TU WIEN Informatics

Optimizing Elevator Control with a Destination Registration System

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Jonas Kompauer, BSc
Matrikelnummer 11776872

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Projektass.in Dipl.-Ing.in Maria Bresich, BSc

Wien, 27. März 2025

Jonas Kompauer	Günther Raidl

TU WIEN Informatics

Optimizing Elevator Control with a Destination Registration System

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Jonas Kompauer, BSc
Registration Number 11776872

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Projektass.in Dipl.-Ing.in Maria Bresich, BSc

Vienna, March 27, 2025

Jonas Kompauer	Günther Raidl

Erklärung zur Verfassung der Arbeit

Jonas Kompauer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 27. März 2025

Jonas Kompauer

Danksagung

Ich möchte mich an dieser Stelle bei meinen Betreuern Günther Raidl und Maria Bresich bedanken, die mich über die gesamte Dauer meiner Masterarbeit mit ihrem Wissen und ihrer Erfahrung unterstützt und motiviert haben. Ohne ihr kontinuierliches Feedback und die wertvollen Diskussionen wäre diese Arbeit nicht umsetzbar gewesen.

Weiters möchte ich mich bei meiner Familie bedanken, die mir stets den Rücken freigehalten und mich immer ermutigt hat, weiterzumachen.

Ich möchte mich auch bei meinen Freunden bedanken, die mir geholfen haben, in frustrierenden Phasen der Arbeit nicht den Mut zu verlieren. Großer Dank gilt speziell Anoki und Flo, die während ihrer eigenen Masterarbeiten durch ähnlich schwierige Phasen gegangen sind. Der Austausch mit ihnen hat mir gezeigt, dass ich nicht der Einzige bin, der mit solchen Problemen zu kämpfen hat. Besonders möchte ich mich auch bei meiner besten Freundin Viola bedanken, die sowohl in guten als auch in schlechten Zeiten immer ein offenes Ohr für mich hatte und mir stets Rückhalt gab. Ohne sie wäre dieses Projekt nicht möglich gewesen.

Acknowledgements

I would like to take this opportunity to thank my supervisors, Günther Raidl and Maria Bresich, who supported and motivated me throughout the entire duration of my master's thesis with their knowledge and experience. Without their continuous feedback and valuable discussions, this work would not have been possible.

Furthermore, I would like to thank my family, who always had my back and continuously encouraged me to keep going.

I would also like to thank my friends, who helped me stay positive during frustrating phases of my work. Special thanks go to Anoki and Flo, who went through similarly challenging times during their own master's theses. Sharing our experiences made me realize that I was not the only one dealing with these difficulties. I would also like to give a special thanks to my best friend Viola, who always had an open ear for me in both good and bad times and continuously supported me. Without her, this project would not have been possible.

Kurzfassung

In unserer modernen Gesellschaft sind Aufzüge aus größeren Gebäuden nicht mehr wegzudenken. Aufzugsgruppensteuerungssysteme sind Mehrfachaufzugsysteme, die den Personentransport in einem Gebäude optimieren und die Beförderungskapazität erhöhen. Während herkömmliche Aufzugssysteme einen einzelnen Knopf auf jeder Etage haben, um einen Aufzug zu rufen, verfügen modernere Systeme über ein Zielregistrierungssystem, bei dem die Fahrgäste vor dem Betreten des Aufzugs ihr gewünschtes Zielstockwerk auswählen können und dann einem Aufzug zugewiesen werden. Die Zuweisung von Fahrgästen zu Aufzügen kann als Optimierungsproblem betrachtet werden, bei dem das Ziel darin besteht, die durchschnittliche Wartezeit der Fahrgäste zu minimieren.

In dieser Arbeit werden wir drei verschiedene Ansätze zur Lösung des Aufzugszuweisungsproblems für Aufzugssysteme mit einem Zielregistrierungssystem entwickeln. Der erste Ansatz ist ein gemischt-ganzzahliges lineares Programm (engl. mixed integer linear program, MILP), das das Problem zunächst als statisches Optimierungsproblem behandelt, bei dem alle Informationen über zukünftige Fahrgastankünfte im Voraus bekannt sind. Dieses MILP wird dann erweitert um es im dynamischen Optimierungsproblem, bei dem das System Entscheidungen nur auf der Grundlage der aktuellen Informationen über die Fahrgäste und Aufzüge treffen muss, auch einsetzen zu können. Gemischt-ganzzahlige lineare Programme sind exakte Methoden und können optimale Lösungen liefern, aber sie sind rechenintensiv und skalieren nicht gut mit der Größe des Systems. Für das statische Problem können wir ein gegebenes Szenario optimal lösen, während wir für das dynamische Problem einzelne Entscheidungspunkte optimal lösen können, aber die Gesamtlösung ist nicht garantiert optimal. Um die Bewegungen eines Aufzugs im Laufe der Zeit zu modellieren, verwendet das MILP einen Netzwerkflussgraphen und Miller-Tucker-Zemlin (MTZ) Ungleichungen um gültige Lösungen zu garantieren. Der zweite Ansatz, der sich auf das dynamische Problem konzentriert, ist eine greedy (gierige) Heuristik, um die Fahrgäste den Aufzügen nach einer gierigen Strategie zuzuweisen, wobei versucht wird, so viele Fahrgäste so schnell wie möglich abzuholen. Der Greedy-Algorithmus ist einfach und kann schnell Lösungen für das Problem finden, für komplexere Szenarien sinkt aber im Allgemeinen die Qualität der Lösung. Der dritte Ansatz ist ein Reinforcement Learning (RL) Ansatz, der einen Proximal Policy Optimization (PPO) Algorithmus verwendet und ebenfalls nur für die Lösung des dynamischen Problems entwickelt wurde. Um die Komplexität zu verringern, verwendet das Modell nicht die gesamte Zustandsinformation, sondern wertet Teilinformationen für jede gültige Aktion einzeln aus und wählt damit die

beste Aktion für jeden Aufzug aus. Das Modell wird für bestimmte Szenarien trainiert, die reale Verkehrsmuster simulieren.

Wir bewerten und vergleichen die drei Ansätze für verschiedene Szenarien, unterschiedliche Aufzugssystemgrößen und unterschiedliche Verkehrsmuster. Die Experimente zeigen, dass der dynamische MILP Ansatz für kleine Instanzen im Allgemeinen die besten Lösungen findet, aber rechenintensiv ist und bei größeren Instanzen nur schwer gültige Lösungen findet. Der greedy Ansatz ist in Bezug auf die Rechenzeit am effizientesten und liefert für die meisten der getesteten Instanzen sinnvolle Lösungen. Der Reinforcement Learning Ansatz reagiert empfindlich auf die Hyperparameter, welche für jedes System abgestimmt werden müssen, um gute Lösungen zu liefern. Die Leistung des RL Ansatzes übertrifft den Greedy-Ansatz für kleinere Systeme und Szenarien, in denen viele Fahrgäste gleichzeitig warten, verschlechtert sich jedoch mit zunehmender Systemgröße, insbesondere wenn die Anzahl der Aufzüge steigt.

Abstract

Modern society relies on elevators for vertical transportation in high-rise buildings. Elevator Group Control Systems (EGCSs) are multi-elevator systems designed to optimize passenger transportation within a building, increasing efficiency and capacity. While traditional EGCSs have a single button on each floor to call an elevator, more advanced systems have a destination registration system, where passengers can select their desired destination floor before entering the elevator and are then assigned to an elevator. The assignment of passengers to elevators can be seen as an optimization problem, where the goal is to minimize the average waiting time of passengers.

In this work, we will develop three different approaches to solve the elevator assignment problem for EGCSs with a destination registration system. The first approach is a mixed integer linear program (MILP) that tackles the problem first as a static optimization problem, where all the information about future passenger arrivals is known in advance, and then as a dynamic optimization problem, where the system has to make decisions only based on the current information about the passengers and elevators and the problem is re-solved when new requests become available. Mixed integer linear programs are exact methods and can provide optimal solutions, but they are computationally expensive and do not scale well with the size of the system. For the static problem, we can solve a given scenario optimally, while for the dynamic problem we can solve single decision points optimally, but the overall solution is not guaranteed to be optimal. To model the movement of an elevator over time, the MILP uses a network flow graph and Miller-Tucker-Zemlin (MTZ) constraints for ensuring feasible solutions. The second approach, focusing on the dynamic problem, is a greedy algorithm, which employs a heuristic to assign passengers to elevators based on a greedy strategy, trying to load as many passengers as fast as possible. The greedy algorithm is simple and generates solutions relatively quickly, but often struggles with finding good solutions for complex scenarios. The third approach is a reinforcement learning (RL) approach, which uses a Proximal Policy Optimization (PPO) algorithm and also only aims to solve the dynamic problem. To reduce complexity, rather than using the whole information of the state, the model evaluates only partial information for each valid action individually and picks the best action for each elevator. The model is trained for specific scenarios which simulate real-world traffic patterns.

We evaluate and compare the three approaches on different scenarios, different elevator

system sizes and different traffic patterns. The experiments show that the dynamic MILP approach generally performs the best for rather small instances in terms of solution quality, but is computationally expensive and struggles to find valid solutions for larger instances. The greedy approach is the most efficient in terms of computation time and provides good solutions for most of the tested instances. The reinforcement learning approach is sensitive to hyperparameters and needs to be fine-tuned for each system individually to provide good solutions. The RL approach outperforms the greedy approach for smaller systems and scenarios where many passengers are waiting at the same time, but gets worse with increasing system size, especially when the number of elevators increases.

Contents

CHAPTER 1

Introduction

Elevators are a common means of transportation in high-rise buildings or even smaller buildings with only few floors. The capability of vertical transport is crucial in modern society, as the construction of larger and larger buildings is pursued and travelling with elevators is often not a convenience but a necessity. Many people rely on elevators on a daily basis, especially elderly people and people with disabilities, who may not be able to use stairs at all. While transporting people to different floors is the most common use of elevators, they are also used in various other applications such as transporting goods in warehouses or factories. This makes elevator systems an important part of many people's lives, and thus they need to be efficient, reliable and accessible. While single elevator systems are common in smaller buildings, larger buildings often use multiple elevators to transport people to different floors. These systems are called Elevator Group Control Systems (EGCSs) and aim to greatly improve the efficiency of elevator systems by implementing dispatching algorithms that optimize the movement of elevators in the building. While there are different measurements of efficiency in EGCSs, the most common one is the minimization of the average waiting time (AWT) of passengers. This aims to provide a better user experience and to reduce the time passengers spend waiting for an elevator, while other measurements such as the minimization of the energy consumption of the elevators are also important especially in the context of sustainability and environmental concerns. When talking about elevator systems, one needs to distinguish between how much information the system has about the passengers. In the most basic and widely used form, each floor has a single button to indicate that a passenger needs a ride to a different floor and inside the elevator are multiple buttons, one for each floor, to indicate the desired destination floor. A more advanced form of EGCS has not just a single button on each floor, but rather two buttons, up and down, to signal the direction the passenger wants to travel. This provides more information to the system, which can then group passengers with the same direction of travel together in the same elevator. Taking it a step further, we can use an EGCS with a destination registration

system, where no buttons are inside the elevator, but rather a panel outside on each floor, so that a passenger can select their desired destination floor before entering the elevator, providing even more information to the system, and allowing for more efficient grouping of passengers. The assignment of waiting passengers to elevators can then be handled with a ticket system, where each passenger will be notified if the elevator is ready to take them to their desired floor. While the destination registration system can be more efficient than the basic system, it is also more complex and requires more advanced algorithms to find a sensible assignment of passengers to elevators. The problem of assigning passengers can be seen from two different perspectives, the offline (or static) perspective, where the information about the passengers (arrival and destination floors as well as arrival time) is known in advance and the system can plan the assignment of passengers to elevators in advance, and the online (or dynamic) perspective, where the information about the passengers is not known in advance and the system has to make decisions based on the current information iteratively over time. While the online perspective is more realistic and challenging, the offline perspective can be used as a benchmark to compare the performance of different approaches. Although there exists some literature on the topic of elevator group control systems with a destination registration system [2, 31, 34, 35, 27], the problem is still not extensively researched and there is still room for improvement in the field, especially with the rise of machine learning and reinforcement learning techniques, which have shown to be able to solve complex problems in various fields.

1.1 Aim of the Work

The aim of the work is as follows:

- Formalize the elevator control process with destination registration systems as a well-defined static and dynamic optimization problem.
- Design a mixed integer linear programming (MILP) model to solve the static optimization problem, as well as the dynamic optimization problem by iteratively solving the static problem with new information as new passengers arrive.
- Develop a greedy approach for the dynamic problem, where elevators are controlled by a heuristic that aims to load as many passengers as fast as possible.
- Design and implement a reinforcement learning approach that learns a policy based on the current state of the system and can assign actions to elevators.
- Experimentally evaluate and compare all developed approaches on different scenarios, different elevator system sizes and different traffic patterns.

1.2 Methodology

The methodology for this work is as follows:

1. **Literature Review.** In order to understand, improve and extend the already existing approaches for EGCSs with and without destination registration systems, an extensive literature research is done.

2. **Systematic Definition.** In order to formulate the problem in a well-defined, intuitive and practical manner, different, already existing, formulations are analyzed, unified and extended.

3. **Design of New Approaches.** The primary objective of this thesis is to introduce our own approaches for addressing these challenges by utilizing state-of-the-art methods. Specifically, we will model the problem as a mixed integer linear program and apply reinforcement learning techniques to tackle the problem. The former is an exact method and can provide optimal solutions, but requires more time as the instance size increases. The latter does not guarantee an optimal solution but can provide good solutions for large instances in reasonable time. Additionally, a greedy heuristic approach is designed to have a comparison to a naive approach, which is expected to be fast but to perform worse than the reinforcement learning approach.

4. **Implementation of Different Algorithms.** In order to apply and evaluate the designed approaches, all three methods are implemented to solve the defined problem.

5. **Data Generation.** To compare the different approaches in a scientific and objective way, data will be artificially generated, which will encompass a wide range of realistic scenarios, traffic patterns, and peak loads, enabling us to assess the performance of the algorithms under different conditions.

6. **Evaluation and Comparison.** The evaluation and comparison will concentrate on all implemented approaches. We will assess their effectiveness by utilizing metrics such as the average waiting time and examine the efficiency of each approach by considering factors such as runtime, which is especially interesting when thinking about real-world usage.

1.3 Key Results

Our main results of this work show that the static MILP approach is able to provide optimal solutions for tiny instances and, expectedly, outperforms the dynamic approaches. However, when increasing the size of the elevator systems and the length of the scenarios, the computational complexity of the static problem becomes too high to be practical. Solutions for large instances, if any are found by the MILP in reasonable time, are far from optimal. Comparing the dynamic approaches, the dynamic MILP approach is able to provide the best solutions for most of the tested instances, but also struggles occasionally to find valid solutions for large instances within the given time limit. The reinforcement learning approach shows strength in scenarios, where many orders arrive

within a short time, but also struggles as the elevator system size increases. The greedy approach is the most time efficient and provides good solutions for most of the tested instances, in some cases even outperforming all other approaches. In real-world scenarios the dynamic MILP approach would not be practical, as the computational complexity is too high to be able to solve the problem in real-time. The reinforcement learning approach and the greedy approach both show potential to be used in real-world scenarios, but the reinforcement learning approach needs to be fine-tuned for each specific elevator system and traffic pattern to be able to provide good solutions.

1.4 Structure of the Work

The remainder of this thesis is structured as follows: Chapter 2 provides an overview of the related work in the field of elevator group control systems with and without destination registration systems and focuses on the different approaches that have been used to solve the problem. Next, in Chapter 3, we describe the fundamentals of mixed integer linear programs and reinforcement learning. Chapter 4 provides a formal definition of the problem of controlling elevators in an EGCS with a destination registration system, as well as assumptions we made to be able to handle the complexity of the system. In Chapter 5, we present our mixed integer linear program approach for both the offline and online variant. The heuristic approach with a greedy strategy is introduced in Chapter 6. The last approach, using reinforcement learning, is presented and discussed in Chapter 7, where we define the Markov Decision Process and the architecture of the underlying neural network, with the observation and action space of the model. Chapter 8 presents the experiments and results of the different approaches. Finally, Chapter 9 concludes the work and provides an outlook on future work.

CHAPTER 2

Related Work

There are multiple variants of Elevator Group Control Systems (EGCSs) that have been proposed and tackled. The most prominent and most researched one is the single-car elevator system, where each elevator shaft has only one elevator car. While there exist systems with only one shaft and elevator car, having multiple elevator shafts is more common in large buildings, like hospitals or office buildings, where the number of floors is high and the number of passengers is large. These are also more interesting in terms of research, as the complexity of the system increases drastically with the number of elevator shafts. While the goal of an EGCS is to optimize the performance of the elevator system by seeing the problem as online (or dynamic) problem, where we adjust by re-optimizing the current assignments each time new information of arriving passengers is gathered, this problem can also be seen as an offline (or static) problem where all the information about the passengers, i.e. the arrival times, arrival floors and destination floors, is known in advance and the goal is to find an optimal assignment and scheduling of the elevators to minimize the performance metric. This is a more theoretical approach and can be used to compare the performance of different algorithms. This is most interesting when using exact methods, so we know the theoretical best performance of the system. In this chapter we will give an overview of the different types of elevator systems, a brief overview of offline approaches and the different online approaches to solve the problem of efficiently transporting passengers between floors.

2.1 Elevator Group Control Systems

Ever since the Otis Elevator Company developed the first elevator controller based on electronic microprocessors in 1979 [8], the research on classical Elevator Group Control Systems has been ongoing and many different algorithms and approaches have been developed. The EGCS is traditionally defined as a system that controls multiple elevator cars in a building, with each car having its own shaft. There exist hall buttons on each

floor, one for each direction, and car buttons inside the elevator car, which are used to select the desired floor of the passenger. The goal is to meet all the demands of the passengers and transport them between floors in an efficient way, which is done by dispatching elevators to pick up certain passengers, hence it is also called the elevator dispatching problem. The dispatching process needs to optimize some performance metric, with the most common one being the average waiting time (AWT) of passengers, which is the average time a passenger has to wait from the moment they request an elevator until they enter it. Another similar and common metric is the average travel time (ATT), which is the average time a passenger spends in the elevator from the moment they enter it until they leave it. The ATT is mostly used in combination with the AWT to calculate the passengers' average journey time (AJT), which is the sum of the AWT and the ATT. The problem of dispatching elevators in a building is in general NP-complete [14], and in real-world applications, this problem needs to be solved in real-time, which makes it even more challenging and required innovative solutions, especially as there are changing patterns of traffic like up-peak, where most passengers arrive at the ground floor and want to go up, or down-peak, where most passengers arrive at any non-ground floor and want to go down to the ground floor.

