Multi-agent Simulator for Reinforcement Learning-based Elevator Dispatching

Michael Morckos
Robotics and Automated Systems (RAS) Research Group
German University in Cairo-(GUC)
New Cairo City-Main Enterance, Al Tagamoa Al Khames, Egypt
Email: michael.morckos@student.guc.edu.eg

Alaa Khamis
Pattern Analysis and Machine Intelligence (PAMI) Lab
University of Waterloo
200 University Avenue West, N2L3G1 Waterloo, ON, Canada
Email: akhamis@pami.uwaterloo.ca

***Abstract*—In this paper, a four-elevator, ten-floor multi-agent elevator system software simulator is illustrated. The system consists of two main entities: the first entity is the elevator car which is simulated as a bundle of interconnecting collaborative agents each doing a certain function of the car; the second entity is the dispatcher which uses a Reinforcement Learning-based algorithm to handle the dispatching process. The algorithm is a direct application of Artificial Intelligence and is able to learn and improve by experience.**

***Index Terms*—Stochastic process, multi-agent, machine learning, reinforcement learning, Q-learning.**

I. Introduction

Elevator dispatching has long been a wide and diverse research area. Due to the economic importance of elevators, many scientists and researchers have researched and developed numerous dispatching techniques. The primal purpose of all elevator dispatching techniques is reducing the passengers waiting time and providing high quality service. Since the elevator system environment is highly variable and stochastic, most of the dispatching techniques couldn't yield optimal results. Some scientists explored the usage of artificial intelligence applications in the dispatching problem.

This paper describes the development of a software agent-based simulator for an elevator system expandable up to ten floors and four elevators. A simplified version of the reinforcement learning-based dispatching algorithm proposed by Crites and Barto [1] is implemented in the dispatcher of the system. The purpose of the dispatcher is to improve the systems response to passengers requests by minimizing the elapsed time from the moment a call button is pressed till the passenger is served (average waiting time). The dispatcher also aims to minimize the system time, which is the traveling time of a passenger inside the elevator till he/she is dropped off at the destination floor. The main components of the simulator are listed as follows:

- The elevator car: which includes all the functionalities of a real life elevator car; moving, stopping, opening/closing the door and taking the passengers requests through the internal buttons panel.
- The dispatcher: which is responsible for assigning elevator cars to serve different requests from different floors.
- The graphical user interface: which provides interactive and illustrative visualizations of the simulated system.

The paper is organized as follows: section 2 gives a detailed description of the elevator-dispatching problem and highlights three techniques to solve this problem. Section 3 addresses reinforcement learning as a solution for elevator dispatching problem. In section 4, the detailed design of the proposed simulator is presented followed by demonstrating the effect of different attributes on the dispatching algorithm behavior in section 5. Finally section 6 summarizes the paper and introduces proposed future work.

II. Elevator Dispatching

Elevators play an important rule in people's daily life since they greatly facilitate transportation between the different floors of buildings. In a typical situation a passenger at a certain floor calls an elevator and waits for its arrival; the waiting time is greatly affected by many conditions in the system such as the number of elevators, passengers traffic, number of floors...etc. The most important issue that affects the passengers waiting time is how elevators behave, that is; the "dispatching" strategy that the elevators follow in serving all passengers requests at different floors [2]. For instance, how would elevators behave at peak time to ensure minimal waiting time for all passengers, how would they behave if there are few or no requests or the bulk of the requests are coming from a certain floor [2]. The task of monitoring and dispatching of the elevator cars is the responsibility of the dispatcher.

Larger elevator systems have highly variable environments with huge number of different situations, thus the dispatching problem is considered an ideal real life example of a stochastic optimal control problem of a great economic importance [2]. A stochastic problem is a highly variable and non-deterministic problem that it can yield different outcomes at different encounters of the same conditions. In order to demonstrate this, Crites and Barto [1] studied a model of an elevator system consisting of ten floors and four elevator cars [2]. They estimated the number of the different states in the system as follows: each car has ten buttons for the ten floors and thus there are 2^{40} different combinations of the 40 buttons of the cars. In addition, all the ten floors have two call buttons except

for last and ground floors, which have one button each and so there are 2^{18} different combinations of all call buttons. Moreover, there are 18^4 different combinations of the different positions and directions of the cars. Summing up all possibilities, the system had roughly 10^{22} different states, not counting for possible hidden states. Due to this huge number of states, conventional algorithms based on dynamic programming are strictly not suitable. Using dynamic programming with such a system would take one thousand years just scanning the states space [2]. Many scientist and researchers have proposed different solutions and techniques to enhance the dispatching process. In the following subsections three of these techniques are presented. Most of these techniques are concerned with reducing the average waiting time and the efficiency of service.

