Proceedings of the SMART–2021, IEEE Conference ID: 52563
10th International Conference on System Modeling & Advancement in Research Trends, 10th–11th December, 2021
Faculty of Engineering & Computing Sciences, Teerthanker Mahaveer University, Moradabad, India

2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART) | 978-1-6654-3970-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/SMART52563.2021.9675307

Performance Benchmarking and Analysis of Various Elevator Dispatching Strategies

Rucha Kulkarni[1], Chirag Jain[2], Luv Gupta[3], Pravesh Ganwani[4], Varsha Hole[5]
[1,2,3,4,5]*Department of Information Technology, Sardar Patel Institute of Technology Mumbai, India*
E-mail: [1]rucha.kulkarni101@gmail.com, [2]chiragajain291@gmail.com, [3]luvkoolg15@gmail.com, [4]praveshganwani@gmail.com, [5]varsha hole@spit.ac.in

***Abstract*—In this modern era, high-rise buildings and skyscrapers are almost everywhere and the average number of floors in a building is almost 40. Thus, elevators play a crucial role in minimizing the time taken by people to reach their desired floor. We as computer scientists need to analyze how the elevators function and thus need to study the working of elevators to improve upon the existing algorithms. The following paper is a comparative study of various elevator dispatching algorithms, their working, and the results after simulating those algorithms. It first describes the Backtracking algorithm followed by the Q-learning approach and finally the Deep Q-learning approach to tackle the optimal control problem of elevator dispatching to minimize the waiting time of the passengers. The present research was conducted to study and investigate the merits and demerits of each of these techniques and how efficiently they solve the problem at hand.**

Keywords: Backtracking, Q-learning, Deep Q-learning, Rein-forcement Learning, Elevator Dispatching

I. Introduction

Different elevators offer distinctive service levels to their passengers based on their unique dispatching systems. Unlike conventional elevator systems, dispatching systems take into consideration multiple parameters to significantly improve efficiency and reduce the waiting time of their passengers. Designing optimal dispatching algorithms to optimize traffic flow through a building has gained significant importance to improve passenger handling efficiency.

Elevator dispatching system is a DES (Discrete Event System) optimal control problem dealing with multiple lifts in a multi-storey building satisfying multiple user requests to reduce the average waiting time for each passenger. DES problems often deal with different states of the entire system and every event inside the system is independent of every other event. Therefore, a single problem could be dealing with several different events inside such a system.

In this paper, we have researched, implemented, and com- pared three different approaches to designing an optimal elevator dispatching system with a primary focus on minimizing the average waiting time of passengers.

The first approach focuses on the dispatching system as an approximate dynamic programming problem and has been solved using the backtracking algorithm. The main motive behind using the backtracking algorithm is to solve the DES computational problem quickly by eliminating invalid solu- tions. Since the time and space complexity of the backtracking approach increases exponentially with an increase in number of elevator parameters, the second approach introduces an agent-environment model to optimize the solution by exploring the environment and exploiting already known information about it. This approach leverages the use of Q-Learning algorithm to reduce the computational expense and improve results and performance.

The final approach utilizes the concepts of Deep Learning to further optimize the Q-Learning approach to support increased state spaces. This approach inspired from paper [1] is heavily focused on designing an optimal neural network that holds the state of each elevator and decides the next step for the elevator based on its input parameters that resemble the environment.

II. Related Work

By applying the concept of reinforcement learning to a singular elevator system, this paper [5] makes a comparison between the performance Q-value iteration based and Q-learning based reinforcement learning algorithms. The former approach gave a better result with optimized low waiting times,as compared to the latter approach.

This paper [4] studies several elevator dispatching systems and the algorithms they work on-including Estimated Time to Dispatch (ETD), Destination Control System (DCS), and CompassPlus Destination System (CDS), and compares the efficiency of the algorithm based on a wide range of pa- rameters. The paper also throws light on the behavioral and environmental aspects of designing such elevator dispatching systems.

A novel method [2] based on deep convolutional and recur- rent neural networks has been proposed to solve the problem of elevator group control systems. The algorithm designed as an end-to-end self-learning algorithm gave optimal results, which were better than the traditional dispatching algorithms.

This paper [1] demonstrates a reinforcement learning-based adaptive order dispatching algorithm. The application men- tioned in the paper is for the field of the semiconductor industry, however, by working on the designs of the

algorithm, the scope of the applications can be widened to work for several other cooperative multi-agent systems such as the elevator dispatching systems.