2.1.1 Offline Approaches

As already mentioned, the offline approach is to find an optimal (w.r.t. optimization criterions) assignment and scheduling of elevators, given all the information about the passengers in advance. As this is not applicable in real-world scenarios, this approach has not been researched as much as online approaches, however, it is still interesting as it can give a lower bound on the performance of the system. Sun et al. [32] proposed a two-level heuristic approach to solve the offline problem with full information about the passengers, where the first level is to assign the passengers to the elevator cars and the second level is to schedule the elevator cars with the assigned passengers to minimize the average waiting time. Exact approaches to find the optimal solution are by Shen et al. [30], where they used a branch-and-bound algorithm to find the optimal solution of different traffic patterns, and Xu et al. [42], who used a mixed integer programming approach to solve the problem for single elevator systems.

2.1.2 Exact Online Methods

In contrast to offline approaches, exact online methods tackle the online variant of the elevator dispatching problem by reoptimizing the current assignment of the elevators each time new information of arriving passengers is gathered. Exact methods from Pepyne et al. [25] and Levy et al. [22] solve each snapshot of the problem optimally, which is often computationally expensive and thus not suitable for large scale problems. Additionally, the optimal solution is only optimal for the current snapshot and does not take into account future snapshots, which can lead to suboptimal solutions in the long run.

2.1.3 Expert Systems

One of the first approaches to tackle the dynamic online problem were expert systems, which used a set of predefined rules, so called "expert rules". These rules can be based on the experience of engineers in the field of vertical transportation and are used for dispatching the elevators. Another approach to derive expert rules is to gather information by finding good solutions (w.r.t. the average waiting time) of different generated traffic instances of a building, e.g. using simulated annealing, and then comparing the good solution to solutions of conventional methods. Then crucial aspects and strategies can be extracted and formed as rules, which are then used to guide the elevator dispatching and improve the performance [36]. While expert systems brought significant advancements in the early days of elevator control, they have some limitations. The main limitation is the reliance on the predefined rules and knowledge of the system structure, which, when confronted with the dynamic and complex environment of a building, can not always adapt to the changing conditions [45].

2.1.4 Other Heuristic Approaches

Different approaches based on heuristics have been proposed to solve the elevator dispatching problem. These approaches are based on a set of rules that are used to guide the elevator system and make decisions. These can be based on multiobjective formulations with different criteria, like the waiting time, cabin-load factor, preferential zones and stop calls [21], or a combination of decision rules and a cost function that is used to calculate how good the set of suggested rules are [37]. An additional heuristic approach is to obtain solutions by using a tree search. The algorithm is searching a tree of possible actions and their outcomes, with heuristic estimations of the cost of each action to limit the search space. Child nodes of all nodes are possible actions from the node's state, which represent the assignment of not assigned hall calls to an elevator and also an idle action. To further limit the search space they only considered one elevator at each level of the search. After each node on the current level is assigned with the heuristic value, the best action (w.r.t. the heuristic value) is chosen and the next level of the tree is searched, this is done until all hall calls are assigned. One problem with such an approach is the time used to search the tree, which needs to be long enough to get a good approximation of the optimal solution, but also short enough to be able to make decisions in reasonable amount of time. The computational complexity increases with the number of waiting passengers on different floors, which makes this approach less suitable for large buildings with many floors and passengers [14].

2.1.5 Fuzzy Control

Fuzzy Control is another approach to solve the elevator dispatching problem, which is based on *Fuzzy logic*. *Fuzzy logic* is a multi-valued mathematical logic that aims to solve problems with the introduction of vagueness in reasoning. Rather than the usual binary (true or false) decision making of classical (Boolean) logic, *Fuzzy logic* uses degrees of

truth, which are represented by a value between zero and one. Thus making this approach suitable for complex systems which include uncertainty and imprecision [24].

A fuzzy controller for an EGCS consists of four main components: the fuzzification interface, the knowledge base, the inference engine and the defuzzification module. An important first step is to determine crucial state information, like the elevator position, passenger counts, wait times and target requests. The fuzzification interface then converts the crisp input values, i.e. information of the elevator systems state, into fuzzy values and uses the knowledge base to generate a degree of membership for each value. The inference engine then uses these membership values to determine which rules of the knowledge base are activated and generates a result for each value. These results are then combined in the defuzzification module to convert back into crisp outputs and actual control actions. The knowledge base, similar to the expert rules, is a set of predefined rules, based on expert knowledge, that are used to guide the elevator system [18].

A main advantage of *Fuzzy Control* is the ability to handle vague concepts but also the fast response time, which is crucial for real-time applications. However, the performance is dependent on the quality of the knowledge base and the membership functions, which often need to be tuned based on the specific scenario [45].

Basic *Fuzzy Control* systems are also often limited by being unable to reschedule the elevators until they completed the chosen action, making them less flexible and thus not as suitable for dynamic environments, where a crucial part is to adjust to new information. Fernández et al. [9] improved the *Fuzzy Control* approach by using a closed loop system, which reappraises the parameter inputs for the fuzzy inference in every iteration and thus allows new information to be incorporated into the decision making process and allows past assignments to be reevaluated and changed.

2.1.6 Genetic Algorithms

Another widely used approach to improve the performance of an EGCS is the use of *Genetic Algorithms* (GA). *Genetic Algorithms* are a type of evolutionary algorithm that are used to find good solutions to problems by using principles of natural selection and genetics, where a population of solutions is evolved over generations to find the best solution. Early approaches used GAs as a tuning tool to find the best control parameters for some heuristic based elevator dispatching algorithm with dozens of parameters, based on the state of the system [11]. Another strategy is to use the GA directly by encoding the decisions of each elevator as chromosomes of twice the number of floors, where each half represents the up calls and down calls respectively. The fitness is calculated based on an estimation of the time it takes for the elevator to complete all current calls [5].

2.1.7 Neural Networks

A logical next step to improve the performance of an EGCS is to use *Neural Networks* (NNs) for either directly controlling the elevators or to improve already existing approaches. Early approaches by Whitehall et al. [40] describe a general method of prediction. It

estimates the remaining response time for an elevator car to reach a hall call and chooses the elevator to assign accordingly. This system is able to train the neural network while the elevator system is in operation, which allows the system to adapt to changing conditions of traffic. While the adaptability of the system is a great advantage, it is limited to long term changes, rather than being able to do drastic changes in real-time. A system that uses NNs as improvement for an existing system was proposed by Naoki et al. [17] by combining a neural network with fuzzy logic to a fuzzy neural network, where the NN predicts the specific traffic patterns of arriving passengers and thus changes control parameters for the fuzzy logic. This approach is able to handle sudden changes in the environment with the fuzzy reasoning while still being able to adapt to long term changes with the neural network. Another approach uses the output of the neural network directly as input for a fuzzy logic system. This is done by transforming the output signal via an activation function to values between zero and one, which then represents the degree of membership for the fuzzy logic and thus control the actions of the elevator system. While this approach can be very flexible, it also is limited by the quality of the membership functions and the chosen fuzzy rules, which can drastically affect the performance of the system [45].

2.1.8 Reinforcement Learning

A more recent approach to solve the elevator dispatching problem in an efficient manner is to use *Reinforcement Learning* (RL). *Reinforcement Learning* is method of machine learning where an agent learns to make decisions by interacting with an environment and receiving rewards or penalties for its actions. The agent learns to maximize the cumulative rewards over time by learning a policy, which is a mapping from a state to an action. A more detailed explanation of *Reinforcement Learning* will be given in Chapter 3.2, as it is a crucial part of this thesis. The first to tackle the ECGS problem with *Reinforcement Learning* were Crites et al. [6]. They created the state space by including information about the pressed buttons, hall call buttons on each floor and car buttons in each elevator car, the approximated position of the elevator and the direction of the cars. The action space was defined by either “move up” or “move down” if the elevator has stopped at a floor, or “stop at next floor” or “continue past next floor” if the elevator is moving.

Zhou et al. [46] used a method called Genetic Network Programming (GNP), an evolutionary computation method based on Genetic Algorithms and Genetic Programming which uses a direct graph structure as genes, and combined it with reinforcement to combine the advantages of both methods.

With the evolution of RL algorithms, Deep Neural Networks are merged with RL to create *Deep Reinforcement Learning*, which has been shown to outperform traditional RL algorithms in many tasks. Wei et al. [39] use a *Deep Reinforcement Learning* method called asynchronous advantage actor–critic (A3C) to solve the elevator dispatching problem, which uses a deep convolutional and recurrent neural network to learn the optimal dispatching of elevators. They used the same state construction as Crites et al. [6],

while using a simpler action space, which only includes the action to stop at the next floor or continue past it for each elevator. The reward function was defined as the average squared waiting time of all passengers which are currently waiting for an elevator. The results showed an increase in performance compared to traditional algorithms.

2.2 Elevator Group Control System Variants

While the traditional EGCS is the most common and most researched system, there exist other variants of elevator systems that have been proposed and tackled. With the increasing demand of more efficient elevators, new systems have been developed to meet the requirements of modern buildings. In this section we will describe some of the most common variants of elevator systems, which contain more information about the system or even have a different building structure than the traditional EGCS.

2.2.1 Destination Registration Systems

One major drawback of the traditional EGCS is the inefficiency when it comes to handling multiple passengers waiting on the same floor but having different destination floors. This is due to the fact that the elevator system does not know the destination of the passengers when they arrive at the elevator hall but only when all passengers have entered the elevator car and pressed their respective floor button. This can lead to inefficiencies like having passengers with the same destination floor in different elevator cars, resulting in unnecessary stops which could have been avoided by grouping the passengers together. By adding a destination terminal at each floor, passengers can enter their desired destination floor before entering the elevator car, which allows the elevator system to make its decision more informed and efficient. This system is called *Elevator Group Control System with a Destination Registration System* (EGCS-DRS) and will be thoroughly explained in Chapter 4.

Early works on this variant of the EGCS done by Beielstein et al. [2] already made use of neural networks in combination with evolution strategies to judge the performance of the elevator system. The neural network was used to predict the best parameter configuration for the evolution strategy. Sorsa and Siikonen [31] had a different approach, where they mainly focused on the up-peak scenario. They calculated the maximum handling capacity of an elevator group and tried to balance (and thus raise) the handling capacity of all elevators by dividing all destination floors between the elevator cars, such that only one car stops at each floor to deliver passengers. Additionally they showed that when overlapping of the served floors is allowed, i.e., multiple cars can deliver passengers to the same floor, the handling capacity is more even across elevators but also smaller. Different approaches were proposed by Tanaka et al. [34, 35], where they used a branch-and-bound algorithm to find good elevator assignments, and by Ruokokoski et al. [27], where they formulated the problem as a mixed integer linear program. These are different to the previous approaches, as they are both exact approaches to find the optimal solution for snapshots of the system, so when the algorithm terminates, the solution for the current

state is guaranteed to be the best possible, but in the long run might be suboptimal as it does not have full information about the future. This comes with the drawback of being computationally expensive, especially for large buildings with many floors and passengers. To simplify the problem they only considered a single elevator shaft with a single elevator car, which reduces the complexity because there is no need to coordinate multiple elevator cars. Although we focus only on multi elevator systems, this approach is still interesting as it shows the potential and limitations of exact methods in elevators systems. These limitations make it unsuitable for use in a real-time scenario, as we will discuss in Chapter 5 about our mixed integer linear programming approach.

2.2.2 Double-Deck Elevators

A variant of the traditional EGCS which not only has more information about the system and the passengers but also a different building structure is the double-deck elevator system. In contrast to the single-deck, double-deck elevators are two elevator cars stacked on top of each other in the same shaft, allowing more passengers to be transported in one trip, while also allowing to serve two adjacent floors at the same time. This introduces a new challenge at the ground floor, where passengers can choose to go to the upper or lower car, which can lead to inefficiencies and unnecessary stops if not supervised by the system, thus making it almost necessary to use destination registration systems (or sometimes called *Destination Floor Guidance System* (DFGS)) to gather information about the passengers' destinations, so the system can make informed decisions and guide the passengers to the correct elevator car. Both Zhou et al. [47] and Yu et al. [43] used GNP to tackle the double-deck elevator system with a destination registration system, where they combined GNP with Reinforcement Learning and Ant Colony Optimization (ACO), respectively. GNP is good at a more broad global search, but struggles with converging to good solutions, thus using it in combination with RL or ACO to intensify the local search leads to better results.

2.2.3 Multi-Car Elevators

Another variant which has been proposed is the multi-car elevator system, where multiple elevator cars are in the same shaft. This system is, again, more complex than the traditional EGCS, as the elevator cars not only need to be coordinated across shafts but also especially within the shaft. Transporting passengers between floors is not as straight forward as the cars can block or delay each other. As for the other variants, the multi-car elevator system has been tackled with different approaches, like a heuristic approach by Valdivielso et al. [38] who mainly divided the floors into zones to be serviced by specific elevators, or using a genetic algorithm [15] to tackle the elevator assignment. Further research has been done by using GNP [44] or reinforcement learning to select from a set of predefined strategies [16].

CHAPTER 3

Methodology

This chapter will present and explain the methodology used in our approaches to solve the elevator dispatching problem for EGCS. We will first describe the basics of mixed integer linear programming (MILP), followed by discussing Reinforcement Learning (RL) and the applied Proximal Policy Optimization (PPO) policy gradient method.

3.1 Mixed Integer Linear Programming

In this section we introduce the basics of Mixed Integer Linear Programming (MILP) based on Chapter 1 and 10 of the work of Bertsimas and Tsitsiklis [3] and Chapter 1 of the book by Wolsey [41].

A mixed integer linear program is used to mathematically model optimization problems and is a variant of a linear program (LP), with the aim to minimize or maximize a linear objective function subject to linear equality and inequality constraints. A general LP contains of a set of decision variables $x = (x_1, \ldots, x_n)$, a cost vector $c = (c_1, \ldots, c_n)$ and a linear cost function $c'x = \sum_{i=1}^{n} c_i x_i$ and linear equality and linear inequality constraints. The general form of an LP for a minimization problem is given by:

$$\begin{aligned} \text{minimize} \quad & c'x && (3.1)\\ \text{subject to} \quad & Ax \geq b && (3.2)\\ & x \geq 0 && (3.3) \end{aligned}$$

where formulation 3.1 is the objective function to be minimized, combining the decision variable and the cost vector. Formulation 3.2 is the set of linear inequality constraints, where A is a $m \times n$ matrix, with m being the number of constraints and n the number of decision variables, and b is a $m \times 1$ vector. Formulation 3.3 sets the domain of the

decision variables. Note that it is generally possible to convert a maximization problem into a minimization problem by multiplying the cost vector with -1, and vice versa. It is also possible to convert an equality constraint $a_i'x = b_i$ into inequality constraints by splitting them into two inequality constraints $a_i'x \leq b_i$ and $a_i'x \geq b_i$ to get the previously defined general form of an LP.

The goal of an LP is to find the values of the decision variables that minimize the objective function while satisfying the (in-)equality constraints. If we can find values that satisfy all constraints, we say that the LP is feasible, and if we can find values that satisfy all constraints and minimize the objective function, we say that the solution is an *optimal feasible solution*. It is however possible that the LP is either infeasible, meaning that no values of the decision variables can satisfy all constraints, or unbounded, which means that the objective function can be made arbitrarily small (or large), so the optimal cost is said to be $-\infty$ (or ∞).

One of the main advantages of LPs is, that they can be solved efficiently, even for large instances, in polynomial time. There are different algorithms used for the solving process, with the most common being the simplex algorithm and the interior-point method. While the simplex is the most widely used algorithm, it has a worst-case exponential time complexity, but in practice it is very efficient. The interior-point method is generally slower than the simplex algorithm, but has a polynomial time complexity, which makes it better suited for worst-case scenarios.

A mixed integer linear program is an extension of a linear program, where some (but not all) of the decision variables are restricted to be integer values. We can formulate the general form of a MILP as follows:

$$\begin{aligned} \text{minimize} \quad & c'x + d'y && (3.4)\\ \text{subject to} \quad & Ax + By \geq b && (3.5)\\ & x \geq 0 && (3.6)\\ & y \geq 0 \text{ and integer} && (3.7) \end{aligned}$$

where x, c', A and b are the same as in an LP, $y = (y_1, \ldots, y_p)$ is another set of decision variables, $d = (d_1, \ldots, d_p)$ is another cost vector and B is a $m \times p$ matrix with p being the number of integer decision variables.

While LPs can be solved efficiently, MILPs are generally harder to solve and are in general NP-complete. The main reason for this is that the integer constraints make the feasible region of the problem non-convex and restricting the solution space to discrete points, making it harder to find an optimal solution.

3.2 Reinforcement Learning

This section will introduce the basics of Reinforcement Learning (RL) based on Chapter 1, 3 and 13 of the work of Sutton and Barto [33], which is also great source for a more detailed and in depth explanation of RL.

Reinforcement Learning is one of the three main machine learning paradigms, next to supervised and unsupervised learning, which mainly differ in the way they learn from data:

- Supervised leaning is learning from a labeled dataset, which is given to the model to learn from. This dataset contains input-output pairs, where the input is the data and the output is the desired prediction. The goal of the model is to learn from the dataset to make accurate predictions on new, unseen data.
- Unsupervised learning takes unlabeled data as input, with a common goal of finding hidden patterns in the data. For example, clustering techniques fall into this category, where the goal is to group similar data points together.
- Reinforcement learning algorithms learn by interacting with an environment, where the agent learns to take actions to maximize a reward signal. The processes of the environment are not known to the agent, so it must learn them by trial and error.

While reinforcement learning may seem similar to unsupervised learning, because it also learns data without a clear correct answer, the main difference is that uncovering hidden patterns in the environment itself does not solve the problem of maximizing the reward signal, but it still may be helpful. The agent must be able to learn from its own experience interacting with the environment. Another important difference between RL and the other two paradigms is that reinforcement learning has to deal with the trade-off between exploration and exploitation. Exploration is the process of trying out new actions to learn more about the environment and thus may find new, favorable actions, while exploitation is the process of favoring actions that have already been shown to yield high rewards. The agent has to find the right balance between exploration and exploitation to find good solutions and to not get stuck in suboptimal solutions.