A. Priority Dispatching

Priority dispatching is a technique used in multi-car elevator systems which involves assigning priorities to different passengers' calls and dispatching elevators accordingly to them [3]. In a typical modern computer-controlled system, there are two kinds of passenger's requests; the first is made at a floor in the building and is called a "hall call", the second is made inside the elevator car by pushing a floor number button, this is called a "car call" [3]. Whenever a hall call is made, the call is registered in the dispatcher which keeps track of all assigned calls [3]. When an elevator car responds to the call by either reaching the destination floor or serving a passenger at a specific floor, the call is erased from the system.

When a passenger makes a hall call at a specific floor, the dispatcher assigns a car arbitrarily to serve that call. However, most typical systems don't keep track of the times at which calls are made and don't even have information regarding the proximity of the assigned car to the target floor, so the dispatched car may not be the nearest one. Moreover, the car will serve all intermediate calls (car or hall calls) made along its direction to the target floor regardless of their arrangement. Consequently, the passenger with the original call might have to wait a long time before being served. Systems operating in this manner will often has the elevator cars gathered at certain areas along the building instead of being evenly distributed and thus other portions of the building won't be served efficiently.

A proposed solution to this problem is to make the dispatcher prioritize hall calls received from passengers. In such a system, a hall call with the longest waiting time is classified as a priority call and when this happen, the system assigns the nearest car that has served all its car calls to serve the priority call ignoring any hall calls made later [3]. In order to increase the efficiency of the system, once a priority call is designated by the dispatcher, a "priority zone" can be virtually registered which includes one or two floors above or below the floor from which the priority call originated. Thus calls from these floors can be arrange according to waiting times and will be served by a single elevator car as long as they are on the same direction as the car (priority cluster). This technique is efficient on the long run since it reduces passengers' waiting times and keeps the elevator cars evenly distributed [3]. However, it doesn't cover all possibilities.

B. Multiple Term Objective Function and Instantaneous Assignment

The system assignment of cars to different hall calls is based upon a number of parameters all based on the average waiting time of the passenger. Since the elevator environment is highly variable, the system may reassign the call to different cars in the system many times before the call is finally served by one of the cars. The passenger comes to know which car among the others that will serve him/her only moments before the car's door opens [4]. Average registration time [4] is the time elapsed from pressing a hall button until the call is served or canceled (in some elevator systems), it's considered an important parameter in measuring performance [4]. A good average registration time might be a good indicator but it can be deceiving as well; since it may include very long registration times originated from a large number of shorter registration times. The reason is that an elevator car might bypass a floor with a hall request simply because it was not assigned to serve it.

This dispatching technique employs Instantaneous Car Assignment (ICA) which states that once a car is assigned to serve a call, the assignment is not shifted to another car [4]. Once a hall call button is pressed, the passenger instantly knows which car is assigned and thus has the opportunity to be right at the door when the car arrives, maximizing the know-and-go time (time to know which car is assigned and to get in it) [4]. However, traditional systems used reassignments in order to ensure an optimal outcome based on the continuously changing conditions and so, this technique is concerned with reducing the average registration time, maximizing the know-and-go time and achieving an optimal car assignment. This technique uses several parameter; Remaining Response Rime (RRT), Predicted Registration Time(PRT), Maximum Predicted Registration Time (maxPRT) and finally, the Relative System Response(RSR). Moreover, it uses what is called the "objective function" which is a mathematical function using the previously mentioned parameters along with user defined ones to determine which elevator car in the system is to be assigned to serve a certain call. The car which yields the minimum object value calculated by the object function is the one assigned to serve the call [4].