This paper [3] proposed a method for group control elevator dispatching systems based on Siemens S7-1200 PLC, with the primary aim of reducing the riding time, waiting time, and other such parameters of the system.

III. Backtracking

In this approach, we construct a tree where each node represents the floor on which the elevator is currently present. This is a recursive approach that uses the DFS (Depth First Search) technique to make a series of decisions to solve the problem while also keeping a track of what has been processed in previous steps. Once the initial solution has been found which is when all the requests have been served by the elevator, the algorithm starts to backtrack the tree to find alternate solutions which can potentially be better. The backtracking algorithm thus finds the most efficient way in which the elevator can serve all the requests.

A. Algorithm

Consider the example where we need to serve two requests – Floor 2 to Floor 4 & Floor 3 to Floor 0. Also, the elevator is resting on Floor 0.

Each request consists of 3 parameters (Source, Destination & Ctr [Initially 0]) as shown in Table I. The Ctr can take 3 values: 0 – Passenger not picked up, 1 – Passenger picked up & 2 – Passenger Dropped (i.e., Request Satisfied). The root of the tree will be the floor where the elevator is currently present at. Important point to note here is that the nodes in the tree are generated using the request we need to satisfy and hence all the nodes will have values as that of the source or destination floor of any of the requests.

We start the recursive call by passing the function with the starting Floor. When we are generating new Nodes (From now on we will call them Floor) we go through all the requests and check the value of ctr. For each request if the Ctr value is 0 then the new Floor generated has Floor number equal to the Source Floor. If the Ctr value if 1 then the new Floor generated has Floor number equal to the Destination Floor. And finally, if the Ctr value is 2 then we do nothing. Table I shows the status of counter for the two requests before starting the algorithm.

Table I. CTR Values Before Starting the Algorithm

	Source	Destination	Ctr
Request 1	Floor 2	Floor 4	0
Request 2	Floor 3	Floor 0	0

In this case both the requests have Ctr value 0 hence the new Floors generated have value of 2, 3 i.e., source floors. The approach of Backtracking follows DFS, hence now we move onto the next Floor i.e., Floor 2.

Table 2. CTR Values After Servicing 1ST Request

	Source	Destination	Ctr
Request 1	Floor 2	Floor 4	1
Request 2	Floor 3	Floor 0	0

Now Request 1's Source is reached, and we have attended the passenger hence now we update the Ctr to 1. Table II shows the status of the counter for both the requests after we have picked up the passenger making the 1st request.

Once a new Floor is generated, we again loop through all the requests and check if the new Floor generated is the Destination for any of the requests. When we loop through – we also check whether their Ctr value is 1 because only if the Ctr value is 1 then we can drop them to their destination (Ctr value 1 means we have picked them up) and then we update the Ctr value for that request to 2.

Now the request 1 has Ctr value 1 hence the new Floor generated from this Floor will have the Destination value i.e., 4, while the request 2's Ctr value is 0 and hence new Floor generated because of it is the Source Floor i.e., 3. The state of the tree as of now can be seen by looking at the first 4 nodesif we move to the left from the source nodes which are the nodes 0, 2, 4 and 3.

Table 3. CTR Values After Completing 1ST Request

	Source	Destination	Ctr
Request 1	Floor 2	Floor 4	2
Request 2	Floor 3	Floor 0	0

This process is followed recursively and hence the finaltree is generated. Fig. 1. shows the representation of the tree generated by the code for the example stated above.

After the entire tree is generated, we find all the paths from source node and their costs as shown in the above repre- sentation. The Costs are calculated by finding the difference between two adjacent nodes in the path (as shown on the arrows) and adding them together.

After calculating all the cost, the lowest value is found,and that Path associated with that cost is the Optimal Path to serve all the Requests. In Fig. 1. we can see that there aretwo Optimal Paths to serve the required requests both havinga Cost of 8 units.

