

Optimizing Elevator Control with Reinforcement Learning

DRAFT

Omar Elbaghdadi

A thesis presented for the degree of
Bachelor Econometrics and Operations Research



School of Business and Economics
Vrije Universiteit Amsterdam
The Netherlands
May 25, 2018

Thesis Title

Thesis Subtitle

Author Name

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1	Reinforcement Learning	3
2	Model of the System	4
2.1	System Dynamics	4
2.2	Traffic and Passenger Arrival	5
2.3	State Space	5
2.4	Action Set	7
2.5	Performance Measures and Reward	8
3	The Algorithm	9
3.1	Q -learning	9
3.1.1	Function Updates	10
3.1.2	Calculating Omniscient Reinforcements	10
3.1.3	Making Decisions and Updating Q -Values	11

2 Model of the System

This section introduces the model of the elevator system studied here. The modelling approach is similar to the one used by [7, 4]. We use this model in all further analysis.

2.1 System Dynamics

The system dynamics are parameterized as follows:

- Number of floors: ?.
- Number of elevator cars: 1 **for now?**. We consider only a single elevator to simplify the problem.
- Elevator floor time (time it takes for an elevator to pass a floor at full speed): 1.45 seconds.
- Elevator stop time (time it takes for the elevator to decelerate, open and close the doors, and accelerate again): 7.19 seconds.
- Elevator load time (the time it takes for a passenger to enter or exit the elevator): random variable from a 20th order truncated Erlang distribution with a range from 0.6 to 6 seconds and a mean of 1 second.
- Capacity of the cars: 20 passengers.

Passengers can arrive on any floor, where they can press either the up or down button. We call this a hall call. After an elevator arrives at the passenger's floor, they get in and press a button to a floor in the same direction as the elevator. We call this a car call.

An elevator system can be very complex, with events such as passenger and elevator arrivals happening asynchronously. Even though time is continuous, it is still possible to model the elevator system as a discrete event system, where significant events happen at discrete times, with the time between them taking on a continuous value. In this case, the constant discount factor γ used in most discrete-time reinforcement learning algorithms is inappropriate. Instead, we take into account the amount of time between events using a time varying discount factor [1]. This is also done to take into account the asynchronicity of events that is well suited to the elevator model. We define returns as the total discounted reward received over time:

$$\int_0^{\infty} e^{-\beta\tau} r_{\tau} d\tau \quad (1)$$

where we integrate over continuous time, instead of the usual discrete-time reinforcement:

$$\sum_{t=0}^{\infty} \gamma^t r_t.$$

Here, r_t is the immediate cost in the system at discrete time t , r_{τ} is the instantaneous cost at continuous time τ and $\beta > 0$ is a parameter used to control the rate of exponential decay. A system with a higher β cares more about rewards that are closer in time. The environment is thusly modeled as a discrete-time Markov Decision Process with a finite number of states and actions. Using this approach, the states and actions can be fully specified.

The model parameters have been chosen such that the state space is finite and reasonably sized. This is of great importance when using algorithms in which it is necessary to store values for each state, so called tabular methods. If the state space is too large, running time and memory capacity can become intractable. We would need to approximate the value functions.

It is not possible to observe the full state of the system. After a button is pressed, the elevator controller does not know if another passenger arrived after the first one. One way to deal with this is assuming *omniscience*. We assume that in every state, the elevator controller knows how many passengers are waiting at a floor, when they arrived and where they are going. This can then be used to calculate the reinforcement signals. An important thing to note here is that it is not the controller that is receiving this extra information, it is the critic *evaluating* the controller. This information is only important in the training phase. Once the controller is trained, it can be implemented in a real system without needing the extra knowledge.

Another possibility is to let the system learn using only information that would be available to the system on-line. The arrival times of passengers after the first would have to be estimated, so that expected costs can still be computed. We use omniscient reinforcements, since they are slightly more accurate and results will not differ by a large margin [3].

2.2 Traffic and Passenger Arrival

It is important to take into account the traffic profile at a time. General building traffic profiles have been identified [5]. Four important profiles are up-peak, down-peak, inter-floor, and lunchtime. We will concern ourselves only with the down-peak traffic profile. Down-peak is a traffic pattern in which passengers are primarily moving down to the ground floor. An example is people going home at the end of a business day in an office building. **We will assume every arriving passenger wants to go to the ground floor. (maybe)**

We model the arrival of passengers as a Poisson process with rate parameter λ being the expected number of people arriving at a floor each minute. The rate can vary across floors **and time (maybe)**. We set it to λ . For every floor, passenger interarrival time is drawn from an exponential distribution with rate λ .