RRT is the time an elevator car takes to reach a target floor. Each car has a RRT value and it's calculated cumulatively. Cars with low RRT are usually chosen by the dispatcher to serve long waiting hall calls [5]. However, RRT alone is not enough in achieving an optimal assignment; a second parameter PRT is used which is the amount of time elapsed from the moment a hall call is made. For example, having two cars, A and B where A is at floor 10 and B at floor 12 in a building, car A is moving downwards to serve a call at floor 6 when a call at floor 9 is made. Based on RRT data, car B should be reassigned to serve the call at floor 9. However, this would increase the PRT of the call at floor 6.

Thus car B should be assigned to serve the call at floor 9. The objective function includes a user defined penalty parameter which determines the maximum time a passenger can wait without getting uncomfortable. If the PRT of a car is very low or very high compared to the penalty parameter, the car is given a penalty since if the PRT is very high it means a long waiting time. The PRT shouldn't also be too low since there might be more urgent calls to serve [4]. The third parameter in the objective function is maxPRT which is like PRT but used with considerably long waiting times. The fourth parameter RSR defines the average elevator system response to different calls.

C. Dispatching by Predicting Peak Times [6]

This technique is concerned with following different dispatching strategies at different periods in the day based on the prediction and classification of passengers traffic density into time periods. It's well suited for large buildings with multiple elevator cars and it's distinguished for being *artificially intelligent*. Passengers traffic in a building can be classified into time periods according to density, for instance; up-peak and down-peak periods. An up-peak period is distinguished by high traffic density, it usually occurs early at mornings and at afternoons (when people are coming to work and when they are leaving respectively). Down-peak periods are distinguished by below normal traffic density, this is usually during the rest of the day and at evenings.

In an up-peak period in the morning, in order for the system to activate an up-peak strategy, it predicts the number of passengers arriving at the lobby based on previous data and information gathered, then it compares this number to a certain threshold. If the number is above the threshold then an up-peak period is confirmed and the right strategy is deployed. The threshold is a constant pre-determined value which is usually a percentage of the people in the whole building. Moreover, to increase the accuracy and efficiency of the system, the threshold value can be recalculated in order to be able to distinguish other periods. For instance, when an up-peak period is confirmed, the system can estimate the number of passengers leaving the lobby to inter-floors over a time interval, then the percentage of an elevator car capacity is compared to this number. One one hand, if the number is greater than that percentage, the system determines that it has shifted to an up-peak period late and consequently, the threshold value is reduced. On the other hand, if the number is smaller than the percentage, it's an indicator that the system has shifted to an up-peak period early, thus the threshold value is increased.

III. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning [7, 8] technique categorized as an unsupervised learning [7] technique, it features an interactive intelligent agent with an explicit goal to achieve. An RL agent works by "learning" how to perform actions in response to different situations/states perceived from its surrounding environment [2]. These actions affect and alter the state of the environment and in return, the environment responds by a reward signal to the agent. The agent's aim is to maximize the rewards received from the environment on the long run through achieving its goal in an efficient way.

A distinctive feature of RL making it unique among most of the other machine learning techniques is that it doesn't require external assistance or supportive learning materials in learning. The RL agent depends entirely on its interaction with the environment to formulate the right actions at the right states by deploying a trial-and-error approach in order to come up with an optimal policy in dealing with the different environment situations; an optimal policy is the policy which will yield maximum rewards on the long run [2].

The RL agent is the intelligent entity enclosing the RL algorithm whose task is to learn and make decisions. The learning environment is responsible for providing the agent with the current systems state, modifying the states based on the agents actions and providing rewards.

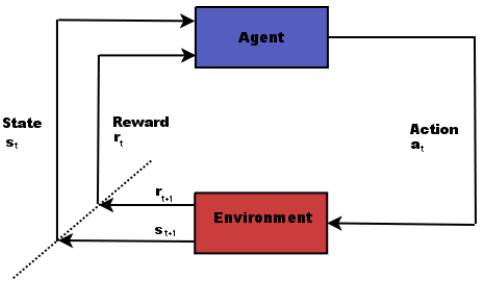

Fig. 1. The Reinforcement Learning model.

As shown in Fig. 1, in an abstract RL framework the agent and the environment interact on basis of discrete timing, $t = 0, 1, 2, 3...$. At a single communication at a specific time t, the agent perceives the current state of the environment s_t, where $s_t \in S$, the set of all possible environment states. Based on the state, an action a_t is taken, where $a_t \in A(s_t)$, the set of all possible actions available for the state s_t. On the next time cycle, the environment responds by a reward r_{t+1}, where $r_{t+1} \in R$, the set of rewards. The environment shifts to a new state s_{t+1} , which is again perceived by the agent and the cycle continues [2].

