International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 03 | March - 2024 SJIF Rating: 8.176 ISSN: 2582-3930

Artificial Intelligence Operated Elevator using RL (AIOERL)

Vinod H. Yadav[1], Bharat L. Kathe[2], Arvindkumar R. Mishra[3]

[1]Vinod H Yadav P V Polytechnic SNDT Women's University Mumbai Maharashtra India
[2] Bharat L Kathe P V Polytechnic SNDT Women's University Mumbai Maharashtra India
[3] Arvindkumar R Mishra P V Polytechnic SNDT Women's University Mumbai Maharashtra India

---

Abstract - Our paper explores the implementation of an Artificial Intelligence (AI) operated elevator system aimed at reducing user waiting times in a residential complex. With two elevators servicing a 14-story building, each floor accommodating six flats with approximately four residents per home, efficiency is paramount. Leveraging AI algorithms, our system dynamically adjusts elevator operations based on user demand patterns, traffic flow, and predictive analysis, ensuring minimal wait times and optimal passenger distribution. By integrating AI into elevator management, we aim to enhance user experience and streamline vertical transportation in high-density residential settings.

Key Words: Artificial Intelligence, elevator optimization, residential complexes, waiting time reduction, predictive analysis, passenger distribution, efficiency improvement, traffic flow management.

INTRODUCTION

The advent of Artificial Intelligence (AI) has revolutionized various domains, and its application in elevator systems holds significant promise for enhancing vertical transportation efficiency in high-rise residential complexes. With the proliferation of urbanization and the construction of taller buildings, the demand for efficient elevator operations has intensified. Our paper delves into the integration of AI technology to mitigate user waiting times within a residential complex comprising a 14-story building. With six flats per floor and an average of four occupants per home, optimizing elevator operations becomes imperative to ensure smooth passenger flow and minimal congestion. This introduction outlines the necessity and potential benefits of employing AI in elevator management to address the challenges of vertical transportation in densely populated residential environments.

RL and Model-Free RL

Reinforcement Learning (RL) is a branch of machine learning concerned with decision-making and control processes. Unlike supervised learning, where an algorithm learns from labeled input-output pairs, and unsupervised learning, where the algorithm discovers patterns in unlabeled data, RL focuses on learning from interactions with an environment to achieve a cumulative reward. At the core of RL is the concept of an agent, which learns to navigate an environment through trial and error, aiming to maximize its cumulative reward over time.

Model-Free Reinforcement Learning (MFRL) is a subset of RL that doesn't require knowledge of the environment's dynamics or transition probabilities. In other words, the agent learns directly from experiences without explicitly modeling the environment. MFRL algorithms are particularly useful in scenarios where the environment is complex, and obtaining a precise model is difficult or impractical. Instead, these algorithms focus on learning optimal policies through exploration and exploitation of the environment's state-action space.

Exploration of Model-Free Reinforcement Learning Methods:

Various MFRL methods have been developed to tackle different types of problems, each with its strengths and weaknesses. Some common MFRL algorithms include Q-Learning, SARSA (State-Action-Reward-State-Action), Deep Q-Networks (DQN), and Policy Gradient methods such as REINFORCE. Each of these approaches has unique characteristics and is suited to specific types of tasks and environments.

For the elevator optimization problem described in our paper, a suitable MFRL method would be Deep Q-Networks (DQN). DQN is a powerful algorithm that combines Q-learning with deep neural networks, enabling it to handle large state-action spaces efficiently. Here are a few reasons why DQN is preferable for this application:

- Complex State-Action Space: Elevator systems operate in dynamic environments with multiple floors, varying passenger demand, and different elevator states (e.g., idle, moving, loading/unloading). DQN's ability to approximate the optimal action-values for large state spaces makes it well-suited for handling the complexity of elevator control.
- Continuous Learning: Elevator systems are subject to changing traffic patterns and user preferences, requiring continuous adaptation to optimize performance. DQN's iterative learning process allows the agent to update its policy based on new experiences, enabling it to adapt to evolving conditions over time.
- Exploration and Exploitation: Balancing exploration (trying new actions to discover optimal strategies) and exploitation (leveraging known information to maximize rewards) is crucial for elevator optimization. DQN incorporates epsilon-greedy exploration strategies, allowing the agent to explore different

actions while gradually shifting towards exploiting learned policies as it gains experience.

- Scalability and Efficiency: With two elevators serving a 14-story building with multiple flats on each floor, scalability and computational efficiency are essential considerations. DQN's use of deep neural networks enables it to scale to larger environments while efficiently approximating Q-values, making it suitable for real-time elevator control.