B. Results

To evaluate the performance of the algorithm several simulations were ran on our machine using Java and a graph

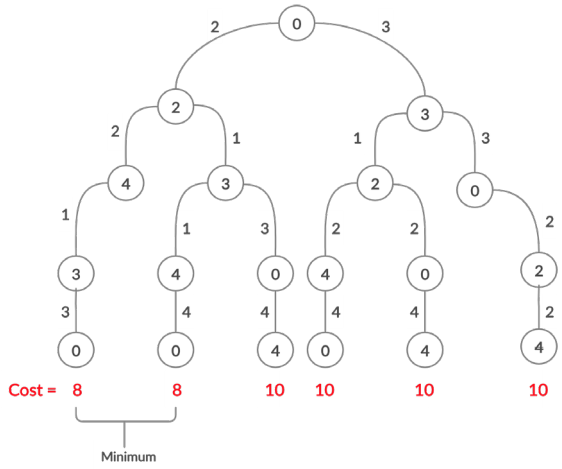

Fig. 1: Final Tree after Processing All Requests

was plotted as shown in Fig. 2. For a given number of floors (5, 7, 9 and 11 floors) in a building the amount of time taken for the algorithm to run was plotted against the number of requests. To simulate the algorithm random requests (2 to 10) were generated between two floors and the algorithm was ran to find the running time. For each iteration of request the algorithm was ran for 5 epochs and the average was calculated. It was found that the running time are almost exactly same for any number of floors in the building but as the number of requests increase the running time increases. Initially the running time increased by a factor of 3-5 and as the number of requests increase the running time increases by a factor of 10 with the highest value being 9491.254 ms when the number of requests are 10. Also, when the number of request reach 11 then Java throws a OutOfMemoryException which is expected as the tree grows beyond the heap size at that point.

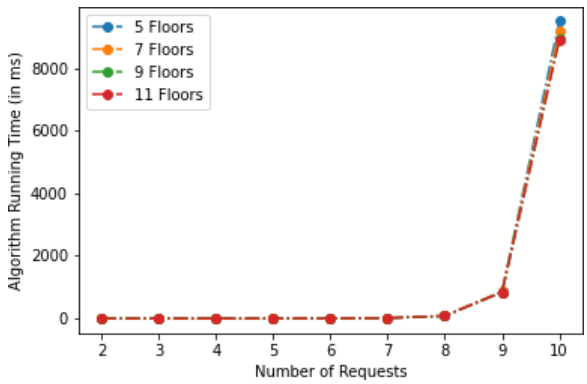

Fig. 2: Graph Showing Running Time of the Algorithm as the Number of Requestsare Increased

C. Conclusions

The time complexity of the backtracking approach is very high and proves to be very costly with an increase in pa- rameters such as number of requests, due to the increase in number of nodes in the tree. Thus, a solution that ensures faster convergence of the algorithm to find the optimal approach without going through every possible solution is required to overcome this issue and to ensure a more efficient optimization process. This propels us in the direction of Reinforcement Learning, specifically Q-learning.

IV. Q-Learning

The elevator can be considered a person who is trying to optimize a problem at hand and thus being rewarded. This is where Reinforcement Learning comes into play and specifically Q-Learning. The elevator is an agent trying to maximize the process of serving requests made by passengers in a building that is nothing but the environment. Thus, itis being rewarded for minimizing the waiting time of thepassengers.

A. Implementation

The implementation which is discussed below is based on the model proposed by Xu Yuan, Lucian Busoniu & Robert Babuska [5]. A single elevator system is simulated in the Elevator Dispatching Model using the Q-learning approach.

a) Assumptions and System Design: Certain assumptions have been made to minimize the complexity of the imple- mentation, one of them being that the system experiences a down-peak traffic pattern that primarily occurs in commercial buildings and offices. Also, if one or more than one passenger is waiting on a floor then the number of passengers waiting will be considered as one.

The down peak elevator pattern is primarily characterized by a single destination floor (i.e, the ground or zeroth floorin our implementation) and multiple floors (non-zero floors) from where the elevator loads the passengers. The system implemented has following other variables:

- Number of floors: 5
- Elevator capacity: 4 passengers
- Elevator Velocity: 3 m/s
- Inter-floor Travel Time: 2 s
- Stop-Time: 2 s (Time taken for the elevator to unload andload the passengers at any floor)
- Passenger Arrival Uniform Distribution: 0.6875, 0.0625, 0.09375, 0.09375, 0.0625 for events e = 0, 1, 2, 3, 4(Here if e=0 is selected then no passenger arrives butif e¿0 then passenger arrives at floor num ‘e’ with the probability specified above)

b) State Space and Complexity Analysis: The state space of the implementation is modelled as follows:

$$x = [c_1, c_2, c_3, c_4, p, v, o]^T \quad (1)$$

Here,

- c_i indicates call requests on a given floor i = 1, 2, 3, 4,
- p represents the discrete elevator positions which are 0,1, 2, 3, 4,
- v represents the discrete vertical velocity which can takethe values -3, 0, 3 m/s,
- o represents the discrete elevator occupancy which canbe 0, 1, 2, 3, 4.

