DEGREE PROJECT, IN COMPUTER SCIENCE , FIRST LEVEL
STOCKHOLM, SWEDEN 2015

Elevator Control Using Reinforcement Learning to Select Strategy

ANTON JANSSON
KRISTOFFER UGGLA LINGVALL

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

KTH Computer Science
and Communication

Elevator Control Using Reinforcement Learning to Select Strategy

ANTON JANSSON
KRISTOFFER UGGLA LINGVALL

Degree Project in Computer Science, DD143X
Supervisor: Vahid Mosavat
Examiner: Örjan Ekeberg

Stockholm, Sweden 2015

Abstract

In this thesis, we investigated if reinforcement learning could be applied on elevator systems to improve performance. The performance was evaluated by the average squared waiting time for the passengers, and the buildings considered were apartment buildings.

The problem of scheduling elevator cars is an NP-hard problem, and no optimal solution is known. Therefore, an approach where the system learns a strategy instead of using a heuristic, should be the easiest way to get near an optimal solution.

A learning system was constructed, where the system was trained to use the best scheduling algorithm out of five in a given situation, based on the prevailing traffic. The purpose of this approach was to reduce the training time that was required in order to get good performance and to lower the complexity of the system.

A simulator was then developed, in which the different algorithms was implemented and tested in four different scenarios, where the size of the building and the number of elevator cars varied. The results generated by the simulator showed that reinforcement learning is a great strategy to use in buildings with 16 floors and three or four elevator cars. However, reinforcement learning did not increase the performance in buildings with 10 floors and two to three elevator cars. A possible reason for this is that the variation in performance between the different scheduling algorithms was too small in these scenarios.

Sammanfattning

Hisschemaläggning där reinforcement learning väljer strategi

I denna rapport har vi undersökt huruvida reinforcement learning är användbart för att öka prestandan för hissystem i lägenhetshus. Prestandan bedömdes efter de genomsnittliga kvadrerade väntetiderna för resenärerna.

Schemaläggningsproblemet för hissar är NP-svårt och ingen optimal lösning är känd. Att lösa problemet med hjälp av ett system som lär sig hur det ska agera, bör således vara en enklare strategi för att komma nära den optimala lösningen än att använda sig av en heuristik.

Ett självlärande system konstruerades, där systemet tränades att använda den bäst lämpade schemaläggningsalgoritmen med avseende på rådande trafikförhållanden. Det fanns totalt fem olika algoritmer att välja bland. Anledningen till att detta gjordes i stället för att systemet skulle lära sig en komplett strategi, var för att sänka träningstiden som krävdes för att åstadkomma bra resultat och för att minska komplexiteten.

En simulator utvecklades sedan, där de olika algoritmerna implementerades och testades på fyra olika scenarion, där storleken på byggnaden och antalet hissar varierade. Resultaten som genererades visade att reinforcement learning fungerar utmärkt på byggnader med 16 våningar och tre eller fyra hissar. På byggnader med tio våningar och två till tre hissar är det dock inte lika användbart och där bör i stället enklare algoritmer användas. En möjlig förklaring till detta är att prestandaskillnaderna mellan algoritmerna var för små under dessa scenarion.

Contents

Chapter 1

Introduction

It is vital to have a well-performing elevator system in highly populated buildings. Particularly in high-rise buildings, where the elevator usually is the only method of traveling between floors in a reasonable time. Passengers expect the elevator to serve them immediately and take them to their destination floor as fast as possible. However, while ordinary people see this as something that should not be hard to do and gets frustrated when this takes some time, the problem is not easy at all to solve. The task of assigning elevator cars to calls to minimize the average waiting time is as a matter of fact an NP-hard problem even with no car capacity restrictions [1].

There are different approaches towards decreasing the average waiting time, such as increasing the number of elevator cars or the capacity of each car. However, that kind of changes are not always possible to accomplish due to structural limitations of the building or might not be financially viable. An alternative and often a more applicable solution would be to optimize the control system for the elevator in the building. There is a collection of different strategies to control an elevator system, but an optimal policy for elevator group control is not yet known [2]. Because of this, developing a near-optimal policy using traditional methods is very difficult and an alternative approach is to let the computer develop a policy on its own. This approach to solve problems is known as *machine learning* and we will focus on the specific method known as *reinforcement learning.* Reinforcement learning is a form of unsupervised machine learning, where the system does not know what the optimal solution to the problem is. It must instead discover what the optimal solution is by itself.

1.1 Problem Statement

Is the application of reinforcement learning a viable strategy for reducing the average squared waiting time for the passengers in an apartment building's elevator system?

1.2 Scope

This thesis only focuses on elevator systems in apartment buildings, since it would not be possible to cover all sort of buildings where elevators are used. Apartment buildings are common and elevators are frequently used in them.

This thesis will primarily evaluate elevator systems after the *average squared waiting time*, where the waiting time is defined as the time between a passenger's arrival and entrance into a car [2]. There are also other ways of measuring the performance of elevator systems, such as *average system time*, where the system time is the total time it takes from a passenger's arrival to its destination, and regular *average waiting time* [2]. However, most of these other measuring techniques does not take into consideration that long waiting times are devastating for the passengers affected. For example, two passengers with waiting times 2 and 20 seconds have the same average (11 seconds) as two passengers whose waiting time is 11 seconds each. Using squared waiting time instead, the average times are 202 and 121 seconds^2 for the respective pair of passengers. This is consequently a better measuring technique to avoid long waiting times.

1.3 Purpose

Elevators have an essential role in today's society and we think that they will be even more important in the future when cities might grow taller instead of wider. The purpose of this thesis is to investigate if reinforcement learning can be used to create a control strategy that will reduce the waiting times for the elevator passengers in apartment buildings. More specifically, this thesis will investigate if reinforcement learning can be used to select what control algorithm that should be used, based on the prevailing traffic circumstances.

1.4 Disposition

In the next section, section two, the reader will be introduced to the area of elevator scheduling and reinforcement learning.

In the third section, the method and the tools to be used will be explained and motivated. This is to obtain credible results. All assumptions that are made and the test data that will be used are addressed here.

In section four the results obtained will be presented. These are then interpreted and discussed in the following section.

In the sixth section, conclusions for the problem statement are made based on the prior discussion.

Finally, all the references used in the thesis are listed.

1.5 Terminology

Key terms used in this thesis.

Lobby	The ground floor in a building.
Hall call	A call to the elevator system made from one of the floors in the building. A hall call only considers the desired direction.
Car call	A call made from inside an elevator car to a specific floor.
Control system	The central system that manages all elevator cars. For instance, it decides which car that shall handle a specific call and where a car shall go when it becomes idle.

Chapter 2

Background

To be able to invent a good strategy for elevator control, a lot has to be known about the elevator traffic in the building that the elevator shall serve. In this section, general traffic patterns for elevators will be specified and after that a theory that can be used to simulate this traffic — the Poisson process — will be explained. Subsequently, the most used strategies to handle the different traffic types will be described and a handful of algorithms using these strategies will be defined. This is followed by a sub-section devoted to the main subject, reinforcement learning.

2.1 Traffic Patterns

The traffic in an elevator system depends principally on the passenger arrivals. Generally, there are three different kinds of traffic in an elevator system: *up-peak*, *down-peak* and *two-way* traffic. In this section, each traffic type will be explained and it will be specified when these occur in an apartment building during a regular day.

2.1.1 Up-peak traffic

Up-peak traffic is characterized by that most of the passengers arrives at one single floor, typically the lobby, but have several destination floors [2]. In an apartment building, this is the case on the evenings when the residents are coming home from their jobs and other daily occupations.

2.1.2 Down-peak traffic

Down-peak traffic is almost the same as up-peak traffic but in the opposite direction. This type of traffic have multiple arrival floors and a single destination floor, most commonly the lobby [2]. This scenario occurs primarily in the mornings when the residents are heading to their daily occupations. The down-peak traffic will therefore not be as high and concentrated on weekends as on the weekdays since many of the residents will stay at home the whole day or at least longer than on the weekdays. This is also true for the up-peak traffic on the weekends.

2.1.3 Two-way traffic

During most of the day, the traffic, however, goes in both directions. Some residents travel from their apartments down to the lobby, and some in the opposite direction. This traffic is called *two-way lobby traffic*. There is also another form of two-way traffic called *two-way interfloor traffic*, where the users of the elevator travels between two floors different than the lobby [2]. This is not a common scenario in an apartment building if there are no special rooms such as a laundry room that is not on the ground floor.

2.2 Poisson Process

Under the assumption that the arrival time for two passengers is independent of each other, the arrival model in an elevator system can be described by a *Poisson process* [3]. Measurements on installed elevator systems in buildings has proven this to be a reasonable assumption [2]. The Poisson process defines that the probability of a certain number of arrivals at a given time t follows the *Poisson distribution* and the probability distribution of the time until the next arrival follows the *exponential distribution* [3].

To determine if a passenger has arrived, the *cumulative distribution function* (CDF) of the exponential distribution can be used. It is defined as $F_X(x) = 1-e^{-\lambda x}$, where λ is the average rate of arrivals per a time unit and x is the time since the last arrival [3]. $F_X(x)$ will describe the probability of an event occurring after the time x has passed. To determine if the event has occurred, a random value is generated from a uniform distribution in the interval $[0, 1]$. If the value is less or equal to the probability, the event has occurred.

Another approach is to calculate the time when the next arrival will occur after an arrival has been generated. The idea is to generate a random point from a uniform distribution on the function $F_X(x)$ y-axis and locate the x value for the point. The function $F_X(x)$ has an analytical inverse, meaning that the stochastic variable X can be calculated as $X = \frac{-ln(U)}{\lambda}$, where U is a random number generated from a uniform distribution in the interval $(0, 1]$ [12].

2.3 Control Strategies

Four general approaches to control elevators can be identified. Those will be explained below. There also exists many simple control strategies that do not fall into a specific category. One example is the *Collective control* strategy where the elevator cars always stop at the nearest call in their running direction [2].

