

## Reinforcement Learning for Elevator Control\*

Xu Yuan \* Lucian Bușoniu \*\* Robert Babuška \*\*

\* ASML Netherlands B.V., The Netherlands  
(e-mail: xu.yuan@asml.com)

\*\* Delft University of Technology, The Netherlands  
(e-mail: i.l.busoniu@tudelft.nl, r.babuska@tudelft.nl)

**Abstract:** Reinforcement learning (RL) comprises an array of techniques that learn a control policy so as to maximize a reward signal. When applied to the control of elevator systems, RL has the potential of finding better control policies than classical heuristic, suboptimal policies. On the other hand, elevator systems offer an interesting benchmark application for the study of RL. In this paper, RL is applied to a single-elevator system. The mathematical model of the elevator system is described in detail, making the system easy to re-implement and re-use. An experimental comparison is made between the performance of the Q-value iteration and Q-learning RL algorithms, when applied to the elevator system.

Keywords: reinforcement learning; elevator control; reinforcement learning benchmark.

### 1. INTRODUCTION

Today's urban life cannot be imagined without elevators. A large number of modern elevator systems have been installed in most high-rise buildings. The elevator controllers assign elevators to service passengers requests in real-time while optimizing the overall service quality, e.g., by minimizing the waiting time and/or the energy consumption.

Classical examples of elevator control strategies include the collective control strategy (Siikonen, 1993) and the zone approaching strategy (Strakosz, 1983). With the collective control strategy, an elevator services the nearest call request in its current direction. However, with this control strategy several elevators sometimes stop at the same floor at the same time, and this increases the waiting time of passengers on other floors. With the zone approaching strategy, a building is divided into several vertical zones, and elevators are assigned to serve passengers in a particular zone. Elevators serve call requests within their predeterminate zones when they are available, and park there when they are idle. For the zone approaching strategy, the number of zones and the number of elevators within a zone are usually determined a-priori, which decreases the flexibility of this overall suboptimal control strategy.

In addition to being suboptimal, heuristic control strategies for elevator systems are difficult and costly to design. They can also become inappropriate if new traffic patterns arise that were not anticipated during the design process (Walczak and Cichosz, 2006).

In an attempt to address these drawbacks, the application of learning to elevator control was proposed (Crites and

Barto, 1998; Makaitis, 2003). Reinforcement learning (RL) in particular has been the focus of much research (Crites, 1996; Crites and Barto, 1998; Pepyne et al., 1996; Zhou et al., 2005; Walczak and Cichosz, 2006). Rather than relying on pre-designed control strategies, RL algorithms learn control policies (strategies) on the fly (Sutton and Barto, 1998; Kaelbling et al., 1996; Bertsekas, 2007). RL can be applied to general nonlinear, stochastic processes, and can obtain an optimal control policy without using a model of the controlled process. It is therefore useful when the control problem is complex or insufficiently known. For elevator systems, this means that reinforcement learning control can be applied without knowing in advance the traffic patterns (i.e., how the passengers will arrive). An optimal policy can be obtained that outperforms any heuristic strategy. Furthermore, to a certain extent the controller will be able to adapt on-line to time-varying traffic patterns.

Some authors regard each elevator as an agent that makes its own decisions autonomously, leading to a multi-agent view of the elevator system (Crites and Barto, 1998; Walczak and Cichosz, 2006). Using multi-agent learning for the elevator system can benefit from the advantages of distributed systems: scalability and robustness to single-point failures.

From a different perspective, elevator systems can serve as useful benchmark applications for the study of RL. Elevator control is well-suited for RL, because reward signals encoding relevant performance indices are easy to obtain (e.g., maximum or average waiting times), but a model of the task is difficult to derive, and an optimal control policy is unknown (most policies used in practice are heuristic). The passenger arrivals are best described as stochastic variables, and RL is suited for stochastic control tasks. The complexity of simulated elevator systems can be varied significantly. For instance, the number of state

\* This research was financially supported by Senter, Dutch Ministry of Economic Affairs within the BSIK-ICIS project "Interactive Collaborative Information Systems" (grant no. BSIK03024).

variables can range from a few to the order of tens; discrete variables (e.g., integer elevator positions), or continuous variables (e.g., waiting times) can be included. When the task is simple enough, exact RL algorithms can be applied, whereas more complex variations can be used to study approximate RL techniques. One or several elevators can be considered, and in the latter case, multi-agent RL algorithms can be employed.