A. Q-Learning

Q-learning is an off-policy Temporal Difference (TD) [2] reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy. This algorithm works by directly approximating an optimal action-value function Q* regardless of the current working policy. Thus this algorithm works independently of the policy, which greatly simplifies the problem and speeds up convergence [2]. The policy still determines which state-action pairs are encountered and updated, but as long as all pairs are constantly updated the algorithm works perfectly. The one-step Q-learning is represented by the equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

Noting $max_a Q(s_{t+1}, a_t)$, the algorithm follows a greedy approach in choosing actions, that is, for each state s, the algorithm selects the action a from the set $A(s)$ where $Q(s, a)$ has the highest value, thus figuring out an optimal action by experience.

For *semi-Markov* continuous problems like elevator dispatching a modification of the Q-learning equations is used for more accuracy [2]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\int_{t_1}^{t_2} e^{-\beta(\tau - t_1)} \cdot r_\tau d\tau + e^{-\beta(t_2 - t_1)} max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

where $e^{-\beta(t_2 - t_1)}$ is a variable discount factor depending on the time interval between the action and transition to a new state.

B. Reinforcement Learning-based dispatching Algorithm

The dispatching algorithm used in simulator dispatcher is the one-step Q-learning. The main components are arranged as follows:

- **Dispatching agent**: is the RL agent whose task is to receive hall calls, acquire the possible actions for each call from the environment, uses the algorithm to pick a certain action and finally applies the action by dispatching an elevator car to serve the call.
- **Dispatching environment**: keeps track of the current state of the system and provides the agent with the action list for a specific hall call.
- **Dispatching state**: the state is a snapshot of the environment at an instance. It basically holds four information data about each elevator in the system as well as information about all hall calls. Information about an elevator car include its ID, current floor, direction and its car calls. The state representation is a simplified version of that in Crites and Barto simulator [2].
- **Dispatching action**: an action in the system is choosing an elevator car to serve a specific hall call.
- **Reward function**: the passengers waiting time is the main evaluation metric. A positive reward is given when the waiting time is below sixty seconds, a negative reward is given otherwise.
- **Dispatching policy**: the policy adheres to the conventional elevator protocols which state that an elevator doesn't change its direction as long as there are requests in its current direction. This policy is not variable.

In its simplest form, Q-learning usually uses lookup hash tables for storing $Q(s, a)$ tuples. However, since the dispatching environment contains huge number of states, any linear storage would be very inefficient. The algorithm uses a nonlinear artificial neural network trained by back-propagation [9] for storing state-action pairs. The network updates and learns after receiving a certain number of pairs.

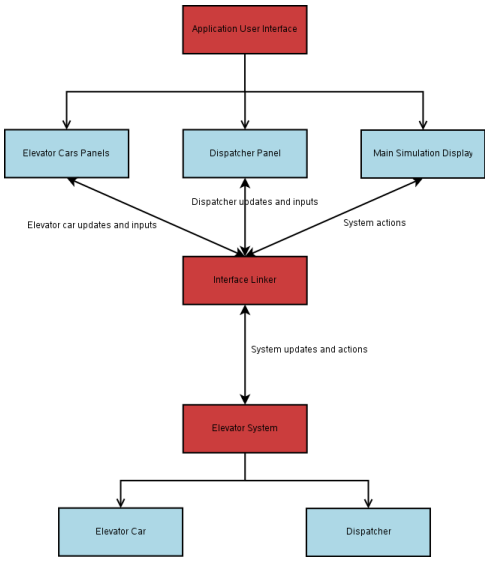

Fig. 2. Simulator architecture.

IV. Multi-agent Simulator

The simulator design was intended to be distributive and non-centralized, meaning that all the tasks performed should be carried out collaboratively by multiple entities with no entity doing the whole job. These entities were to have little or no centralized controller and if so the controller would only issue high level commands to the underlying entities. In order to achieve this a multi-agent design approach was adopted. For the agents implementation, JADE (**J**ava **A**gent **DE**velopment Framework) was used [10, 11]. JADE is an open source, Java-based platform for agent applications, it allows the development of multiple interactive agents and handles their peer-to-peer communications [11]. A JADE *Agent* is a single *Java Thread* and it includes one or more *Behaviours*, which are the functions and tasks performed by that agent. Moreover, JADE incorporates a database of the multiple running agents that keeps track of all exchanged messages and data. The simulator is mainly composed of three main layers as shown in Fig. 2 and illustrated in the upcoming subsections.