Reinforcement learning normally consists of different elements:

- The *environment*, which is the world the agent interacts with. The environment is defined by a set of states, actions, and rewards, and the agent interacts with the environment by taking actions and receiving rewards. The environment changes its state based on the actions the agent takes.
- A *state*, which is a representation of the environment at a certain time.

- An *observation*, which is a representation of a state that the agent can observe. The observation can be the same as the state, but it can also be a partial representation of the state, where some information is hidden from the agent.
- An *agent*, which learns about the environment by interacting with it. The agent takes actions in the environment and receives rewards based on these actions. The agent's decisions can either be based on a value function, which estimates the value of an action (for a specific state), or a policy, which selects actions without estimating the value of the action by learning the parameters of the policy.
- An *action*, which is a decision the agent makes to interact with the environment. The action can be discrete, meaning that the agent can choose from a finite set of actions, or continuous, meaning that the agent can choose from a continuous set of actions.
- A *reward signal*, which is a value the agent receives from the environment after taking an action. The reward signal defines good and bad actions and is used to guide the agent to take actions that maximize the reward signal.

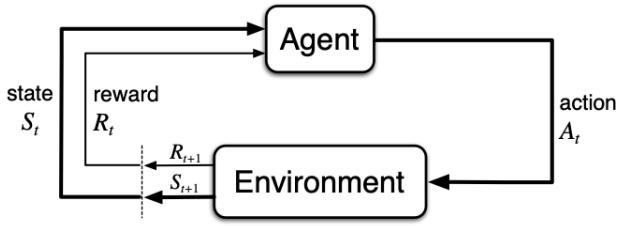


Figure 3.1: The agent-environment in a Markov decision process. The agent interacts with the environment by taking actions and receiving rewards. The environment then changes its state and the agent receives a new state and reward [33].

A reinforcement learning problem is generally formulated as a Markov Decision Process (MDP), which can be formally defined as follows:

Definition 1 *A Markov decision process is a four-tuple $\langle S, A, P, R \rangle$, where S is the set of states, A is a function that maps states to actions, P is the transition probability function $P\colon S \times S \times A \to [0, 1]$, which gives the probability $P(s' \mid s, a)$ of transitioning from state s to state s' given action a, and R is the reward function given by $R\colon S \times S \times A \to \mathbb{R}$, which gives the reward $R(s, s', a)$ of taking action a in state s resulting in state s' as scalar value.*

Figure 3.1 shows the interaction between the agent and the environment in a MDP. The agent receives at each timestep t the current state S_t and the reward R_t from the environment, and then selects an action A_t based on the agent's policy. The environment

then changes its state to S_{t+1} based on the chosen action and the agent receives the new state and reward.

A policy for RL algorithms is formally defined as a mapping from states to probabilities of selecting each possible action. We say that if an agent follows a policy π at time t, the probability of taking action a in state s is given by $\pi(a \mid s)$. A value function $v_\pi(s)$ is then the expected return the agent receives when starting in state s and following policy π. Formally we define $v_\pi(s)$ for MDPs as:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \forall s \in S, \tag{3.8}$$

where γ is the discount factor, which is a value between 0 and 1 that determines the importance of future rewards compared to immediate rewards, and $\mathbb{E}_\pi$ is the expected value of a random variable when the agent follows policy π. To define a value to a specific action a which is taken in a state s following policy π, we define the action-value function $q_\pi(s, a)$ as:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \tag{3.9}$$

The goal of the reinforcement learning agent is to find the optimal policy π^*, which maximizes the expected return. The optimal value function $v_{\pi^*}(s)$ is then defined as

$$v_{\pi^*}(s) = \max_\pi v_\pi(s), \forall s \in S, \tag{3.10}$$

and similarly the optimal action-value function $q_{\pi^*}(s, a)$ is defined as

$$q_{\pi^*}(s, a) = \max_\pi q_\pi(s, a), \forall s \in S \text{ and } a \in A. \tag{3.11}$$

3.3 Proximal Policy Optimization

While there are many different reinforcement learning algorithms, in this thesis we will focus on the Proximal Policy Optimization (PPO) algorithm [29], which is an actor-critic method. Actor-critic methods combine both principles of value-based methods and policy-based methods. Value-based methods focus on estimating an optimal action-value function $q_{\pi^*}(s, a)$, while policy-based methods focus on finding the optimal policy π^* directly. For policy-based methods a policy's parameter vector θ is introduced, which is used to define $\pi(a \mid s, \theta) = Pr\{A_t = a \mid S_t = s, \theta_t = \theta\}$ as the probability that the agent takes action a in state s with the policy parameters θ. The policy parameters can then be improved by using gradient ascent

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta), \tag{3.12}$$

where the parameter vector is updated in the direction of the gradient by the approximation of some objective function $\nabla J(\theta)$.

The actor in the actor-critic method is the policy, which is used to select the appropriate actions given a state, which is updated by using the policy gradient method but instead of a proportional update of the policy parameters, the critic is used to evaluate the new policy and adjust the policy gradient. In PPO the idea is to prevent the policy from changing too much at once. Policy-based methods change the parameter vector θ, but the change is not proportional to the change of the policy π_θ itself, this can lead to large fluctuation in the policy and thus to bad performance. PPO uses a method called trust region policy optimization (TRPO) [28], where the similarity of both policies is measured with the Killback-Leibler divergence. The PPO algorithm uses a clipped version of the TRPO, where the objective function is modified to prevent the policy from changing too much at once.

Apart from the good performance of PPO, it is known in general to be relatively easy to tune, as there are only a few parameters that need to be adjusted, and the sensitivity of these parameters is low compared to other algorithms. This makes PPO a good choice for many reinforcement learning problems, which is why it is currently a state-of-the-art algorithm in RL and is used in this thesis to solve the elevator dispatching problem for EGCS-DRS.

CHAPTER 4

Problem Description

Elevator Group Control Systems with a destination registration system extend the traditional EGCS by allowing passengers to select their destination floors before entering the elevator. While the idea of such systems was first proposed in the 1960s, the technology to implement it was not ready until the 1990s, when Schindler Elevator first introduced a commercial system with a destination registration system. This was done by a central control panel (10-button keyboard) in the elevator hall of each floor. With this new feature, the elevator system gains more information about the passengers and can make more informed decisions about which elevator to assign to which passenger. Not only can passengers be assigned and grouped more efficiently on each floor, but also other information can be derived, like the number of passengers waiting at each floor and the number of passengers in each elevator, assuming that passengers behave as expected and do not miss entering the elevator. For six car elevator systems, the theoretical handling capacity can be increased by 50% when using a destination registration system compared to traditional systems. This can lead to a reduction in the number of elevators needed, especially in high rise buildings, or generally to a reduction in the waiting time for passengers when using the same number of elevators under high traffic conditions [13].

In this chapter, we will introduce the specifications and assumptions related to the elevator system that will be referenced in the following chapters and used for our simulations and experiments. This clarification is crucial due to the wide variety of elevator systems documented in the literature, each with its own set of specifications and assumptions. Additionally, there is currently no general standard for Elevator Group Control Systems, particularly for those utilizing a destination registration system. Section 4.1 will provide a detailed description of the problem we are trying to optimize, while Section 4.2 will list all assumptions we make about the elevator system and the passengers. These assumptions are necessary to simplify the problem and make it more manageable.

4.1 Problem Specification

The problem we are trying to solve is to optimize the assignment of passengers to elevators in an EGCS-DRS and to control the movement of the elevators accordingly. An elevator system consists of k identical elevator cars, each in its own shaft, and m floors. Each floor has an elevator hall from which all elevators are accessible, i.e., each elevator can stop at every floor and (un)load passengers. The elevator cars have a given capacity C, which limits the number of passengers that can be transported at once. Each passenger p arrives at an elevator hall on some specific (arrival) floor a_p at some time c_p, registers their desired destination floor b_p on the control panel and is assigned a ride ID. This ride ID is used to inform the passenger when the elevator arrives with which they are supposed to travel, which is in contrast to other definitions in the literature ([13, 20]), where the passengers are immediately assigned to an elevator number, when they register their destination floor, and should enter the elevator with this number as soon as it arrives and opens its doors. This, however, is a restriction for the system as the elevator assignment is fixed and cannot be changed in the future, even if a more efficient assignment would be possible. With our definition we remain flexible until the elevator arrives and thus use the latest information to make the best decision, even though this might be less convenient for the passengers. Each elevator has (potentially multiple) *trips*, which are defined as travels in one direction, i.e., a trip is either an upward, when the starting floor is lower than the ending floor, or downward travel, if the ending floor is lower than the starting floor. Each trip can have multiple stops on floors between the starting and ending floor, where passengers can enter or leave the elevator, but on each trip, all passengers in the elevator have to have the same travel direction. Additionally, passengers can also arrive in passenger groups for a joint ride (to the same destination), by specifying their group size at the control panel, a group must travel in the same elevator at the same time. This can also be used when transporting bulky things, to reserve enough space. We call such an entity an *order*, which can be a single passenger, a group of passengers or an item to be transported. The goal is to minimize the total time it takes for all passengers to board the elevator which is equivalent to minimizing average waiting time. With the group size incorporated, the AWT is then defined as follows:

$$\text{AWT} = \frac{\sum_{p \in P} (u_p - c_p) \cdot d_p}{\sum_{p \in P} d_p}, \tag{4.1}$$

where P is the set of all orders, u_p is the time the passengers of order p enter the elevator, c_p is the time the order p arrives at the elevator hall and d_p is the group size of order p. In our model all events that occur, like the arrival of orders, orders entering the elevator or the stop of an elevator, can occur at any moment. For practical purposes, the times of occurrence of these events are rounded to the nearest second. This approach maintains a realistic representation of the system's dynamics while making it more manageable.

Tackling this problem for both the offline and online case we need to define some instance-dependent constants to represent information from the state of the system. As both

cases are similar but have some differences, we will first briefly define common constants, which both problem variants need, and in Subsection 4.1.1 and Subsection 4.1.2 we will define case specific constants for the offline and online case respectively. The following constants are used in both cases:

- $k \in \mathbb{N}$: the number of elevator cars in the system;
- $E = \{1, \ldots, k\}$: the set of all elevator cars;
- $m \in \mathbb{N}$: the number of floors in the building;
- $F = \{1, \ldots, m\}$: the set of all floors;
- $C \in \mathbb{N}$: the capacity of each elevator;
- $T(j, j') : F \times F \to \mathbb{R}^+$: the time each elevator needs to move from floor j to floor j' without a stop inbetween;
- $T_{\mathrm{L}} \in \mathbb{N}$: the average duration for one passenger to board or exit an elevator.

These constants describe only the basic information about the elevator system, whereas additional information about the orders and elevators will be defined separately as required in the following subsections for the offline and online case. To solve these problems we also need to define the structure of a solution. A solution is represented as a sequence of *system-actions*, where each system-action is a list of *elevator-actions*, one for each elevator. An *elevator-action* is the information about the next move an elevator makes. Each elevator-action *act* is structured two-fold:

- The target floor $act^{\mathrm{target}} \in F$ the elevator is supposed to make its next stop;
- the orders act^{load} to load into the elevator from the floor the elevator makes its current stop.

These elevator-actions cover all possible actions an elevator can make, as they can either move to a different floor or stay at the current floor, and they can interact with the orders at the current floor by loading them into the elevator. Combining these actions for all elevators in the system gives us a system-action. Note that each elevator-action must be feasible and abide by the rules of the elevator system defined above and the assumptions we make about the system in Chapter 4.2. The disembarkation of passengers is not explicitly modeled in the system-actions, as it is assumed that all passengers leave the elevator at their destination floor.

4.1.1 Offline Problem

As already briefly mentioned in Chapter 2, this problem can be seen either as an offline problem or as online problem. In the offline case, we have a static problem where we know all the information about the system and the orders (including their arrival times and destination floors) in advance and can make informed decisions and plan the whole trip of each order perfectly without any uncertainty about the future. A major advantage of the offline case is that we can utilize the information about future orders to decrease the downtime of the elevators, for example, sending idle elevators to arrival floors before orders actually arrive and enter their destination floor into the control panel could remove the waiting time for the order which otherwise would have to wait for the elevator to react to the new information. Another possibility, exclusive to the offline case, is that waiting for multiple orders to arrive is feasible, as we know when they will arrive and where they want to go, so if multiple orders show up in close temporal proximity, it may be beneficial to not immediately load the first order into the elevator, but wait for the other orders to arrive and only load the optimal combination of orders into the elevator. This way, we utilize the capacity of the elevator efficiently and it further allows us to group orders with identical destinations together, which can also decrease the number of stops the elevators have to make. Because we cannot (generally) predict when people will arrive and where they want to go, we use this case only as a theoretical case, but it provides a valid lower bound for solutions for the online case.

In general, we will tackle this problem by solving a scenario, a specific time window where different numbers of orders arrive at different times. To work with the offline case, we need to define the initial state of the system by extending the definitions from the previous section:

- $n \in \mathbb{N}$: the number of orders arriving in this scenario;
- $P = \{1, \ldots, n\}$: the set of all orders (i.e., passenger groups) in this scenario;
- $a_p, b_p \in F$, $\forall p \in P$, $a_p \neq b_p$: the arrival and destination floors of order p, respectively;
- $c_p \geq 0$, $\forall p \in P$: the arrival time of order p;
- $d_p \in \{1, \ldots, C\}$, $\forall p \in P$: the group size of order p;
- $f_i \in F$, $\forall i \in E$: the starting floor of elevator i.

Here we added some crucial information about the orders for the offline case. As we have full knowledge of the scenario, we know exactly the arrival floor and time, the desired destination floor and the group size of each order considered in this scenario. Another important aspect is the starting floor, each scenario starts with some initial position of each elevator, which in most cases would be the ground floor, but can be any floor of the system. A solution to a scenario is then defined as a sequence of system-actions.

4.1.2 Online Problem

The online case is more realistic, as it tries to mimic a real-world scenario where we get information about the orders only when they arrive at the elevator hall. We need to make decisions based on the information we have at the time, which can lead to suboptimal results, even if the decision would be optimal for the current state. For a clearer view on the difference, consider the following, simple example: We have an elevator system with only one elevator with a capacity of five and two orders. The first order has the group size of 1, arrives at floor 1 at time 1 and wants to go to floor 4. The second order has the group size of 5, arrives at floor 3 at time 5 and wants to go to floor 1. For simplicity, we say it takes five seconds to go from one floor to an adjacent floor and (un)loading orders takes one second per passenger of the order. In the online case, we would start at time 1 and need to make our first decision, which would be to load the first order at floor 1 into the elevator, as we cannot know that the next order will arrive at floor 3. We would deliver the first order at floor 4 at time 18 (one second to load the order, then go from floor 1 to 4 in 15 seconds and unload for one second) and then go to floor 3 to pick up the second order at time 23 and deliver it to floor 1. We can calculate the AWT for this scenario by using Equation 4.1, where we get $\frac{(1-1)\cdot 1+(23-5)\cdot 5}{1+5} = 15$. For the offline case we would know all the arrival times beforehand and could see that it would be better to first go to floor 3 and pick up the second order at time 11 and then go to floor 1, deliver the second order and pick up the first order at time 31. This would lead to an AWT of $\frac{(11-5)\cdot 5+(31-1)\cdot 1}{5+1} = 10$. This example shows that the online case can lead to suboptimal results, even if the decision would be optimal for the current state. This is the challenge we face when trying to optimize the elevator system in an online setting.

When considering the online problem for the EGCS-DRS, we basically work with a scenario similiar to the offline problem, but rather than dealing with all the information at once, we take system-action only at specific points in time. These decision points are defined to be at the following events:

- when a new order arrives; and
- when an elevator reaches stops at a floor and passengers have (possibly) been dropped off.

While one could argue that the information about the system does only change when new orders arrive and not when an elevator stops at a floor, thus needing the decision points only to be at new arrivals, some approaches prefer to plan only to the next stop and then choose the next system-action at the next decision point. For us this is important as system-actions only include the immediate next stop for each elevator and not the whole trip to all waiting orders.

We also need to clearly extend the initial *state* definition with all the information about the elevator system and orders known at the current time (and decision point). This is defined as follows:

- $t \in \mathbb{N}$: the current time;
- $f_i^{\text{last}} \in F, \forall i \in E$: the last floor elevator i has stopped;
- $t_i^{\text{last}} \geq 0, \forall i \in E$: the time elevator i has stopped at f_i^{last} and unloaded all affected orders;
- $f_i^{\text{next}} \in F, \forall i \in E$: the floor elevator i is planning to stop next, which is equal to f_i^{last} if the elevator is currently not moving;
- n^{elev}: the number of current orders in elevators;
- n^{wait}: the number of current orders waiting for an elevator;
- $P^{\text{known}} = \{1, \ldots, n^{\text{elev}} + n^{\text{wait}}\}$: the set of all orders either waiting or loaded in an elevator;
- $P^{\text{elev}} \subseteq P^{\text{known}}, |P^{\text{elev}}| = n^{\text{elev}}$: the set of all orders currently in an elevator;
- $P^{\text{wait}} \subseteq P^{\text{known}}, |P^{\text{wait}}| = n^{\text{wait}}$: the set of all orders currently waiting for an elevator;
- $a_p, b_p \in F, \forall p \in P^{\text{known}}, a_p \neq b_p$: the arrival and destination floors of order p, respectively;
- $c_p \geq 0, \forall p \in P^{\text{known}}$: the arrival time of order p;
- $d_p \in \{1, \ldots, C\}, \forall p \in P^{\text{known}}$: the group size of order p;
- $e_p \in E, \forall p \in P^{\text{elev}}$: the elevator the order p is currently in.

Here we introduce specific information for decision points needed to solve the dynamic problem. We introduce additional information about each elevator to correctly represent the state of the system. We need the last stopped floor and last stop time, as well as the next planned stop, to be able to calculate the time of the next stop of the elevator, if it is currently in motion. Additionally we need the specific information about currently waiting and currently loaded orders, to correctly calculate the load of each elevator and know which floors the elevators need to stop at.

4.2 Assumptions

As we can never fully capture the complexity of a real-world elevator system, we need to make assumptions about the inner workings of the systems to simplify the problem and make it manageable for our approaches. This section will list all assumptions we make about the system and the passengers.