Hence the Cardinality of the State Space described above willbe: $2^4 * 5 * 3 * 5 = 1200$

c) Action Set and Reward Function: The discrete action set for the elevator controller is -1, 0, 1 wherein, the -1 indicates downward acceleration of the elevator, 1 indicates upward acceleration of the elevator and 0 indicates halt orstop action. The reward function is given by:

$$\rho(x) = -\sum_{i=1}^{4}(c_i - o) \quad (2)$$

Here, c_i indicates call requests on a given floor i, o rep- resents discrete elevator occupancy. The reward at state st ata given time t is obtained by negating the total number of passengers waiting to enter the elevator at all floors as

wellthe number of passengers inside the elevator waiting to reach their destination.

d) Simulation: To simulate the algorithm, we ran the algorithm for 100 trials with each trial running for 500-time steps. During each time-step we generate a new call request on each floor using the passenger arrival distribution. Once a passenger is spawned on a floor then we store its arrival time. Then until the passenger is dropped to his desired floor this time keeps on increasing. Once the passenger is dropped to his desired floor, we calculate the waiting time for that passenger. The sum of the waiting times for all the passengers is calculated and averaged across the 500-time steps and 100 trials to find the performance of the Agent/ Controller in terms of average waiting times.

B. Results

To analyze the results of the Q-learning implementation a graph was plotted showing the trend of average Waiting Time against the number of episodes. We noticed that as the number of episodes increased, the average waiting time decreased (overall). The graph shows some spikes and a lot of variances as the number of episodes increase which is attributed to the fact that the passengers were spawned on various floors randomly to run the simulation. Due to this randomness some simulations saw the agent making bad decisions thus increasing the overall Waiting time. Theoretically, if we run the algorithm to infinity then a general trend of the waiting time decreasing can be seen. Also, in a practical scenario the best q-table can be captured by finding the global minima of the graph. The waiting time starts at 42s and then sees a series of variant spikes which keep on occurring till around 23 episodes after which it starts decreasing to the global minima. The highest value of waiting time was measured to be 113.45s while the lowest value came out to be 4.96s.

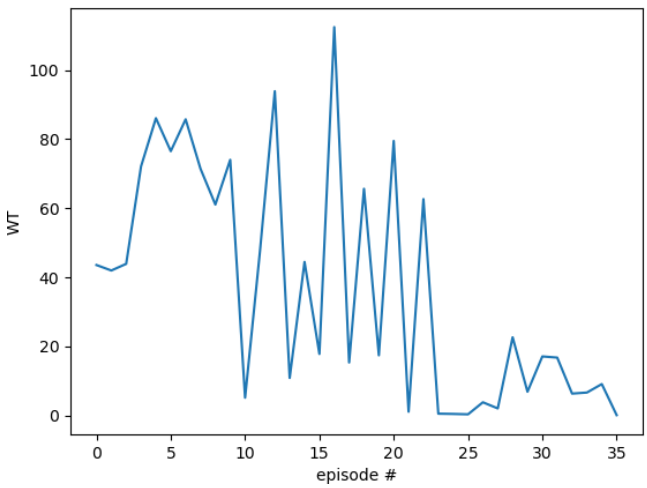

Fig. 3: Average Waiting Time vs No. of Episodes

V. Deep Q-Learning

The primary objective of Q-Learning is to find the optimal Q-Function that satisfies the Bellman optimality equation. However, the limitation of Q-Learning using value iteration is the size of the state space. As the state space size increases, the time taken to traverse all the states and iteratively update the Q-values proportionately increases.

For large state spaces, instead of directly computing the Q- values, we can use a function approximator to estimate the optimal Q-values.

Deep Q-Learning scales the existing Q-Learning algorithm to make it efficient to work with large state spaces. This approach utilizes deep neural networks that will estimate the Q-values when given a state-action pair for the environment which will ultimately approximate the optimal Q-function.

