Optimizing Vertical Transportation: A Comparative Study of Elevator Dispatching Algorithms in Dynamic Environments

EMRE CECANPUNAR, Akdeniz University, Türkiye
MELIK SAVAŞ, Akdeniz University, Türkiye
DOĞAN ENSAR PAPUÇCUOĞLU, Akdeniz University, Türkiye
DOĞUKAN ÇELİK, Akdeniz University, Türkiye
EFKAN ERTAŞ, Akdeniz University, Türkiye
FURKAN ŞENOL, Akdeniz University, Türkiye
MURAT AK, Akdeniz University, Türkiye

Efficient elevator systems are essential for high-rise buildings, especially as cities grow taller and more crowded. The Elevator Dispatching Problem (EDP) focuses on assigning elevators to passenger requests in a way that improves performance metrics like wait times and energy consumption. This paper examines traditional algorithms such as First-Come-First-Serve (FCFS) and SCAN, as well as more advanced methods that use machine learning and dynamic optimization. We analyze how these approaches handle challenges like unpredictable traffic and various user needs. Finally, we discuss the potential for AI-based adaptive systems to make elevators smarter, more scalable, and better suited for future intelligent and larger buildings.

CCS Concepts: • **Theory of computation** → **Algorithm design techniques**; • **Mathematics of computing** → *Queueing theory*; • **Hardware** → Building automation.

Additional Key Words and Phrases: Elevator Dispatching, Optimization, Dynamic Traffic, Queueing Theory, SCAN Algorithm, AI in Dispatching, Reinforcement Learning, Vertical Transportation, High-Rise Building Systems, Energy Efficiency, Real-Time Systems, Machine Learning Applications

ACM Reference Format:
Emre CECANPUNAR, Melik SAVAŞ, Doğan Ensar PAPUÇCUOĞLU, Doğukan ÇELİK, Efkan ERTAŞ, Furkan ŞENOL, and Murat AK. 2024. Optimizing Vertical Transportation: A Comparative Study of Elevator Dispatching Algorithms in Dynamic Environments. *J. ACM* 37, 4, Article 111 (August 2024), 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Over the years, many methods have been developed to solve the Elevator Dispatching Problem (EDP). Early algorithms like First-Come-First-Serve (FCFS) and SCAN provided simple solutions, focusing on fairness and systematic traversal of floors. However, these approaches often struggled with inefficiency during high-request scenarios or when requests were unevenly distributed.

Authors' Contact Information: Emre CECANPUNAR, Akdeniz University, Antalya, Türkiye, emreleno@gmail.com; Melik SAVAŞ, Akdeniz University, Antalya, Türkiye, meliksavas583@gmail.com; Doğan Ensar PAPUÇCUOĞLU, Akdeniz University, Antalya, Türkiye, ; Doğukan ÇELİK, Akdeniz University, Antalya, Türkiye, ; Efkan ERTAŞ, Akdeniz University, Antalya, Türkiye, ; Furkan ŞENOL, Akdeniz University, Antalya, Türkiye, ; Murat AK, Akdeniz University, Antalya, Türkiye, .

ACM 1557-735X/2024/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

To improve performance, advanced methods like Shortest Seek Time First (SSTF) and LOOK were introduced. These algorithms optimize travel distance by prioritizing nearby requests, but they can lead to fairness issues, such as high waiting times for far-off floors.

More recently, artificial intelligence (AI) has become a promising tool in elevator scheduling. Techniques like reinforcement learning and predictive analytics allow systems to adapt dynamically to changing traffic patterns, providing better scalability and responsiveness

Despite these advancements, challenges remain in achieving real-time efficiency, balancing fairness with energy savings, and integrating hybrid solutions effectively.

This paper aims to fill these gaps by evaluating traditional and modern methods, comparing their performance, and exploring future directions in AI-powered elevator systems.

2 Classical Elevator Dispatching Algorithms

Classical algorithms form the foundation of elevator dispatching and provide simple yet effective methods for handling passenger requests. In this section, we discuss four popular algorithms: **First-Come-First-Serve (FCFS)**, **Shortest Seek Time First (SSTF)**, **SCAN**, and **LOOK**. Each algorithm has its unique strengths and weaknesses, making them suitable for different scenarios. Their implementations, performance, and limitations are explained below. Full code for some of these algorithms can be accessed at [3]

2.1 First-Come-First-Serve (FCFS)

The FCFS algorithm processes requests in the order they are received. Its simplicity ensures fairness since every request is treated equally. However, this often leads to inefficiencies, especially in high-traffic scenarios where requests are spread across multiple floors.