2.3 State Space

An elevator controller maps state information to decisions. The definition of the state is therefore of great importance. It needs to properly reflect the current state of the system, measured by quantities that can be practically observed. This includes:

- the state of hall calls and their waiting times,
- awaiting car calls and their waiting times for all elevators,
- position of all elevators,
- moving direction of all elevators,
- velocity of all elevators,
- calls to which particular elevators are allocated.

If we assume the state is defined as above, it is possible to estimate the size of the resulting state space. Let n be the number of floors and m the number of elevators **to notation?**, the size of the state space can be estimated as follows:

- at most $n - 1$ up hall calls: 2^{n-1} possibilities,
- at most $n - 1$ down hall calls: 2^{n-1} possibilities,
- at most n car calls for each elevator: 2^n possibilities,
- possible locations for each elevator: n possibilities,
- possible elevator directions for each elevator: 3 possibilities (up, down or stopped),

which gives a state space size of $2^{n-1} \cdot 2^{n-1} \cdot 2^n \cdot n \cdot 3$ in a situation where each elevator is considered as a separate learner. The amount of states rises tremendously in the number of floors. The problem already gets computationally intractable for a relatively few number of floors. To combat this and further reduce the size of the state space, we aggregate the states into a more compact representation, while hopefully keeping the most relevant information. Instead of receiving information regarding exactly which hall and car calls are active, the controller only receives:

- the number of up and down awaiting hall calls higher and lower than the elevator's current position,
- the number of car calls to floors in the current moving direction.

Taking into account this particular aggregation of states, we can recalculate the size of the state space as follows:

- the number of remaining **up** hall calls from floors **higher** than the current position: at most $n - 1$ possibilities,
- the number of remaining **down** hall calls from floors **higher** than the current position: at most $n - 1$ possibilities,
- the number of remaining **up** hall calls from floors **lower** than the current position: at most $n - 1$ possibilities,
- the number of remaining **down** hall calls from floors **lower** than the current position: at most $n - 1$ possibilities,
- the number of remaining car calls in the current moving direction: at most $n - 1$ possibilities,
- the current position: n possibilities,
- the current moving direction: 3 possibilities,

which make for a total of $(n-1)^4 \cdot (n-1) \cdot n \cdot 3$ possible states. This is a vastly smaller amount than before the aggregation. In fact, it is not exponential in n anymore. Such a size would allow for us to use tabular methods instead of value function approximators, given moderate sizes of n and m . Note also that a large subset of states will (almost) never be visited under normal conditions, reducing the effective state space size even further. **probabilistic analysis number of states visited on average**

2.4 Action Set

Now that we have the state space defined, we can move on to defining the actions we can take. What action the elevator is able to take will depend on the state of the system.

We define the actions as follows:

- If the elevator is moving, it can either
 - stop at the next floor, or
 - continue past the next floor.
- If the elevator is not moving, it can either:
 - go up, or
 - go down.

The system considered is event-based. There are two main types of events. Events of the first type concern waiting times, such as passenger arrivals and transfers in and out of cars. Events of the second type are car arrival events, which are potential decision points for the elevator controllers. A car moving between floors generates a car arrival event when it reaches a point at which it must decide whether to stop at the next floor, or continue past it. Sometimes, the controlling agent is restricted to take a certain action. If a passenger wants to get off at the next floor, it has to stop at the next floor. A potential decision point is only considered a decision point if the choice of action is not constrained.

This approach equals that of [4, 3], but differs from the approach used in [7]. They suggest that instead of using the 2 aforementioned actions, an elevator chooses a floor to serve and commits to that action, under a couple of other constraints. This has wider applicability to practical elevator systems, where the stopping distance for an elevator moving at full speed is longer than half of the distance between floors. Another advantage is the ability to announce the arrival of the car several seconds beforehand. Passengers are given more time to go towards the entrance of the elevator. Although it has practical advantages, it makes the learning task more difficult. Instead of 2 possible actions, at most n actions can be taken at each state.

There are additional restrictions on what actions can be taken:

- When at the bottom and top floors, we can not choose the action go down and go up respectively. We cannot continue past the bottom and top floors.
- We can not turn in a single action. If the current direction is up, for example, we have to first stop the elevator before we can go down.
- A car cannot turn until it has served all the calls in its current direction.
- A car can stop at a floor only if there is a call from this floor or there is a car call to this floor.
- A car cannot stop to pick up passengers at a floor if another car is already stopped there.
- Given a choice between moving up and down, it should prefer to move up (since the down-peak traffic tends to push the cars toward the bottom of the building).