A main factor in the complexity of the problem is the allowed behavior of the elevators when serving passengers. In real-world systems, elevators have different restrictions and

rules they need to follow and not all systems behave the same, thus making it necessary for us to define the behavior of the elevators.

1. Each elevator car has an initial floor at which it starts.
2. If there are no known orders to be served, the car stays at the last floor it stopped.
3. The car does not stop at a floor without loading or unloading any passengers there.
4. The passengers of an order enter an elevator car, if the elevator car stops at the passengers' current floor and the order is called with its ride ID. The passengers always enter the elevator they are assigned to take.
5. If the car stops at a floor, all (and only) passengers whose destination floor is this floor get off the elevator car.
6. People only use a single elevator to reach their destination floor.
7. Once an elevator stops at a floor, the selection of passengers who should enter this elevator at this floor cannot change.
8. The elevator car is not allowed to pass a floor without stopping in which an on-board passenger wants to get off.
9. The elevator car cannot change the direction before all the calls in the current movement direction have been completed.
10. A moving elevator can only stop at its originally scheduled target floor or be rescheduled to a different target floor that is at least two floors away from the current position in the current movement direction; i.e., an elevator cannot stop always immediately but needs to decelerate.
11. It takes a constant time T_{L} per passenger for each order to get on or off a car; no additional time is counted in for the arrival of passengers as well as time it would take to open and close the doors.
12. The travel time of an elevator to move from floor a to a different floor b follows a function $T(\Delta)$ of both floor numbers given, with $\Delta = |a - b|$.

Assumptions 1–3 are made to specify the behavior of the elevators. While Assumption 1 is a general assumption about the starting position of the elevator at the begin of a scenario, Assumption 2 specifies the behavior of the elevator when there are no passengers to be served. While one could argue that in specific scenarios it might be better to move the elevator to a different floor (e.g. the ground floor), this assumption is the most general and allows for the most flexibility in the system. Assumption 3 further restricts unnecessary stops of the elevator, which could be confusing for the passengers and are not necessary.

Assumptions 4 and 5 basically restrict the behavior of the passengers to make no mistakes by pressing wrong buttons or changing their mind. This helps to make the problem more manageable, so we always know that the information about the passengers is correct and our information about the system, e.g., the number of passengers in each elevator, is always up to date and accurate. Assumption 6 further restricts passengers so that they cannot make a detour by first traveling to a different (non destination) floor and then to their destination floor.

Assumptions 7–9 aim to prohibit elevator operations which are psychologically undesirable for passengers. Once an elevator stops and opens its doors, passengers should know if they should enter the elevator or not, even if new information arises after the elevator has stopped (Assumption 7). Assumption 8 and Assumption 9 both ensure that once an order has entered the elevator, the elevator will only move towards their destination floor and not away from it, which otherwise could be confusing for the passengers. Furthermore, they also state that all passengers loaded into the same elevator car for the same trip must have the same desired direction of travel.

Assumption 10 is made to ensure that the elevator system behaves more realistically, as in real-world scenarios the elevator needs to decelerate before stopping at a floor and, especially if there are passengers inside the elevator, it is not possible to stop immediately.

Because we work with a simulated elevator system, we also need to specify some time constraints for certain processes of the elevators. Assumption 11 specifies that the (un)loading of orders takes constant time for each passenger of this order, i.e., larger orders need longer to (un)load, and that we further ignore the time for opening and closing doors, which is again a simplification of the real world. The travel time of an elevator between two floors is given by Assumption 12, which we define as a function of the distance between the two floors.

CHAPTER 5

MILP Formulations

In this chapter, we present and discuss our approach to solving the elevator assignment problem for EGCS-DRS with mixed integer linear programming formulations. Firstly we introduce some basic definitions used for both offline and online problems. Then we present the MILP formulation for the offline problem. Lastly we extend and adapt it to the online problem.

5.1 General Definitions

One of the main challenges in the formulation of the MILP model is that we need to model the entire route of each elevator so that we can calculate an optimal assignment. A route of an elevator is the sequence of all its trips from the start to the end of a scenario, so basically all the stops it needs to make to deliver all its orders. When tackling the offline and online problem, we, in general, need multiple trips for each elevator to serve all orders, and because we need some kind of ordering for the trips, we will enumerate them, starting at trip one, and define odd trips to always go upward, i.e., $j < j'$ holds for starting floor j and destination floor j', and even trips always go downward, i.e., $j > j'$ holds for starting floor j and destination floor j', so each trip only ends when the elevator changes direction. As both offline and online problems use the same structure, we will use the same definitions for both problems but in the remainder of this thesis, we will, depending on whether we have an offline or online problem, substitute P' with the set of all orders P or with the set of currently known orders P^{known} respectively:

- $P^1 = \{p \in P' \mid a_p < b_p\}$ and $P^0 = \{p \in P' \mid a_p > b_p\}$: the sets of orders going upward and downward, respectively;
- $s^{\max} = 2 \cdot |P'|$: a maximum number of trips of each elevator;
- $S = \{1, \ldots, s^{\max}\}$: the set of all possible trips of each elevator;

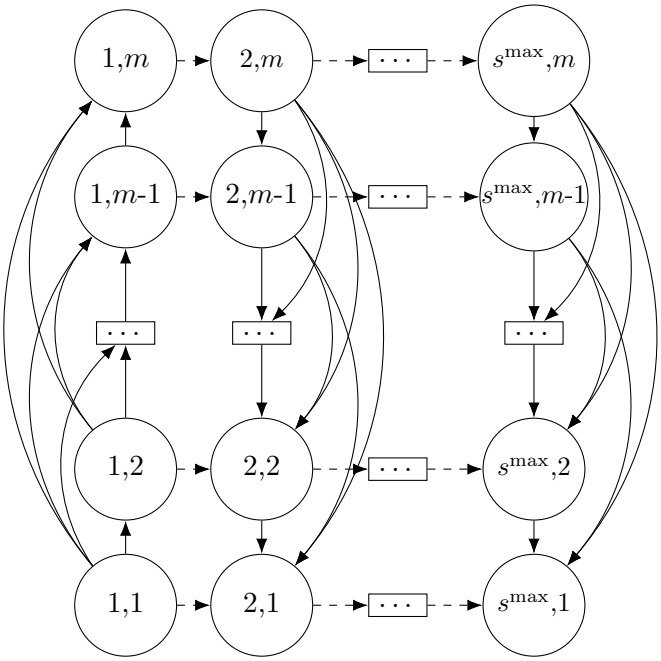


Figure 5.1: Directed graph G without the artificial destination node with m floors and $s^{\max}$ trips, when $s^{\max}$ is even.

- $S^1 = \{s \in S \mid s \bmod 2 = 1\}$ and $S^0 = \{s \in S \mid s \bmod 2 = 0\}$: the sets of upward and downward trips, respectively.

An important aspect of the MILP of both the offline and online problem is that we do not know in advance how many trips each elevator will need to serve all orders. This poses a challenge as we need to model the entire route of each elevator and thus need an upper bound for the maximum number of trips $s^{\max}$. We define $s^{\max}$ as $2 \cdot |P'|$, which is derived from the fact that in the worst case we only have a single elevator and each order needs to be transported separately, so the elevator needs a trip to pick up the order and a trip to deliver the order (in a different direction).

Keeping track of an elevator's movement in the system is an additional challenge, where we use a network flow approach to model it. To do this, we need to define a directed graph $G = (V, A)$, where $V = V' \cup \{\top\}$, $V' = \{(s, j) \mid s \in S, j \in F\}$ is the set of all trip and floor combinations and $\top$ is the artificial destination node for all flows. The set of arcs $A = A_1 \cup A_2 \cup A_3 \cup A_4$ consists of four subsets:

- $A_1 = \{((s, j), (s, j')) \mid s \in S^1, (s, j) \in V', j' \in F, j < j'\}$;
- $A_2 = \{((s, j), (s, j')) \mid s \in S^0, (s, j) \in V', j' \in F, j > j'\}$;
- $A_3 = \{((s, j), (s + 1, j)) \mid (s, j) \in V', s < s^{\max}\}$;

- $A_4 = \{(v, \top) \mid v \in V'\}$.

Nodes in this graph represent stops of an elevator at a specific floor at a specific trip. The arcs represent the possible movements of the elevator between stops, which are limited by the direction of the trip, upward trips have only arcs to a higher floor (A_1) and downward trips have only arcs to a lower floor (A_2). To connect consecutive trips, we need to add arcs between the last stop of a trip and the first stop of the next trip (A_3). Each node has also an arc to an artificial destination node (A_4), which is used to model the flow of the elevator and marks the end of the elevator's last trip of a scenario. An example of such a directed graph for an even number of $s^{\max}$ (without the artificial destination node for better visual representation) is shown in Figure 5.1. We can see that the graph has a grid like structure where each column represents a trip and each row represents a floor. The solid arcs represent the possible movements of the elevator within a trip, and the dashed arcs represent the connections between the trips.

5.2 Offline Problem

As stated in Section 4.1, the offline problem of assigning elevators is to determine an optimal assignment of elevators to orders with full knowledge of the future. Based on the general definitions above, we can now formulate the MILP model for the offline problem. Firstly, we describe the decision variables as follows:

- $x_{ivv'} \in \{0, 1\}$, $\forall i \in E$, $\forall (v, v') \in A$: 1 iff elevator i moves from node v to node v';
- $z_{pis} \in \{0, 1\}$, $\forall p \in P^{s \bmod 2}$, $\forall i \in E$, $\forall s \in S$: 1 if order p travels in the s-th trip of elevator i, otherwise 0;
- $t_{iv} \geq 0$, $\forall i \in E$, $\forall v \in V'$: the time elevator i arrives at node v if visited and after unloading all affected orders;
- $u_p \geq 0$, $\forall p \in P$: the boarding time of order p.

These decision variables are used to generate the solution of the MILP model. We need to know the movement of the elevator between the floors ($x_{ivv'}$), the time the elevator arrives at each floor (t_{iv}), the boarding time of each passenger (u_p) and the assignment of each order to a trip of an elevator (z_{pis}).

Secondly, the MILP model is defined as follows:

$$\min \quad \sum_{p \in P} d_p \cdot (u_p - c_p) \tag{5.1}$$

$$\text{s.t.} \quad \sum_{\substack{p \in P^1 \\ a_p \leq j < b_p}} d_p z_{pis} \leq C \qquad \forall i \in E, \forall j \in F, \forall s \in S^1 \tag{5.2}$$

$$\sum_{\substack{p\in P^0\\ b_p<j\le a_p}} d_p z_{pis} \le C \qquad \forall i\in E, \forall j\in F, \forall s\in S^0 \tag{5.3}$$

$$\sum_{j=a_p+1}^{b_p} x_{i(s,a_p)(s,j)} \ge z_{pis} \qquad \forall p\in P^1, \forall i\in E, \forall s\in S^1 \tag{5.4}$$

$$\sum_{j=a_p}^{b_p-1} x_{i(s,j)(s,b_p)} \ge z_{pis} \qquad \forall p\in P^1, \forall i\in E, \forall s\in S^1 \tag{5.5}$$

$$\sum_{j=b_p}^{a_p-1} x_{i(s,a_p)(s,j)} \ge z_{pis} \qquad \forall p\in P^0, \forall i\in E, \forall s\in S^0 \tag{5.6}$$

$$\sum_{j=b_p+1}^{a_p} x_{i(s,j)(s,b_p)} \ge z_{pis} \qquad \forall p\in P^0, \forall i\in E, \forall s\in S^0 \tag{5.7}$$

$$\sum_{s\in S^r}\sum_{i\in E} z_{pis} = 1 \qquad \forall r\in\{0,1\}, \forall p\in P^r \tag{5.8}$$

$$\begin{aligned} &t_{i(s,j)} + T(j,j')\cdot x_{i(s,j)(s,j')} + \\ &\quad T_{\mathrm{L}}\cdot \sum_{\substack{p\in P^{s \bmod 2}\\ j=a_p\vee j'=b_p}} z_{pis} d_p \le \\ &\quad t_{i(s,j')} + (M + T_{\mathrm{L}}2C)\cdot(1 - x_{i(s,j)(s,j')}) \end{aligned} \qquad \forall i\in E, \forall s\in S, \forall((s,j),(s,j'))\in A \tag{5.9}$$

$$\begin{aligned} &t_{i(s,j)} + T(j,j')\cdot x_{i(s,j)(s,j')} + \\ &\quad T_{\mathrm{L}}\cdot \sum_{\substack{p\in P^{s \bmod 2}\\ j=a_p\vee j'=b_p}} z_{pis} d_p \ge \\ &\quad t_{i(s,j')} - (M + T_{\mathrm{L}}2C)\cdot(1 - x_{i(s,j)(s,j')}) \end{aligned} \qquad \forall i\in E, \forall s\in S, \forall((s,j),(s,j'))\in A \tag{5.10}$$

$$t_{i(s,j)} \le t_{i(s+1,j)} + (1 - x_{i(s,j)(s+1,j)})\cdot 2M \qquad \forall i\in E, \forall((s,j),(s+1,j))\in A \tag{5.11}$$

$$t_{i(s,j)} \ge t_{i(s+1,j)} - (1 - x_{i(s,j)(s+1,j)})\cdot 2M \qquad \forall i\in E, \forall((s,j),(s+1,j))\in A \tag{5.12}$$

$$u_p \ge t_{i(s,a_p)} - (s\cdot M)\cdot(1 - z_{pis}) \qquad \forall s\in S, \forall p\in P^{s \bmod 2}, \forall i\in E \tag{5.13}$$

$$t_{i(s,a_p)} \ge u_p - (s^{\max}\cdot M)\cdot(1 - z_{pis}) \qquad \forall s\in S, \forall p\in P^{s \bmod 2}, \forall i\in E \tag{5.14}$$

$$\sum_{((1,f_i),v')\in A} x_{i(1,f_i)v'} = 1 \qquad \forall i\in E \tag{5.15}$$

$$\sum_{(v',v)\in A} x_{iv'v} - \sum_{(v,v')\in A} x_{ivv'} = 0 \qquad \forall i\in E, \forall v\in V'\setminus\{(1,f_i)\} \tag{5.16}$$

$$\sum_{(v,\top)\in A} x_{iv\top} = 1 \qquad \forall i\in E \tag{5.17}$$

$$\sum_{((s,j'),(s,j))\in A} x_{i(s,j')(s,j)} + \sum_{((s,j),(s,j'))\in A} x_{i(s,j)(s,j')} - 1 \leq \sum_{\substack{p\in P \\ j\in\{a_p,b_p\}}} z_{pis} \quad \forall i \in E, (s,j) \in V' \tag{5.18}$$

$$x_{i(s,j)((s+1),j)} \leq \sum_{\substack{p\in P^{s \bmod 2} \\ j=b_p}} z_{pis} + \sum_{\substack{p\in P^{(s+1) \bmod 2} \\ j=a_p}} z_{pis} \quad \forall i \in E, (s,j) \in V' \setminus \{(1,f_i)\}, s < s^{\max} \tag{5.19}$$

$$x_{ivv'} \in \{0,1\} \quad \forall i \in E, \forall (v,v') \in A \tag{5.20}$$

$$z_{pis} \in \{0,1\} \quad \forall s \in S, \forall p \in P^{s \bmod 2},\ i \in E \tag{5.21}$$

$$t_{iv} \geq 0 \quad \forall i \in E, \forall v \in V' \tag{5.22}$$

$$u_p \geq c_p \quad \forall p \in P \tag{5.23}$$

The objective function 5.1 minimizes the sum of all waiting times for all orders, weighted by their group size, which is equivalent to minimizing the AWT. The capacity of each elevator is enforced by (5.2) and (5.3), no elevator can carry more than C passengers when travelling from one floor to another. We introduce (5.4) – (5.7), for linking the elevator and the trips the passengers take, each order travelling upwards (downwards) needs to be picked up and travel to a floor higher (lower) than the arrival floor, and each order needs to arrive at the destination floor from a floor lower (higher) than the destination floor.

As specified in Assumptions 5 and 6 about the system and the orders travelling, we need to limit a person to only take one trip of one specific elevator, this is done in (5.8).

Important for solving the MILP is also tracking the time the elevators need to travel between floors, here we use a Miller-Tucker-Zemlin (MTZ) [23] formulation for Constraints 5.9 and 5.10 to ensure that the time between the stopping on a floor j and the arrival on the next floor j' is exactly the travel time $T(j,j')$ of the elevator plus the time T_{L} it takes to load the passengers at the departure floor and unload them at the destination floor. Note that stopping time of an elevator is always before loading new orders in and after unloading the orders at the destination floor. The main goal of a MTZ formulation is that these inequalities hold even when $x_{i(s,j)(s,j')} = 0$, i.e., when this path in the graph is not taken by the elevator. This requires a large ("Big-M") constant $M = (m-1) \cdot (T(1,2) + T_{\mathrm{L}}2C)$, to inactivate the constraints when we do not need them. For Constraints 5.9 and 5.10 we also need to add a constant value of $T_{\mathrm{L}}2C$ to M for a valid MTZ formulation, because in case of $x_{i(s,j)(s,j')} = 0$ we cannot set the loading time on the left side of the inequality to zero without losing linearity, thus we need to account for this on the right side. The value of M is derived from the maximum possible span two time points can have in the same trip, which is the time from the first (or last) floor to the last (or first) floor assuming that the elevator is always full and stops at each floor

where it fully unloads and again loads to maximum capacity.

Similarly we need to ensure that we link the time across consecutive trips, which is done in (5.11) and (5.12), where we use the same "big-M" to inactivate the constraints when we do not need them, this time multiplying M with two, because the span between these two points in time can be the maximum time it takes the elevator to travel from the first floor to the last floor and back.

To get the boarding time of the passengers we need (5.13) and (5.14), where we, again, need a "big-M" constraint but need to increase it by multiplying it with s, the trip number, in (5.13) because the largest value $t_{i(s,a_p)}$ can have is the time it would take an elevator i to stop at every floor and (un)load the maximum amount of passengers for every trip and floor before (s, a_p), which results in $s \cdot M$. For the Inequality (5.14) we need to multiply M with $s^{\max}$, as u_p can be at most the time it would take an elevator to make all trips with maximum stops and loading time.