Algorithm Overview:

- The elevator starts from a given floor and services requests sequentially.
- No prioritization is given to proximity or travel distance, which can result in unnecessary movements.

Performance Scenarios:

- **Best Case:** Requests are sequential, minimizing travel distance.
  - *Example:* Starting floor 0, requests [1, 2, 3, 4].
  - *Output:* Path: [0, 1, 2, 3, 4], Total Distance: 4.
- **Worst Case:** Requests are scattered, maximizing travel distance.
  - *Example:* Starting floor 0, requests [10, 1, 15, 2].
  - *Output:* Path: [0, 10, 1, 15, 2], Total Distance: 34.

FCFS is easy to implement and useful for systems with low traffic but becomes inefficient when demand increases or requests are diverse.

2.2 Shortest Seek Time First (SSTF)

SSTF selects the closest request to the elevator's current position, reducing travel distance. While this method is efficient, it can lead to increased waiting time where far-off requests remain unsatisfied if new closer requests keep arriving.

Algorithm Overview:

- At each step, the elevator identifies and services the nearest request.
- This requires recalculating distances dynamically as requests are completed.

Performance Scenarios:

- **Best Case:** Requests are clustered around the elevator.

  - *Example:* Starting floor 5, requests [4, 6, 5, 7].
  - *Output:* Path: [5, 5, 4, 6, 7], Total Distance: 3.
- **Worst Case:** Continuous arrival of new, closer requests.
  - *Example:* Starting floor 0, initial requests [10, 20, 30], but new requests [1, 2, 3] keep appearing.

SSTF works well in static systems with fixed requests but struggles in dynamic environments where fairness is critical.

2.3 SCAN (Elevator Algorithm)

The SCAN algorithm, also called the "Elevator Algorithm," services requests by moving in one direction (up or down) until the furthest request is reached, then reverses direction. This approach balances fairness and efficiency by systematically addressing all requests.

Algorithm Overview:

- The elevator moves in a fixed direction, fulfilling requests as it travels.
- Once it reaches the highest or lowest requested floor, it reverses direction to handle remaining requests.

Performance Scenarios:

- **Best Case:** Requests are in a single direction.
  - *Example:* Starting floor 0, requests [1, 2, 3, 4].
  - *Output:* Path: [0, 1, 2, 3, 4], Total Distance: 4.
- **Worst Case:** Requests are scattered across multiple directions.
  - *Example:* Starting floor 0, requests [10, 1, 15, 2].

SCAN is effective in moderately busy systems with a mix of scattered and clustered requests, though unnecessary traversal to highest or lowest floor can increase travel time in some cases.

2.4 LOOK (Optimized SCAN)

LOOK is an optimized version of SCAN that avoids unnecessary travel to unrequested floors. Instead of moving to the building's borders, it "looks ahead" to determine the furthest active request in the current direction and reverses direction accordingly.

Algorithm Overview:

- The elevator dynamically adjusts its direction based on pending requests.
- This minimizes unnecessary traversal and improves efficiency.

Performance Scenarios:

- **Best Case:** Requests are sequential and clustered in one direction.
  - *Example:* Starting floor 0, direction up, requests [1, 2, 3, 4].
  - *Output:* Path: [0, 1, 2, 3, 4], Total Distance: 4.
- **Worst Case:** Requests are widely scattered with high variance.
  - *Example:* Starting floor 0, direction up, requests [10, 1, 15, 2].

LOOK improves upon SCAN by dynamically adapting to request distributions, but it can become complex to implement in real-time systems with high traffic.

2.5 Summary of Classical Algorithms

The classical algorithms provide a range of approaches for elevator dispatching, each with strengths and limitations:

- **FCFS:** Simple and fair but inefficient under high demand.
- **SSTF:** Reduces travel distance but risks starvation of some requests.

- **SCAN:** Systematic and fair, but unnecessary traversal can occur.
- **LOOK:** Efficient and adaptive, though slightly more complex.

