s^{\blacktriangle}[s^{car\_floor}] = 1, \quad s^{car\_floor} = 1\} \\ S_{last\_floor}^{no\_\blacktriangledown hall\_call} = \{s^{\blacktriangledown}[s^{car\_floor} - 1] = 0, s^{car\_floor} = NF\} \\ S_{last\_floor}^{\blacktriangledown hall\_call} = \{s^{\blacktriangledown}[s^{car\_floor} - 1] = 1, s^{car\_floor} = NF\} \end{Bmatrix} \quad (21)$$

$$A_{end} = \begin{Bmatrix} A_{first\_floor}^{no\_\blacktriangle hall\_call} = \{\uparrow, \bullet\} \\ A_{first\_floor}^{\blacktriangle hall\_call} = \{\uparrow, \Uparrow\} \\ A_{last\_floor}^{\blacktriangledown no\_hall\_call} = \{\downarrow, \bullet\} \\ A_{last\_floor}^{\blacktriangledown hall\_call} = \{\downarrow, \Downarrow\} \end{Bmatrix} \quad (22)$$

VI. Reward Definition in Elevator Control

Reward signals can be positive, zero or negative based on the goal of the problem. In elevator control, a major concern is to serve whereas minimizing average passenger waiting time. In conventional directional elevator control, a hall call represents at least one passenger, however there may be a passenger queue for the same call. Therefore, number of passengers is equal to or greater than number of hall calls. Nevertheless, reducing the average waiting time of hall calls can help reducing average waiting time of passengers, especially in a balanced arrival distribution. Consequently, at first we defined reward signal as the negative of sum of waiting times of hall calls. However, in experiments this definition resulted in interesting, undesired outcomes in certain states. For instance, if the car is idle and if there is not any hall call, it is observed that optimal policy gave the moving action. This is due to the fact that moving and waiting actions yields the same reward which is zero. Similarly, another effect of solely relying on waiting times, was observed in state where there is not any hall call, but car call. In this case since the reward is zero, waiting and moving actions yield the same zero reward. As a result, if waiting action is selected travelling passengers in the car also wait. Considering these two consequences, we add two terms to the sum of waiting hall calls which are waiting penalty and moving penalty. Accordingly, reward of action a in state s is defined as follows:

$$R(s,a) = -N_{hc}.\Delta t - penalty_{waiting}.N_{cc}.\Delta t - penalty_{move} \quad (23)$$

where N_{hc} is the number of hall calls, N_{cc} is the number of car calls and Δt is the period of taking action. Waiting penalty prevents the idle car from movement and moving penalty prevents the car with passengers from waiting.

VII. Simulation Results

Elevator traffic is composed of three components which are named as incoming, inter-floor and outgoing. These components are specified as ratios with regard to overall traffic and sum up to 1. In other words, traffic components represent probability distribution of passengers according to their destination intentions. To simulate elevator traffic we used the simulator shown in Fig. 3 and described in [20]. This simulator allows initially specifying configuration ranges to try different combinations of traffic components, cars and floors.

In simulations, a sample building and elevator are set up with the parameters given in Table II. Due to the state space's memory constraints the proposed control method is applied for single elevator. Using a step size of 0.1 we tried 66 possible combinations. Initial number of waiting passengers is chosen as 2, 3, and 4. As the reference method for comparison Nearest Car Method is used [1]. In each simulation total waiting time of passengers and hall calls are recorded. Waiting time of hall call is also named as system response time. For a single waiting passenger, passenger waiting time and system response time of call is equal. For each combination, simulation is run for 1000 times and mean of total times (Mean_Total) are calculated. Results are given as differences (ΔMean_Total) between Mean_Total of the reference method's outputs and Mean_Total of the proposed method's output sums.