To make use of the previously defined directed graph G we need to ensure that the flow of the elevator is correct. We first define the origin of the flow in (5.15), where the elevator starts at the initial starting floor, and the target of the flow in (5.17), where the elevator ends at the artificial destination node. The flow conservation is defined in (5.16), where the flow of the elevator is conserved at each floor, i.e., the flow into a floor (of a specific trip) is equal to the flow out of the floor. This controls the movement in a way that the elevator has to leave each floor it visits until it reaches the final destination of its route.

We also introduce symmetry breaking constraints where we ensure that the elevator does not take any stops without (un)loading passengers (5.18) and that the elevator makes no unnecessary trips (5.19). Lastly we have the domain constraints for the decision variables in (5.20) – (5.23), where $x_{ivv'}$ and z_{pis} are binary, t_{iv} is non-negative and u_p is at least the arrival time of the order.

5.3 Online Problem

In contrast to the offline problem, the MILP model for the online problem does not have full knowledge of the future but also needs to consider the current state of the system. While in the offline problem we dealt with a whole scenario where we have all the information and the system is in an initial state, for the online problem we need a dynamic approach. We will still use scenarios for evaluation, but the model itself only can handle the current state and finds an optimal solution with the given information. Because elevators might be currently in motion in the given state we want to solve, we need to introduce some additional definitions based on the definitions stated above and the definitions of the online problem in Section 4.1.2:

- $E^{\text{up}} = \{i \mid i \in E, f_i^{\text{last}} < f_i^{\text{next}}\}$: the set of all elevators which currently travel upwards;

- $E^{\text{down}} = \{i \mid i \in E, f_i^{\text{last}} > f_i^{\text{next}}\}$: the set of all elevators which currently travel downward;
- $F_i^{\text{up}} = \{a \in F \mid t_i^{\text{last}} + T_{\text{L}} \cdot \sum_{p \in P_i^{\text{load}}} d_p + T(f_i^{\text{last}}, a) \leq t, f_i^{\text{last}} < a\}$: the set of all floors which have been passed by elevator i on the current trip;
- $F_i^{\text{down}} = \{a \in F \mid t_i^{\text{last}} + T_{\text{L}} \cdot \sum_{p \in P_i^{\text{load}}} d_p + T(f_i^{\text{last}}, a) \leq t, a < f_i^{\text{last}}\}$: the set of all floors which have been passed by elevator i on the current trip;
- $P_i^{\text{load}} = \{p \in P^{\text{elev}} \mid e_p = i \wedge a_p = f_i^{\text{last}}\}$: the set of all orders in elevator i which embarked at the last stop.

Here we extended the defined sets and parameters by E^{up} and E^{down} to track the direction of currently moving elevators and F_i^{up} and F_i^{down} to track the floors which have been passed by the elevator on the current trip. This is needed as we do not have accurate information about the current location of the elevators, but we can infer the direction and approximate location based on the last (and next) stop of the elevator, the time since the last stop and the loaded orders.

Furthermore we, again, introduce the decision variables for the MILP model, which are very similar to the offline problem, but use only the currently known orders:

- $x_{ivv'} \in \{0, 1\}$, $\forall i \in E$, $\forall (v, v') \in A$: 1 iff elevator i moves from node v to node v';
- $z_{pis} \in \{0, 1\}$, $\forall p \in P^{s \bmod 2}$, $\forall i \in E$, $\forall s \in S$: 1 iff order p travels in the s-th trip of elevator i, otherwise 0;
- $t_{iv} \geq 0$, $\forall i \in E$, $\forall v \in V'$: the time elevator i arrives at node v if visited and after unloading all affected orders;
- $u_p \geq 0$, $\forall p \in P^{\text{known}}$: the boarding time of order p.

Another important step for correctly representing the current state of the system in the MILP model is to set some decision variables defined above, so that the model basically looks at a static state. Because our directed graph G contains arcs which may not be valid for the current state, we need to fix some movement of the elevator between the floors via the $x_{ivv'}$ variables. We also need to state currently loaded orders via the z_{pis} variables, and the time the elevator arrives at each floor via the t_{iv} variables. Formally this is done by the following constraints:

- $z_{pe_ps} = 1$, $\forall s \in \{1, 2\}$, $\forall p \in P^{s \bmod 2} \cap P^{\text{elev}}$: setting orders currently in an elevator;
- $x_{i(1,f_i^{\text{last}})(2,f_i^{\text{last}})} = 1$, $\forall i \in E^{\text{down}}$: setting the elevator to move down;
- $t_{i(1,f_i^{\text{last}})}$ is equal to t if the elevator i is currently not moving, i.e., $f_i^{\text{last}} = f_i^{\text{next}}$, otherwise t_i^{last}, $\forall i \in E$: setting the time of the last stop of the elevator, if currently not moving use the current time,

- $x_{i(1,f_i^{\text{last}})(1,a)} = 0$, $\forall i \in E^{\text{up}}$, $\forall a \in F_i^{\text{up}} \cup \{\max(F_i^{\text{up}})+1, \max(F_i^{\text{up}})+2\}$, $f_i^{\text{last}} < a < f_i^{\text{next}}$: prohibiting stops which the elevator already passed by or cannot stop in time in an upward trip, only stops between the last stop and the next planned stop are considered;
- $x_{i(2,f_i^{\text{last}})(2,a)} = 0$, $\forall i \in E^{\text{down}}, \forall a \in F_i^{\text{down}} \cup \{\min(F_i^{\text{down}})-1, \min(F_i^{\text{down}})-2\}$, $f_i^{\text{next}} < a < f_i^{\text{last}}$: prohibiting stops which the elevator already passed by or cannot stop in time in a downward trip, only stops between the last stop and the next planned stop are considered;
- $x_{i(1,f_i^{\text{last}})(1,f_i^{\text{next}})} = 1$, $\forall i \in E^{\text{up}}, \max(F_i^{\text{up}})+2 > f_i^{\text{next}}$: fixing stops if the elevator is not more than two floors away from the next planned stop and going up;
- $x_{i(2,f_i^{\text{last}})(2,f_i^{\text{next}})} = 1$, $\forall i \in E^{\text{down}}, \min(F_i^{\text{down}})-2 < f_i^{\text{next}}$: fixing stops if the elevator is not more than two floors away from the next planned stop and going down.

With some decision variables fixed we can now formulate the MILP model for the online problem. The objective function and the constraints are very similar to the offline problem, but we need to adapt them to the current state of the system:

$$\min \sum_{p \in P^{\text{known}}} d_p \cdot (u_p - c_p) \tag{5.24}$$

$$\text{s.t.} \sum_{\substack{p \in P^1 \\ a_p \le j < b_p}} d_p z_{pis} \le C \qquad \forall i \in E, \forall j \in F, \forall s \in S^1 \tag{5.25}$$

$$\sum_{\substack{p \in P^0 \\ b_p < j \le a_p}} d_p z_{pis} \le C \qquad \forall i \in E, \forall j \in F, \forall s \in S^0 \tag{5.26}$$

$$\sum_{j=a_p+1}^{b_p} x_{i(s,a_p)(s,j)} \ge z_{pis} \qquad \forall p \in P^1 \setminus P^{\text{elev}}, \forall i \in E, \forall s \in S^1 \tag{5.27}$$

$$\sum_{j=a_p}^{b_p-1} x_{i(s,j)(s,b_p)} \ge z_{pis} \qquad \forall p \in P^1, \forall i \in E, \forall s \in S^1 \tag{5.28}$$

$$\sum_{j=b_p}^{a_p-1} x_{i(s,a_p)(s,j)} \ge z_{pis} \qquad \forall p \in P^0 \setminus P^{\text{elev}}, \forall i \in E, \forall s \in S^0 \tag{5.29}$$

$$\sum_{j=b_p+1}^{a_p} x_{i(s,j)(s,b_p)} \ge z_{pis} \qquad \forall p \in P^0, \forall i \in E, \forall s \in S^0 \tag{5.30}$$

$$\sum_{s \in S^r} \sum_{i \in E} z_{pis} = 1 \qquad \forall r \in \{0,1\}, \forall p \in P^r \tag{5.31}$$

$$t_{i(s,j)} + T(j,j') \cdot x_{i(s,j)(s,j')} + T_{\mathrm{L}} \cdot \sum_{\substack{p \in P^{s \bmod 2} \\ j = a_p \vee j' = b_p}} z_{pis} d_p \leq t_{i(s,j')} + (M + T_{\mathrm{L}} 2C) \cdot (1 - x_{i(s,j)(s,j')}) \qquad \forall i \in E, \forall s \in S, \forall ((s,j),(s,j')) \in A \tag{5.32}$$

$$t_{i(s,j)} + T(j,j') \cdot x_{i(s,j)(s,j')} + T_{\mathrm{L}} \cdot \sum_{\substack{p \in P^{s \bmod 2} \\ j = a_p \vee j' = b_p}} z_{pis} d_p \geq t_{i(s,j')} - (M + T_{\mathrm{L}} 2C) \cdot (1 - x_{i(s,j)(s,j')}) \qquad \forall i \in E, \forall s \in S, \forall ((s,j),(s,j')) \in A \tag{5.33}$$

$$t_{i(s,j)} \leq t_{i(s+1,j)} + (1 - x_{i(s,j)(s+1,j)}) \cdot 2M \qquad \forall i \in E, \forall ((s,j),(s+1,j)) \in A \tag{5.34}$$

$$t_{i(s,j)} \geq t_{i(s+1,j)} - (1 - x_{i(s,j)(s+1,j)}) \cdot 2M \qquad \forall i \in E, \forall ((s,j),(s+1,j)) \in A \tag{5.35}$$

$$u_p \geq t_{i(s,a_p)} - (s \cdot M) \cdot (1 - z_{pis}) \qquad \forall s \in S, \forall p \in P^{s \bmod 2}, \forall i \in E \tag{5.36}$$

$$t_{i(s,a_p)} \geq u_p - (s^{\max} \cdot M) \cdot (1 - z_{pis}) \qquad \forall s \in S, \forall p \in P^{s \bmod 2}, \forall i \in E \tag{5.37}$$

$$\sum_{((1,f_i^{\mathrm{last}}),v') \in A} x_{i(1,f_i^{\mathrm{last}})v'} = 1 \qquad \forall i \in E \tag{5.38}$$

$$\sum_{(v',v) \in A} x_{iv'v} - \sum_{(v,v') \in A} x_{ivv'} = 0 \qquad \forall i \in E, \forall v \in V' \setminus \{(1, f_i^{\mathrm{last}})\} \tag{5.39}$$

$$\sum_{(v,\top) \in A} x_{iv\top} = 1 \qquad \forall i \in E \tag{5.40}$$

$$\sum_{((s,j'),(s,j)) \in A} x_{i(s,j')(s,j)} + \sum_{((s,j),(s,j')) \in A} x_{i(s,j)(s,j')} - 1 \leq \sum_{\substack{p \in P^{\mathrm{known}} \\ j \in \{a_p, b_p\}}} z_{pis} \qquad \forall i \in E, (s,j) \in V' \tag{5.41}$$

$$x_{i(s,j)((s+1),j)} \leq \sum_{\substack{p \in P^{s \bmod 2} \\ j = b_p}} z_{pis} + \sum_{\substack{p \in P^{(s+1) \bmod 2} \\ j = a_p}} z_{pis} \qquad \forall i \in E, (s,j) \in V' \setminus \{(1, f_i^{\mathrm{last}})\}, s < s^{\max} \tag{5.42}$$

$$x_{ivv'} \in \{0,1\} \qquad \forall i \in E, \forall (v,v') \in A \tag{5.43}$$

$$z_{pis} \in \{0,1\} \qquad \forall s \in S, \forall p \in P^{s \bmod 2},\ i \in E \tag{5.44}$$

$$t_{iv} \geq 0 \qquad \forall i \in E, \forall v \in V' \tag{5.45}$$

$$u_p \geq c_p \qquad \forall p \in P^{\mathrm{known}} \tag{5.46}$$

Looking at the MILP for the online problem, it is very similar to the offline problem, but

it is adjusted to the dynamic nature of the problem. The constraints are mostly the same and the overall idea of the model is identical, but here we use only the currently known orders for the inequalities. One important difference is that we need to exclude orders, which are already on an elevator, from some constraints like (5.27) and (5.29), because they do not need to be picked up anymore, but it is important that we still consider them in Constraints 5.28 and 5.30, because they still need to be delivered. We also essentially replace the starting floor used in the offline problem with the f_i^{last} variable, which is the last stop of the elevator, where we can start our flow of the elevator.

With the formulation of the MILP we can now solve a scenario of the online problem by iteratively solving the MILP for each time a new order arrives. Because the MILP model always calculates the current best route for each elevator until all known orders are delivered, we can use this information at decision points where an elevator makes a stop but no new order arrives. On new order arrivals we solve the MILP for the current state and use the new solution for making decisions, overwriting solutions from previous time points if necessary, as we have now more information about the system and can make better decisions. This way we can solve the online problem with the MILP model and always have the best possible solution for the current state of the system.

CHAPTER 6

Greedy Algorithm

In this chapter, we present our second approach, a greedy algorithm, for tackling the online elevator assignment problem for EGCS-DRS. In contrast to the MIP approach, the greedy algorithm is a heuristic method that does not guarantee an optimal solution. However, it is computationally substantially less expensive, especially for larger scenarios, thus making it more suitable for online applications.

We approach this with an elevator centric perspective, where we try to find fast pickups for each elevator considering the positions of the waiting orders and the elevator, as well as the current choices from other elevators. The algorithm is based on the idea that picking up orders is the most important action an elevator can take, as the average waiting time of orders only counts time until the pickup, so we want to pick them up as fast as possible, disregarding the group size and distance to the destination floors of the orders. We consider here, again, only a snapshot of the state at the designated decision points and try to find a good solution for it. This is done for each decision point of the scenario and in the end the solution for the whole scenario can be constructed by concatenating the solutions of the individual decision points.

The base structure is shown in Algorithm 6.1, which is called to return the greedily best found system-action, which contains elevator-actions for all elevators. In contrast to the online MILP, where we calculated the whole route for the current state, we only find the next best action for each elevator and do not plan every stop and pickup in advance, reducing computational complexity as well as logical complexity for designing the algorithm. We first mark all waiting orders P^{wait} from state s as unconsidered and create a system-action, initialized with empty actions for each elevator. In Lines 7-10 we loop until we have chosen an elevator-action for each elevator. For each elevator without already fixed action, we find the currently best elevator-action with *getBestEvAction*, which returns the elevator-action *act*, the orders o it considers picking up, and the value of *act*, which is used to prioritize the elevator-actions. The orders o represent orders which cannot currently be picked up by the elevator in this elevator-action, (e.g., when

the elevator is currently moving or has no orders waiting on its current floor) but rather are considered to be picked up in the future by this elevator. Line 11 returns the elevator from the heap with the lowest value elevator-action, which is our highest priority, as it represents the time the next pickup is planned for this elevator. We remove the elevator from the list of remaining elevators in Line 13 and mark all orders which are either picked up by the elevator-action (act^{load}) or considered to be picked up in the future by this elevator as considered in Lines 14 and 15. As we cannot load orders which are on a different floor than the elevator currently is, we need to mark them as considered, otherwise other elevators could also try to target this order for pickup, which might lead to multiple elevators travelling to the same floor for (possibly) only a single order, resulting in unnecessary travel for some elevators and (possibly) increased waiting times for other orders. We then repeat the process until all elevators have an elevator-action assigned.

Algorithm 6.1: Base Greedy Algorithm

```
   Input  : State s
   Output : System-action A
 1 Function runGreedyAlgorithm():
 2     Unconsidered ← s.P^wait ;
 3     A ← array of size s.k;
 4     Elev ← s.E;
 5     while Elev is not empty do
 6         H ← empty Min-heap;
 7         for each e in Elev do
 8             act, o, value ← getBestEvAction(s, e, Unconsidered);
 9             H.add(value, act, o, e);
10         end
           // Get elevator-action with highest priority, i.e.,
              lowest value
11         value, act, o, e ← H.pop();
12         A[e] ← act;
13         Elev.remove(e);
           // Mark picked up orders and targeted orders as
              considered
14         Unconsidered.remove(act^load);
15         Unconsidered.remove(o);
16     end
17     Return A;
```

A crucial part of the algorithm is the function *getBestEvAction*, which is shown in Algorithm 6.2. Here we decide based on the current state s, the elevator e and the unconsidered orders u, what the best elevator-action is, i.e., which orders we plan to load or which order we consider for pickup and at what time we load the next order. For

Algorithm 6.2: Elevator-Action Selection

Input : State s, Elevator e, Unconsidered orders u
Output : Elevator-Action act, Considered orders c, Value v

Function `getBestEvAction()`**:**
  if *e is currently not moving* **then**
    if *u is empty and e has no orders loaded* **then**
      $act \leftarrow (s.f_e^{\text{last}}, [])$;
      $c \leftarrow []$;
      $v \leftarrow \infty$;
    else if *u is empty and e has orders loaded* **then**
      $f \leftarrow$ getNearestDropoff(s, e);
      $act \leftarrow (f, [])$;
      $c \leftarrow []$;
      $v \leftarrow \infty$;
    else if *u is not empty and orders are waiting on the current floor of e* **then**
      $pickup \leftarrow$ getBestPickup(s, e, u);
      f, $toConsider \leftarrow$ getBestOrderStop(s, e, u, $pickup$);
      $act \leftarrow (f, pickup)$;
      $c \leftarrow toConsider$;
      $v \leftarrow s$.t;
    else
      f, $nextPickUpTime$, $orders \leftarrow$ getNextOrderPickup(s, e, u);
      $act \leftarrow$ (f, []);
      $c \leftarrow orders$;
      $v \leftarrow nextPickUpTime$;
    end
  else
    if *u is empty* **then**
      $act \leftarrow (s.f_e^{\text{next}}, [])$;
      $c \leftarrow []$;
      $v \leftarrow \infty$;
    else
      f, $nextPickUpTime$, $orders \leftarrow$ getNextOrderPickup(s, e, u);
      $act \leftarrow (f, [])$;
      $c \leftarrow orders$;
      $v \leftarrow nextPickUpTime$;
    end
  end
  Return (act, c, v);

different states of the system and elevators we need to decide for different elevator-actions. We consider the following cases:

1. If e is currently not moving, i.e., f_e^{last} is equal to the current time t of s (lines 2–24):
   a) If there are no unconsidered orders (i.e. u is empty) and e has no orders loaded, we choose the next elevator-action for e to be staying at the current floor (lines 3–6).
   b) Else if there are no unconsidered orders and e has orders loaded, the elevator-action for e is to go to the nearest destination floor of the currently loaded orders in e, which is found by *getNearestDropoff* (lines 7–11).
   c) Else if there are unconsidered orders on the same floor as the current stop of e, pick up as many orders as possible. The function *getBestPickup* returns a greedy choice of pickup orders, considering capacity and traveling direction and prioritizing them in non-increasing number of passengers and breaking ties by choosing nearer destination floors. Additionally, set the next planned stop of e by finding the nearest floor where either a currently loaded order of e wants to go or an unconsidered waiting order wants to be picked up. This is done by the function *getBestOrderStop*, which considers that a waiting order must travel in the same direction as the (possibly) currently boarded orders and must not exceed the capacity to be able to be picked up. If we travel to a stop to pick up waiting orders also return the orders to consider (lines 12–17).
   d) Else we call the function *getNextOrderPickup* to find the order, which we can pick up in the nearest future. The function returns the next floor the elevator should stop, which is either a destination floor of a loaded order or, if an order can be picked up before we deliver the next order, the floor of the next pickup (lines 18–23).
2. Else (e is currently moving) (lines 24–35):
   a) If there are no unconsidered orders, set the next elevator-action of e to stop at the currently next planned stop of e (lines 25–28).
   b) Otherwise, call the function *getNextOrderPickup* and find the nearest floor and order(s) to pick up (lines 29–34).