2.3.1 Zoning approaches

The zoning approach divides the building into zones, where an elevator car only serves calls within its own zone and parks there when it is idle. The main purpose of this approach is to minimize the number of stops that an elevator car has to make and is primarily designed for down-peak traffic. The zones can either be static, where the elevator cars are permanently assigned to one zone or dynamic, where the cars can switch zones. [4]

2.3.2 Search-based approaches

In search-based approaches, the space of possible car assignments is searched through and the assignment that will optimize some property of the system, such as the average waiting time, is selected. In general there are two types of search strategies: *greedy* and *non-greedy* ones. In greedy strategies, a car is assigned to an elevator call when the call is registered and this assignment is not changed later when new information arrives. In non-greedy strategies, the car assignment can be delayed or changed when new information arrives. This means that greedy algorithms requires less computation time but sacrifices performance compared to non-greedy algorithms. [2]

2.3.3 Rule-based approaches

Rule-based approaches consist of rules of the form “IF situation THEN action” [2]. In the IF-clause, a logic known as *fuzzy logic* may be used to identify if the clause is satisfied or not. Fuzzy logic grades how truthful the statement is in the range $[0, 1]$. This approach may be used to switch the scheduling strategy when a specific traffic pattern emerges. [5]

2.3.4 Learning approaches

Instead of using heuristic approaches, learning approaches may be used. In learning-based strategies, the system adapts the control strategy based on circumstances influencing the building that the elevator system is used in, such as the current traffic. Since there exists no optimal policy for elevator control, some learning approaches such as supervised learning become less practical. This is because these require training data with a desired output value, which may not exist in practice. One possibility is to use reinforcement learning, which do not require a predefined output value for the training data. [2]

2.4 Algorithms

Based on the above-named strategies, there are a number of algorithms that is often used in practice. In this section, five of these will be defined and explained.

2.4.1 Longest Queue First (LQF)

In Longest Queue First, idle elevator cars always prioritize the floor with the passenger that has waited the longest. When an elevator car is moving, it will stop at a floor if it has a hall call in the same direction as the car is traveling [2]. One disadvantage with this is that a phenomenon known as *bunching*, where multiple elevator cars arrive at the same floor simultaneously, often occurs in practice [8].

2.4.2 Zoning

A building with N floors is divided into m zones, where m is the number of elevator cars. Each zone get at least $\lceil N/m \rceil$ floors and one elevator car. If the number of zones is not divisible by the number of floors, $N \bmod m$ zones will get one additional floor.

The boundaries of the zones are static, which means that the floors assigned to a specific zone are not changed during the course of the simulation. The zones do only consider the arrival floor of a passenger and not the destination. This makes the zoning approach only viable in down-peak scenarios. When an elevator car is idle, it is sent to the floor in the middle of its zone. When a car is moving, it can pick up passengers in the same movement direction inside its zone.

2.4.3 Round-Robin

In the Round-Robin algorithm, the elevator cars are assigned to calls in a sequential way, where the first car get the first call, the second car the second call and so on. The goal of the algorithm is to get equal load among all elevator cars and is not primarily designed to perform very well time-wise. [4]

2.4.4 Up-Peak Group Elevator

Up-Peak scheduling is a variant of the Round-Robin algorithm, designed for use in up-peak scenarios. The only difference is that when an elevator car is idle, it moves down to the lobby to be prepared for upgoing passengers. [4]

2.4.5 Estimated Time of Arrival (ETA)

Rong, Hakonen and Lahdelma [9] developed an Estimated Time of Arrival based elevator group control algorithm, which is defined in this section. Elevator group control algorithms using ETA are based on the Three Passage concept. This declares that there are three types of hall calls that can be made to an elevator car [9]:

- P1: Passage One calls can be served by the elevator car in its current travel direction.
- P2: Passage Two calls can be served after the elevator car has reversed its travel direction *once*.

- P3: Passage Three calls can be served after reversing the direction *twice*.

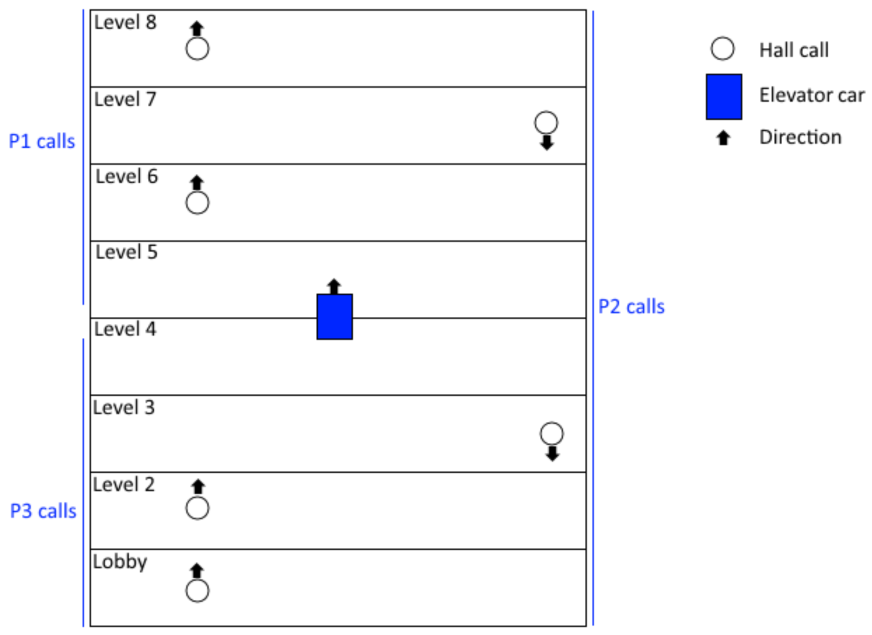

Figure 2.1: A descriptive picture of the Three Passage concept. The elevator car is traveling upwards and there are passengers waiting to be handled by the elevator car on different floors. [4]

The floors that cause an elevator car to change direction is called the *reversal floors* [4]. For each car, the algorithm calculates the cost to serve a call and chooses the car with the lowest. The total cost of allocating a hall call to elevator car i, is defined as

$$t_i^{total} = \sum_{j=1}^{n_i} t_{i,j}^{delay} + t_i^{attending} \tag{2.1}$$

where n_i is the number of passengers that has been assigned to elevator car i but not yet been served, $t_{i,j}^{delay}$ is the delay that the new call will cause to passenger j, who has been assigned to car i but not yet served, and $t_i^{attending}$ is the estimated time to handle the new call. How $t_{i,j}^{delay}$ and $t_i^{attending}$ is estimated can be seen in appendix A. [9]

2.5 Reinforcement Learning

Imagine that you want to train a computer to find the shortest path through a maze. From each position, the computer can only perform one out of four movement actions: move up, down, right or left. Given the maze and a position, the computer knows if it has reached the exit of the maze and what moves that are legal to make. Without any further information, how can this problem be solved?

One solution is to use a table that for each pair of position and movement action, contains a value that describes how good it was to perform that certain action at the position. This value can be based on that it cost the computer 1 to make a move, and it is rewarded with 100 if it reaches the exit of the maze. If the computer would have multiple attempts at trying to find a way through the maze, the table should gradually contain the best path through it. This is an example of how reinforcement learning works.

2.5.1 Definition

A reinforcement learning system consists of six main elements: an agent, an environment, a policy, a reward function, a value function and in some cases a model of the environment.

The *agent* is the learner and the decision-maker and the *environment* is everything outside the agent that it interacts with. It is the environment that gives the agent rewards.

The *policy* defines the behavior of the agent at a given time and is a mapping from the perceived state of the environment to the actions to take in the state.

The *reward function* defines the goal of the problem. The function maps each perceived state of the environment to a single number — the *reward* — which indicates the desirability of that state for the agent. The reward function defines the features of the problem that is faced by the agent and may serve as the basis for changing the policy.

The *value function* defines what is good in the long run, in contrast to the reward function which defines what is good in the immediate case. The *value* of a state is the total amount of reward that an agent can expect to gather in the future when starting from the state.

The *model of the environment* imitates the behavior of the environment. The purpose of the model is to plan which actions to take in the future by considering possible upcoming situations where the actions are actually performed. Reinforcement learning systems that do not use a model are *trial-and-error learners.*

An *episode* is a scenario that the agent will be trained on. An episode ends when a special state known as the *terminal state* is reached, which causes the state of the agent and the environment to be reset and the episode to be repeated. The information about the rewards and values is saved between the episodes in order for the agent to be able to learn anything. In some scenarios, there exists no terminal state and hence no episodes. In those cases, the interaction between the agent and the environment goes on continually. [6]

2.5.2 Exploration and exploitation

One of the challenges when using reinforcement learning, that is not present in other forms of learning methods, is the trade-off between exploration and exploitation. The agent *explores* action by trying actions that it has not tried before and *exploits* actions by performing actions that it already knows is generating a good reward [6].

In general there are two different approaches to handling exploration: *on-policy* and *off-policy*. In on-policy methods, the agent always performs exploration and tries to find the best policy that maintains exploration of possible actions [6]. This means that the selected action is not always the best one, based on the estimated reward. In off-policy methods, the agent can learn different policies for behavior and estimation. The behavior policy usually maintains exploration and the estimate policy may consist of actions which have not been tried. This means that an agent trained using an off-policy method may learn strategies that were not experienced during the learning phase [7].

2.5.3 Action selection policies

The action selection policies are used by the learning methods to determine which action to take.

The *ε-greedy* policy will normally choose the action with the highest estimated reward, but will with a probability of ε choose a random action independent of any action-value estimate. It is also possible to let ε decay over time, decreasing the amount of random actions taken. This method ensures that the optimal actions are discovered if enough trials are done.

One of the drawbacks with the ε-greedy method is that it selects random actions uniformly. This means that the worst possible action is selected with the same probability as the second best action. The *softmax* method avoids this problem by weighting each action according to its action-value estimate. Actions are always selected randomly, but the weight is taken into consideration [7].

The most common softmax method uses the Boltzmann distribution. The method chooses the action a on the t:th time step with the probability of

$$\frac{e^{Q_t(a)/\tau}}{\displaystyle\sum_{i=1}^{n} e^{Q_t(i)/\tau}} \tag{2.2}$$

where τ is a positive parameter called the *temperature*. A high temperature means that all actions are equally probable to be selected and a low temperature means that actions that have a great difference in their value estimate causes a great difference in their selection probabilities. As the temperature approaches zero, the probability of selecting the action with the highest estimated value gets close to 1 [6].