A. The Elevator System

Considering any elevator system, it's mainly composed of a number of elevator cars, which transport the passengers along different floors and the dispatcher, which is responsible for assigning the cars to reply for passengers requests. The elevator system is composed of two main components:

- **Elevator car**: this component houses the implementation of all the agents of the elevator car and its shaft. It simulates the real life functionalities such as opening/closing the door, starting/stopping the driver motor, keeping track of and serving passengers' requests issued from the car's buttons panel. As shown in Fig. 3, the elevator car and its shaft are implemented as separate entities; that is, all the physical parts of the car are implemented as simple non-intelligent agents as follows:

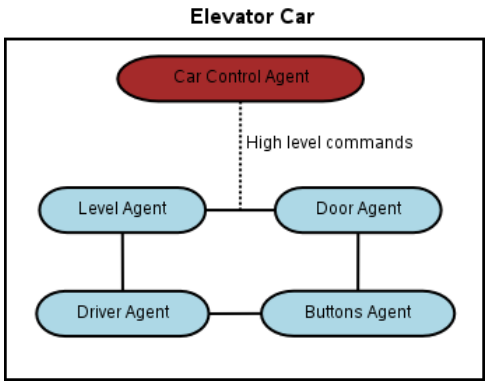

Fig. 3. Agent-based elevator car.

- **The Level agent**: represents the level sensor, which is responsible for monitoring the current level of the car.
- **The Driver agent**: represents the drive motor, which transports the car to the desired floor in the right direction.
- **The Door agent**: represents the elevator car's door.
- **The Buttons Panel agent**: represents the buttons panel inside the car, which is used by the passengers to register their desired destination floors.
- **The Elevator Control agent**: acts as an interface between the car and the dispatcher agent and it issues high level commands to the underlying agents. This agent controls the overall behavior of the car.

The main advantage of using multiple agents is distributivity, which distributes the work load and ensures decentralization. Also, this design approach would ease the integration of the simulator with a hardware model.

- **Dispatcher**: it's the most important component in the system. It keeps track of all issued hall requests and the status of each elevator car (position, direction and car requests) and it assigns cars to serve hall requests accordingly. The dispatcher is implemented as a single JADE agent and can either use the RL algorithm or a conventional technique (explained later) in the dispatching process based on the user's choice. For the RL dispatching algorithm the open-source package PIQLE (**P**latform **I**mplementing **Q**-**LE**arning) is used [12]. This package provides implementations for the Q-learning algorithm, neural networks and templates for environments, agents and actions. The agent, environment, states and actions are all custom implemented to suit the problem. The dispatcher contains four behaviors; two of them handling the two dispatching techniques. Also, it contains two classes representing the RL agent and environment.
  Fig. 4 shows the sequence diagram of a typical RL dispatching scenario.

B. Interface Linker

The Interface Linker is a JADE agent responsible for initializing the elevator, dispatcher and GUI components and handling communication between the GUI and all the other components. It updates the GUI based on commands from the system components and also updates them based on the GUI feedback. This is a special kind of JADE agents called *GuiAgent* which is capable of receiving events from GUI.

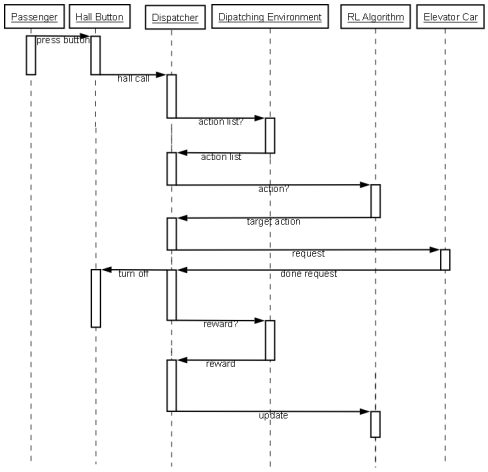

Fig. 4. RL dispatching sequence diagram.