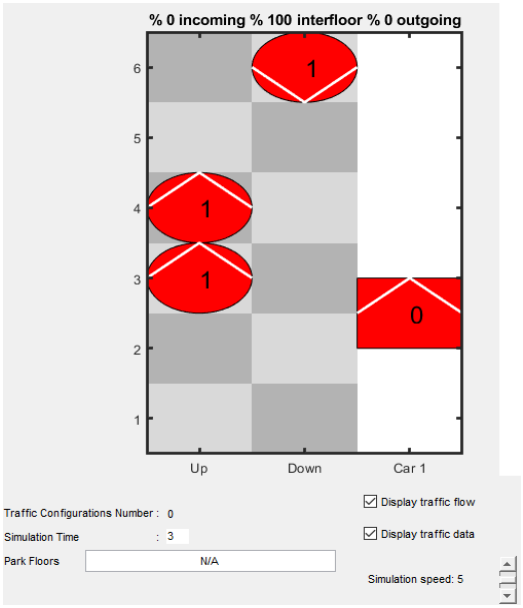

Figure 3. Elevator traffic simulator.

TABLE II. BUILDING AND ELEVATOR CONFIGURATION

Number of Floors	Elevator Speed	Floor Height	Door Opening Time	Door Closing Time	Transfer Time
6	1.5 m/s	3 m	2 s	2 s	3 s

TABLE III. ΔMEAN_TOTAL OF PASSENGER WAITING TIME AND HALL CALL WAITING TIME FOR NP = 2

Inc	Inter-floor										
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
0	5.8	4.8	4.4	3.7	3.3	2.9	2.6	2.2	1.9	1.4	1.3
0.1	4.9	3.8	3.9	3.1	2.8	2.6	1.9	1.5	1.1	0.9	-
0.2	4.1	3.3	2.9	2.7	2.0	1.9	1.7	1.1	0.6	0.0	-
0.3	3.6	3.0	2.3	2.1	1.7	1.3	0.7	0.6	-	-	-
0.4	3.1	2.2	1.7	1.6	1.1	0.7	0.5	-	-	-	-
0.5	2.3	1.7	1.8	1.1	0.7	0.4	-	-	-	-	-
0.6	1.9	1.5	1.0	0.5	0.1	-	-	-	-	-	-
0.7	1.4	1.0	0.6	0.1	-	-	-	-	-	-	-
0.8	0.9	0.5	0.1	-	-	-	-	-	-	-	-
0.9	0.6	0.0	-	-	-	-	-	-	-	-	-
1	0.0	-	-	-	-	-	-	-	-	-	-

Since ΔMean_Total values of waiting times and hall call times are same for 2 passengers, results for 2 passengers are given as a single table, in Table III. For 3 and 4 initial passengers ΔMean_Total results of passenger and hall waiting times are given through Tables IV–VII. In each condition MDP based control resulted in waiting times less than or equal to the reference method's. Especially in heavy outgoing traffic reference method operates poorly due to the undesired bunching elevators. To handle this, a solution is to collect passengers starting from the highest floor. This explicit rule based method is used only under heavy outgoing traffic in directional collective control or used in single push button systems where inter-floor traffic is not expected [1]. This type of control is named as down collective control (so-called up-distributive, down collective) where all hall calls are regarded as down calls. The proposed method finds out this strategy by framing the problem as MDP and using dynamic programming. In general terms, rule based approaches try to find different set of rules for different conditions which is not always feasible. On the other hand, in the proposed approach, optimal policy includes optimal actions for different states.

TABLE IV. ΔMEAN_TOTAL OF HALL CALL WAITING TIME FOR NP = 3

Inc	Inter-floor										
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
0	14.9	13.4	12.4	10.6	9.5	7.9	5.8	4.7	4.2	2.9	2.2
0.1	13.1	11.0	9.9	8.7	6.8	6.2	4.6	3.7	1.9	1.6	-
0.2	10.6	9.2	7.4	6.6	5.2	4.4	2.7	2.0	1.5	-	-
0.3	8.1	6.7	5.6	4.6	4.1	2.9	1.8	0.9	-	-	-
0.4	6.7	5.5	4.4	3.3	2.6	1.5	0.7	-	-	-	-
0.5	5.6	4.5	3.7	2.5	1.3	0.4	-	-	-	-	-
0.6	4.2	2.9	2.4	1.3	0.3	-	-	-	-	-	-
0.7	3.2	2.1	1.2	0.2	-	-	-	-	-	-	-
0.8	2.3	0.9	0.1	-	-	-	-	-	-	-	-
0.9	1.2	0.0	-	-	-	-	-	-	-	-	-
1	0.0	-	-	-	-	-	-	-	-	-	-