A. Model Specifications

Each elevator is modeled as an independent agent having 6 different states (3 motion states & 3 intent states). Whenthe elevator is in motion, it can have either of the 3 states – MOVING UP, IDLING, MOVING DOWN. When is elevator is idle and intends to start moving, it can have either of the 3 states – INTENT IDLE, INTENT UP, INTENT DOWN.

The observation space of an individual elevator holds 4 features – elevator position, carrying weight, hall-call status & in-elevator requests. There are 2 primary decision epochs i.e., sequence of times at which the decisions/actions are taken – Elevator Arrival (Upon reaching a particular floor) & LoadingFinished (Passenger being dropped off or picked up). Action space size for all the elevators is 10 i.e., there are 10 different actions an elevator can take whenever a decision epoch is triggered.

If the elevator is moving when Elevator Arrival is set off, at that point it needs to choose if it needs to stop on the next floor (MOVE-IDLE) or continue to move in the same direction (MOVE-MOVE). If the elevator is idling when Elevator Arrival is triggered, then it needs to choose an intended moving direction (IDLE-INTENT-UP, IDLE-INTENT-DOWN or IDLE-INTENT-IDLE) and then a passenger loading event is queued by the environment (intent needs to be declared so that passengers in the waiting area can decide whether they want to enter the elevator or not).

When the Loading Finished event is triggered, the elevatoneeds to choose an action that corresponds to its' declared intent. (For example, if the intent was IDLE-INTENT-UP, then it chooses between IDLE-UP-IDLE, or IDLE-UP-MOVE, basically whether it wants to stop at the next floor up or not). The average moving, idling & loading time for every elevator is 3 units, 5 units & 1 unit respectively. Every elevator accumulates discounted cost or rewards between every 2 decision epochs. Paper [19] presents an omniscient reinforcement scheme where every elevator *i* has an associated reward *R[i]*, where the total rewards it has received since itslast decision *(d[i])* is stored. This reward function is given as

$$\Delta R[i] = \Sigma_p e^{\beta(t_0 - d[i])}\left(\frac{2}{\beta^3} + \frac{2w_0(p)}{\beta^2} + \frac{w_0^2(p)}{\beta}\right) - e^{-\beta(t_1 - d[i])}\left(\frac{2}{\beta^3} + \frac{2w_1(p)}{\beta^2} + \frac{w_1^2(p)}{\beta}\right) \quad (3)$$

where t_0 is the time of the last event & t_1 is the time of the current event. Every passenger p waiting between t_0 & t_1, $w_0 p$ & $w_1 p$ is the total time the passenger has waited at t_0 & t_1

B. Methodology

Every elevator has an associated deep neural network that has been optimized using the Adam's learning rate optimiza- tion algorithm that makes use of squared gradients to scale the learning rate of every single parameter.

During each trial, passenger spawns on each floor are assumed to follow Poisson Distribution. For Poisson rate λ, the arrival time for the nth passenger follows the Gamma Distribution –

$$\Gamma(n, \tfrac{1}{\lambda}) \quad (4)$$

The average weight of every passenger follows the normal distribution. Finally, all the elevators are spawned randomly on arbitrary floors which completes the environment and is now ready for training.

The environment steps through the simulation until the next decision epoch is reached at which point the environment iterates through the list the actions to be taken and schedules them for all the decision elevators. The environment then prepares the state representation for every decision elevator and updates the rewards attained by them during this interval. For every iteration, the environment randomly selects a decision agent, and computes its Q-values for every output state. The minimum of these Q-values is used to compute the targeted Q-value and update the parameters of the neural network.

Now that the neural network for the decision agent has been updated, the environment needs to choose a specific action the decision agent takes. Softmax action distribution is used to compute the probability distribution on the legal action state of the decision agent. The exploration rate of the model controls the temperature which is used to control the randomness of the softmax distribution. The environment samples a legal action from the probability distribution of the legal action state and appends it to the list of actions to be taken. The iteration ends when the specified number of simulation hours are completed. The updated neural network for every elevator is saved as a final model which can be used for evaluating the efficiency of the algorithm for testing purposes.

C. Results and Observations

The model was tested for 2 elevators and 5 floors and it was able to adapt itself in a dynamic environment after 750 simulation hours. A graph was plotted against average waiting time of all passengers against the simulation time.