Fig. 5. GUI.

C. Graphical User Interface

This layer is responsible for the interactive user interface of the simulator. It includes panels, frames, buttons, menus...etc. as shown in Fig. 5.

V. Experimental Test Cases

In this section statistical data will be presented demonstrating the RL dispatching algorithm behavior in response to variations in some of the major algorithm attributes. Moreover, the performance of the RL dispatching algorithm is compared to that of a conventional dispatching technique by varying some of the system's attributes. The average passenger waiting time is the main evaluation metric in all the test cases. For comparison purposes, an extra dispatching technique was

implemented which works by dispatching the elevator car that will take the least time to serve a certain passenger (least system time).

A. Number of Elevator Cars

In this section the effect of the number of elevator cars on the dispatching process is illustrated. The number of cars is variable from one up to four cars and the number of floors is kept constant (ten floors). The door timeout time was set to 5 seconds. The simulation was run twice for each of the RL dispatching algorithm and the conventional dispatching techniques. During a single run 60 passenger waiting time readings were obtained and divided into 3 clusters of 20 readings each. The average of each cluster is then calculated. In the simulation roughly the same conditions were applied for both the RL dispatching and the conventional dispatching where congestion along all the floors was simulated by multiple requests from all elevator cars and hall buttons as well. Fig. 6 shows the significant difference between the average passenger waiting times using the two techniques. When there is a single elevator car, the effectiveness of the RL algorithm is not illustrated and it yielded roughly the same results as the conventional technique. However, as the number of elevator cars increases, the RL algorithm produced the best results.

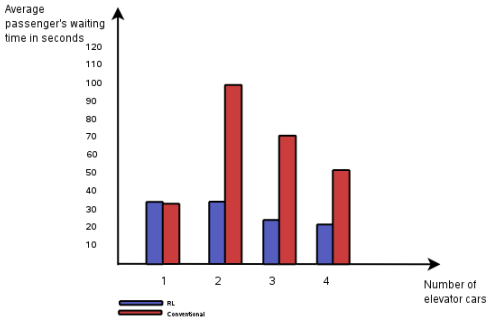

Fig. 6. Comparison between RL and conventional dispatching.

B. Reinforcement Learning attributes

In this section, the effects of the major RL attributes on the dispatching behavior will be illustrated. The test cases included are exclusive to the RL dispatching technique.

1) Alpha attribute: Recalling the Q-learning equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

α represents the learning rate of the algorithm, it normally fluctuates between 0 and 1 and has a significant effect on how fast the algorithm converges. In order to illustrate the effect of α, three different simulations were carried out with three different values of α. 40 readings were taken in each run and divided into two clusters. When α is 0.7 the average waiting times are high and the gab between the two clusters is small, indicating a slow learning rate. When α is 0.95, the average waiting times have significantly decreased and the gab increased, thus indicating a steep learning rate. It's concluded

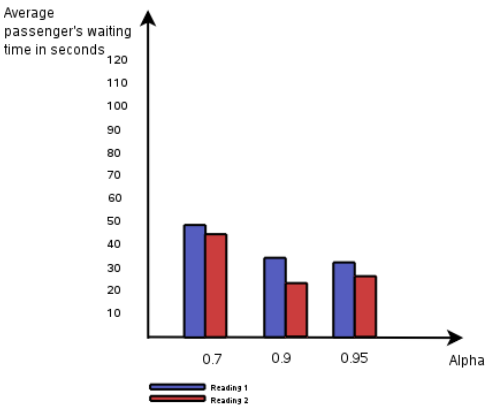

Fig. 7. Alpha effect.

that α can have a significant effect on the algorithm. Fig. 7 illustrates this effect.

2) Gamma attribute: In the Q-Learning algorithm, γ is the reward discounting rate, with each reward given the agent actually receives a discounted version of it. The purpose of discounted rewards is to motivate the agent to achieve higher returns. Like α, the value of γ can fluctuate between 0 and 1. Proper increase in γ enhance the learning rate and makes the agent more farsighted. Decreasing γ can halt the learning ability of the agent; to illustrate this effect a test similar to that made for the α was carried out with three clusters of readings. Fig. 8 illustrates the γ effect. Although the results are not very demonstrative due to some fluctuations, it's shown that when γ is low the algorithm doesnt behave optimally because the agent is aiming at maximizing the immediate rewards not the return (long term rewards).