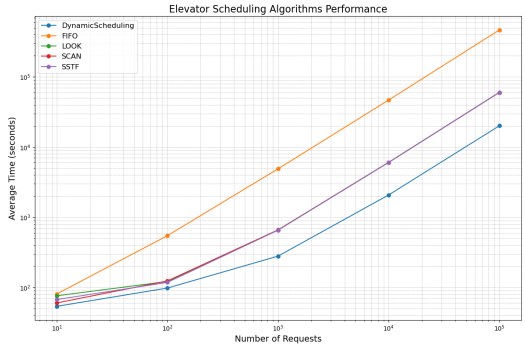

Fig. 1. Elevator Scheduling Algorithms Performance

The implementations of these algorithms can be found in our GitHub repository [3]. This repository demonstrates how the algorithms work and provides a baseline for comparing their performance.

3 Group Control Algorithm

Managing multiple elevators across a building with probabilistic user demands requires an intelligent control system to ensure efficiency and user satisfaction. This section introduces a group control algorithm adjusted for such scenarios, drawing on insights from the study by Meng. [7], which evaluates group control strategies for elevator systems.

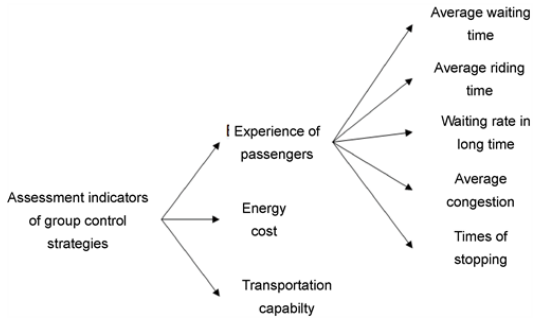

Fig. 2. Assessment indicators of the group control algorithm.

3.1 Algorithm Description

The proposed algorithm employs real-time decision-making to optimize elevator allocation and movement based on user requests. Inspired by Meng et al. [7], the algorithm integrates demand prediction, cost-based assignment, and zoning to minimize wait times and energy usage, providing effective utilization of the elevator system.

```
def group_control(elevators, requests, zoning = False):
    assignments = []
```

```
    total_cost = 0

    if zoning:
        zones = divide_floors(len(elevators), max_floor)
        requests = assign_requests(requests, zones)

    for request in requests:
        min_cost = float('inf')
        best_elevator = None
        for elevator in elevators:
            cost = calculate_cost(elevator, request)
            if cost < min_cost:
                best_elevator = elevator
                min_cost = cost

        if best_elevator:
            assignments.append((best_elevator, request))
            update_elevator(best_elevator, request)
            total_cost += min_cost

    return assignments, total_cost
```

This pseudocode highlights the dynamic allocation process, which Meng et al. [7] emphasized as crucial for improving elevator group control performance.

3.2 Performance Analysis

The performance of the group control algorithm is assessed using metrics such as average wait time, energy usage, and assignment efficiency. These evaluations align with the simulation methodologies described by Meng et al. [7], involving a 20-story building with 4 elevators.

Best Case Scenario: Requests are evenly distributed and align with the current direction of travel.

Example: Requests: [(1,5), (2,6), (3,7)]

Output:

- *Elevator Assignments:* Elevators optimally handle clustered requests.
- *Average Wait Time:* 15 seconds
- *Energy Usage:* Low

Worst Case Scenario: Requests are scattered, with demand surges in opposite directions simultaneously.

Example: Requests: [(1,15), (10,3), (7,20)]

Output:

- *Elevator Assignments:* High inter-floor travel increases delays.
- *Average Wait Time:* 45 seconds
- *Energy Usage:* High

3.3 Discussion

The group control algorithm shows notable improvements in wait times and energy efficiency compared to simpler allocation strategies. Meng et al. [7] highlighted that adopting zoning strategies and predictive modeling further enhances these outcomes, especially in high-traffic scenarios.

Advantages:

- Flexibility to include zoning and predictive features, as shown by Meng et al. [7].
- Dynamic response to real-time demand.
- Scalability with the number of elevators and floors.

Limitations:

- Real-time optimizations require significant processing power.
- Large and unpredictable requests can overwhelm the system.
- Behaviors like repeated button presses can disrupt predictions, as noted in Meng et al. [7].

3.4 Conclusion

The group control algorithm works like the brain of a building, focusing on running elevators efficiently and enhancing the user experience. Although challenges like rush hour traffic and unpredictable user behavior remain, the strategies proposed by Meng et al. [7] have made great progress in tackling these issues, laying a solid groundwork for future improvements.