For different learning rates, the graphs showcased signifi- cantly different behavior and it was noticed that for an optimal value of learning and exploration rate, the model was able to achieve an optimum wait time in the range of 300 to 400 sim time (time unit for the SimPy environment).

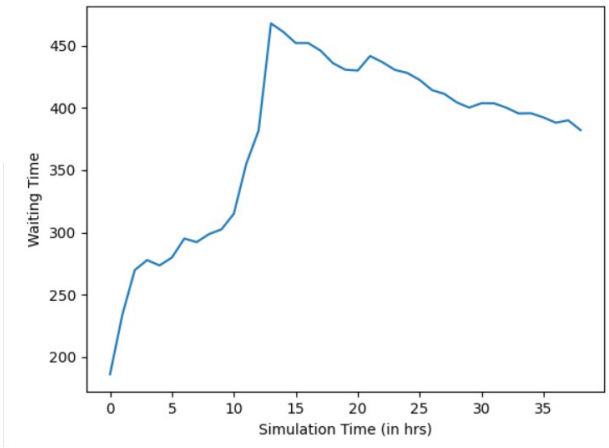

Fig. 4. Average Waiting Time vs Simulation Time for α = 0.0001

Fig. 4 shows a graph plot for learning rate = 0.0001 and exploration rate = 0.992. The graph shows an exponential rise in the waiting time initially since the model first starts to explore the environment (i.e., random MOVING UP, IDLING, MOVING DOWN actions). However, with more training the model learns to adapt itself to earn more rewards which results in decreasing average wait time.

Fig. 5 shows a graph plot for learning rate = 0.0003 and exploration rate = 0.992. The graph again shows a slightly steep rise in the waiting time in the first half followed by a stabilizing second half.

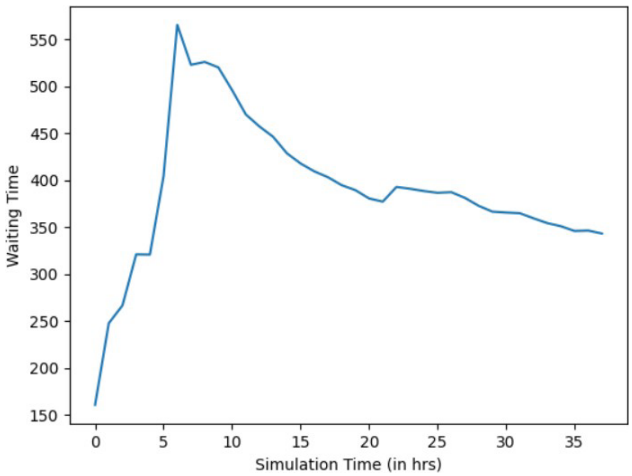

Fig. 5. Average Waiting Time vs Simulation Time for α = 0.0003

Fig. 6 shows a graph plot for learning rate = 0.38 and exploration rate = 0.992. For a very large learning rate, the graph attains the highest peak value quite early and then slowly decreases to achieve the optimal waiting time value.

For one particular iteration, the model traps itself in a cycle. Fig. 7 shows a graph plot for learning rate = 0.001 and

exploration rate = 0.992. The above graph shows a steep risein the average waiting time since one elevator loops into IDLEstate and therefore only one elevator keeps working.

From the above graphs, it is evident that the optimal average waiting time remains the same irrespective of the learning rateof the model. However, with increasing learning rates, the timeto achieve the optimal result decreases.

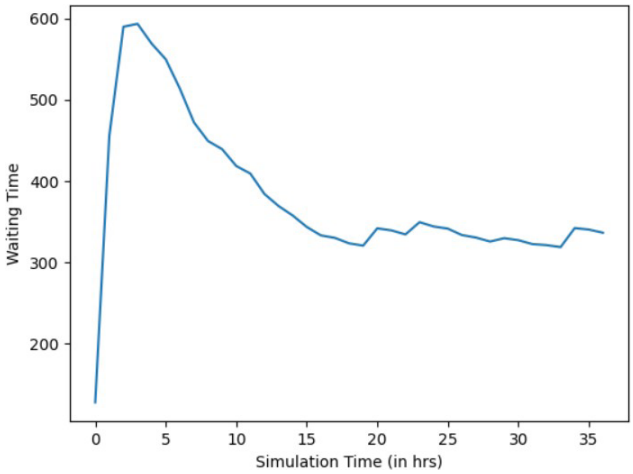

Fig. 6: Average Waiting Time vs Simulation Time for $\alpha = 0.38$

VI. Comparative Analysis

This section analyses and compares the three approaches discussed in the previous sections and gives an overview of the differences among these solutions. The analysis is done on various parameters such as Time Complexity, Space Complexity, Reward Function, Optimising Factor, etc., which are summarized in Table IV.