Importantly we need to comply with all our assumption stated in Section 4.2 and thus we need to correctly handle the different cases of the elevator state, especially Assumption 9 which states that we cannot load orders which travel in a different direction than the currently loaded orders. Looking at the cases above where u is not empty, meaning we have unconsidered orders we can pickup, and no orders are on the current floor of the elevator, we cannot just (always) rush to the nearest order and pick them up but rather may need to deliver orders first. The function *getNextOrderPickup* considers all these constraints where we may currently have loaded orders in the elevator e which are not

yet delivered. If all waiting orders have a different travel direction than the loaded orders or are not between the current floor f of e and the next destination floor f' of an order loaded in e, then we need to first unload orders at f' and again search for orders we could pick up. This may repeat until all loaded orders are unloaded, and we can pick up any waiting order or until we can pick an order midway which travels in the same direction. For calculating the planned arrival time at the next pickup floor, we need to consider all intermediate stops we may have to make to deliver the currently loaded orders, including the time it takes the load and unload orders and the time it takes to travel between floors.

As we want to prioritize elevators which can immediately pick up orders on their current floor over elevators which need to travel to pick up orders, we set the values of these elevators to the time of the current state, which is the minimum value we will have. For planned pickups on other floors we will need to calculate the planned arrival time, thus prioritizing elevators which need to travel less to pick up orders. As we may end up with all the waiting orders considered, some elevators might not have any elevator-action assigned yet and thus will not plan to pick up any order. But as picking up orders is the only action elevators can do to interact with the environment with a direct impact on the other elevators, we can ignore the prioritization of the elevators in such a case and set the value to ∞ for these elevators. This will result in the elevator-actions of the remaining elevators, which are finishing their current plan of delivering orders or waiting on a floor, being chosen in order of their ID.

Let us consider the example shown in Table 6.1 where we have three orders and three elevators, the time of the current state is 8 and for simplicity we set the capacity of the elevators to be one. We define the loading time $T_{\mathrm{L}} = 1$ and the travel time $T(1) = 6$ and $T(k) = T(1) + k$ for $k > 1$. Two orders, 2 and 3, are currently waiting on floor 1 and 4 respectively, while order 1 is currently loaded in elevator 1, which is on its way to floor 3 and started its trip on floor 1 at time 0. The elevators 2 and 3 are both currently on floor 1 and are ready to pick up orders. In Table 6.2a we see that orders 2 and 3 are both unconsidered before the first iteration of the loop starting in Line 5 of Algorithm 6.1. In the first iteration, we start with elevator 1 and call function *getBestEvAction* where we go to line 29 in Algorithm 6.2 because the elevator is currently moving and the unconsidered orders u is not empty, as it still contains all the waiting orders, namely 2 and 3. Now *getNextOrderPickup* is called, which finds the nearest next pickup, but as the elevator is currently fully loaded it cannot pick up any other orders, thus is required to first unload order 1 and then find the nearest order to pickup. As elevator 1 delivers order 1 at floor 3, the nearest order to pick up is order 3 on floor 4. From the information given in state s we can calculate when order 1 should be disembarked, which is the time the elevator left floor 1, which is 0, plus the loading time $T_{\mathrm{L}} \cdot d_1 = 1$ plus the travel time $T(|b_1 - a_1|) = 8$ plus the unloading time $T_{\mathrm{L}} \cdot d_1 = 1$, which sums up to 10. We then can calculate the arrival time at floor 4, which is the time it left floor 3 plus the travel time to floor 4, so $10 + T(|4 - 3|) = 16$, this is now the *nextPickUpTime* we return, together with the target floor for the elevator-action, which is still floor 3 because we need to deliver order

Table 6.1: Example of the state at a decision point.

(a) Order data

p	a_p	b_p	d_p	e_p
1	1	3	1	1
2	1	5	1	-
3	4	1	1	-

(b) Elevator data

e	f_e^{last}	t_e^{last}	f_e^{next}
1	1	0	3
2	1	8	1
3	1	8	1

Table 6.2: Content of the *Unconsidered* array *before* each iteration and the content of the min-heap H and system-action A *after* each iteration of the while loop.

(a) **Before** first iteration

Unconsidered	[2, 3]

(b) **Before** second iteration

Unconsidered	[3]

(c) **Before** third iteration

Unconsidered	[]

(d) H **after** first iteration

e	*value*	a	o
1	16	(3, [])	[3]
2	8	(5, [2])	[]
3	8	(5, [2])	[]

(e) H **after** second iteration

e	*value*	a	o
1	16	(3, [])	[3]
3	17	(4, [])	[3]

(f) H **after** third iteration

e	*value*	a	o
3	∞	(1, [])	[]

(g) A **after** first iteration

e	$A[e]$
1	-
2	(5, [2])
3	-

(h) A **after** second iteration

e	$A[e]$
1	(3, [])
2	(5, [2])
3	-

(i) A **after** third iteration

e	$A[e]$
1	(3, [])
2	(5, [2])
3	(1, [])

1 first, and the considered orders, being only order 3. For elevators 2 and 3 we call *getBestEvAction* and go to line 12 of Algorithm 6.2, because there are unconsidered orders left and some orders are waiting on the same floor as the elevator. Function *getBestPickup* just returns order 2 because there are no other orders, and *getBestOrderStop* returns floor 5 because the elevator only needs to deliver order 2. The greedily best action is then for both elevators to pickup order 2 and go to floor 5, the value is 8, the time of the current state. The result of the first iteration is then the min-heap shown in Table 6.2d, where the elevator-actions for elevator 2 and 3 are both the lowest value, so we break the tie by choosing the lowest index, thus we choose elevator 2 and the system-action A contains now this elevator-action, shown in Table 6.2g. We remove elevator 2 from the list of remaining elevators and mark order 2 as considered.

The second iteration has now only order 3 not considered (Table 6.2b), and for elevator 1 the next best elevator-action has not changed, but for elevator 3 we now have to find another order, as order 2 is already considered, and we end up in line 18 in Algorithm 6.2. The next best elevator-action for elevator 3 is to go to floor 4 because order 3 is waiting there, we calculate the arrival time which is just the current time 8 plus the travel time $T(|4-1|) = 9$, resulting in a planned arrival time of 17. The min-heap after the second iteration is now shown in Table 6.2e, where we choose elevator 1 as the next best action, as it would pick up order 3 sooner than elevator 3. We update the elevator-action list in Table 6.2h, remove elevator 1 from the list of remaining elevators and mark order 3 as considered.

The third iteration has now no orders unconsidered (Table 6.2c), but we still need to choose an elevator-action for elevator 3. We go to line 3 in Algorithm 6.2 because u is empty and we have no orders loaded, thus we return the elevator-action to stay at the current floor with the value set to ∞ as shown in Table 6.2f, as we have no orders to pick up. The final elevator-action list is then shown in Table 6.2i, which is the result of the greedy algorithm for this example state.

CHAPTER 7

Reinforcement Learning

This chapter will introduce our third and final approach to solving the elevator dispatching problem for EGCS-DRS, namely reinforcement learning. We will focus, as in the previous greedy approach, only on the online variant of the problem, where we do not have access to future information and make decisions based on the current state of the system.

Section 7.1 will first introduce the concept of reinforcement learning and describe the Markov Decision Process, after which we will describe the architecture of our reinforcement learning model in Section 7.2.

7.1 Markov Decision Process

The Markov Decision Process is a central concept in reinforcement learning, as already touched on in Chapter 3.2. In this section, we will define and describe the MDP for the elevator dispatching problem, specifically we need to define the state and observation, the action space and the reward function.

The *state* of the MDP is the current state (at the time of decision-making) of the system, with all information about orders and elevators, and is as defined and described in Section 4.1.2. When using reinforcement learning, the state of the system is all the information we know, but the agent might not have access to all of it or might not be able to use all of it. Thus, we need to define the *observation* of the system, which is a representation of the state that the agent can make use of, which, in our case, is only partial information about the state and furthermore is processed and simplified such that the agent and the underlying machine learning model can work with it (more easily). For the elevator dispatching problem in EGCS-DRS we have actions for the whole system and not just for one single elevator, thus our action and observation space is not specific for each elevator but rather for all of them together. Our goal was to make the observation as compact as possible, avoiding large multidimensional arrays which would be needed

if we wanted to exactly model each waiting order and their desired target floor, and to include only the most relevant information for the agent to make decisions. We define the observation as the concatenation of sub-observations for each elevator e, where the sub-observation for e is as follows:

- The next planned stop of e one-hot encoded;
- The direction of e one-hot encoded with three values: up, down, and idle;
- The sum of the sizes of orders in e, normalized by the maximum capacity of the elevator;
- The target floors of the orders in e binary encoded;
- The time to reach each floor from the current position of e, normalized with the travel time from the ground floor to the top floor;
- For each floor the sum of the sizes of orders which can be picked up by e and travel upwards, normalized by the maximum capacity of the elevator;
- For each floor the sum of the sizes of orders which can be picked up by e and travel downwards, normalized by the maximum capacity of the elevator;
- For each floor f the sum of the sizes of orders which can be picked up by e on the current floor and have f as destination floor, normalized by the maximum capacity of the elevator and capped at 1.

Using one-hot encoding, binary encoding and normalization is an important step in the preprocessing of the observation, as it represents the information in a way that is easier to work with instead of using raw values like floor numbers or number of orders.

The *action space* of the MDP is the set of all possible actions that the agent can take in a given state. In the elevator dispatching problem, the action space is a subset of all possible and valid system-actions, as defined in Section 4. When choosing actions we need to be careful, as a system-action might be invalid, even if all the elevator-actions are valid in isolation. For example, consider the following example for two elevators $e1$ and $e2$: If $e1$ and $e2$ both are waiting on the same floor f and there is only one order on some other floor f', a valid elevator-action for $e1$ would be to go to f' and pick up the order. The same elevator-action would also be valid for $e2$, as the goal is to pick up the order and without information of the other elevator we do not break any rules or assumptions, but in context of the whole system-action we know that both elevators would reach the order at the same time and only one of them can pick it up, thus making the system-action invalid (w.r.t. Assumption 3). Another improvement we made to the action space is to not allow all elevators to wait at the same time while there are orders waiting to be picked up. This is especially important as we consider each decision point in isolation and do not have any memory of previous decisions and previous states. Thus,

if we allow elevators always to wait, they might learn that waiting for more orders to arrive is beneficial in some cases and then not pick up the orders that are already waiting, which, on the one hand, would be a psychological factor for the orders, as they would arrive at an elevator hall and see that the elevators are not moving, maybe thinking that the system is not working correctly. On the other hand, it would also be a bad decision for the system, as decision points are only on elevator stops and order arrivals, thus if all elevators are waiting and a new order arrives, and the system decides for all elevators to wait, the order would be waiting for the next decision point, which could be a long time. Even if we have decision points in a short time span, the system still does not know that the state has not changed in the meantime and would make the same decision, to wait, again.

Another important part of the MDP is the *reward function*, which is a function that assigns a scalar value to each state-action pair, representing the immediate reward the agent receives when taking an action in a given state. The reward function is crucial in reinforcement learning, as it guides the agent to learn a policy that maximizes the expected cumulative reward. Rewards are given at each decision point (immediate rewards) and at the end of an episode (terminal reward). An episode is the whole simulation of a scenario, starting from the initial state and then delivering orders until there are no more orders arriving. In the elevator dispatching problem, a first thought might be that the reward function is straight forward, as we could just use the negative AWT of the whole episode as the terminal reward and let all immediate rewards be zero. But this would not be an effective approach, as the agent would not be able to learn from the immediate actions and only will get feedback on how well it performed at the end of each episode. Thus, we need to incorporate immediate rewards with the goal of having the cumulative reward (the sum of all rewards in an episode) be in line with our objective function, the AWT, but as RL aims for the agent to learn a policy that maximizes the reward we would need to adjust the reward function to cumulate to the negative AWT. Although this approach may seem reasonable and similar reward functions seem to work for similar approaches in the literature [39], we found that it did not perform at all for our problem. The main reason for this is the learning of the model, as the agent needs to learn the policy from scratch and in a way that it can generalize it to unseen states we need a lot of different states and actions to be tried. We will describe the training process in more detail in Section 8.3, but we generally use many different scenarios to train the model and in general these scenarios have different characteristics, like different number of orders arriving and the time span in which they arrive, thus the AWT of a scenario is not a good measure of how well the agent performed, as it is highly dependent on the scenario. We need to give the reward expressiveness to guide the agent in the right direction but also need to incorporate the AWT in some way, thus we use a reward function which cumulates to the negative AWT of the episode divided by the greedily best found AWT of the scenario. Formally, the reward function $R(s, s', a)$ with the current state s, the state of the next decision point s' and action a, is defined as

follows:

$$R(s, s', a) = \begin{cases} -\frac{-\delta^+ \cdot W + \varepsilon}{G_{\text{awt}}^{\text{greedy}} \cdot N + \varepsilon} & \text{if the episode has not ended yet;} \\ (1 + \frac{W}{N})^2 \cdot -C_{\text{err}} & \text{if the episode ended with some orders not delivered;} \\ 0 & \text{otherwise;} \end{cases} \tag{7.1}$$

where δ^+ is the time difference between s' and s, $W = \sum_{p \in P^{\text{wait}}} d_p$ is the sum of sizes of orders waiting in s', $G_{\text{awt}}^{\text{greedy}}$ is the AWT solution of our greedy approach (Section 6) of the scenario and $N = \sum_{p \in P} d_p$ is the sum of the size of all orders in the scenario. To avoid division by zero (if $G_{\text{awt}}^{\text{greedy}} = 0$) we add $\varepsilon = 1e^{-10}$ to both numerator and denominator of the fraction. We also, sometimes, need to prematurely end an episode, if the episode takes too long to finish, in this case we end the episode and use the second reward, where C_{err} is some constant value, which in our case can just be 1 as we do not need to penalize the agent more, because if we end prematurely the reward is (generally) already a large negative value. We say an episode ends prematurely if the episode takes longer than $n \cdot 2M$, with $M = (m-1) \cdot (T(1,2) + T_{\text{L}}2C)$, which is the worst case time it would take to deliver all orders separately. For a whole scenario (which did not end prematurely) the cumulative reward of all immediate rewards is then approximately (because of ε) $\frac{G_{\text{awt}}}{G_{\text{awt}}^{\text{greedy}}}$, where G_{awt} is the AWT of the RL solution for this scenario. With this reward function we can guide our agent with the help of our greedy approach, as we have a baseline of how good it performs in relation to the greedy solution. While one could argue that the agent could learn the greedy solution and then just follow it, the agent is also incentivized to improve the greedy solutions, as better solutions will lead to rewards closer to 0, and because we only use negative rewards, the optimal solution would be to have a reward of 0 at each decision point. An additional improvement of the reward function would be not to use the greedy solution but rather the MILP approach, which could be either used in the offline or the online variant, with the offline variant being more desirable as it gives us a lower bound. We found that the MILP approach is computationally too expensive (for either variant as we will find out in Section 8.4), thus not suitable for training the model as training would take too long.

7.2 Architecture

While the Markov Decision Process defines the environment and the agent, we need a model to learn the policy of the agent. In this section, we will describe the general architecture used for the reinforcement learning approach.