In this paper, RL is applied to a single-elevator system. The mathematical model of the elevator system is described in detail, making the system easy to re-implement and re-use. This is in contrast to other works on RL for elevator systems, where the model is usually not fully described (Crites and Barto, 1998; Pepyne et al., 1996). In our experiments, we compare the performance of the control policy computed with the model-based Q-value iteration algorithm, with the performance of policies found by the model-free Q-learning algorithm.

The rest of this paper is organized as follows. In section 2, the necessary background on RL is introduced, including the model-based Q-value iteration algorithm and the model-free Q-learning algorithm. Section 3 describes in detail our elevator system model. Section 4 presents our RL experiments with the elevator system. Section 5 concludes and closes the paper.

## 2. REINFORCEMENT LEARNING

This section presents the necessary elements of reinforcement learning (RL). RL searches for a control policy (a mapping from states to control actions), so as to maximize a cumulative reward signal (Sutton and Barto, 1998; Kaelbling et al., 1996).

In RL, the learning controller (agent) interacts with the controlled process (environment) in discrete time. At time step  $t$ , the controller measures directly the state  $x_t \in X$  of the environment, and applies an action  $u_t \in U$  according to its policy,  $u_t = \pi(x_t)$ . The environment makes a transition to a new state, according to its stochastic dynamics: the probability of ending up in  $x_{t+1}$  as a result of action  $u_t$  in  $x_t$  is  $f(x_t, u_t, x_{t+1})$ . Simultaneously, the controller receives a scalar reward  $r_{t+1} = \rho(x_t, u_t, x_{t+1})$  that measures the quality of the state transition. The reward says nothing directly about the long-term performance or how it can be improved. Then the cycle repeats for the next sample  $t + 1$ , and so on.

The learning goal is to maximize the expected value of the return:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \quad (1)$$

where  $\gamma \in [0, 1]$  is the discount rate. The discount rate can be regarded as encoding uncertainty about future events. So, the goal of the agent is to maximize infinite-horizon returns, while only receiving feedback about immediate transitions.

This can be achieved by learning a state-action value function. The value of the state-action pair  $(x, u)$  under the policy  $\pi$ , denoted by  $Q^\pi(x, u)$ , represents the expected return when starting in state  $x$ , taking action  $u$  and following policy  $\pi$  thereafter:

Table 1. The Q-value iteration algorithm

<b>Input</b>	$f, \rho, \gamma, \text{convergence threshold } \theta$
Initialize $Q_0$ arbitrarily, e.g. $Q_0(x, u) = 0$ , for all $x \in X, u \in U$	
<b>Repeat</b>	$k = 0$
<b>For</b> each $x \in X, u \in U$	
$Q_{k+1}(x, u) =$	
$\sum_{x' \in X} f(x, u, x')[\rho(x, u, x') + \gamma \max_{u' \in U} Q_k(x', u')]$	
<b>EndFor</b>	$k = k + 1$
<b>Until</b> $\max_{x, u}  Q_k(x, u) - Q_{k-1}(x, u)  < \theta$	
<b>Output</b> $\pi^*(x) = \arg \max_{u \in U} Q_k(x, u) \quad \forall x \in X$	

$$Q^\pi(x, u) = E^\pi\{R_t|x_t = x, u_t = u\} \quad (2)$$

where  $E^\pi\{\cdot\}$  denotes expectation under the stochastic dynamics  $f$ , given that the controller uses policy  $\pi$ .

The optimal action value function  $Q^*$  is defined as the maximum Q-function over all the policies:

$$Q^*(x, u) = \max_\pi Q^\pi(x, u). \quad (3)$$

Once  $Q^*$  is known, an optimal policy (i.e., one that maximizes the return) can be found by a maximization over the action argument:

$$\pi^*(x) = \arg \max_{u \in U} Q^*(x, u). \quad (4)$$

This is the greedy policy in  $Q^*$ .