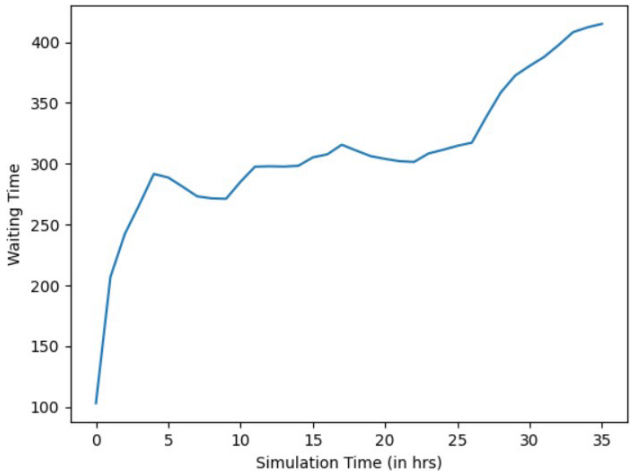

Fig. 7: Trapped Model (i.e., One Elevator Stops Working)

Table IV: Comparative Analysis

Parameter	Backtracking	Q-Learning	Deep Q-Learning
Optimising Factor	Recursion	Model-Less Dynamic Programming	Deep NeuralNetwork
Time Complexity	Increases as thenumber of requests increase because of the growing requesttree. The time complexity of this algorithm is not dependent on the number of floors because the algorithm solves the problem by fulfilling the requests irrespective of the floors	The Q-Learningalgorithm needsto be trained for multiple episodes before it starts showing some results. As the number of training parameters grow large so does the performance. This way the upper bound onthe time complexity is infinite	Remains the same irrespective of the number ofrequests, floorsor elevators
Space Complexity	Increases exponentially asthe number of requests increase. In our simulation we reached OutOfMemo-ryException for11 requests which shows that this approach isn't optimal when we have small space constraints	The Q-table increases exponentially with the increase in number of floors. In our simulation for a5 storey building the state space came out to be 1200, similarlyfor a 10 storey building the state space increases to 1,53,600	Since, every elevator has its own neural network to storeits parameters, the space complexity of this method increases with number of ele-vators/ models.
RewardFunction	It is the summation of the costs that are incurred when the elevator is transitioning between different floors to serve passenger requests	It is the negative of the summation of difference between the callrequests and thediscrete elevatoroccupancy for agiven floor. Themathematical expression is mentioned in equation (2)	It is calculatedin a way such that each elevator accumulates itsrewards after every decision. The reward obtained in every epoch depends on thepassenger waiting times and is mentioned in equation (3).
Limitation	The algorithm proves to be costly and inefficient with the increase in the number of nodes due to increase in parameters suchas number of requests, number of floors, etc	The efficiency of the algorithm decreases and leads to a less optimal performance asthe size of the Q-table increases to accommodate the increasing number of Q-values	The model initially requires a significant amount of timeto train itself toachieve optimalvalues. Since, this approach relies on usinga function approximator, the estimated Q-value may ormay not be optimal

VII. Conclusion

Implementation of the Q-learning model gave rise to the realization that the model would be suitable only for environ- ments with discrete and finite state and action spaces. A possi- ble alternative to overcome this limitation and for extending Q-learning to richer environments could include implementation of the model by applying function approximators to learn the value function, taking states as inputs, instead of storing the full state-action table. However, since deep neural networks are powerful function approximators, it seems more logical to try to adapt them for this purpose.

VIII. Future Work

In the future, the algorithms can be studied in terms of the scalability of the optimized methodology. In addition to that, the Model-less Q-learning approach can be further optimized by working on the global minima across all episodes by fetching the corresponding Q-values. The Deep Q-learningapproach can be further extended in such a manner that the system predicts the destination floor without taking input fromthe passenger. The algorithm could take into consideration several parameters such as passenger's floor request history, etc. to offer a personalized optimal experience to the user.