Tackling the elevator dispatching problem with reinforcement learning is a challenging task, as the state space is complex and defining an observation suitable for a neural network that would exactly capture all aspects of a state would be huge. The action space is not as vast but still complex, as we need to consider invalid system-actions especially when multiple elevators are involved. We use the Proximal Policy Optimization (PPO) algorithm, which is a policy gradient method, as it is well suited for discrete action

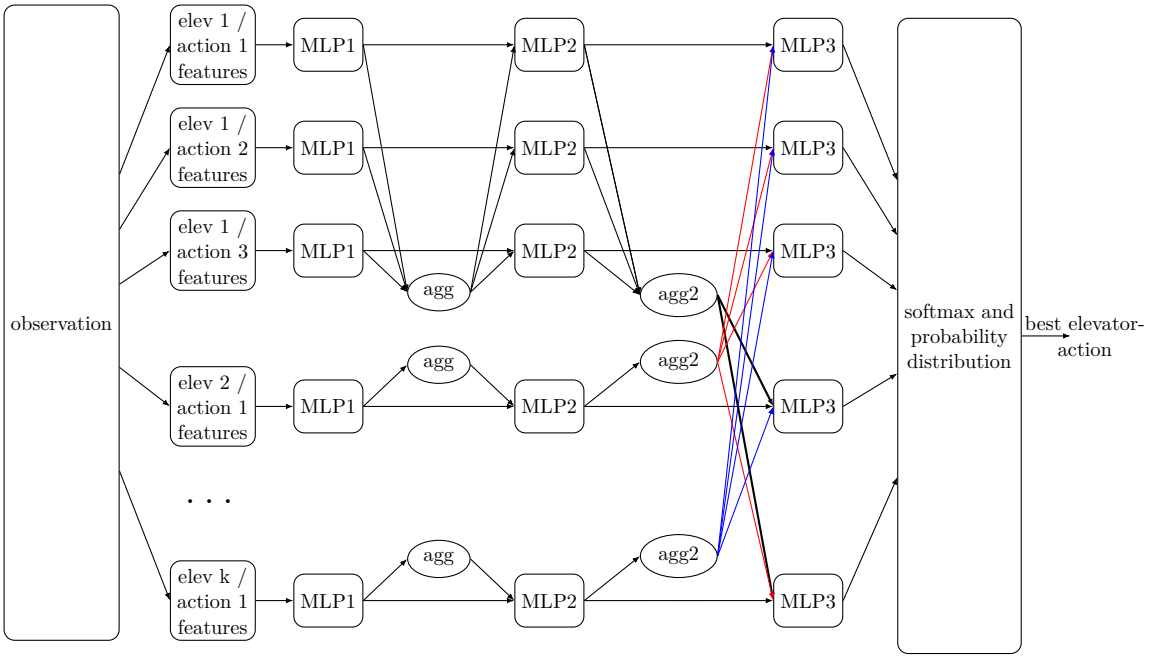


Figure 7.1: Neural Network Model for the Reinforcement Learning Approach with k elevators. Different arrow colors are only for better visualization.

spaces and has been shown to work well in practice [29]. The neural network model of the agent is a major component of PPO and is responsible for learning the policy and in our approach it is used to find the act^{target} of the next elevator-action act we apply. Our approach for the neural network is different to classical RL approaches as we do not use the whole observation as input but rather use sub-observations for each elevator and valid target floor combination and then decide on the best act^{target} by choosing the floor with the highest value. The neural network model is shown in Figure 7.1 and works as follows:

At each decision point of a scenario, the observation is the input, which is then split into sub-observations for each elevator. This sub-observation is then split again and extended to create an elevator-action target observation which has an elevator feature part, where the information about the elevator e is encoded:

- The next planned stop of e one-hot encoded;
- The direction of e one-hot encoded with three values: up, down, and idle;
- The sum of the sizes of orders in e, normalized by the maximum capacity of the elevator;
- The target floors of the orders in e binary encoded;

and an action target part, where the information about the target floor f of the elevator-action is encoded:

- A one-hot encoded vector of the target floor f;
- The time to reach floor f from the current position of e, normalized with the travel time from the ground floor to the top floor;
- The sum of the sizes of orders on floor f which can be picked up by e and travel upwards, normalized by the maximum capacity of the elevator;
- The sum of the sizes of orders on floor f which can be picked up by e and travel downwards, normalized by the maximum capacity of the elevator;
- The sum of the sizes of orders which can be picked up by e on the current floor and have f as destination floor, normalized by the maximum capacity of the elevator and capped at 1.

Each elevator-action target observation is then passed through the first phase of the network, the multilayer perceptron (MLP) MLP1. The input size of MLP1 is dependent on the number of floors in the building ($\mathcal{O}(m)$), as we use one-hot encoding for the sub-observations. The output of MLP1 is a single value, which is grouped by the elevators and aggregated to generate compressed information of all valid target floors of an elevator. Each output of MLP1 is then combined with the aggregated value of the same elevator and passed through the second phase of the network, the multilayer perceptron MLP2, so each target floor can use the information of other possible target floors for this elevator. We may lose some information by the aggregation, but we can keep the input size of MLP2 constant ($\mathcal{O}(1)$) and independent of the number of valid floors. The output of MLP2 is, again, only a single value, which is again grouped by the elevators and aggregated, but this time the aggregation is used as information for other elevators. The input of the next phase, another multilayer perceptron MLP3, is then the output of MLP2 and the aggregated information of all other elevators, which is again a constant ($\mathcal{O}(1)$) input size. The output of MLP3 for each elevator-action combination is then passed through a softmax function, which then gives us a probability distribution over all valid elevator-action targets. When using the model for predictions, the best elevator-action is chosen by the highest probability, but when training the RL model we do not always choose the best elevator-action, but rather sample from the probability distribution, as this is a key part of the PPO algorithm to explore the action space and not get stuck in local optima. Additionally, PPO uses a value net to estimate the value of a state, which calculates the advantage of an action, then updates the policy based on the advantage. The input of the value net are the logits from the output of MLP3 before the softmax is applied. The value net is not shown in Figure 7.1, as it is not part of the decision-making process, but rather used for training the model.

After deciding on the best elevator-action we only have a single elevator-action rather than a full system-action, but as briefly touched on before, finding valid system-actions

is not straightforward in the RL context, thus we find the full system-action by using an autoregressive solution construction. We decide on an elevator-action, then update the observation based on the chosen elevator-action and repeat the process with the new observation until we have a full system-action. In each iteration we use the same neural network model structure but ignore the information of the elevator we already decided on. This increases the computational complexity of the model, as we need to run the neural network multiple times for each decision point, but it is necessary to find valid system-actions, as complex invalidation rules are hard to model in the reinforcement learning model. The increased computational effort is reduced as we do not consider invalid elevator-actions (e.g. floors where no (un)loading happens), thus saving a large amount of computational resources for each iteration, as in practice (depending on the load of the system) only a fraction of available floors are also valid elevator-action targets. The exact structure of the MLPs, the value net, and the aggregation function are not fixed and can be adjusted, which we will further explore in Chapter 8, where we will discuss the number of layers and the number of neurons in each layer as well as the activation function used for the MLPs, and further compare different aggregation functions used for the aggregation of the output of the MLPs. Note that while the Figure 7.1 shows different nodes for MLP1 (also MLP2 and MLP3), the weights of these subnetworks are shared across all nodes (for MLPs with the same name) and are updated only after an episode has ended.

After finding the best floor for each elevator we need the second part of the elevator-action act, the orders to load on the current floor act^{load}. While this could also be done by an additional MLP, we decided to use a greedy function to not further increase the complexity of the system. This is done in a similar way as in the greedy approach, but we need to consider that we are given a floor and have to find the best orders to load in the elevator. For this approach we only need to look at a certain state of the system, namely where the elevator is currently waiting at a floor f with waiting orders, and for the target floor f' holds $f \neq f'$, i.e., the elevator is not waiting. For all other states we cannot pick up any orders in this elevator-action, thus always return an empty list. To decide on the orders we first discard any order p on the current floor that has their destination floor b_p in the opposite direction of the target floor, i.e., if $f < f'$ then $b_p < f$ or if $f > f'$ then $b_p > f$, or have a destination floor that is between the current floor and the target floor, i.e., $f < b_p < f'$, as the elevator will drive past floor b_p without stopping, thus making the elevator-action invalid. After filtering the invalid orders we sort the remaining orders by their group size in non-increasing order and then by distance to their destination floor in non-decreasing order. The group size is (greedily) more important as larger groups have a bigger impact on the AWT, making it more desirable to load them earlier. We then iterate through the sorted orders and pick them up if the elevator has enough capacity left, this is done for every elevator, resulting in a full system-action for the current decision point.

CHAPTER 8

Experiments

In this chapter, we evaluate the performance of the proposed solving approaches for the elevator dispatching problem for EGCS-DRS. First, we specify the technical setup of the experiments. Then, we describe the different traffic patterns in elevator systems which we consider for the evaluation, and the training process of the reinforcement learning agent. Finally, we present the results of the experiments and discuss the findings.

8.1 Experimental Setup

This computational study is conducted on the computational cluster provided by the Algorithms and Complexity Group of TU Wien. We performed all experiments on a single core of an Intel Xeon E5-2640 v4. All solving approaches are implemented in Julia[1] 1.11.3 [4], while the reinforcement learning agent is realized by using the Python library Stable Baselines3 (SB3) (version 2.3.2) [26] from Julia via the PyCall library (version 1.96.4) [19] with Python[2] 3.12.7 [10]. The MILP approach was implemented with the JuMP package (version 1.24.0) [7] with Gurobi[3] 10.0.3 [12] as underlying solver.

8.2 Instance Generation

For testing our solving approaches, we consider three different types of traffic patterns inspired from real world use cases. Traffic patterns specify the structure of elevator traffic, i.e., the arriving floor and destination floor of orders. These three are the most prominent in the literature [8, 39, 32], thus are the most interesting to consider, although there exist different additional patterns. The three patterns are as follows:

[1] https://www.julialang.org
[2] https://www.python.org
[3] https://www.gurobi.com

- **Up Peak**: In this scenario, the majority of the orders arrive at the ground floor and want to travel to higher floors. The system is under high load, and we must choose wisely which orders to assign and load in which elevator. This scenario represents, for example, a morning rush hour in an office building. The ground floor is the bottleneck floor, where the orders are concentrated.

- **Down Peak**: In this scenario, the majority of the orders arrive at (different) non-ground floors and want to travel to the ground floor. Here the difficulty lies in the fact that the orders are distributed over different floors and the elevators may need to pick up orders on different floors in the same trip. This scenario represents, for example, the end of a work day in an office building, where the employees want to leave the building.

- **Inter-floor**: This scenario is less specific than the other two, here orders arrive at different floors and want to travel to different floors. This is the most general case and has no clear bottleneck floor, in contrast to the other two scenarios, where the ground floor either has the most orders arriving or the most orders leaving. This scenario represents, for example, an office building during working time, where employees move between different floors for meetings or lunch.

Table 8.1: Expected traffic load for traffic patterns.

	Ground-to-Upper	Upper-to-Ground	Random
Up peak	90%	10%	0%
Down Peak	10%	90%	0%
Inter-floor	0%	0%	100%

Each traffic pattern has defined rates of the travel directions of all arriving orders, these are defined in Table 8.1. In our experiments the up peak scenario has 90% of the orders arriving at the ground floor traveling up and 10% of the orders arriving at other floors traveling to the ground floor. The down peak scenario is defined to have 90% of the orders arriving at non-ground floors traveling to the ground floor and 10% of the orders arriving at the ground floor traveling up. For the inter-floor scenarios, we define that 100% of the orders are random, i.e., there is no specific distribution of orders, they arrive at random floors and travel to random floors.

Another important aspect is the temporal distribution of the arrival of the orders. In our experiments, we consider all arrivals of a given scenario to follow an inhomogeneous Poisson distribution, which describes the distribution of the orders in a given interval and defines the probability of a certain number of events occurring in a fixed interval of time. The inhomogeneous Poisson distribution is defined by a rate function $\lambda(t)$, which specifies the rate of arrivals at time t. The rate function for our generated scenarios with the total number of orders n and the maximum arrival time T of any order for a scenario

at time t is defined as follows:

$$\lambda(t) = \begin{cases} \frac{n \cdot 0.1}{T \cdot 0.2} & \text{if t} \leq \text{T} \cdot 0.2; \\ \frac{n \cdot 0.8}{T \cdot 0.7} & \text{if T} \cdot 0.2 < \text{t} \leq \text{T} \cdot 0.7; \\ \frac{n \cdot 0.1}{T \cdot 0.1} & \text{if T} \cdot 0.7 < \text{t}. \end{cases} \tag{8.1}$$

The rate function is designed to model the arrival in a way that the majority of the orders, namely 80% of n, are expected to arrive in the middle of the time interval, while in the first 20%, as well as in the last 10% of the time interval, only 10% of the orders are expected to arrive in each case. This models the fact that in real-world scenarios, some orders may arrive earlier or later than others, which can effect the performance as the elevator system has different states when the majority of orders arrive, because elevators may have already delivered some orders to different floors and are now waiting there.

For the main experiments, we consider two different elevator systems, the first system is a small building with two elevators and ten floors, the second system is a larger building with four elevators and 30 floors. For each system, we generate scenarios for two main cases: one where a large number of orders arrive within a short period of time and another where orders are more temporally distributed. The first case consists of 60 orders arriving within ten minutes, while the second case consists of 120 orders arriving within one hour. For each case, we consider the three traffic patterns described above. We further set the capacity of the elevators to five, and we define the loading time $T_{\text{L}} = 1$. The travel time is defined as $T(1) = 6$ and $T(k) = T(1) + k$ for $k > 1$, which accounts for the elevator's acceleration dynamics, and necessitates a longer travel time for a single floor compared to moving across multiple floors and is loosely based on the values given by Tanaka et al. [34].

8.3 Hyperparameter Tuning and Training of the Reinforcement Learning Agent

This section will discuss the hyperparameter optimization and the training of the reinforcement learning agent for the elevator dispatching problem. The general training for the RL is done by the Proximal Policy Optimization Implementation from the Stable Baselines3 framework, but we need to provide the training data, which are scenarios the agent has to navigate through and learn from. An important part of learning is that we need to make sure that the agent is able to learn from many different scenarios and has seen enough different states of the system, so it can generalize well to new scenarios. To be able to do this, we need to generate a large number of scenarios with specific traffic patterns and different arrival rates. Each episode of the training consists of a different scenario with a traffic pattern depending on the model. We (almost) always train a model for a specific traffic pattern, so each episode in the learning process has the same pattern. The scenarios are generated by randomly picking a number of orders between 10 and 20 and a random maximum arrival time between 10 and 200, then the instance is generated as described in the previous section. Training with these smaller scenarios

showed to work for all models except for the model training on the large system and the up peak pattern, which exhibited difficulties in the learning process. Thus we used, for this model only, scenarios of 60 orders and maximum arrival times between 600 and 1200 for the training, which increased the performance of this model but also required slightly more time to train.

There exist different hyperparameters for the RL agent, which need to be optimized to be able to learn effectively from different scenarios. For hyperparameter optimization, different frameworks already exist, we chose Optuna [1] which uses state-of-the-art algorithms for sampling hyperparameter values and also implements efficient pruning of unpromising trials. We optimize the following hyperparameters with the given ranges, where the ranges are chosen based on the default values of the PPO implementation in Stable Baselines3 and some preliminary experiments we did before the optimization process:

- **Learning Rate**: The learning rate of the agent, which specifies how much the agent should change its policy based on the reward. The learning rate is optimized in the range of $[0.00001, 0.5]$ but sampled log-uniformly, i.e., smaller values are preferred over larger values.
- **Learning Rate Schedule**: The learning rate schedule of the agent, which specifies how the learning rate should change over time. The learning rate schedule is sampled from $\{\textit{constant}, \textit{linear}, \textit{quarter}\}$, where *constant* means the learning rate is constant over time, *linear* means the learning rate decreases linearly over time to zero, and *quarter* means the learning rate decreases linearly to a quarter of the initial learning rate over time.
- **Gamma**: The discount factor of the agent, which specifies how much the agent should value future rewards. The discount factor is sampled from $\{0.9, 0.95, 0.99, 0.995, 0.999, 1\}$.
- **Batch Size**: The batch size of the agent, which specifies how many samples the agent should use for one update of the policy. The batch size is sampled from $\{32, 64, 128, 256, 512\}$.
- **N Epochs**: The number of times each batch of collected data is used for an update. The number of epochs is sampled from $\{5, 10, 15, 20\}$.
- **N Steps**: The number of different scenarios the agent observes before updating the policy. The number of steps is sampled from $\{1024, 2048, 4096\}$.
- **Ent Coef**: The entropy coefficient of the agent, which specifies how much the agent should explore. The entropy coefficient is optimized in the range of $[0, 0.1]$ but sampled log-uniformly.

- **Vf Coef**: The value function coefficient of the agent, which specifies how much the agent should value the value function. The value function coefficient is optimized in the range of $[0, 1]$.
- **Clip Range**: The clip range of the agent, which specifies how much the policy should be clipped to keep the changes of the policy small. The clip range is sampled from $\{0.1, 0.2, 0.3\}$.
- **Target KL**: The target Killback-Leibler divergence (KL) of the agent, which specifies how much the policy should change. If the change of the policy is larger than the target KL, the current update is stopped. The target KL is sampled from $\{0.005, 0.01, 0.02, 0.05, 0.1, 0.3, 0.5, 1, 3\}$.
- **MLP Size**: The size of the MLP1 and the value net of the agent. The size of both MLPs is sampled from $\{small, medium1, medium2, large\}$, where *small* means the MLP has 1 hidden layer with 64 neurons, *medium1* means the MLP has 2 hidden layers with 128 neurons, *medium2* means the MLP has 3 hidden layers with 64 neurons, and *large* means the MLP has 3 hidden layers with 256 neurons.
- **Activation Function**: The activation function of the MLPs of the agent. The activation function is sampled from $\{tanh, relu\}$.
- **Aggregation Function**: The aggregation function used for aggregating MLP1 and MLP2. The aggregation function is sampled from $\{mean, max, sum\}$.
- **Learn Steps**: The number of steps the agent should learn from. The number of learn steps is sampled from $\{100000, 200000, 300000, 400000, 500000, 600000, 700000\}$.

Note that there exist other hyperparameters for the PPO implementation in Stable Baselines3, but we found that these hyperparameters have the most impact on the performance of the agent. All other hyperparameters not listed here are kept at the default values. Furthermore, we limit the number of different hyperparameter trials of the optimization to 100 to save computational resources as the optimization showed to be computationally expensive, especially for the larger system, thus not every possible combination of hyperparameters is tested, but rather a subset of them. We took the results from the optimization and further did some manual tuning to refine the hyperparameters, especially the learn step parameter was not optimized with Optuna, but rather decided by some preliminary experiments done with the other hyperparameters fixed. Besides, we decided that when optimizing the MLP size we only optimize for MLP1 and the value net as the input of MLP2 and MLP3 is relative small in comparison, thus requires fewer neurons. We fix the number of hidden layers to one and the number of neurons for each layer to 64 for MLP2 and MLP3.

Important to note is that the (input) size of the neural network of the RL is dependent on the number of elevators and the number of floors of the elevator system, resulting in varying performance on different systems. Thus, we cannot expect to find general good

hyperparameters for the agent, but need to optimize them for each system individually as larger systems generally need longer to learn and thus may require adjustments, e.g. increasing the learning rate. We also discovered that when training on larger systems there is no single set of hyperparameters which works best for all traffic patterns, but rather we need to find the hyperparameters for each traffic pattern individually, as the agent needs to learn different strategies for different traffic patterns, even if the system is the same.

For our experiments, we trained RL models on each combination of an elevator system with either one of the three specific traffic patterns described in the previous section or with all three combined (referred to as the "All Peak Pattern"). In the latter case, a random traffic pattern is selected for each episode. For the small system, a single set of hyperparameters is chosen because the configuration performed well across the different patterns, while for the large system, each pattern-specific model has different hyperparameters, which are shown in Table 8.2.

Table 8.2: Hyperparameters for the RL agent for different elevator system sizes and traffic patterns.