A central result in RL states that the optimal value function  $Q^*$  is the unique solution of the Bellman optimality equation:<sup>1</sup>

$$Q^*(x, u) = \sum_{x' \in X} f(x, u, x')[\rho(x, u, x') + \gamma \max_{u' \in U} Q^*(x', u')]. \quad (5)$$

The model-based Q-value iteration algorithm computes the right-hand side of the Bellman equation iteratively, starting from an arbitrary Q-function  $Q_0$ . The algorithm is given in Table 1. Although Q-value iteration has large computational expenses, works only offline and requires a model, it has some very useful properties. Namely, the algorithm is deterministic, has only two parameters to set ( $\gamma$  and  $\theta$ ), and ensures monotonous convergence to  $Q^*$  (Bertsekas, 2007).

Among the algorithms that do not require a model of the environment, one of the most widely used is Q-learning (Watkins and Dayan, 1992). The Q-learning algorithm iteratively estimates  $Q^*$  from interaction with the environment, using the update formula:

$$Q_{t+1}(x_t, u_t) = Q_t(x_t, u_t) + \alpha_t[r_{t+1} + \gamma \max_{u \in U} Q_t(x_{t+1}, u) - Q_t(x_t, u_t)] \quad (6)$$

Here,  $x_t$ ,  $x_{t+1}$ , and  $r_{t+1}$  are state and reward values observed by the controller while interacting with the environment, and  $\alpha_t$  is the learning rate at time  $t$ . The Q-learning algorithm is given in Table 2.

Under the following conditions, Q-learning converges to  $Q^*$  as  $t \rightarrow \infty$  (Watkins and Dayan, 1992; Jaakkola et al., 1994):

<sup>1</sup> A state space with a finite number of elements is assumed.

Table 2. The Q-learning algorithm

<b>Input</b>	$\gamma, \alpha_t$ , exploration parameters (e.g., $\varepsilon, \tau$ )
Initialize	$Q_0$ arbitrarily, e.g. $Q_0(x, u) = 0$ , for all $x \in X, u \in U$
Initialize	$x_0$
<b>Repeat</b> at each time step $t$ :	
Choose $u_t$ in $x_t$ using policy derived from $Q_t$ (e.g., Boltzmann)	
Apply $u_t$ , observe $r_{t+1}, x_{t+1}$	
$Q_{t+1}(x_t, u_t) = Q_t(x_t, u_t) +$	
$\alpha_t[r_{t+1} + \gamma \max_{u \in U} Q_t(x_{t+1}, u) - Q_t(x_t, u_t)]$	

- Explicit, distinct values of the Q-function are stored and updated for each state-action pair.
- The sum of the squares of  $\alpha_t$  is finite, whereas the sum of  $\alpha_t$  is infinite.
- The controller keeps trying all actions in all states with nonzero probability.

The last condition can be satisfied if the controller does not always choose actions that are greedy for the current Q-function, but also includes randomness in its action choice. This is called exploration. In this paper, the controller chooses at each step the greedy action (4) with probability  $(1 - \varepsilon)$  where  $\varepsilon \in (0, 1)$ , and with probability  $\varepsilon$  draws an action from the Boltzmann distribution:

$$p_t(x, u) = \frac{e^{Q_t(x, u)}/\tau}{\sum_{u' \in U} e^{Q_t(x, u')/\tau}} \quad (7)$$

where  $p_t(x, u)$  is the probability of selecting  $u$  in  $x$ , and  $\tau$  is called the temperature. The temperature typically decreases over time. When  $\tau \rightarrow 0$ , (7) is equivalent with greedy action selection (4). When  $\tau \rightarrow \infty$ , the action selection is purely random. For  $\tau \in (0, \infty)$ , higher-valued actions have a greater chance of being selected than lower-valued ones (Sutton and Barto, 1998).