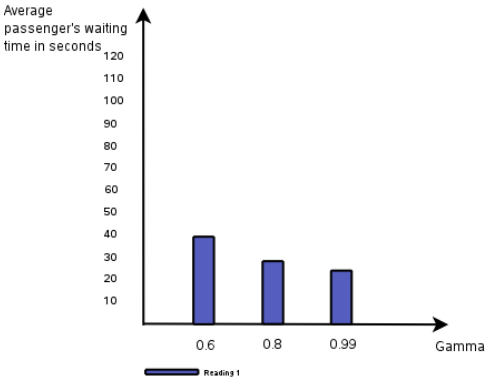

Fig. 8. Gamma effect.

3) Alpha Decay attribute: The α decay is a constant multiplied by α on each algorithm update. It constantly varies the value of α to establish some balance and randomness. A high α decay value will result in a slow decrease in α and vice versa. Test cases of this attribute yielded similar results as in the case of α.

VI. Conclusion

A multi-agent simulator for a four-elevator, ten floor elevator system has been developed. The multi-agent design proved to be very useful to cope with the problem of the complexity of the elevator system. The implemented machine learning-based dispatching algorithm proved to be working properly after several runs. In the future, it's planned to implement the complete version of the Q-learning algorithm and to implement other intelligent dispatching algorithms. Moreover, work is currently being done to integrate the simulator with the hardware model shown in Fig. 9.

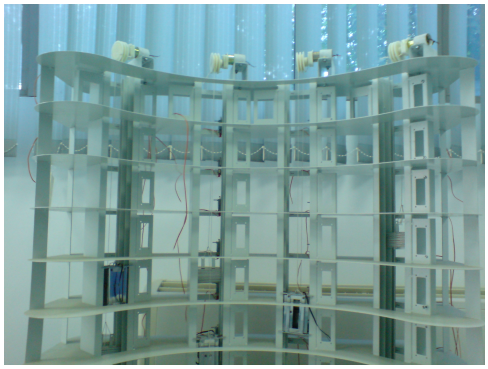

Fig. 9. Elevator system hardware platform.

References

[1] R. H. Crites and A. G. Barto, "Improving Elevator Performance using Reinforcement Learning". In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.). Advances in Neural Information Processing Systems. Proceedings of the 1995 Conference, pp. 1017-1023 MIT Press, Cambridge, MA.

[2] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2005.

[3] Stichweh, James H., Hornung, Stephen A., Duckwall III, Paul, "Elevator Control System with Priority Dispatching Capability". White CO Inc. US Patent no. 3645361, 1972.

[4] Powell, Bruce A. (Canton, CT), Williams, John N. (Coventry, CT), "Elevator Dispatching with Multiple Term Objective Function and Instantaneous Elevator Assignment". Otis Elevator Co. US Patent no. 5388668, 1995.

[5] Powell, Bruce A. (Canton, CT), Williams, John N. (Coventry, CT), "Elevator Dispatching based on Remaining Response Time". Otis Elevator Co. US Patent no. 5146053, 1992.

[6] Thangavelu, Kandasamy, "Artificial Intelligence-based learning system predicting "Peak-Period" times for elevator dispatching". Otis Elevator Co. US Patent no. 5035302, 1991.

[7] Nils J. Nilsson, *INTRODUCTION TO MACHINE LEARNING: AN EARLY DRAFT OF A PROPOSED TEXTBOOK*, December 4, 1996. Stanford University, Stanford, CA.

[8] Williams, David R. (Carlisle, MA, US), Hill, Jerey (Westford, MA, US), "Machine learning". US Patent no. 20050105712, 2005.

[9] K. Gurney, *An Introduction to Neural Networks*. UCL Press, Gunpowder Square, London EC4A 3DE, UK, 1996.

[10] Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood, *Developing Multi-Agent Systems with JADE*. ISBN: 978-0-470-05747-6, Wiley, April 2007.

[11] JADE: Java Agent DEvelopment Framework, http://jade.tilab.com.

[12] PIQLE: Platform Implementing Q-LEarning, http://sourceforge.net/projects/piqle.