2.5.4 Value functions

Almost all reinforcement learning algorithms contain an estimate of the value function, which estimates how much reward that can be expected for the agent given a state or a state-action pair.

Let $V^{\pi}(s)$ be the value of state s under the policy π and $Q^{\pi}(s, a)$ the value of taking the action a in the state s under the policy π. $V^{\pi}(s)$ is known as the *state-value* function for the policy π and $Q^{\pi}(s, a)$ the action-value function for the policy π [6].

The action-value can be stored in a table, but when the complexity of the problem increases (for example when the number of possible states gets higher) this approach becomes less viable and some form of approximation has to be used. One common approach is to use something called a neural network to estimate the functions [2].

Temporal difference learning

Temporal difference methods do not need a model of the environment and uses experience to estimate the value function. Unlike other forms of reinforcement learning methods, temporal difference learning does not require the final reward to be generated in order to update the value for a state. The method only has to wait until the next time step, which makes it applicable in continuous scenarios. [6]

Q-learning

Q-learning (2.3) is an off-policy algorithm which learns an optimal policy independent of the one being followed. The algorithm works by observing a state s before selecting an action a using the selection policy. The action a is then executed and its reward r is calculated and the new state s' is observed. The action-value is then updated for the state using r and the maximum possible reward for the new state.[6]

Let t be the current time, s_t the state, a_t the action, r_t the reward, $\alpha \in [0, 1]$ the learning rate, $\gamma \in [0, 1]$ the discount factor and max_a the reward for taking the optimal action in the next state. A learning rate of 0 means that the agent does not learn anything and a learning rate of 1 means that the agent only considers the most recent information. A discount factor of 0 means that the agent only considers the most recent rewards and a discount factor near 1 will make the agent strive for long-term rewards [7].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_{a_{t+1}}(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)] \tag{2.3}$$

Using Q-learning, reinforcement learning can be implemented as in listing 2.1.

```
// Initalize the Q-table to some value, e.g. 0
$Q(s,a) \leftarrow 0$

// Repeat for each episode
for $episode \leftarrow 1$ $to$ $maxEpisodes$ do
    // Initalize the state
    $s \leftarrow ()$

    // Repeat until the terminal state has been reached
    while $\neg terminal(s)$ do
        // Select an action using a policy (e.g. $\varepsilon$-greedy)
        $a \leftarrow selectAction(Q, s)$

        // Take the action $a$ and observe the new state $s'$ and the
           reward $r$
        $(r, s') \leftarrow takeAction(a)$

        // Update the action-value table
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'}(Q(s',a')) - Q(s,a)]$
        $s \leftarrow s'$
    end
end
```

Listing 2.1: Reinforcement learning using Q-learning.

Sarsa

The Sarsa algorithm (2.4) is an on-policy algorithm and the main difference from the Q-learning algorithm is that the maximum reward for the next state is not necessarily used for updating the action-value. A new action is selected using the same policy as the one that selected the original action. The notation in the algorithm is the same as for Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \qquad (2.4)$$

Using Sarsa, reinforcement learning can be implemented as in listing 2.2.

```
// Initalize the Q-table to some value, e.g.  0
$Q(s, a) \leftarrow 0$

// Repeat for each episode
for $episode \leftarrow 1$ to $maxEpisodes$ do
    // Initalize the state
    $s \leftarrow ()$

    // Select an action using a policy (e.g.  $\varepsilon$-greedy)
    $a \leftarrow selectAction(Q, s)$

    // Repeat until the terminal state has been reached
    while $\neg terminal(s)$ do
        // Take the action $a$ and observe the new state $s'$ and the
           reward $r$
        $(r, s') \leftarrow takeAction(a)$

        // Select a new action using a policy (e.g.  $\varepsilon$-greedy)
           from the new state
        $a' \leftarrow selectAction(Q, s')$

        // Update the action-value table
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    end
end
```

Listing 2.2: Reinforcement learning using Sarsa.

2.5.5 Reinforcement learning in elevator systems

In 1998, Crites and Barto used reinforcement learning to optimize elevator scheduling [2]. They used a system where each elevator car were its own agent and made decisions independent of the other agents' decisions. Each agent was only able to make a small number of actions, such as if the elevator car should stop at the next floor or continue past it. To make the system more realistic, some constraints were applied to the action selection process. An elevator car could not continue past a floor if it contained a passenger that wanted to go off at the floor and it could not change direction before all calls in the current movement direction had been handled.

In their model, the state of the agents was represented by which of the hall buttons that were pressed on each floor, which car buttons that were pressed in each car, what floors all the cars were at and in which direction they were traveling. Using this model, the number of states in the system was over 10^{22} for a building with 10 floors and four elevator cars. They used the Q-learning algorithm with some modifications since a tabular approach was not practical due to the size of the state space. [2]

The action-value function, $Q(s, a)$, was approximated using a nonlinear neural network that used different kinds of inputs to approximate the output. The input to the network was data such as the state of the nine hall buttons used for down travel, the location, speed and direction of the elevator car. [2]

Two approaches were used. In the first one, each elevator car were given its own action-value function and in the second one the action-value function was shared between all elevator cars. The reward function used was the negative sum of the average squared waiting times for the passengers currently waiting in the halls for an elevator car. [2]

The traffic that the system was trained on were down-peak scenarios and in order to get good results, 60,000 hours of simulated training time was required. Both approaches greatly outperformed scheduling algorithms such as Longest Queue First and Zoning but only beat more advanced algorithms such as Empty the System Algorithm (ESA) by a small margin. [2]

Chapter 3

Method

To be able to determine if reinforcement learning is a good strategy to use to reduce the waiting times for the passengers in an elevator system, we had to compare that strategy against other scheduling algorithms. This was done using a simulator where the passenger timings in the elevator system were measured for the different algorithms.

Following in this section, our model for the elevator will be defined and the simulator used for the testings will be explained. Afterward, the implemented reinforcement learning system will be described in details, followed by the scenarios that were simulated. Finally, we will define how data was gathered to get trustworthy results and what data that was collected.

3.1 Elevator Model

In our simulations, a simplified model of the elevator car has been used to facilitate the implementation of the system and the measurements to it. Unlike elevator cars in the real world, these did not have any form of acceleration or deceleration and instead were able to go from stationary to full speed in no time. This was compensated by a stop time and a start time that occurred after the car stopped and before it left. The elevator cars were able to stop at a floor if the call to it was made just before the elevator car passed it. This is not a realistic behavior, but since the model was the same for all control strategies, it should not have had any influence on the results. The elevator control system moreover knew the exact amount of people that was waiting on each floor, where they were traveling and the number of passengers in each elevator car. The system dynamics for the elevator cars were based on Crites and Barto's approximations [2] and the used parameters were:

- Floor time (the time to move one floor): 1.5 seconds.
- Stop time (the time to stop the elevator car): 2.6 seconds.

- Door time (the time to open or close the doors): 1 second.
- Load time (the time for one passenger to enter or leave the car): 1 second.
- Start time (the time to start again after a stop): 2.6 seconds.
- Capacity: 8 passengers.

Some general constraints applied to an elevator car regardless of what control strategy that was used. Due to passengers expectations, it was not possible for a car to continue past a floor that was the destination for a passenger in the car. It could neither change direction before all calls in the current direction had been handled. It was impossible for a car to stop at a floor to pick up more passengers if its capacity had been reached. Lastly, if a car had stopped at a certain floor, a passenger always boarded it if it was heading in the correct direction and had capacity left.

3.1.1 State machine

An elevator car in the system was represented by a finite-state machine, with the states: *idle*, the car is not moving and does not have any passengers on board; *moving*, the car is moving towards a floor and may have passengers on board; *stopped*, the car has stopped at a floor and passengers may board it; *accelerating*, the car is accelerating and cannot take on any new passengers; *decelerating*, the car is decelerating towards a stop; *turning*, the car is changing its movement direction.

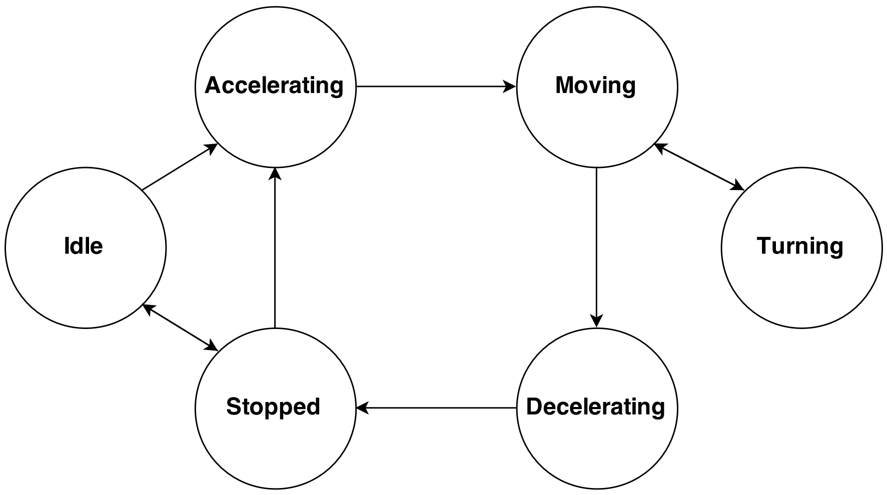

Figure 3.1: The state machine used for an elevator car. An arrow from one state to another means that there exists a transition between those states.

Below, the transitions between the states are described, where the first level indicates the current state and the possible transitions are listed underneath.

- Idle
  - → **Stopped:** A passenger boards the elevator car.
  - → **Accelerating:** The control system sends the elevator car towards a floor.
- Accelerating
  - → **Moving:** The time *start time* has passed since the elevator car started accelerating.
- Moving
  - → **Decelerating:** A passenger in the elevator car wants to go off on the next floor or the destination of the elevator car is the next floor.
  - → **Turning:** The elevator car needs to reverse its movement direction. This only occurs if the elevator car is empty and needs to change direction to serve a passenger.
- Decelerating
  - → **Stopped:** The time *stop time + door time* has passed since the elevator car started decelerating.
- Stopped
  - → **Accelerating:** The *load time + door time* has passed since the last passenger boarded the elevator car and is reset for each passenger. Only the passengers that have boarded the car since the last stop are considered.
  - → **Idle:** The elevator car is empty and there are no passengers that want to board on the current floor.
- Turning
  - → **Moving:** The time *stop time + start time* has passed since the elevator car started turning.