To improve the learning speed of Q-learning, an eligibility trace can be used. The resulting algorithm, called  $Q(\lambda)$ , updates at each time step not only the Q-value of the last state-action pair, but also those of state-action pairs previously encountered, with weights that decay exponentially as the pairs go further back in time. The weights are called eligibility values (Peng and Williams, 1996; Sutton and Barto, 1998).

### 3. ELEVATOR SYSTEM MODEL

This section introduces the model of the elevator system considered. The model is parameterized by the following variables:

- The number of elevators (positive integer). In general, there are several elevators in elevator systems. In order to simplify the problem, we assume here that the system consists of a single elevator.
- The number of floors (positive integer). Set here to 5.
- The height of a floor (positive real). Set here to 6 m.
- The elevator speed (positive real). Here we set it to 3 m/s. This means the elevator takes 2 s to travel between two adjacent floors.
- The elevator capacity (positive integer). Set here to 4 passengers.
- The stop time, i.e., the sum of the intervals needed by the passengers to enter and exit the elevator on a floor. The stop time is set to 2 s. The fact that the

stop time is equal to the floor time ensures intervals between decisions made by the controller are always the same. This means that the elevator system can be modeled as a discrete-time system with the sample time of  $T_s = 2$  s.

The size of the state space can be varied by changing the number of elevators, the number of floors, and the elevator capacity. The particular numbers chosen here ensure that the cardinality of the state space is finite and relatively small, allowing the direct application of RL algorithms without resorting to less-understood, approximate versions of these algorithms.

Our model contains a few simplifications with respect to a real elevator system. We assume that at most one passenger is waiting on each floor. In other words, if one passenger arrives on a floor where there is already another waiting passenger, the number of waiting passengers on that floor is still viewed to be one. The assumption of having at most one waiting passenger per floor makes it reasonable to set the elevator capacity of 4.

In the sequel, a down-peak traffic pattern is assumed. This pattern occurs in office buildings in the afternoon, when people are leaving work for home. In down-peak traffic, passengers have many departure floors and a single destination floor, the ground floor. At each sample time, passengers arrive probabilistically in the following way. A sample is drawn from a discrete probability distribution with support  $\{0, 1, 2, 3, 4\}$ , with the corresponding probabilities of  $\{0.6875, 0.0625, 0.09375, 0.09375, 0.0625\}$ . The event  $e = 0$  means no passenger arrives; an event  $e > 0$  means a passenger arrives at floor  $i = e$ . It is easy to compute that the average passenger arrival rate is 1 person every 6.4 seconds. Given this rate, the elevator is capable to serve all the passengers.

#### 3.1 State-Action Space and Performance Measures

The state space of the elevator system is discrete and has 7 dimensions. The state signal  $x$  is composed of the following variables:

$$x = [c_1, c_2, c_3, c_4, p, v, o]^T. \quad (8)$$

where:

- $c_i, i = 1, 2, 3, 4$ : Binary values, representing call requests (call flags) on each floor  $i$ . There is no call request on the ground floor in the down-peak traffic scenario.
- $p$ : Discrete elevator position, taking values in  $\{0, 1, 2, 3, 4\}$ .
- $v$ : Discrete vertical velocity taking values in  $\{-3, 0, 3\}$  m/s.
- $o$ : Discrete elevator occupancy, taking values in  $\{0, 1, 2, 3, 4\}$ . The number 0 means no passengers are inside the elevator; the number 4 means that the elevator is at its maximum capacity.

The cardinality of the state space is:

$$2^4 \cdot 5 \cdot 3 \cdot 5 = 1200 \quad (9)$$

The elevator controller can choose among three discrete actions:  $\{-1, 0, 1\}$ . The  $-1$  action accelerates the elevator downwards, 1 accelerates it upwards, and 0 stops the

elevator. Two constraints are enforced on actions, corresponding to physical constraints in the elevator system.

- (1) The controller cannot choose the action 1 when the elevator is on the top floor. Similarly, it cannot choose the action -1 when the elevator is on the ground floor.
- (2) The elevator cannot switch directions during a single sample. This means the 0 action must be taken between the 1 and -1 actions.