2.5 Performance Measures and Reward

We need a way to measure the performance of the method we are applying. The goal is to minimize some function of passengers' waiting time. We consider the average passenger waiting time, which is generally considered a primary objective [5]. The waiting time of a passenger is defined as the time between the passenger's arrival at the floor and the passenger's entry into a car. Other possible performance measures are system time and the fraction of passengers waiting more than T seconds, where T is typically 60. System time is defined as the waiting time combined with the passenger's travel time.

In a Reinforcement Learning setting, reinforcement is usually formulated as maximizing some reward. In this case, however, it is more convenient to talk about minimizing costs of some sort. We want to define these costs such that minimizing them will lead to a lower average waiting time. The cost function is based on the definition of a reinforcement for an elevator system as in [3, 4].

Since time between events is continuous, we consider instantaneous costs at continuous time moments. For $\tau \in [t_1, t_2]$, where t_1 is the time of taking an action for an elevator (i.e. deciding to continue past the next floor or stop) and t_2 is the time when the next decision is required, the reinforcement r_τ is defined as follows:

$$r_\tau = \sum_p (\tau - t_1 + w_p)^2 \quad (2)$$

where w_p is the amount of time each passenger p waiting at time t_2 has already waited at time t_1 . Special care is needed to handle any passengers that begin or end waiting between t_1 and t_2 . See Section ??

3 The Algorithm

There are various methods and philosophies to solving Reinforcement Learning problems. The basic idea for most algorithms is to estimate reward values for states. These values are then used to make sure we take actions so that we stay in high-value states. Q is a function that maps state-action pairs (s, a) to a numerical value. This value represents the expected total discounted return after taking action a in state s . These values are then used to approximate the optimal policy π_* .

3.1 Q -learning

We use the Q -learning algorithm [2], which can be categorized as an off-policy Temporal Difference (TD) control method [6], where the Q -update is generally defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (3)$$

$$= (1 - \alpha)Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right]. \quad (4)$$

It is, in essence, a stochastic approximation to the Bellman Optimality equations. The advantage of this method is that we do not need to fully specify a model of the environment. We do not need to know, for instance, state transition probabilities and the distribution of reinforcements. These can be difficult to specify in an environment as complex as the elevator system. Instead, we use a simulator to generate episodes.

Q -learning is an off-policy method, because it does not necessarily learn the same policy that is being followed. The learned action-value function, Q , directly approximates the optimal action-value function q_* , independent of the policy being followed. The policy being followed still determines which state-action pairs visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters α_t , Q has been shown to converge to q_* with probability 1.

The Q -learning algorithm is given below.

MAKE NICER, INCLUDE REFERENCE TO EARLIER FORMULA

Algorithm 1 Q -learning (off-policy TD control) for estimating $\pi \approx \pi_*$

- 1: Initialize $Q(s, a)$, for all $s \in S, a \in A(s)$, arbitrarily, and $Q(\text{terminal} - \text{state}, \cdot) = 0$
 - 2: **repeat**
 - 3: Initialize S
 - 4: **repeat**
 - 5: Choose A from S using policy derived from Q (e.g. ϵ -greedy)
 - 6: Take action A , observe R, S'
 - 7: $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) \right]$
 - 8: $S \leftarrow S'$
 - 9: **until** (S is terminal)
 - 10: **until** episodes ends
-

3.1.1 Function Updates

We need to modify Algorithm (??) to meet our requirements. We now minimize over costs instead of maximizing over rewards. Since we are acting in continuous time, it is necessary to change the Q function update rule described in (4). For a time interval $[t_1, t_2]$ as considered above, the update rule for the Q -value for state s_{t_1} and action a_{t_1} is given at time t_2 by

$$Q(s_{t_1}, a_{t_1}) \leftarrow (1 - \alpha)Q(s_{t_1}, a_{t_1}) + \alpha \left[r_{[t_1, t_2]} + e^{-\beta(t_2 - t_1)} \min_a Q(s_{t_1}, a) \right] \quad (5)$$

where the total discounted cost for time interval $[t_1, t_2]$, $r_{[t_1, t_2]}$, is given by

$$\int_{t_1}^{t_2} e^{-\beta(\tau - t_1)} \sum_p (\tau - t_1 + w_p)^2 d\tau. \quad (6)$$

which can be solved by parts to yield

$$\sum_p e^{-\beta w_p} \left[\frac{2}{\beta^3} + \frac{2w_p}{\beta^2} + \frac{w_p^2}{\beta} \right] - e^{-\beta(w_p + t_2 - t_1)} \left[\frac{2}{\beta^3} + \frac{2(w_p + t_2 - t_1)}{\beta^2} + \frac{(w_p + t_2 - t_1)^2}{\beta} \right]. \quad (7)$$

3.1.2 Calculating Omniscient Reinforcements

We have to be careful in doing the accounting of reinforcements.