TABLE V. ΔMEAN_TOTAL OF PASSENGER WAITING TIME FOR NP = 3

Inc	Inter-floor										
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
0	16.1	15.0	13.7	11.8	10.4	9.3	6.7	5.2	4.4	2.8	2.4
0.1	14.6	12.4	10.8	9.5	7.7	6.5	4.9	4.1	2.2	1.8	-
0.2	11.4	9.7	8.3	7.2	5.6	4.8	2.9	2.1	1.5	-	-
0.3	8.6	7.0	5.6	5.0	4.4	3.1	1.8	1.0	-	-	-
0.4	6.8	5.4	4.4	3.3	2.5	1.6	0.8	-	-	-	-
0.5	5.7	4.3	3.2	2.3	1.2	0.4	-	-	-	-	-
0.6	3.6	2.4	1.9	1.0	0.3	-	-	-	-	-	-
0.7	2.3	1.5	1.0	0.2	-	-	-	-	-	-	-
0.8	1.4	0.5	0.1	-	-	-	-	-	-	-	-
0.9	0.7	0.0	-	-	-	-	-	-	-	-	-
1	0.0	-	-	-	-	-	-	-	-	-	-

TABLE VI. ΔMEAN_TOTAL OF HALL CALL WAITING TIME FOR NP = 4

Inc	Inter-floor										
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
0	26.9	23.1	20.2	17.4	13.8	11.1	8.9	6.8	5.9	3.6	2.8
0.1	20.3	17.1	13.7	12.3	10.3	8.2	6.6	4.8	2.9	2.3	-
0.2	15.0	12.5	11.5	9.9	8.1	5.8	4.5	2.4	1.6	-	-
0.3	12.7	10.1	8.4	6.7	5.2	3.6	2.4	0.8	-	-	-
0.4	10.1	7.9	6.0	5.2	3.5	2.0	0.7	-	-	-	-
0.5	8.0	5.9	4.9	3.3	2.1	0.5	-	-	-	-	-
0.6	6.3	4.4	3.4	1.7	0.2	-	-	-	-	-	-
0.7	4.7	3.8	2.2	0.1	-	-	-	-	-	-	-
0.8	4.1	1.9	0.1	-	-	-	-	-	-	-	-
0.9	2.1	0.0	-	-	-	-	-	-	-	-	-
1	0.0	-	-	-	-	-	-	-	-	-	-

TABLE VII. ΔMEAN_TOTAL OF PASSENGER WAITING TIME FOR NP= 4

Inc	Inter-floor										
	0	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
0	32.7	28.0	25.3	21.8	17.9	14.4	11.5	8.8	7.7	5.0	3.4
0.1	24.9	20.9	17.5	15.1	12.7	9.7	8.0	5.6	3.7	2.9	-
0.2	17.8	15.0	15.0	12.6	9.0	6.9	5.8	3.1	1.9	-	-
0.3	14.3	11.5	8.5	6.9	5.4	3.9	2.6	0.8	-	-	-
0.4	10.3	8.2	6.0	4.9	3.0	1.8	0.8	-	-	-	-
0.5	6.6	5.0	3.7	2.7	1.6	0.6	-	-	-	-	-
0.6	4.7	2.8	2.1	1.3	0.3	-	-	-	-	-	-
0.7	2.7	2.2	1.0	0.2	-	-	-	-	-	-	-
0.8	1.5	0.7	0.1	-	-	-	-	-	-	-	-
0.9	0.5	0.0	-	-	-	-	-	-	-	-	-
1	0.0	-	-	-	-	-	-	-	-	-	-