The reward function is:

$$\rho(x) = - \sum_{i=1}^4 c_i - o \quad (10)$$

where  $c_i$  is the call request on the  $i$ -th floor;  $o$  represents the elevator occupancy. The reward at state  $x_t$  is the negative of the number of all the passengers waiting at time  $t$ , including passengers waiting for the elevator on the floors, and passengers inside the elevator waiting to exit on the ground floor. In other words, the only situation for which the reward is 0 is when no passengers are waiting in the system. The optimal policy will attempt to drive the system into this situation in minimum time, thereby transporting passengers to their destination in minimum time.

The controller efficiency is evaluated through the waiting time of the passengers. One passenger's waiting time is the time since she arrives on some floor until she reaches her destination, including the time spent waiting for the elevator at the departure floor, and the time spent inside the elevator waiting to exit at the destination floor. In the sequel, the passengers' waiting time is used to compute two performance measures:

- The *average waiting time*, an instantaneous quantity computed as the average of the waiting time over all the passengers currently in the system.
- The *trial waiting time*, which characterizes an entire run (trial) of the simulator. This is an average over all the time samples in the trial of the (instantaneous) average waiting time defined above. When computing this quantity, it is assumed that the duration of the trial is finite.

In both cases, a smaller value of the performance measure indicates a better performance of the controller.

### 3.2 Elevator System Dynamics

**Elevator motion.** The elevator spends one sampling interval (2 seconds) from one floor to the next. This period is divided into 10 equal time intervals, denoted by the integer numbers  $h$  ( $h = 1, 2, \dots, 10$ ), during which the elevator moves in small increments between two adjacent floors.

- When  $h = 1$ , elevator's velocity  $v$  changes according to the chosen action  $u$ :

$$v = 3 \cdot u \quad (11)$$

- At each time interval  $h$ , the elevator position  $y$  between two floors is calculated as:

$$y = v \frac{h}{10} T_s \quad (12)$$

This can be regarded as an Euler integration of the (constant) velocity  $v$  with a time step of  $T_s/10$ .

- When  $h = 10$ , the elevator floor position changes as:

$$p \leftarrow p + \frac{y}{6} \quad (13)$$

where the height between floors is 6 m.

**Passenger arrival and departure.** In down-peak traffic, passengers exit the elevator only at the ground floor, and can enter the elevator at any non-zero floor, as long as the total number of passengers in the elevator remains within its capacity.

- When  $h = 0$ , an event  $e$  is drawn from the passenger arrival distribution, and the call request flag  $c_i$  on each floor  $i$  changes as follows:

$$c_i \leftarrow \begin{cases} 1 & \text{if } i = e \\ c_i & \text{otherwise} \end{cases} \quad (14)$$

- When  $h = 10$ , the call request flag changes as follows:

$$c_i \leftarrow \begin{cases} 0 & \text{if } i = p, v = 0, \text{ and } o < 4 \\ c_i & \text{otherwise} \end{cases} \quad (15)$$

where  $o$  is the elevator occupancy. The occupancy changes as:

$$o \leftarrow \begin{cases} o + 1 & \text{if } p > 0, v = 0, c_p = 1, \text{ and } o < 4 \\ 0 & \text{if } p = 0 \text{ and } v = 0 \\ o & \text{otherwise} \end{cases} \quad (16)$$

The elevator system simulator was implemented in Matlab. Matlab was chosen because it offers powerful tools for rapid development and analysis. A screenshot of the graphical user interface (GUI) of the developed simulator is shown in Figure 1.

## 4. EXPERIMENT: Q-VALUE ITERATION AND Q-LEARNING

In this section, we apply the model-based Q-value iteration algorithm and the model-free Q-learning algorithm to the elevator system in down-peak traffic. The control policies computed with the two algorithms are compared using the average waiting time and the trial waiting time, which were introduced in Section 3.1. Each trial has a duration of 500 seconds.

The following heuristic policy is designed as a baseline. The controller randomly chooses a waiting passenger, and takes that passenger to the ground floor without stopping at any non-zero floor. This baseline policy yields trial waiting times of up to 70 seconds.