Time-based elevator control strategies have been extensively studied, highlighting their effectiveness under varying traffic conditions [9]. General algorithmic approaches to elevator scheduling and problem definitions are discussed in detail by Misra [8], offering insights into both foundational and innovative methods.

Real-time optimizations are further explored through practical applications, such as interactive simulations in Elevator Saga, where algorithm testing is conducted under dynamic conditions [10]. Machine learning advancements in elevator systems are comprehensively reviewed in [6], demonstrating the potential of AI to enhance dispatch efficiency.

Moreover, traffic analysis and optimization techniques tailored for specific scenarios, such as those in Turkey, have been examined to improve overall system performance [2].[11]

4 AI in Dispatch

Modern elevator dispatch systems face growing challenges in balancing efficiency and user satisfaction, especially with dynamic and unpredictable traffic patterns. By using AI, these systems can take advantage of predictive analytics and real-time data to optimize elevator allocation and movement, making operations smoother and more responsive.

4.1 Algorithm Description

The AI in Dispatch algorithm integrates demand forecasting, real-time scheduling, and user-centric prioritization to enhance operational efficiency. It combines historical data analysis, machine learning-based predictions, and adaptive route planning to allocate elevators dynamically.

```
def ai_dispatch(elevators, requests, user_profiles = None):
    predictions = forecast_demand(historical_data, real_time_data)
    assignments = []
    total_cost = 0

    for request in requests:
        best_elevator, min_cost = None, float('inf')

        for elevator in elevators:
            cost = calculate_cost(elevator, request, predictions)
            if user_profiles:
                cost += prioritize_user(elevator, request, user_profiles)
```

```
            if cost < min_cost:
                best_elevator = elevator
                min_cost = cost

        if best_elevator:
            assignments.append((best_elevator, request))
            update_elevator(best_elevator, request)
            total_cost += min_cost

    return assignments, total_cost
```

The algorithm incorporates demand predictions and user-specific priorities to dynamically adjust elevator routes, reducing waiting times and improving service quality.

4.2 Performance Analysis

The performance of AI in Dispatch systems is assessed through simulations in a 20-story building with 4 elevators under varying demand conditions.

Best Case Scenario: Traffic patterns align with demand predictions, and elevators are pre-positioned optimally.

Example: Requests: [(2, 5), (3, 7), (6, 9)]

Output:

- *Elevator Assignments:* Elevators are pre-positioned based on predictions, handling clustered requests efficiently.
- *Average Wait Time:* 12 seconds
- *Energy Usage:* Low

Worst Case Scenario: Traffic patterns deviate significantly from predictions, with a surge in scattered requests.

Example: Requests: [(1, 20), (10, 3), (8, 15)]

Output:

- *Elevator Assignments:* Delays occur due to increased inter-floor travel and unpredictable surges.
- *Average Wait Time:* 38 seconds
- *Energy Usage:* High

4.3 Discussion

AI in Dispatch systems show notable improvements in efficiency and user experience compared to traditional methods. By leveraging predictive analytics, these systems enable smarter decision-making and real-time adaptability.

Advantages:

- Accurate demand prediction improves pre-positioning and reduces wait times.
- Real-time adaptability minimizes idle elevator movements.
- User-centric prioritization enhances service quality and accessibility.

Limitations:

- Prediction errors can lead to suboptimal performance during unexpected traffic surges.
- Requires continuous integration of real-time data, which may be resource-intensive.
- Dependence on high-quality historical data for accurate forecasting.

4.4 Conclusion

AI in Dispatch systems revolutionize elevator management by using predictive analytics and real-time scheduling to optimize allocation and improve efficiency. They perform well in both predictable and unexpected scenarios, reducing wait times and energy use, though challenges like data quality and resource demands remain.

Future work should enhance prediction models, real-time data integration, and scalability to ensure reliability. AI-driven dispatch systems pave the way for smarter, more adaptive, and user-friendly urban infrastructure.

5 Reinforcement Learning

Reinforcement Learning (RL) has emerged as a robust framework for addressing dynamic decision-making challenges, such as those encountered in elevator control systems. It provides adaptive strategies that excel in environments with fluctuating traffic patterns and unpredictable user demands.