REPHRASE

In the omniscient reinforcement scheme, which we are using, accumulated cost for each car is updated after every passenger arrival event (when a passenger arrives at a floor), passenger transfer event (when a passenger gets on or off of a car), and car arrival event (when a control decision is made). The amount of cost accumulated between passenger arrival events is the same for all cars since they share the same objective function, but the amount of cost each car accumulates between its decisions is different since the cars make their decisions at different times. Therefore, each car i has an associated storage location, $R[i]$, where the total discounted cost it has incurred since its last decision (at time $d[i]$) is accumulated. At the time of each event, the following computations are performed: Let t_0 be the time of the last event and t_1 the time of the current event. For each passenger p that has been waiting between t_0 and t_1 , let $w_0(p)$ and $w_1(p)$ be the total time that passenger p has waited at t_0 and t_1 respectively. Then for each car i ,

$$\Delta R[i] = \sum_p e^{-\beta(t_0 - d[i])} \left[\frac{2}{\beta^3} + \frac{2w_0(p)}{\beta^2} + \frac{w_0^2(p)}{\beta} \right] - e^{-\beta(t_1 - d[i])} \left[\frac{2}{\beta^3} + \frac{2w_1(p)}{\beta^2} + \frac{w_1^2(p)}{\beta} \right]. \quad (8)$$

3.1.3 Making Decisions and Updating Q-Values

REPHRASE A car moving between floors generates a car arrival event when it reaches the point at which it must decide whether to stop at the next floor or to continue past the next floor. In some cases, cars are constrained to take a particular action, for example, stopping at the next floor if a passenger wants to get off there. An agent faces a decision point only when it has an unconstrained choice of actions. The algorithm used by each agent for making decisions and updating its Q-value estimates is as follows:

1. At time t_1 , observing state s_{t_1} , car i arrives at a decision point. It selects an action a using the Boltzmann distribution over its Q-value estimates:

$$P(\text{stop}) = \frac{e^{Q(x, \text{cont})/T}}{e^{Q(x, \text{cont})/T} + e^{Q(x, \text{stop})/T}}$$

where T is a positive “temperature” parameter that is “annealed” (decreased) during learning. The value of T controls the amount of randomness in the selection of actions. At the beginning of learning, when the Q-value estimates are very inaccurate, high values of T are used, which give nearly equal probabilities to each action. Later in learning, when the Q-value estimates are more accurate, lower values of T are used, which give higher probabilities to actions that are thought to be superior, while still allowing some exploration to gather more information about the other actions. Choosing a slow enough annealing schedule is particularly important in multi-agent settings.

2. Let the next decision point for car i be at time t_2 in state s_{t_2} . After all cars (including car i) have updated their $R[\cdot]$ values as described above, car i adjusts its estimate of $Q(s_{t_1}, a)$ toward the following target value:

$$R[i] + e^{-\beta(t_2-t_1)} \min_{\{\text{stop}, \text{cont}\}} Q(s_{t_2}, \cdot).$$

3. Let $s_{t_1} \leftarrow s_{t_2}$ and $t_1 \leftarrow t_2$. Go to step 1.

References

- [1] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *NIPS*, pages 393–400. MIT Press, 1994.
- [2] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King’s College, Cambridge, 1989.
- [3] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235, 1998.
- [4] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. April 16 1998.
- [5] James Lewis. A dynamic load balancing approach to the control of multiserver polling systems with applications to elevator system dispatching. 1991.
- [6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, 2012.
- [7] Tomasz Walczak and Pawel Cichosz. A distributed learning control system for elevator groups. In Leszek Rutkowski, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing - ICAISC 2006, 8th International Conference, Zakopane, Poland, June 25-29, 2006, Proceedings*, volume 4029 of *Lecture Notes in Computer Science*, pages 1223–1232. Springer, 2006.