In order to run the Q-value iteration algorithm (Table 2), the transition probability function  $f$  was first computed. The discount rate was set to  $\gamma = 0.99$ , and the convergence threshold  $\theta = 0.01$ . This optimal solution has a trial waiting time of 5.13 seconds (computed as an average of 50 experiments, to account for the stochastic passenger arrivals).

The online Q-learning algorithm was run with the same discount rate, and a constant learning rate  $\alpha = 0.38$ . Boltzmann exploration and an eligibility trace were used, as described in Section 2. The exploration probability was set to  $\varepsilon = 0.8$  in the first trial, and was annealed

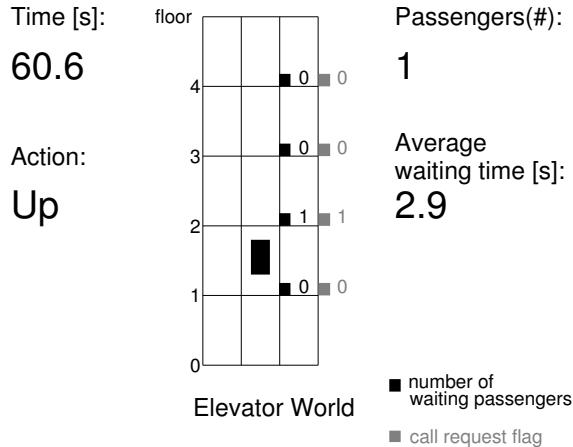


Fig. 1. Graphical user interface (GUI) of the elevator system. On the left, the simulation time is displayed, together with the current action chosen by the controller. On the right, the number of passengers in the elevator and the average waiting time is displayed. Beside each black square is the number of waiting passengers on that floor. Beside each gray square, the call request flag at that floor is given. The large black rectangle represents the position of the elevator.

Table 3. Parameter settings for the Q-learning controller.

Parameters	Value
Number of trials	50
Trial length	500 s
Learning rate $\alpha$	0.38
Discount rate $\gamma$	0.99
Initial exploration probability $\varepsilon$	0.8
Exploration probability annealing $\varepsilon_d$ (per trial)	0.89
Initial exploration temperature $\tau$	11.8
Temperature annealing $\tau_d$ (per time step)	0.998
Eligibility trace annealing $\lambda$	0.68

exponentially at the end of every trial using the annealing factor  $\varepsilon_d = 0.89$ :

$$\varepsilon \leftarrow \varepsilon_d \cdot \varepsilon \quad (17)$$

The exploration temperature in (7) was set to an initial value  $\tau = 11.8$  and was annealed at each sample time with a factor  $\tau_d = 0.998$ , using a formula similar to (17). All the parameters of the Q-learning algorithm are listed in Table 3, together with their values.

Each Q-learning experiment consists of 50 trials during which a control policy is learned. The trial waiting time of the passengers is recorded. A number of 50 such experiments are performed and the results are averaged, to account for the stochastic passenger arrivals. The results are plotted in Fig. 2. The evolution of the trial waiting time is recorded against the number of trials. The learning curve eventually converges to approximately 6 s, from the initial

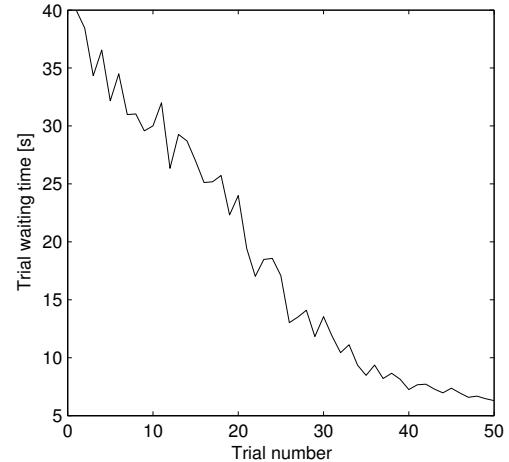


Fig. 2. Evolution of the trial waiting time with Q-learning control. The curve represents the average of 50 independent experiments.