3.2 Simulator

The programming language chosen for the simulator was *Java* since it is the language in which the authors are most comfortable and it was also suitable for the task. Since the computational performance of the elevator system and the scheduling algorithms was not considered, a lower level language such as C++ was not required in order to get better runtime performance. The random generator used was Java's built in (java.lang.Random), which generates random numbers from a uniform distribution. The reinforcement learning system was written using a library called *YORLL* (YOrk Reinforcement Learning Library), developed by the

Reinforcement Learning Research Group at the University of York UK [10]. The library contained the implementation of all the reinforcement learning algorithms discussed in section 2.5. The source code for the simulator is available at GitHub [1].

3.2.1 Simulating time

The system simulated is a real-time system, which means that the time when an event occurs in the simulator is important. The time in the simulator was represented as the number of milliseconds since the start of the simulation. The simulation advanced with a fixed time step where events could occur. An event could be a passenger arrival or an elevator car that stopped, for instance.

The length of the time step can affect the results of the simulation if not chosen carefully. If the time step is too long, the precision of the waiting times will be affected. For example, if the time step is five seconds, the waiting times cannot be lower than five seconds since the simulation always advances with five seconds at a time. On the other hand, the lower the time step gets, the longer the simulation takes to run. The time step chosen was 10 milliseconds since no event occurs at a higher frequency than that in the simulator. Because the simulator always advanced with the same time in each step, the simulation was completely deterministic given that the random generator was given the same seed each time.

3.2.2 Metrics

To describe the handling capacity of an elevator system, the *HC5%* metric is often used. It defines that a certain percentage of the building's population is served within a five-minute window [11]. Since this is an established way of measuring an elevator system's capacity, this method has been used to describe how crowded the elevator system was in the simulation. The traffic density was consequently represented as how many passenger arrivals that occurred in five minutes in proportion to the building's population, in percent.

3.2.3 Passenger arrival model

The building considered in this thesis was an apartment building, where some assumptions were made. It was assumed that there were no public rooms on any floor that had an influence on the elevator traffic. Furthermore, people living on the ground floor were ignored since they rarely (if ever) use the elevator. With those assumptions in mind, we further assumed that the traffic to and from a certain floor, other than the lobby, only depended on how many residents it had. For example, a floor with 100 residents had twice the traffic, both to and from it, as a floor with 50 residents.

[1]`https://github.com/svenslaggare/ElevatorSimulator`

Knowing how many people lived on each floor, the average arrival rate was calculated for the building. The average arrival rate λ is the average number of arrivals per time unit, which in the simulation was one minute long and was calculated as

$$\lambda = \frac{\text{traffic desnity}}{5}. \tag{3.1}$$

The traffic density differed with the time of the day together with the ratio between up, down and interfloor traffic. This data was represented in 10-minute intervals, which means that a day had 144 intervals (6 per hour, 24 hours). To be able to calculate the time until the next arrival, we assumed that the passenger arrivals followed a Poisson process (see section 2.2).

The ratio between the different traffic types was represented as how much of the total traffic that was of a certain type. If, for instance, the up traffic ratio is 0.6, 60 % of all elevator hall calls are made from the lobby. If the up traffic ratio further is 0.35, 35 % of the traffic goes down to the lobby, and 5 % of the traffic is interfloor. That means that totally 40 % of the hall calls comes from floors other than the lobby.

3.3 Reinforcement Learning System

We chose to not use a reinforcement learning system where the system would learn a complete scheduling strategy, but instead one that learned to use the scheduling algorithm best suited for the prevailing traffic. The algorithms that were selectable are defined in section 2.4. This strategy was chosen for the reason that a complete reinforcement learning system would require a very large amount of states (Crites and Barto estimated that a state space with a size of 10^{22} would be required in a 10 floor building with four elevator cars [2]) and hence a neural network would be required to approximate the action-values, which is beyond the scope of this thesis.

3.3.1 Model

The state of the system was defined by the prevailing traffic pattern, which was described by four parameters: *up*, the ratio of passenger traveling from the lobby; *down*, the ratio of passengers traveling to the lobby; *interfloor*, the ratio of passengers traveling between floors other than the lobby and *travels*, the total number of travels. To make the system require less training and to be more realistic, the parameter values was divided into *intervals*. The interval grouped similar values together by rounding them to the closest value divisible by the interval length. For example, if we chose an interval length of 100 for the number of travels, all values between 50 and 149 were considered to be in the same interval (100). Two states were equal if for all the traffic pattern parameters, each value was in the same interval. In our simulations, up, down and interfloor had the interval length of 0.1 and the number of travels the length of 100.

The possible actions for the system were to use one of the scheduling algorithms and the system had the ability to switch scheduling algorithm every 10 minutes. We used 10-minute intervals to give the system enough data about the traffic pattern, and the traffic did not change more often than that. When the 10-minute interval had ended, the reward was calculated for the interval and the action-value table was updated using the learning algorithm. The reward function used was the negative sum of the average squared waiting times for the served passengers in the interval, the passengers still waiting on the floors and the passengers inside the elevator cars. An episode was defined to be an entire day.

3.3.2 Reinforcement learning algorithms

The learning algorithm used was the Q-learning algorithm with $\alpha = 0.05$ and $\gamma = 0.1$. Since the environment was non-deterministic, a low α-parameter was chosen to minimize how much the action-value was updated each time. The action selection policy that was chosen was softmax, using the Boltzmann distribution with $\tau = 200$ at the beginning. τ decayed linearly over the course of the training and became 0 when 55 % of the episodes had been simulated.

3.4 Scenarios

When simulating, different scenarios were used where the building's size and the number of elevator cars differed. To make it possible to run all simulations a sufficient number of times to get reliable results, only two buildings were simulated and two different amount of elevator cars were used for each building. This meant that the total number of scenarios to cover was four. Since elevator scheduling is not very interesting for small buildings with only one elevator car, those were not considered and the minimum number of cars was two.

The traffic profile that was used in the simulations was a complete 24-hour weekday. Since it would take too long time to simulate an entire week, only weekdays were simulated. The traffic density and pattern during the day were specified using the following assumptions:

- Hour 0–4: The traffic density is very low since most people are sleeping. The density first decreases but slowly starts to increase towards the end of the interval. The main traffic type shifts from up to down during the course of the interval.

- Hour 5–10: The people in the building wakes up and start going to their daily occupations. The traffic density increases rapidly until the peak is reached during hour 8. The traffic type is mostly down, but the number of up travels increases progressively after 6 o' clock.

- Hour 11–14: Most of the building's population has left the building and the traffic density decreases. The traffic type is two-way lobby traffic and the amount of up and down travels are around equal.
- Hour 15–19: People are starting to come home from their daily occupations and the traffic density increases. The traffic goes primarily from the lobby.
- Hour 20–23: The traffic density is decreasing since most of the buildings population are home. Most of the travels goes upwards.

The traffic density for the whole day in the building was specified so that the average person in the building made slightly more than two journeys per day. The average up rate, down rate, interfloor rate and traffic density per hour that was used, is specified in its entirety in appendix B.

3.4.1 Medium building

The smaller building had 10 floors (nine excluding the lobby) and a total number of 710 residents. The number of residents on each floor is presented in table 3.1. The building had either two or three elevator cars.

Table 3.1: The number of residents on each floor in the medium building.

Floor 1: 65	**Floor 2:** 80	**Floor 3:** 75
Floor 4: 85	**Floor 5:** 90	**Floor 6:** 90
Floor 7: 75	**Floor 8:** 80	**Floor 9:** 70

3.4.2 Large building

The bigger building had 16 floors including the lobby and a total number of 1,170 residents. The resident distribution over the floors is presented in table 3.2. The building had three or four elevator cars.

Table 3.2: The number of residents on each floor in the large building.

Floor 1: 70	**Floor 2:** 70	**Floor 3:** 75
Floor 4: 85	**Floor 5:** 75	**Floor 6:** 80
Floor 7: 90	**Floor 8:** 90	**Floor 9:** 85
Floor 10: 75	**Floor 11:** 80	**Floor 12:** 75
Floor 13: 80	**Floor 14:** 70	**Floor 15:** 70

3.5 Data Collection

To get credible results, each scenario was simulated 50 times and the average for all data points are presented under results. For each test run, a unique seed was used. This seed was then used to create the random generator used by the simulator for each of the scheduling algorithms. Using the same seed for all of the scheduling algorithms in a test run means that a passenger is always generated at the same time and with the same arrival and destination floor, which makes the results comparable.

For the reinforcement learner, the simulator used a new random seed for each training episode. When the training was over, the same 50 seeds as for the other algorithms were used for the reinforcement learning system, and the data was collected the same way as before. To get good results but still have a reasonable training time, this was set to one year (365 days).

The following data was collected. All the data points were collected both per hour and accumulated for the entire day.

- The number of generated passengers.
- The number of served passengers.
- The average waiting time.
- The average squared waiting time.
- The average ride time.
- The percentage of waiting times over 60 seconds.
- The number of travels *from* the lobby, known as *up travels.*
- The number of travels *to* the lobby, known as *down travels.*
- The number of travels between two floors other than the lobby, known as *interfloor travels.*

Chapter 4

Results

In figure 4.1, the generated traffic for the different scenarios is presented. It is a graphical representation of the table in appendix B and was the same for every building. The actual number of travels for a specific building is presented at the beginning of the sub-section devoted to it.

For every scenario, a table containing the accumulated data will be presented. This is followed by two figures where the first one displays the average squared waiting times over the day (per hour) and the second one the distribution of which scheduling algorithm the reinforcement learner used per hour.