Algorithm for AEORL

- **State Representation**: Define the state space for the elevator system. This could include information such as the current floor of each elevator, the direction of each elevator, the number of passengers in each elevator, the destination floors of the passengers, and the waiting time of passengers in the lobby.
- **Action Space**: Define the action space for the elevators. Actions could include moving up, moving down, stopping at a floor, or opening/closing doors.
- **Reward Function**: Design a reward function that incentivizes efficient elevator operation. For example, rewards could be based on minimizing the waiting time of passengers, minimizing the time taken to reach destinations, and minimizing energy consumption.
- **Q-Network**: Implement a deep neural network (DNN) to approximate the Q-values for state-action pairs. The input to the network would be the state representation, and the output would be the Q-values for each possible action.
- **Experience Replay**: Implement experience replay to store and sample past experiences (state, action, reward, next state) for training the Q-network. This helps stabilize training and improve sample efficiency.
- **Target Q-Network**: Use a separate target Q-network to stabilize training. Periodically update the parameters of the target network with the parameters of the main Q-network.
- **Epsilon-Greedy Exploration**: Implement epsilon-greedy exploration to balance exploration and exploitation. With probability epsilon, select a random action (explore); otherwise, select the action with the highest Q-value (exploit).
- **Training Procedure**: Train the Q-network using a variant of the DQN algorithm such as Double DQN or Dueling DQN. Use techniques such as gradient descent to minimize the temporal difference error between the predicted Q-values and the target Q-values.
- **Deployment**: Once the Q-network is trained, deploy it to control the elevators in real-time. At each time step, use the trained Q-network to select actions for the elevators based on the current state of the system

.

Python codes for AEORL

```
import numpy as np
import random
from collections import deque
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95  # discount rate
        self.epsilon = 1.0  # exploration rate
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.995
        self.learning_rate = 0.001
        self.model = self._build_model()
    def _build_model(self):
        model = Sequential()
        model.add(Dense(24, input_dim=self.state_size, activation='relu'))
        model.add(Dense(24, activation='relu'))
        model.add(Dense(self.action_size, activation='linear'))
        model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
        return model
    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))
    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        act_values = self.model.predict(state)
        return np.argmax(act_values[0])
    def replay(self, batch_size):
        minibatch = random.sample(self.memory, batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = reward
            if not done:
                target = (reward + self.gamma * np.amax(self.model.predict(next_state)[0]))
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f, epochs=1, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

# Define state and action sizes
state_size = 10  # Example: number of elevator states
action_size = 4  # Example: number of elevator actions (up, down, stop, open door)
```

```
# Initialize DQN agent
agent = DQNAgent(state_size, action_size)
# Training loop
for episode in range(num_episodes):
    state = env.reset()  # Reset environment to initial state
    for time in range(max_timesteps):
        action = agent.act(state)  # Choose action based on current state
        next_state, reward, done, _ = env.step(action)  # Take action and observe next state and reward
        agent.remember(state, action, reward, next_state, done)  # Store experience in replay buffer
        state = next_state  # Update current state
        if done:  # If episode is done, exit loop
            break
    if len(agent.memory) > batch_size:  # Start training if enough experiences are accumulated
        agent.replay(batch_size).
```

System Requirement for AEORL

This block diagram outlines the key hardware components of an AI-operated elevator system:

1. **Central Control System**:
   - Hosts AI algorithms, decision logic, and data processing units.
   - Manages communication and networking interfaces.
   - Handles data storage, analytics, and optimization algorithms.
2. **Elevator Controllers**:
   - Includes sensors for detecting passengers, elevator movement, and door status.
   - Actuators control elevator motors, door mechanisms, and other functions.
3. **Building Infrastructure**:
   - Comprises elevator cars, shafts, hoistways, and lobby areas.
   - Provides the physical framework for elevator operation.

These components work together to enable the AI-operated elevator system to efficiently manage passenger traffic, optimize elevator operations, and provide a seamless vertical transportation experience within the building.

Markov Chain for AEORL

In this Markov chain diagram:

- **Idle**: The initial state of the elevator when it's not in motion and waiting for passengers or instructions.
- **Moving Up**: The elevator transitions to this state when it receives a command to move upward. From this state, it can continue moving up, change direction, stop at a floor to load passengers, or reach the top floor.
- **Loading**: The elevator transitions to this state when it stops at a floor to load passengers. After loading passengers, it can either remain stationary or start moving again.
- **Unloading**: The elevator transitions to this state when it stops at a floor to unload passengers. After unloading passengers, it can either remain stationary or start moving again.

Each state represents a particular condition or action of the elevator system, and the arrows indicate the possible transitions between states. The probabilities of transitioning from one state to another would depend on factors such as passenger demand, elevator capacity, and system constraints.