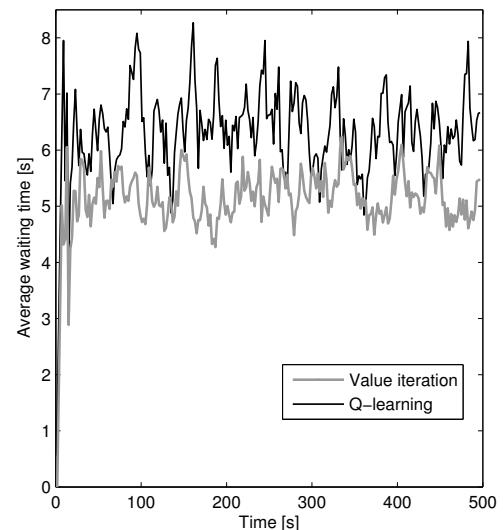


Fig. 3. Average waiting time comparison between the Q-value iteration policy and the Q-learning policy, for a typical single trial.

value of 40 s. The curve does not monotonously decrease because of exploration during the learning process.

It can be seen from these experiments that both Q-value iteration and Q-learning give much better performances than the baseline policy. After 50 trials, Q-learning gives a good performance with a trial waiting time of 6.32 seconds, larger than the optimal performance obtained with Q-value iteration but still close. The reason for the difference is that infinite exploration, one of the fundamental requirements for Q-learning convergence, cannot be satisfied in a finite learning process. Figure 3 also shows the evolution of the average waiting time over a typical single trial, comparing the policy computed with Q-value iteration with a policy computed with Q-learning.

## 5. CONCLUSION

In this paper, we have applied two RL algorithms, Q-value iteration and Q-learning, to a simulated elevator system. The mathematical model of the elevator system was described in detail. In the experiment, the elevator system proved to be a suitable RL benchmark. The elevator system provided a discrete state space with 1200 states, which could be handled both by Q-value iteration and Q-learning. Both algorithms produced good solutions that ensured low waiting times for the passengers, with the Q-learning solution slightly worse than the (optimal) Q-value iteration solution.

A useful direction for future work is the investigation of more realistic elevator system models. For instance, more than one elevator can be included in the system, and more realistic passenger arrival patterns can be considered. Simulating multiple elevators can be useful in studying multi-agent reinforcement learning algorithms.

## REFERENCES

- D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition, 2007.
- R. Crites. *Large-scale dynamic optimization using teams of reinforcement learning agents*. PhD thesis, University of Massachusetts Amherst, 1996.
- R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2–3):235–262, 1998.
- T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.
- L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- D. Makaitis. Evolving fuzzy controllers through evolutionary programming. In *Proceedings 22nd International Conference of the North American Fuzzy Information Processing Society (NAFIPS-03)*, pages 50–54, Chicago, US, 24–26 July 2003.
- J. Peng and R.J. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22:283–290, 1996.
- D.L. Pepyne, D.P. Looze, C.G. Cassandras, and T.E. Djaferis. Application of Q-learning to elevator dispatching. In *Proceedings 13th IFAC World Congress*, pages 317–322, San Francisco, US, 1996.
- M.L. Siikonen. Elevator traffic simulation. *Simulation*, 61: 257–267, 1993.
- G.R. Strakosch. *Vertical Transportation: Elevators and Escalators*. Wiley and Sons, 1983.
- R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- T. Walczak and P. Cichosz. A distributed learning control system for elevator groups. In J.G. Carbonell and J. Siekmann, editors, *Artificial Intelligence and Soft Computing (ICAISC-06)*, volume 4029 of *Lecture Notes in Computer Science*, pages 1223–1232. Springer, 2006.
- C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
- J. Zhou, T. Eguchi, K. Hirasawa, J. Hu, and S. Markon. Elevator group supervisory control system using genetic network programming with reinforcement learning. In

*Proceedings 2005 IEEE Congress on Evolutionary Computation*, pages 336–342, Edinburgh, UK, 2–5 September 2005.