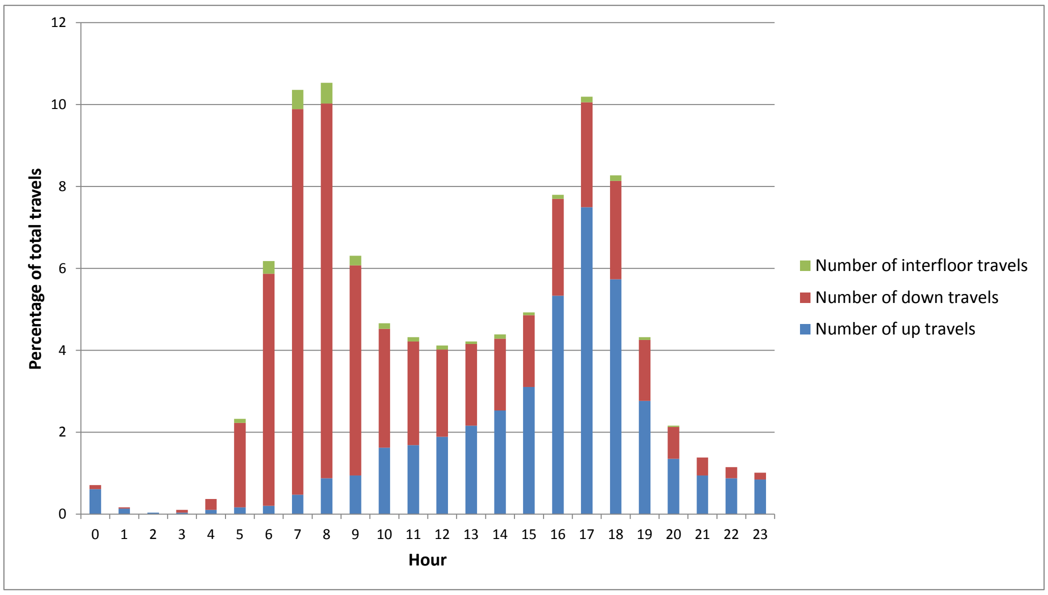

Figure 4.1: Amount of different travel types per hour.

4.1 Medium Building

In the medium building, the total amount of travels made was 1,796. Of those, 750 were up travels, 994 were down travels and 52 were interfloor travels.

4.1.1 Two cars

When two elevator cars were used in the smaller building, reinforcement learning performed slightly worse than the best algorithm, Longest Queue First (table 4.1). It was primary during the early hours, 0–5, and hour 17 that LQF performed better than reinforcement learning (figure 4.2). It did also perform worse than Zoning during hours 7–8.

Table 4.1: The performance of the different scheduling algorithms in the medium building with two elevator cars.

Scheduling algorithm	Longest Queue First	Zoning	Round-Robin	Up-Peak Group Elevator	Estimated Time of Arrival	Reinforcement Learning
Average waiting time	10.50 s	13.64 s	12.29 s	12.33 s	12.76 s	10.61 s
Average squared waiting time	221.07 s^2	302.58 s^2	313.30 s^2	332.38 s^2	294.25 s^2	225.86 s^2
Average ride time	16.37 s	16.35 s	15.13 s	14.99 s	15.44 s	16.05 s
Waiting times over 60 s	0.22 %	0.22 %	1.058 %	1.17 %	0.61 %	0.22 %

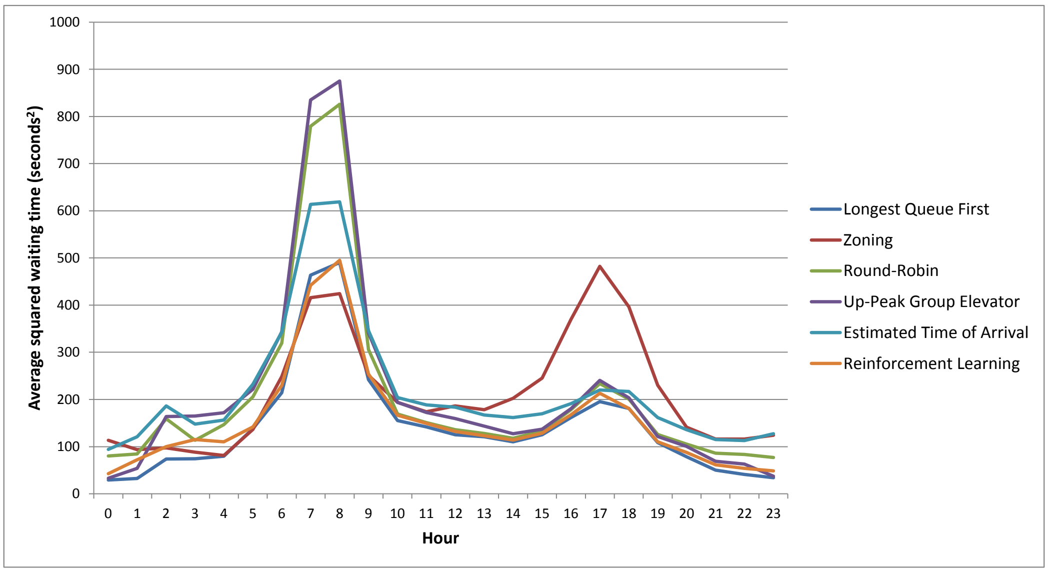

Figure 4.2: The average squared waiting time as a function of time for the different scheduling algorithms in the medium building with two elevator cars.

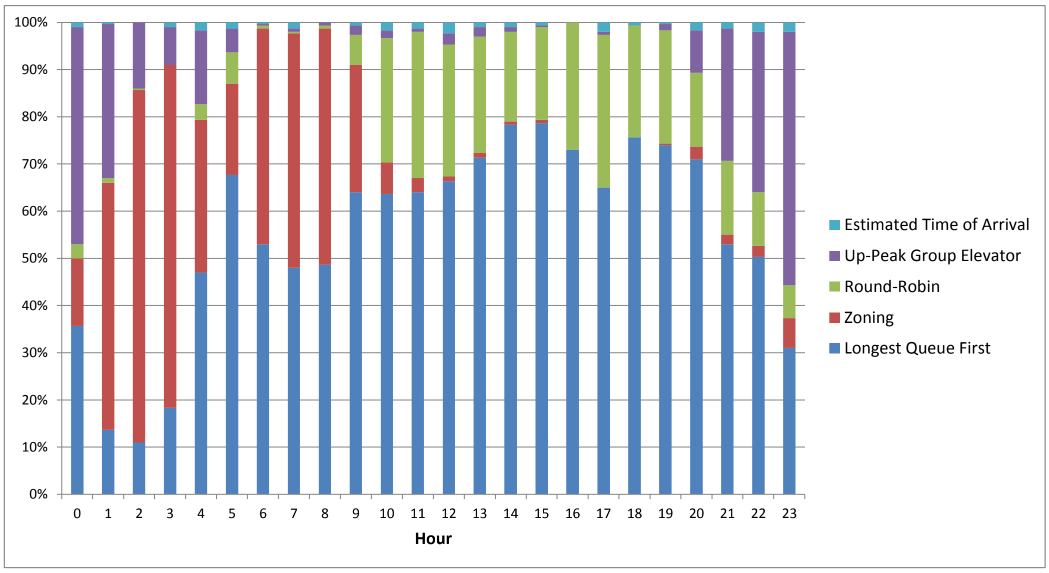

Figure 4.3: The scheduler usage for the reinforcement learning system as a function of time for the medium building with two elevator cars.

4.1.2 Three cars

When three cars were used, reinforcement learning performed slightly worse than both Longest Queue First and Round-Robin (table 4.2). In figure 4.4, we can see that reinforcement learning was outperformed mainly in the hours 7–8 and 16–18. Why can be seen in figure 4.5. For example, between hour 16 and 18, where Round-Robin was the best algorithm, the reinforcement learner used LQF and Up-Peak Group Elevator around the same amount of time as it used Round-Robin. Those algorithms did not perform as good during those hours (figure 4.4).

Table 4.2: The performance of the different scheduling algorithms in the medium building with three elevator cars.

Scheduling algorithm	Longest Queue First	Zoning	Round-Robin	Up-Peak Group Elevator	Estimated Time of Arrival	Reinforcement Learning
Average waiting time	8.24 s	11.29 s	8.26 s	8.91 s	9.58 s	8.16 s
Average squared waiting time	135.47 s^2	225.35 s^2	132.70 s^2	154.56 s^2	171.92 s^2	139.28 s^2
Average ride time	15.94 s	15.99 s	14.89 s	14.80 s	15.12 s	15.48 s
Waiting times over 60 s	0.00 %	0.06 %	0.00 %	0.00 %	0.06 %	0.00 %

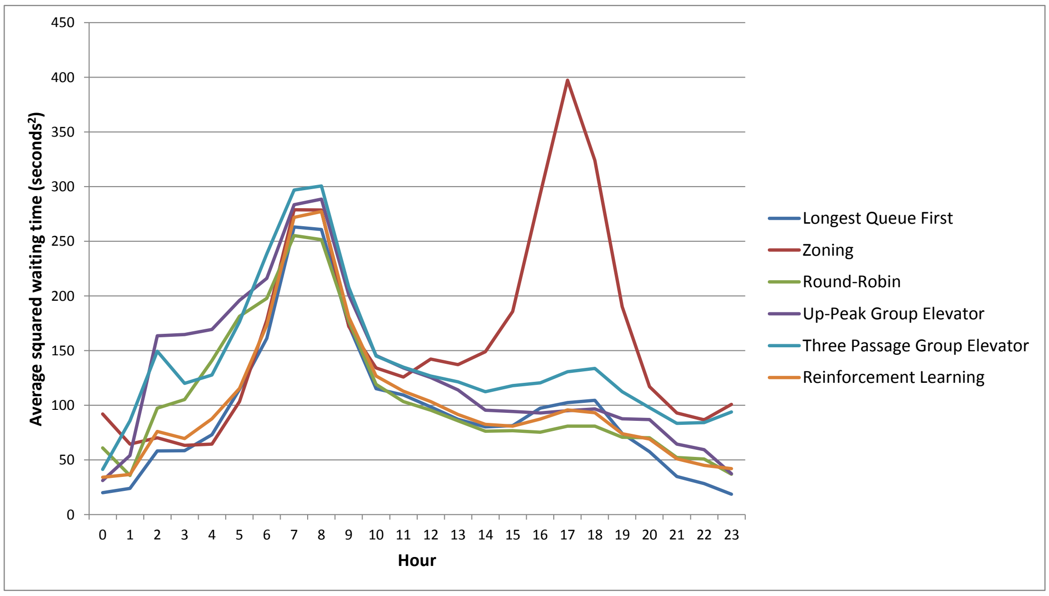

Figure 4.4: The average squared waiting time as a function of time for the different scheduling algorithms in the medium building with three elevator cars.