VIII. CONCLUSION

In this study we considered sequential decision making for elevator control as a reinforcement learning problem and modeled it as a finite MDP. We described state space, action space and dynamics of the model based on the elevator traffic components and solved the MDP using dynamic programming. We used elevator traffic simulator to simulate and verify the theoretically proposed model. For comparison we involved a conventional elevator control method which is known as Nearest Car Method. The proposed method outperformed in most of conditions and in some conditions gave the same result. Since NCM

is not suitable for outgoing traffic, we also tried down collective control and obtained the same results by the proposed method in pure outgoing traffic. As the optimal policy includes optimal actions for each state, it presents generalization capability. However, due to the curse of dimensionality, as the number of floors and elevators increase, the state space will grow exponentially. Therefore, regarding memory and computation concerns, more advanced RL methods which approximates MDP can be tried for more complex cases. In conclusion, in future studies we regard this study as a basis and aim to extend the framework to other model-free RL methods such as Q-learning which do not rely on explicit state-transition probabilities and rewards. A such approach could deal with the complex control problem of elevator group systems and make autonomous driving elevators possible.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

E. O. Tartan conducted software development, experimental design and contributed in writing the paper. C. Ciflikli analyzed the results, supervised paper writing and prepared the final draft. Both authors had approved the final version.

REFERENCES

[1] G. C. Barney and L. Al-Sharif, *Elevator Traffic Handbook*, New York: Taylor and Francis, 2016.

[2] E. O. Tartan and C. Ciflikli, "A genetic algorithm based elevator dispatching method for waiting time," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 424–429, 2016.

[3] P. Cortés, J. Muñuzuri, A. Vázquez-Ledesma, and L. Onieva, "Double deck elevator group control systems using evolutionary algorithms: Interfloor and lunchpeak traffic analysis," *Computers & Industrial Engineering*, vol. 155, 2021.

[4] M. Beamurgia, R. Basagoiti, I. Rodríguez and V. Rodríguez, "Improving waiting time and energy consumption performance of a bi-objective genetic algorithm embedded in an elevator group control," *Soft Computing*, vol. 26, p. 13673–13692, 2022.

[5] A. Vodopija, J. Stork, T. Bartz-Beielstein, and B. Filipič, "Elevator group control as a constrained multiobjective optimization problem," *Applied Soft Computing*, vol. 10, pp. 1568–4946, 2022.

[6] D. Nikovski and M. Brand, "Optimal parking in group elevator control," in *Proc. IEEE International Conference on Robotics and Automation*, 2004, vol. 1, pp. 1002–1008.

[7] D. Nikovski and M. Brand, "Decision-theoretic group elevator scheduling," in *Proc. the Thirteenth International Conference on International Conference on Automated Planning and Scheduling*, 2003, pp. 133–142.

[8] P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*, USA, NJ: John Wiley & Sons, 2017.

[9] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, Cambridge, MA.: MIT Press/Bradford Books, 2018, p. 14.

[10] R. E. Bellman, "A Markov decision process," *Journal of Mathematical Mechanics*, no. 6, pp. 679–688, 1957.

[11] R. E. Bellman, *Dynamic Programming*, Princeton: Princeton University Press, 1957.

[12] R. E. Bellman and S. E. Dreyfus, "Functional approximations and dynamic programming," *Mathematical Tables and Other Aids to Computation*, vol. 13, pp. 247–251, 1959.

[13] R. Howard, *Dynamic Programming and Markov Processes*, John Wiley, 1960.

[14] L. N. Steimle, D. L. Kaufman and B. T. Denton, "Multi-model Markov decision processes," *IISE Transactions*, vol. 53, no. 10, pp. 1124–1139, 2021.

[15] D. Silver, A. Huang, and C. Maddison, "Mastering the game of Go with deep neural networks and tree search," *Nature*, no. 529, pp. 484–489, 2016.

[16] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, 2012.

[17] O. Tartan. (2023). MDP_GridWorld. [Online]. Available: https://github.com/onertartan/RL-GridWorld/blob/main/GridWorld5x5.m

[18] G. D. Closs, "The computer control of passenger traffic in large lift systems," Ph.D. Thesis, University of Manchester, Institute of Science and Technology, Manchester, 1970.

[19] E. O. Tartan and C. Ciflikli, "A model for the visualization and analysis of elevator traffic," *Transportation Planning and Technology*, vol. 42, no. 8, pp. 868–880, 2019.

[20] E. O. Tartan and C. Ciflikli. (2023). ESRA (elevator simulation, research & analysis): An open source software tool for elevator traffic simulation, research and analysis. [Online]. Available: https://github.com/onertartan/elevator-simulator.git