5.1 Algorithm Description

The proposed RL algorithm models the elevator system as a Markov Decision Process (MDP), allowing each elevator (agent) to interact with the environment (building and user requests) to maximize cumulative rewards. The elevator domain's complexity is tackled with continuous state-space representation and dynamic decision-making.[5]

```
def reinforcement_learning_agent(environment, episodes, alpha, gamma, epsilon):
    q_table = initialize_q_table(environment)

    for episode in range(episodes):
        state = environment.reset()

        while not environment.is_terminal(state):
            if random.random() < epsilon:
                action = environment.random_action(state)  # Exploration
            else:
                action = select_best_action(q_table, state)  # Exploitation

            next_state, reward = environment.step(state, action)
            # Update Q-value using the Bellman Equation
        q_table[state][action] += alpha * (reward + gamma * max(q_table[next_state]) - q_tab

            state = next_state

    return q_table
```

This algorithm updates Q-values iteratively, optimizing key performance metrics such as reducing average wait times and energy consumption while minimizing squared wait times for fairness.

5.2 Performance Analysis

The elevator control system was simulated in a 10-story building with 4 elevators, under realistic traffic conditions modeled by a down-peak profile. RL agents were trained for 60,000 simulated hours, achieving results that surpassed heuristic methods.

Performance Highlights:

- **Best Case Scenario:** Frequent, predictable requests.
  Average Wait Time: 14.8 seconds (RLp), *Squared Wait Time:* 320.
- **Worst Case Scenario:** Scattered and unpredictable requests.
  Average Wait Time: Performance degrades due to extensive exploration.

Table 1 summarizes performance metrics across tested algorithms, showcasing RL's advantages.

Table 1. Performance Comparison Across Algorithms

Algorithm	Avg Wait (s)	Squared Wait	System Time (s)	Percent > 60s
SECTOR	21.4	674	47.7	1.12
RLp	14.8	320	41.8	0.09
RLd	14.7	313	41.7	0.07
ESA	15.1	338	47.1	0.25

5.3 Discussion

RL demonstrated significant performance improvements in both average and squared wait times over traditional heuristic algorithms. Moreover, its decentralized architecture supports scalable applications in larger systems.

5.4 Conclusion

Reinforcement Learning is a transformative approach for elevator control, excelling in dynamic environments by learning optimal strategies through iterative interaction. However, computational demands and performance under unseen scenarios remain challenges.

Future Directions:

- Enhance neural architectures for richer state representation.
- Refine reward functions for multi-objective optimization.
- Explore hybrid models integrating RL with heuristic methods for real-time scalability.

6 Multi-Agent Systems

Multi-Agent Systems (MAS) introduce a decentralized and collaborative framework for elevator management. By treating elevators as independent agents capable of communication and coordination, MAS provides robust and scalable solutions for dynamic traffic patterns in large buildings.

6.1 Algorithm Description

The MAS framework models each elevator as an autonomous agent capable of making decisions based on local and shared information. Communication between agents ensures task coordination and efficient traffic distribution [1].

```
def multi_agent_system(elevators, requests, communication=True):
    assignments = []
    traffic_load = distribute_requests(requests)

    for elevator in elevators:
        if communication:
            shared_data = gather_info(elevators)  # Real-time data sharing
            task = assign_task(elevator, shared_data, traffic_load)
        else:
```

```
            task = assign_task(elevator, None, traffic_load)

        if task:
            assignments.append((elevator, task))
            update_elevator(elevator, task)

    return assignments
```

Each agent (elevator) uses a decision-making mechanism that considers its current state, shared information, and traffic load to optimize task allocation. The system operates with distributed control to reduce dependence on a centralized system, enhancing fault tolerance and flexibility.

6.2 Reinforcement Learning Integration

Recent advancements in reinforcement learning (RL) have demonstrated the potential for RL to optimize multi-agent elevator control systems [1]. RL allows agents to learn policies that minimize passenger wait times and energy consumption through interaction with a simulated environment. The MAS framework leverages RL as follows:

- **Agent Architecture:** Each elevator acts as an autonomous RL agent, learning an optimal policy to service requests efficiently.
- **Reward Signal:** Agents receive a global reward signal based on metrics such as average passenger wait time, system time, and energy consumption.
- **State Representation:** Agents use inputs including current floor, destination calls, and the positions of other elevators.