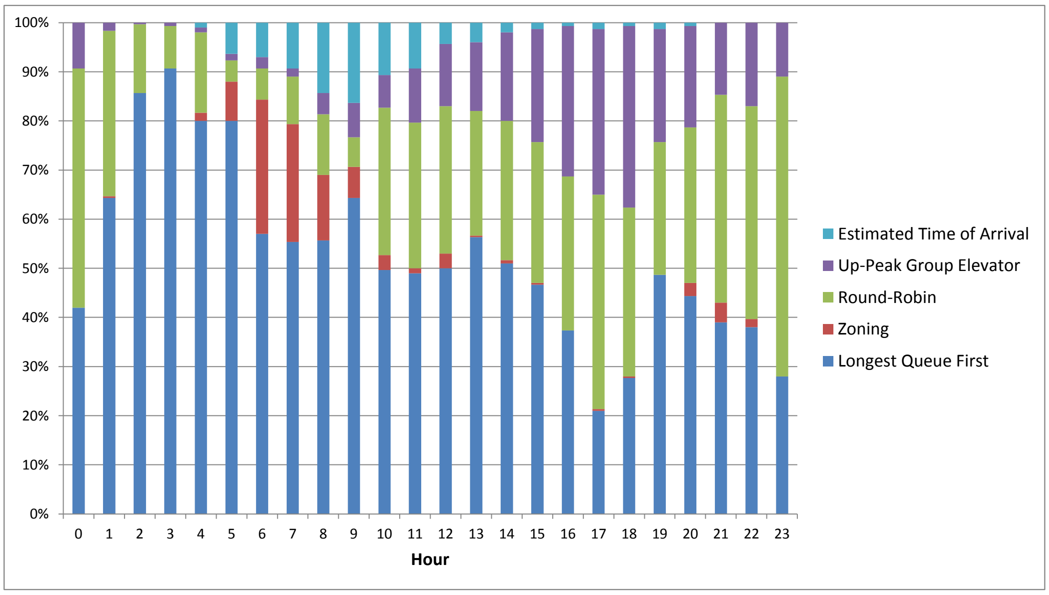

Figure 4.5: The scheduler usage for the reinforcement learning system as a function of time for the medium building with three elevator cars.

4.2 Large Building

In the large building, the total amount of travels made was 2,992. Of those, 1,253 were up travels, 1,651 were down travels and 88 were interfloor travels.

4.2.1 Three cars

In the scenario where three cars were used in the large building, reinforcement learning outperformed all of the other scheduling algorithms by a large margin. The difference in average squared waiting time to the next best algorithm (ETA) was 198.09 s^2 (table 4.3). We can see that reinforcement learning tangent the best scheduling algorithm during almost every hour of the day (figure 4.6).

Table 4.3: The performance of the different scheduling algorithms in the large building with three elevator cars.

Scheduling algorithm	Longest Queue First	Zoning	Round-Robin	Up-Peak Group Elevator	Estimated Time of Arrival	Reinforcement Learning
Average waiting time	17.99 s	23.27 s	21.45 s	22.51 s	18.36 s	15.58 s
Average squared waiting time	904.11 s^2	1,090.03 s^2	1,333.81 s^2	1,426.24 s^2	693.56 s^2	495.47 s^2
Average ride time	23.06 s	23.42 s	19.95 s	19.83 s	20.40 s	21.12 s
Waiting times over 60 s	3.24 %	7.02 %	8.25 %	8.85 %	4.18 %	2.00 %

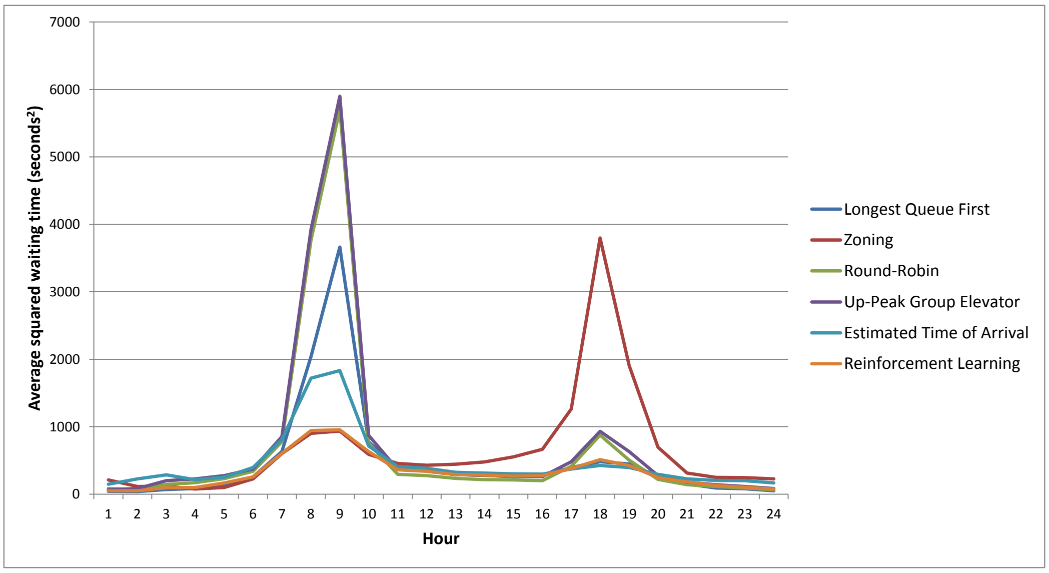

Figure 4.6: The average squared waiting time as a function of time for the different scheduling algorithms in the large building with three elevator cars.

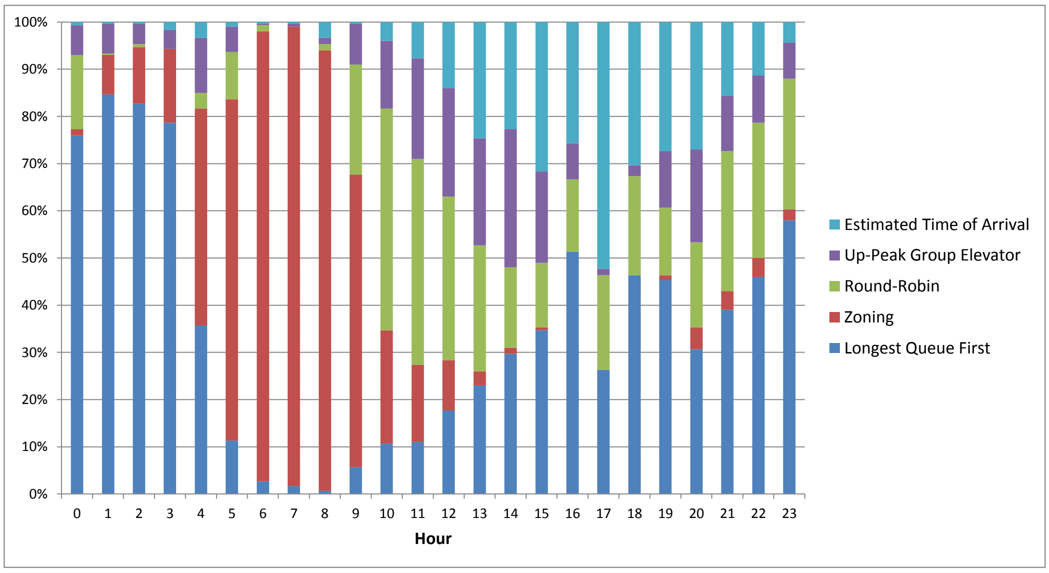

Figure 4.7: The scheduler usage for the reinforcement learning system as a function of time for the large building with three elevator cars.

4.2.2 Four cars

When four cars were used, reinforcement learning again performed the best, but the margin was much lower compared to when three cars were used (table 4.4). Reinforcement learning tangent the best algorithm during most of the day (figure 4.8).

Table 4.4: The performance of the different scheduling algorithms in the large building with four elevator cars.

Scheduling algorithm	Longest Queue First	Zoning	Round-Robin	Up-Peak Group Elevator	Estimated Time of Arrival	Reinforcement Learning
Average waiting time	14.66 s	20.68 s	12.70 s	14.07 s	13.95 s	12.31 s
Average squared waiting time	452.38 s^2	896.47 s^2	362.79 s^2	434.49 s^2	392.33 s^2	330.88 s^2
Average ride time	22.23 s	22.88 s	19.53 s	19.43 s	20.01 s	20.15 s
Waiting times over 60 s	1.40 %	5.58 %	1.24 %	1.57 %	1.44 %	0.80 %

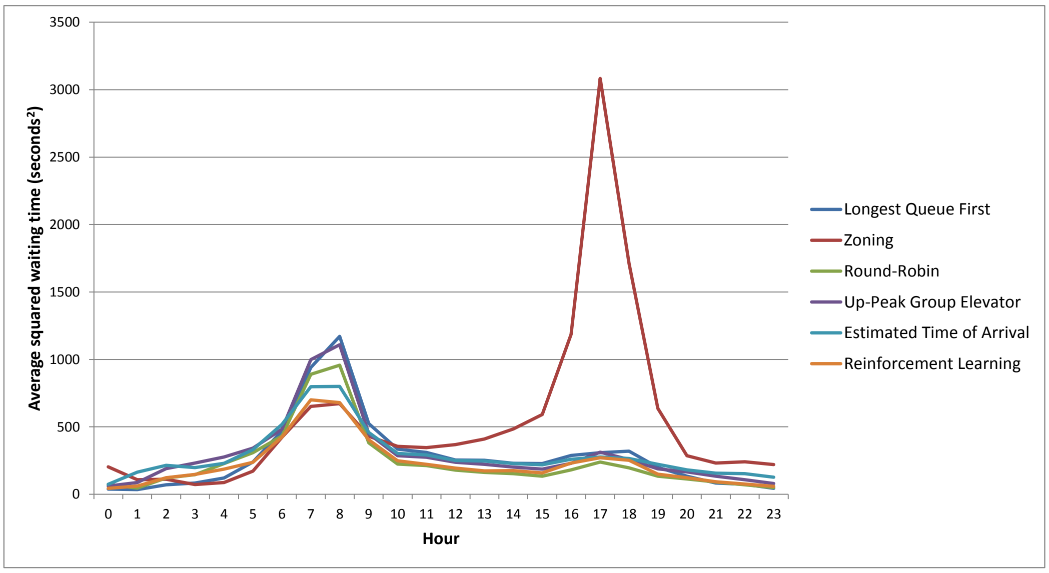

Figure 4.8: The average squared waiting time as a function of time for the different scheduling algorithms in the large building with four elevator cars.