References

[1] Stricker, N., Kuhnle, A., Sturm, R., & Friess, S. (2018). Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Annals, 67(1), 511–514. doi:10.1016/j.cirp.2018.04.041

[2] Q. Wei, L. Wang, Y. Liu and M. M. Polycarpou, "Optimal Elevator Group Control via Deep Asynchronous Actor–Critic Learning," in IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 12, pp. 5245-5256, Dec. 2020, doi: 10.1109/TNNLS.2020.2965208.

[3] Huan, H., Tian, H., Yan, Z., & Yao, Z. (2019). Group control elevator dispatching system based on S7-1200 PLC. 2019 Chinese Automation Congress (CAC). doi:10.1109/cac48633.2019.8996971

[4] Kheshaim, M and Latif, MN and and Kundu, S (2016) A Review of Elevator Dispatching Systems. In: Proceedings of the World Congress on Engineering, 2016, Vol II. International Association of Engineers,pp. 671-673. ISBN 978-988-14048-0-0

[5] Xu Yuan, Lucian Bus¸oniu, Robert Babus˘ka, Reinforcement Learning for Elevator Control, IFAC Proceedings Volumes, Volume 41, Issue 2, 2008, Pages 2212-2217, ISSN 1474-6670, ISBN 9783902661005, https://doi.org/10.3182/20080706-5-KR-1001.00373..

[6] D. L. Pepyne and C. G. Cassandras, "Design and implementation of an adaptive dispatching controller for elevator systems during uppeak traffic," in IEEE Transactions on Control Systems Technology, vol. 6, no. 5, pp. 635-650, Sept. 1998, doi: 10.1109/87.709499.

[7] L. Van, Y. Lin, T. Wu and Y. Lin, "An Intelligent Elevator Development and Management System," in IEEE Systems Journal, vol. 14, no. 2, pp. 3015-3026, June 2020, doi: 10.1109/JSYST.2019.2919967.

[8] A. E. Yildirim and A. Karci, "Group elevator control optimization using artificial atom algorithm," 2017 International Artificial Intelli- gence and Data Processing Symposium (IDAP), 2017, pp. 1-6, doi: 10.1109/IDAP.2017.8090320.

[9] R. M. Hoffman and H. H. Asada, "A multi-track elevator system for E- commerce fulfillment centers," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 4694-4701, doi: 10.1109/IROS.2017.8206341.

[10] J. R. Fernandez and P. A. Cortes, "A survey of elevator group control systems for vertical transportation: A look at recent literature," IEEE Control Syst. Mag., vol. 35, no. 4, pp. 38–55, Apr. 2015.

[11] Q. Wei, B. Li, and R. Song, "Discrete-time stable generalized self- learning optimal control with approximation errors," IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 4, pp. 1226–1238, Apr. 2018.

[12] A. Heydari, "Stability analysis of optimal adaptive control under value iteration using a stabilizing initial policy," IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 9, pp. 4522–4527, Sep. 2018.

[13] Y. Wang, H. He, and C. Sun, "Learning to navigate through complex dynamic environment with modular deep reinforcement learning," IEEETrans. Games, vol. 10, no. 4, pp. 400–412, Dec. 2018.

[14] X. Xu, H. Chen, C. Lian, and D. Li, "Learning-based predictive control for discrete-time nonlinear systems with stochastic disturbances," IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 12, pp. 6202–6213, Dec. 2018.

[15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learn-ing with double Q-learning," in Proc. Conf. Artif. Intell., 2016, pp. 2094–2100.

[16] V. Mnih et al., "Asynchronous methods for deep reinforcement learning,"in Proc. Int. Conf. Mach. Learn., 2016, pp. 1928–1937.

[17] Y. Li, "Deep reinforcement learning: An overview," 2017, arXiv:1701.07274. [Online]. Available: https://arxiv.org/abs/1701.07274

[18] J. Zhang, H. Zhang, and T. Feng, "Distributed optimal consensus control for nonlinear multiagent system with unknown dynamic," IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 8, pp. 3339–3348, Aug. 2018.

[19] Crites, R.H., Barto, A.G. Elevator Group Control Using Multiple Re- inforcement Learning Agents. Machine Learning 33, 235–262 (1998). https://doi.org/10.1023/A:1007518724497.