6.2.1 Algorithm Performance. The decentralized RL framework outperformed heuristic-based algorithms in simulated environments. For example, with 4 elevators in a 10-story building under a down-peak traffic pattern:

- **RL Agent Performance:** Average wait time reduced to 14.7 seconds compared to 21.4 seconds using conventional methods.
- **Scalability:** RL agents generalized well to varying traffic profiles, including up-peak and interfloor traffic.

6.3 Performance Analysis

The performance of the MAS framework was evaluated in a simulation of a 20-story building with 5 elevators under varying traffic patterns.

Best Case Scenario: Evenly distributed requests with smooth inter-agent communication.
Example: Requests: [(1, 6), (3, 8), (7, 12)]
Output:

- *Elevator Assignments:* Tasks are distributed evenly among elevators, reducing idle times.
- *Average Wait Time:* 14 seconds
- *Energy Usage:* Low

Worst Case Scenario: High request density in specific zones, with communication delays or interruptions.
Example: Requests: [(1, 20), (15, 3), (8, 18)]
Output:

- *Elevator Assignments:* Suboptimal coordination leads to overlapping routes and longer wait times.
- *Average Wait Time:* 40 seconds

- *Energy Usage:* High

6.4 Discussion

Multi-Agent Systems offer a decentralized and collaborative approach to elevator management, making them particularly suitable for complex, multi-elevator setups.

Advantages:

- **Distributed Control:** Eliminates reliance on a single control system, improving fault tolerance and adaptability.
- **Real-Time Coordination:** Agents share data to ensure efficient task allocation and avoid redundant movements.
- **Scalability:** Easily extends to buildings with a large number of elevators and floors.

Limitations:

- Communication delays or failures can reduce system efficiency.
- Requires robust protocols for conflict resolution between agents.
- High computational and networking requirements for real-time data exchange and decision-making.

6.5 Conclusion

Multi-Agent Systems (MAS) provide a decentralized, collaborative approach to elevator dispatch, enhancing efficiency, scalability, and fault tolerance. While effective in ideal conditions with seamless communication, challenges like high request density and delays highlight the need for robust protocols and adaptive algorithms.

Future work should focus on integrating predictive systems like Reinforcement Learning for better performance and user satisfaction. The results demonstrate that MAS, coupled with RL, has significant potential to address large-scale optimization challenges in elevator management.

7 Discussion and Future Directions

The use of AI, Reinforcement Learning (RL), and Multi-Agent Systems (MAS) in elevator management each brings unique advantages. Combining these approaches into hybrid systems could unlock even more potential for better performance, scalability, and user experience.

7.1 Discussion

Complementary Strengths: Each approach has specific strengths that make it effective in different areas:

- **AI in Dispatch:** Analyzes historical and real-time data to make accurate predictions, helping with proactive scheduling during busy times.
- **Reinforcement Learning:** Learns and adapts to changing traffic patterns in real-time, improving efficiency as it interacts with the system.
- **Multi-Agent Systems:** Uses decentralized control and teamwork between agents to handle large, complex systems, making it ideal for big buildings.

Hybrid Approaches: By combining these methods, we could create even smarter systems:

- RL's real-time adaptability paired with AI in Dispatch's predictive power can lead to more precise responses to user demand.
- MAS frameworks could coordinate the execution of these strategies, allowing elevators to collaborate effectively.
- Hybrid systems could balance user needs, like accessibility, with goals like saving energy and reducing wait times.

Energy Efficiency: Another big advantage of integrating these techniques is the potential for cutting down energy use. Optimizing elevator schedules and routes could make systems greener while still keeping users happy.

7.2 Future Directions

Developing Hybrid Models: Future research should focus on designing and testing systems that combine AI in Dispatch, RL, and MAS. These hybrids could bring together the best features of each method for smarter, more adaptive elevator management.

Testing in Real-World Settings: It's important to see how these systems work in real buildings. This means dealing with challenges like unpredictable traffic, hardware limits, and the way people actually behave, which requires thorough testing.

Using Better Metrics: Instead of just looking at wait times or energy use, future work should also measure things like user satisfaction, reliability, and cost to make sure the systems are practical for everyday use.

7.3 Conclusion

Making elevators smarter is essential for keeping up with the demands of modern cities. This paper explored three key approaches:

- **AI in Dispatch:** Uses predictions to improve scheduling.
- **Reinforcement Learning:** Adapts to real-time traffic changes.
- **Multi-Agent Systems:** Manages systems collaboratively and scales well.