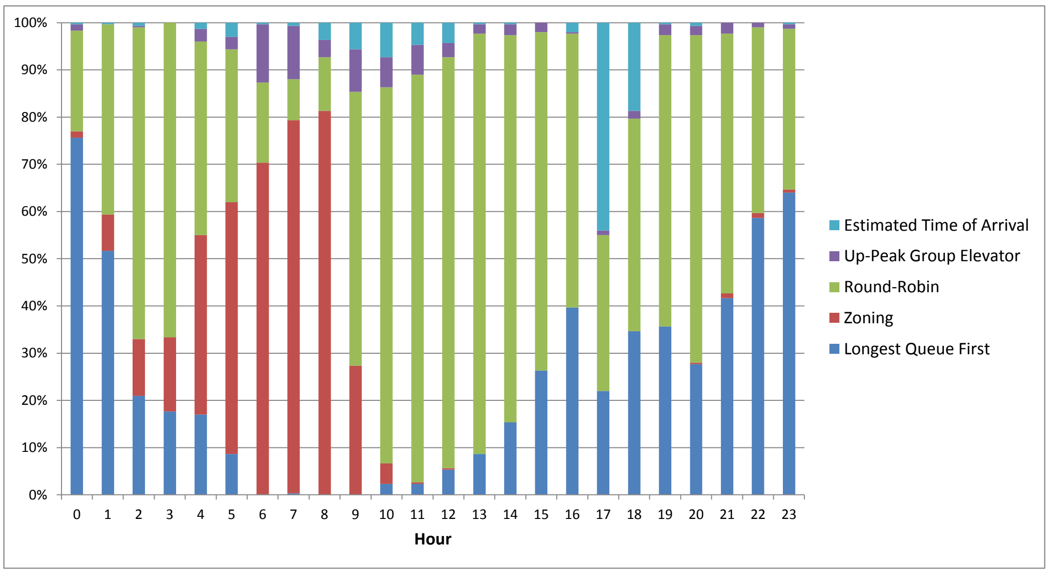

Figure 4.9: The scheduler usage for the reinforcement learning system as a function of time for the large building with four elevator cars.

4.3 Reinforcement Learning

In figure 4.10, the average squared waiting times for the passengers in the large building with three elevator cars is visualized depending on the training time. The same parameters were used — $\alpha = 0.05$, $\gamma = 0.1$, $\tau = 200$ which decayed linearly to 0 until 55 % of the episodes had been trained — for all the simulations.

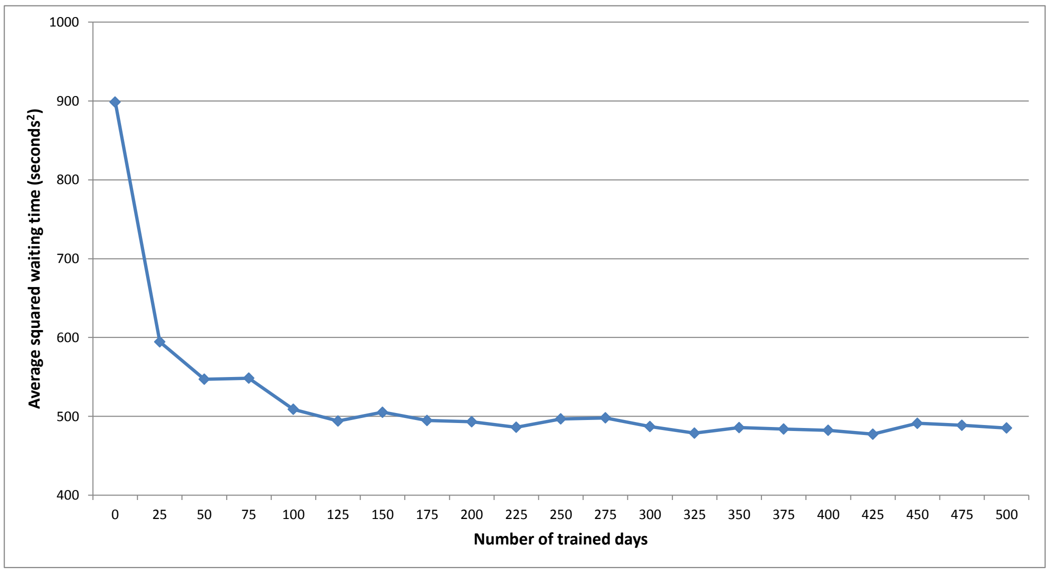

Figure 4.10: The average squared waiting time as a function of the number of training days, in the large building with three elevator cars.

Chapter 5

Discussion

The reinforcement learning scheduling algorithm performed best according to the average squared waiting time in the large building and were only beaten by a small margin in the medium building. Generally, reinforcement learning performed better in the scenarios where the large building was used and especially well when that building only had three elevator cars. One possible explanation to why the reinforcement learning system performed better in the large building than in the small one is that the performance variations between the selectable algorithms were more distinct for the larger building. This implied that the reward the system was given for selecting an algorithm differed greatly between the different algorithms, which meant that the system could choose the best algorithm more easily. Why the difference was bigger when using a lower amount of cars, is probably because the scheduling algorithm gets less important when more elevator cars are added, since the cars then will be more spread out in the building.

A reinforcement learning system using our approach, shall ideally never perform worse than any other algorithm in a time interval. This is not the case for our reinforcement learning system. In every hour diagram (figure 4.2, 4.4, 4.6 and 4.8) it can be seen that the reinforcement learner does not always tangent the best algorithm. In the hour diagrams for the two scenarios with the medium building (figure 4.2 and 4.4), the reinforcement learner differs notably during the down-peak hours, 7–8. In the scenario when two elevator cars were used, the reinforcement learner used about the same amount of Zoning and LQF (the best and the next best in the interval) but the curve is more similar to the LQF curve than a combination of them. In the same interval, but when three cars were used, LQF was most used, primarily followed by Zoning. But, in this case, Round-Robin was clearly the best scheduling algorithm and was used very moderate. The curve also looks more like Zoning's curve than the LQF curve. It is hard to comprehend why the reinforcement learning system makes the decision it does, but it is possible that either our implementation is not perfect or that the reinforcement learning algorithms that we used was not optimal for this field of application. Another possible explanation may be that the used reward function was not good enough.

5.1 Training Time

The training time that was used for the reinforcement learner in the experiments was a complete year. But as figure 4.10 shows, that amount of training time is not truly necessary and it would be possible to go as low as to 125 days without losing significant performance. This is a more reasonable number to be used as training time for a system in a real building. One year of training might be too long, but around four months is more moderate. A real system would probably use a training profile which stretches over a longer time, but where it gains the biggest improvements at the beginning of the usage and then gets better gradually over the years.

The average squared waiting time does not seem to stabilize around a fixed value, but instead tend to stay in the interval 475–495 after 300 days of training. One explanation for this is the non-deterministic nature of the simulator. For each training episode, a different seed was used which means that the traffic patterns were not exactly the same for two episodes. This means that the overall observed traffic was different each training session.

We did only study the training time for one scenario — the large building with three elevator cars. This was because, in this scenario, the performance of the different scheduling algorithms varied the most and, therefore, would best demonstrate how the training time affected the performance.

5.2 Elevator Model and Simulator

In our simulator, the system knew exactly how many people that were waiting on each floor and how long each one had waited. It is unlikely that a real system would have this kind of information. Most elevator systems only know that *at least* one passenger will travel upwards or downwards from a floor and how long that passenger has waited. The extra information this system had was used in the reinforcement learner's reward function. Because of this, it is questionable whether this elevator control strategy can be used in a real elevator system and it must most likely be modified to be usable.

5.3 Reinforcement Learning System

The model that the reinforcement learning system used was a simplified one where the system did not learn a complete new strategy but used other algorithms depending on the prevailing circumstances. This means that the performance of the system was constrained by the performance of the other scheduling algorithm. The system, therefore, was unlikely to develop an optimal strategy, since the algorithms used was only heuristics. Ideally, the system should develop a complete strategy on its own to get the best results. However, such a system is more complex and would require more training time in order to get good performance. Crites and

Barto developed such a system, that required 60,000 hours (almost seven years) of training that outperformed algorithms implemented in this thesis such as Longest Queue First and Zoning but barely beat more advanced algorithms that were not covered here. Our model did not require this many hours of training, so this is an advantage for our model. It is worth noting, though, that we did not use the same model as they did, and the implementations of the simulator and the scheduling algorithms is not the same, so the results is not entirely comparable.

The state model used in the simulation was one that was defined by the prevailing traffic pattern. An alternative representation of the state is to only consider the time of the day and divide it into different intervals. Since all of the episodes are generated by the same scenario and model, the system should then learn a policy that is useful. The problem with this model is that it cannot adapt to changes in traffic during the week or over time. Using the traffic pattern representation, this is not a problem. For example during weekends, the morning traffic might presumably be much sparser compared to weekdays. However, since the state is described by the traffic pattern, it does not matter since the best scheduling algorithm in weekends for a specific traffic should be the same as for weekdays.

When the system changed scheduling algorithm, the decision was made based on the traffic in the current interval and not the upcoming, in which the new algorithm should be used. This is because it would require a model of the future traffic, which would make the system more complex. If the traffic pattern in the new interval would be drastically different from the actual, the system could have chosen the wrong algorithm. However, the interval was chosen to be short (10 minutes) and the traffic did not change much from one interval to the next, which is a realistic behavior. This means that the information that was used for selecting scheduling algorithm was accurate enough.