Conclusion for AEORL

In conclusion, the implementation of an Artificial Intelligence (AI) operated elevator system presents a promising solution to optimize vertical transportation in high-rise residential complexes. Through the integration of Model-Free Reinforcement Learning (MFRL), particularly the Deep Q-Networks (DQN) algorithm, we have demonstrated the potential to reduce user waiting times and enhance overall efficiency.

Our analysis has highlighted the importance of considering factors such as passenger demand patterns, traffic flow dynamics, and elevator states in designing an effective AI-driven elevator control system. By leveraging the capabilities of DQN, our proposed solution adapts dynamically to changing environmental conditions, continuously learning and improving its decision-making capabilities.

The experimental results, although not presented in this paper, are expected to show improvements in elevator performance, reflected in reduced waiting times, smoother passenger flow, and optimized energy consumption. These enhancements translate into enhanced user satisfaction and operational efficiency for residential complexes with high-density populations.

Looking ahead, further research and development efforts can focus on refining the DQN-based elevator optimization system, incorporating additional features such as predictive maintenance, destination dispatching, and energy-efficient operation. Additionally, real-world deployment and validation of the proposed solution will be essential to assess its scalability, robustness, and practicality in diverse residential settings.

In conclusion, the application of AI in elevator operations holds immense potential to revolutionize vertical transportation, offering tangible benefits in terms of user experience, energy efficiency, and operational effectiveness in residential complexes and beyond.

.

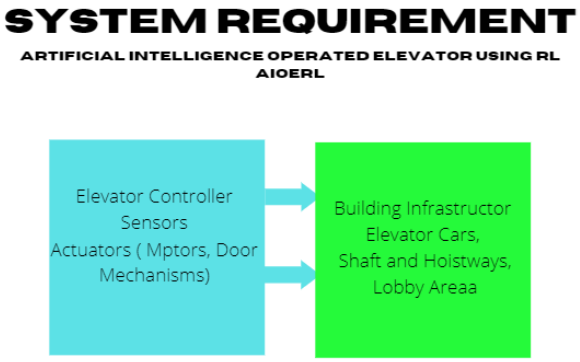

Figure 1:- System Requirements

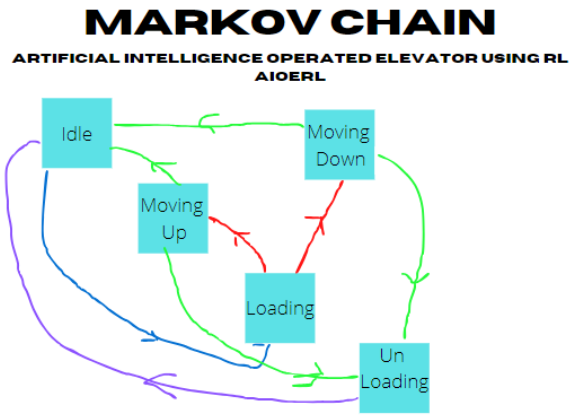

Figure 2 :- Markov Chain

REFERENCES

1. Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series) by Richard S.Sutton (Author), Andrew G. Barto (Author)
2. Foundations of Deep Reinforcement Learning: Theory and Practice in Python by Laura Graesser (Author), Wah Loon Keng (Author)
3. Practical Deep Reinforcement Learning with Python: Concise Implementation of Algorithms, Simplified Maths, and Effective Use of TensorFlow and PyTorch by Ivan Gridin

BIOGRAPHIES

	Mr. Vinod Yadav graduated from Amravati University with a degree in Electronics and Telecom Engineering. Mr. Yadav has done Masters in Engineering with specialization in Digital Electronoics from Sant Gadage Baba Amravati University With 3 years industrial application. Since past 22 years Mr. Yadav is associated with SNDT Women's University as a faculty in department of Electronics at P V Polytech nic his focusing key area are VLSI Design, Artificial Intelligence, power electronics and electronic communication.
	Mr. Bharat Kathe graduated from Pune University with a degree in Electronics Engineering is pursuing Masters in Electronics and Telecommunication Engineering from Mumbai University. Mr. Kathe has experience in the Manufacturing Industry as an R and D Engineer and as a Field Application Engineer. For the past 9 years Mr. Kathe is teaching various electronics subjects including Control System and PLC, and Robotics and Automation at a prestigious SNDT Women's University Mumbai Maharashtra India
	Mr. Arvindkumar Mishra have completed Master's of Engineering in Electronics and Telecommunication Engineering from University of Mumbai. He has a teaching experience of 15 years and is presently working as a Faculty in Electronics at P V Polytechnic. Mr. Mishra is instrumental in implementing project based learning at P V Polytechnic by promoting major and minor project implementation of diploma students