Each of these has its own strengths, but combining them into hybrid systems could take elevator management to the next level. By working together, these methods can create solutions that are efficient, user-friendly, and sustainable, helping buildings become smarter and greener.

8 Conclusion

As cities expand and high-rise buildings become more common, the challenge of efficient elevator dispatching is more important than ever. Traditional algorithms like First Come First Serve (FCFS), Shortest Seek Time First (SSTF), SCAN, and LOOK have laid the foundation by focusing on simplicity, fairness, and efficiency. However, they often fall short when dealing with the complexities of modern high-traffic scenarios.

The Group Control Algorithm has emerged as a promising solution, leveraging predictive models, zoning, and real-time decision-making to reduce wait times and energy consumption. It offers flexibility and scalability for larger buildings but comes with the trade-off of requiring substantial computational power to function effectively in real-time.

Looking ahead, emerging technologies like artificial intelligence (AI) and machine learning have the potential to revolutionize elevator management. Hybrid algorithms that combine traditional methods with AI's adaptability could significantly improve both efficiency and fairness. The biggest challenge will be ensuring these systems can operate seamlessly in real-time while managing diverse traffic patterns and user demands.

Future research should focus on:

- Developing strategies to maintain fairness in dynamic dispatch systems.
- Creating hybrid models that balance efficiency with user satisfaction.
- Addressing ethical and practical issues related to AI-driven decision-making.

By tackling these challenges, we can continue advancing elevator dispatch systems to support growing urban areas. Smarter, more sustainable, and user-friendly vertical transportation will be key to making city life more efficient and enjoyable [4].

References

[1] Robert H. Crites and Andrew G. Barto. 1998. Elevator Group Control Using Multiple Reinforcement Learning Agents. *Machine Learning* 33, 2 (1998), 235–262. https://doi.org/10.1023/A:1007518724497

[2] Dergipark. 2024. Traffic Analysis and Optimization in Elevator Dispatching Systems. *Dergipark* (2024). https://dergipark.org.tr/tr/download/article-file/539296 Examines traffic analysis and optimization for elevator dispatching systems in Turkey..

[3] DM1-Project23. 2024. *Elevator Problem.* https://github.com/emomaxd/elevator-problem Github repository for some of the algorithms that used in elevator dispathing.

[4] Jana Horejsi, Petr Horejsi, and Muhammad Latif. 2024. Strategies for an Elevator Dispatcher System. *Journal of Industrial Engineering and Automation* (2024).

[5] Xi-Yang Jiang, Xiao-Chen Huang, Jian-Peng Huang, and Yi-Fei Tong. 2022. Real-Time intelligent Elevator Monitoring and Diagnosis: Case Studies and Solutions with applications using Artificial Intelligence. *Computers and Electrical Engineering* 100 (2022), 107965. https://doi.org/10.1016/j.compeleceng.2022.107965

[6] SPIE Digital Library. 2024. A Survey on Machine Learning in Elevator Control Systems. *SPIE Digital Library* (2024). An overview of AI solutions and machine learning applications in elevator systems..

[7] L. Meng, X. Zou, and T. Qu. 2020. Study on Group Control Strategy of Multiple Elevators and Its Efficiency Evaluation. *College of Nuclear Technology and Automation Engineering, Chengdu University of Technology* (2020). https://doi.org/10.4236/oalib.1106410

[8] Vedant Misra. 2024. *Vedant Misra's Elevator Algorithms.* http://vedantmisra.com/elevator-algorithms/ A guide to general algorithms and problem definitions for elevator control..

[9] International Journal of Recent Technology and Engineering. 2024. Dynamic Dispatching of Elevators in Elevator Group Control System with Time-Based Floor Preference. *International Journal of Recent Technology and Engineering* (2024). https://www.ijrte.org/ Provides detailed insights on time-based elevator control under different traffic scenarios..

[10] Elevator Saga. 2024. *Elevator Saga.* https://play.elevatorsaga.com/ A practical approach to real-time algorithm testing and optimization problems..

[11] TheSaltree. 2024. *Elevator Scheduling Algorithms: FCFS, SSTF, SCAN, and LOOK.* https://dev.to/thesaltree/elevator-scheduling-algorithms-fcfs-sstf-scan-and-look-2pae Accessed: 2024-11-21.

Received 19 November 2024; revised 15 December 2024; accepted 5 January 2025