Choosing the right parameter values, learning algorithm and action selection policy for the reinforcement learning system, is more of an art than science. We have not found any source that describes a method for choosing the parameter values or algorithms, other than that a low α-value is good in scenarios where the environment is non-deterministic. We did not perform any systematic testing to find the best algorithms and parameters, and they were chosen based on a few tests that was not presented in this thesis.

Chapter 6

Conclusion

The problem statement to be answered in this thesis was if reinforcement learning is a viable strategy to use in elevator systems in apartment buildings, to reduce the average squared waiting time. With the simulation results in mind, it is safe to say that reinforcement learning is a great strategy to use at least in buildings with 16 floors and three or four elevator cars. It can be assumed that it is good even for bigger buildings than that, but there is no proof for it. However, for buildings with 10 floors and two or three cars, reinforcement learning is somewhat unnecessary and a regular Round-Robin or LQF algorithm should be used instead.

The output of the reinforcement learning was a policy that determined which scheduling algorithm to use given a traffic pattern. Since the traffic was only defined by four parameters that could be calculated in a real system, this method could be applied to such a system. However, a modification to the reward function is required since it used information that is not available in a real system.

Bibliography

[1] B. Xiong, P. B. Luh, and S. Chung Chang, "Group Elevator Scheduling with Advanced Traffic Information for Normal Operations and Coordinated Emergency Evacuation," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1419-1424, Apr. 2005.

[2] R. Crites and A. Barto, "Elevator Group Control Using Multiple Reinforcement Learning Agents," *Machine Learning*, vol. 33, pp. 235-262, Nov. 1998.

[3] The University of Alabama in Huntsville: UAH, *The Poisson Process* [Online], [cited 7th March 2015]. Available: `http://www.math.uah.edu/stat/poisson/`.

[4] T. Strang and C. Bauer, "Context-Aware Elevator Scheduling," *21st International Conference on Advanced Information Networking and Applications Workshops*, vol. 2, pp. 276-281, May 2007.

[5] M.-L. Siikonen, *Elevator Group Control with Artificial Intelligence* [Online]. [cited 28th February 2015]. Available: `http://sal.aalto.fi/publications/pdf-files/rsii97a.pdf`.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 2012.

[7] T. Eden, A. Knittel, and R. van Uffelen, *Reinforcement Learning* [Online]. [cited 15th March 2015]. Available: `http://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html`.

[8] D. Nikovski and M. Brand, "Decision-Theoretic Group Elevator Scheduling," *Proceedings of the 13th International Conference on Automated Planning and Scheduling ICAPS03*, pp. 133-142, June 2003.

[9] A. Rong, H. Hakonen, and R. Lahdelma, "Estimated Time of Arrival (ETA) Based Elevator Group Control Algorithm with More Accurate Estimation," *TUCS Technical Report, vol. 584*, Dec. 2003.

[10] D. Kudenko et al., *YOrk Reinforcement Learning Library (YORLL)* [Online]. [cited 20th March 2015]. Available: `http://www.cs.york.ac.uk/rl/downloads/YORLLOverview.pdf`.

[11] J. Koehler and D. Ottiger, “An AI-Based Approach to Destination Control in Elevators,” *AI Magazine*, vol. 23, pp. 59-78, Fall 2002.

[12] D. E. Knuth, *The Art of Computer Programming*, Boston, MA: Addison-Wesley, 1997.

Appendix A

Estimated Time of Arrival Algorithm

All the formulas and explanations for the algorithm come from Rong, Hakonen and Lahdelma's definition [9]. As mentioned in the text, the total cost of allocating a hall call to elevator car i, is defined as

$$t_i^{total} = \sum_{j=1}^{n_i} t_{i,j}^{delay} + t_i^{attending} \tag{A.1}$$

where n_i is the number of passengers that has been assigned to elevator car i but not yet been served, $t_{i,j}^{delay}$ is the delay that the new call will cause to passenger j, who has been assigned to car i but not yet served, and $t_i^{attending}$ is the estimated time to handle the new call.

Expected extra stops and expected farthest floor

Let (k, d) represent an elevator call, where k is the floor of the call and d the direction of the call (up or down). The notation that will be used to calculate the expected number of extra stops and the expected farthest floor is defined below.

n_k^{pass}	The number of passengers behind the hall call.
f_i^k	The number of floors to move for elevator car i, to reach the farthest floor in its current travel direction.
l_i^{net}	The expected net distance to the farthest floor.
s_i^k	The expected number of stops for the f_i^k floors caused by the new hall call (k, d) under the condition that the assignment of elevator car i is empty.
$s_{k,j}^{extra}$	The actual number of extra stops caused by the new call (k, d) before it arrives at the j:th floor.
$s_{k,j}^{mandatory}$	The number of mandatory stops between floor k and floor j, themselves excluded.

$C_{k,j}$	The set of car calls between floor k and floor j, themselves excluded.
$H_{k,j}$	The set of assigned hall calls between floor k and j, excluding themselves, that should be handled before the elevator car arrives at floor j.
f_i^{actual}	The actual farthest floor.
$f_i^{farthest}$	The expected farthest floor that the current hall call can reach.
P_i^k	The probability that none of the passengers that boards the elevator car on floor k with the travel direction d will get off at each ongoing floor.

Under the assumption that a passenger that boards an elevator car on floor k has an equal probability of exiting at one of the f_i^k floors, we have

$$P_i^k = \begin{cases} 1 - 1/f_i^k & \text{if } n_k^{pass} = 1 \\ e^{-n_k^{pass}/f_i^k} & \text{otherwise} \end{cases}. \tag{A.2}$$

Thus

$$s_i^k = f_i^k(1 - P_i^k) \tag{A.3}$$

and

$$l_i^{net} = \begin{cases} 0 & \text{if } f_i^k = 1 \\ \sum_{l=2}^{f_i^k} \prod_{j=f_i^k-l+1}^{f_i^k} P_i^k & \text{otherwise} \end{cases}. \tag{A.4}$$

After l_i^{net} has been calculated, it is uncomplicated to calculate the expected farthest floor.

$$f_i^{farthest} = f_i^{actual} - l_i^{net} \tag{A.5}$$

Then, the number of mandatory stops and extra stops are estimated by

$$s_{k,j}^{mandatory} = |C_{k,j}| + |H_{k,j}| - |C_{k,j} \cap H_{k,j}| \tag{A.6}$$

and

$$s_{k,j}^{extra} = \frac{s_i^k(|j-k| - 1 - s_{k,j}^{mandatory})}{f_i^k}. \tag{A.7}$$

Estimating $t_i^{attending}$

The notation that will be used for estimating the time to handle a new hall call is:

t_s	The stop time of one full stop, which is defined as $$t_s = P_i^k + s_i^k + l_i^{net} + f_i^{farthest} + s_{k,j}^{mandatory} + s_{k,j}^{extra}.$$
C_i	The set of car calls for the i:th elevator car.
H_i	The set of hall calls for the i:th elevator car.
C_i^{before}	The set of car calls for the i:th elevator car to be attended before the hall call (k, d).
C_i^{after}	The set of car calls for the i:th elevator car to be attended after the hall call (k, d).
H_i^{before}	The set of hall calls for the i:th elevator car to be attended before the hall call (k, d).
H_i^{after}	The set of hall calls for the i:th elevator car to be attended after the hall call (k, d).
$t_{i,k}^{nonstop}$	The nonstop travel time to floor number k based on the current motion of elevator car i.

$t_i^{attending}$ is then estimated as follows.

a) If (k, d) is a P1 hall call, then
$t_i^{attending} = t_{i,k}^{nonstop} + t_s(|C_i^{before}| + |H_i^{before}| - |C_i^{before} \cap H_i^{before}| + \sum_{i \in H_i^{before}} s_{i,k}^{extra})$.

b) If (k, d) is a P2 hall call, then

   i) If no P1 hall calls or car calls exist for elevator car i, then $t_i^{attending}$ is the same as in a).

   ii) Otherwise $t_i^{attending}$ can be split into two parts. The first part is the time to handle all of the car calls and P1 hall calls, then the car reverses the direction. The second part is the time to handle (k, d) from the reversal floor. The time in both parts can be estimated separately using the formula in a).

c) If (k, d) is a P3 hall call, then $t_i^{attending}$ can be split into three parts. The first part is the time to finish handling all of the car calls and P1 hall calls (if any), then the elevator car reverses the direction. The second part is the time to finish handling all the P2 hall calls (if any), then the car reverses the direction for the second time. The third part is the time to handle (k, d) from the second reversal floor. The time in all three parts can be estimated separately using the formula in a).

Estimating $t_{i,j}^{delay}$

$t_{i,j}^{delay}$ is the delay time that the new hall call (k, d) will cause to the passenger j in the set H_i^{after}. It is splitted into three parts.

1. One mandatory stop at the destination floor for the hall call if there are no earlier car call made to that floor.
2. The expected number of extra stops caused by the hall call.
3. The additional travel time if the hall call will affect the reversal floor.

Appendix B

Average Traffic per Hour

Hour	Traffic density (HC5%)	Up rate	Down rate	Interfloor rate
0	0.150	83.8 %	11.2 %	5 %
1	0.038	75.7 %	19.3 %	5 %
2	0.018	58.6 %	36.4 %	5 %
3	0.032	41.4 %	53.6 %	5 %
4	0.092	24.3 %	70.7 %	5 %
5	0.500	8 %	87 %	5 %
6	1.275	3.7 %	91.3 %	5 %
7	2.175	4.5 %	90.5 %	5 %
8	2.225	8.5 %	86.5 %	5 %
9	1.333	14.5 %	80.5 %	5 %
10	0.966	33.5 %	61.5 %	5 %
11	0.916	39.5 %	55.5 %	5 %
12	0.866	45.5 %	49.5 %	5 %
13	0.874	51.5 %	43.5 %	5 %
14	0.924	57.5 %	37.5 %	5 %
15	1.050	62.5 %	32.5 %	5 %
16	1.650	68.5 %	26.5 %	5 %
17	2.150	73.5 %	21.5 %	5 %
18	1.750	69.5 %	25.5 %	5 %
19	0.900	63.5 %	31.5 %	5 %
20	0.437	62.5 %	32.5 %	5 %
21	0.300	68.5 %	26.5 %	5 %
22	0.258	74.5 %	20.5 %	5 %
23	0.226	80.5 %	14.5 %	5 %