Hyperparameter	**Small**	**Large (Up)**	**Large (Down)**	**Large (Inter)**	**Large (All)**
Learning Rate	$1.586 \cdot 10^{-4}$	$2.2121 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3.1213 \cdot 10^{-4}$
Learning Rate Schedule	constant	quarter	quarter	quarter	quarter
Gamma	1	1	1	1	1
Batch Size	256	64	64	256	64
N Epochs	15	10	10	10	10
N Steps	4096	4096	4096	4096	4096
Ent Coef	$7.59 \cdot 10^{-7}$	$1.5427 \cdot 10^{-8}$	0	$9.54 \cdot 10^{-7}$	$9.54 \cdot 10^{-7}$
Vf Coef	0.20237	0.48234	0.5	0.28654	0.48234
Clip Range	0.1	0.1	0.1	0.1	0.1
Target KL	0.02	0.5	0.3	0.1	0.5
MLP Size	medium2	medium2	large	large	medium2
Activation Function	tanh	tanh	tanh	tanh	tanh
Aggregation Function	mean	mean	mean	mean	mean
Learn Steps	300000	400000	400000	400000	500000

Looking at the optimized hyperparameter values we see that, while some of them are very similar across system-pattern combinations, there are also interesting differences, e.g., the learning rate, which is lower for the smaller system but stays constant for the whole training duration, while for the larger systems it starts with a higher rate, but it decreases over the learning duration, resulting in larger policy changes in the beginning but more fine-grained changes in the later stages of the learning process. Another important difference is the size of the MLPs, where for the smaller system, the up peak large system and the all peak large system *medium2* is selected, which has 3 hidden layers with 64 neurons, while for the other large systems *large* is chosen, which has 3 hidden layers with 256 neurons. This is interesting as it seems that down and inter peak may require more complex neural networks to learn. Generally one would expect that with increasing size of the system the size of NNs should also be more complex, but when increasing the size of MLPs also the learning gets more complex which may reduce the effectiveness in

some cases. Further we see that the smaller elevator system also requires more epochs, which increases the learning process, as more updates on the policy are done. Another significant difference is the target KL, which is higher for the larger system, allowing for more drastic changes in the policy compared to the smaller system where only small changes are allowed. The batch size is also different across some systems, which is interesting when also looking at the target KL, as the change of the policy is limited by the target KL, so if the change is too large the batch size may not be fully utilized, but if the change is small enough we intensify the learning with a larger batch size. Another important detail is the number of learn steps, which states how many steps the agent should learn from. We found that the all peak pattern needs some additional learn steps as the range of different scenarios is larger than for the other patterns. Generally the number of learn steps should increase with the size of the system, as more information needs to be processed.

Important to note is that some values seem to be generally important, like the gamma value set to one, which means that the agent values rewards which are received later in the episode as much as rewards received earlier in the episode. This seems reasonable as the goal of our rewards is to sum up to the average waiting time of all passengers (normalized with the greedy solution) and if we would value later rewards less, we may not be able to learn effectively as we skew this cumulative reward. Another important similarity is the number of steps, which is set for all systems to 4096, this is due to the fact that it is important to sample enough different scenarios before updating the policy as we deal with vastly different scenarios each episode and need to gather enough information about the current policy and how it performs on different scenarios before updating it. The clip range is also important as it restricts the policy changes (together with the target KL) to keep the policy changes stable, as our agent can be very sensitive in later stages of the learning process, and the policy can diverge drastically if not limited. The optimization process also showed that activation function and aggregation function have a single best value for all systems, namely tanh and mean, respectively.

The training graphs of the RL agents for the small elevator system is shown in Figure 8.1, where we see the average reward of an episode during the learning process. We see that the agent learns effectively and the reward increases over time. The inter-floor pattern shown in Figure 8.1c has a much steeper learning curve, as rewards at the beginning of the training are much lower than rewards for up peak and down peak (Figure 8.1a and Figure 8.1b). In Figure 8.1d we can see that using all three patterns for training results in a learning curve which is in between the learning curves of the individual patterns, starting not as low as inter-floor but also not as high as up/down peak. While it seems that for some agents the rewards have not plateaued yet, experiments showed that increasing the amount of learning steps results most of the time in a decrease of performance, as it seems to overfit to the training data.

The loss of the RL agents for the small system during training is shown in Figure 8.2, where we see that the loss decreases over time for all different agents, which is expected as the agent learns to predict the policy better. We see, similar to the average reward, in

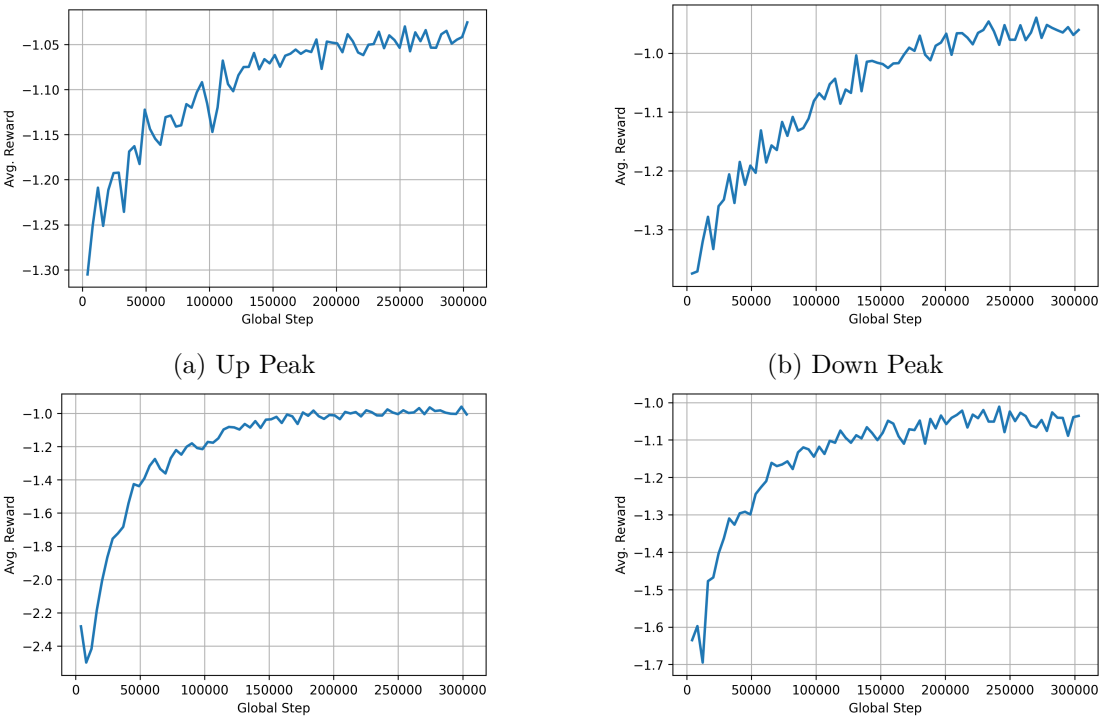
Figure 8.2c that learning from inter-floor patterns starts with a higher loss than learning from up and down peak patterns (Figure 8.2a and Figure 8.2b) and stagnates around the same value as the other patterns but reaches it later in the learning process. The loss for the all peak pattern (Figure 8.2d) is again a middle ground between the individual patterns, starting higher than up and down peak but lower than inter-floor, and reaching lower values earlier than the inter-floor but later than up and down peak pattern.

Looking at the training graphs for the large elevator system in Figure 8.3 we can see for each model that the curve is similar to the small system and the agent learns effectively, as the reward increases over time. We, again, have the up and down peak patterns (Figure 8.3a and Figure 8.3b) with a higher initial reward than the inter-floor pattern (Figure 8.3c), and the all peak pattern has an initial reward in between. The loss graphs of the large system in Figure 8.4 show a difference to the small system loss graphs, as here the up peak pattern (Figure 8.4a) increases the loss in the beginning and then roughly stays the same with high variance. Note that for the up peak training we used larger scenarios which may lead to the more erratic loss graph, but the average reward over time is stable. The down peak pattern (Figure 8.4b) only decreases slightly over time, while the inter-floor pattern (Figure 8.4c) has the most stable loss, which starts with increased loss but decreases over time, and the all peak pattern (Figure 8.4d) is also stable until training step 450000 where sudden large spikes in the loss occur, which may indicate that the agent overfits to the training data.

8.4 Results

In this section, we present the experimental results. We begin by comparing the performance of different solving approaches across the different scenarios described in Section 8.2 and further discuss the limitations encountered for some approaches during the experiments.

The main part of the experiments focuses on the small and large elevator systems, for which we test the short and long instances with each traffic pattern. For each of these cases we present solutions with the offline MILP approach, the online MILP approach, the RL approach trained on the specific traffic pattern and the RL approach trained on all patterns. The results of the greedy approach are mostly used as a baseline to compare the other approaches to. One problem that arises is that the MILP approach, in either offline or online variant, may not solve the instances optimally within a reasonable time frame as we deal with a huge search space as the graph for the network flow formulation grows with the number of orders and the number of floors. To mitigate this issue, we limit the time for the offline MILP to three hours for each instance and for the online MILP we cannot limit the time for the whole instance, but we can limit the time for the computation at each decision point, which is set to ten minutes. The results of the greedy approach for the small elevator system are presented in Table 8.3. In this table, the AWT for the different scenario configurations and for each instance is displayed. We further compare the performance of other approaches against the greedy method, as



(a) Up Peak

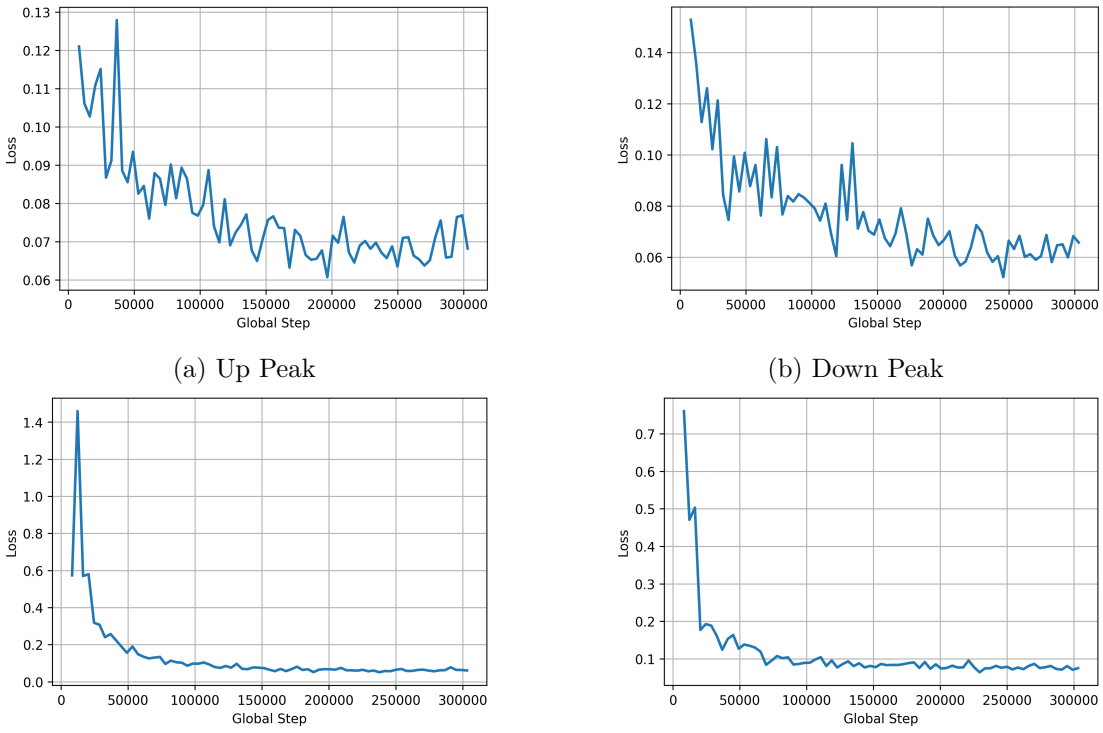
(b) Down Peak

(c) Inter-floor

(d) All Peak

Figure 8.1: Average reward of the RL agent during training for the small elevator system.

shown in Table 8.4, which illustrates the ratio of the AWT of these approaches to that of the greedy approach. Bold values in the table mark the best result across the different approaches. Each result is gathered from a single run of the instance, as the results from all different approaches are generally deterministic. We additionally provide the mean and standard deviation (SD) of the greedy normalized results across all five instances, as well as the relative gap of primal and dual bound for the MILP approaches, which for the online variant is the mean of the relative gaps of all decision points. The relative gap is calculated as $\frac{|z_P - z_D|}{|z_P|}$, where z_P is the primal bound and z_D represents the dual bound. We can see that the offline MILP approach always performs much worse than the other approaches because it always reaches the time limit without finding an optimal solution, which is also shown in the relative gap being always 1.0 thus indicating that the solutions of the bounds are not close at all. The results of the online MILP show that the relative gap is (almost) zero for the long scenarios, i.e., for every decision point the optimal solution was (almost) always found. In contrast, the relative gap for short scenarios is large, but the MILP still finds good solutions. This could be due to the fact that in the beginning of the scenario the number of orders waiting is not as high as later in the scenario, thus the network flow graph is smaller and easier to solve. Early solutions may be (almost) optimal, while later solutions may have a higher gap, but the impact on the overall AWT is not as large as of the early decisions. The strength of the RL approach is



(a) Up Peak

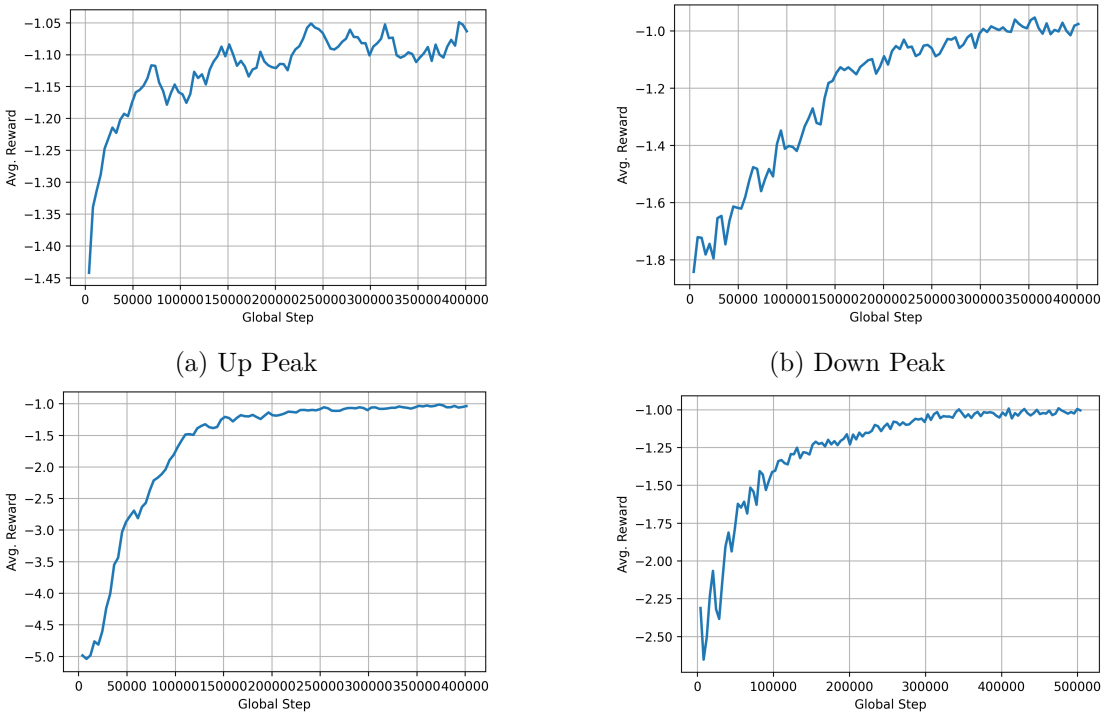
(b) Down Peak

(c) Inter-floor

(d) All Peak

Figure 8.2: Loss of the RL agent during training for the small elevator system.

mostly shown in the short scenarios, where it can learn to navigate the system effectively. There, it outperforms the greedy approach in most cases, especially in the down peak, but is still worse than the online MILP approach. Interestingly the all peak RL approach sometimes outperforms the model trained only on the specific patterns, especially in down peak scenarios, which could indicate that learning states of the other patterns can positively impact the learning process. We can also see in Figure 8.5 that the online MILP approach (almost) always performs best, as it often can find a good solution for most decision points, and we generally see that the specific RL approach performs only a bit better than the all peak RL approach. The boxplots also show that the greedy approach is, in most cases, a good performing algorithm across the different scenarios in comparison to the other approaches. Here we excluded the offline MILP approach as the AWT is too large to be displayed in the plot, without making the other results unreadable. The online MILP still occasionally runs into timeouts and generally takes much longer to solve the scenario than the greedy and RL approaches. Table 8.5 shows the mean solving time for all five scenarios and the maximum time for a single decision point. This confirms that the MILP approach is generally not suitable for real-world applications. The online approach for short instances has a longer runtime than the RL or greedy approach for solving the scenario as it runs into timeouts for most decision points. This is due to the fact that in the short instance the orders are more clustered,



(a) Up Peak

(b) Down Peak

(c) Inter-floor

(d) All Peak

Figure 8.3: Average reward of the RL agent during training for the large elevator system.

so each decision point has to handle many orders. As a result, the network flow graph is larger and more computationally expensive to solve. In contrast, long instances have more spread-out orders, so fewer must be handled at each decision point. The greedy approach yields solutions almost instantly, while the reinforcement learning approach takes a bit longer but can still manage scenarios within a reasonable timeframe.

Table 8.3: Average waiting time of the greedy approach across configurations for the small elevator system in seconds.

Size	**Patt.**	**Inst. 1**	**Inst. 2**	**Inst. 3**	**Inst. 4**	**Inst. 5**	**Mean**	**SD**
Short	Up	73.88	83.65	41.42	30.77	48.87	55.72	22.27
	Down	51.18	66.82	83.33	120.22	71.75	78.66	25.98
	Inter	38.75	38.24	48.90	41.26	72.76	47.98	14.49
Long	Up	11.31	11.14	10.98	11.22	11.19	11.17	0.12
	Down	13.54	12.91	13.92	13.42	13.59	13.48	0.37
	Inter	9.75	10.72	10.27	9.93	10.33	10.20	0.37

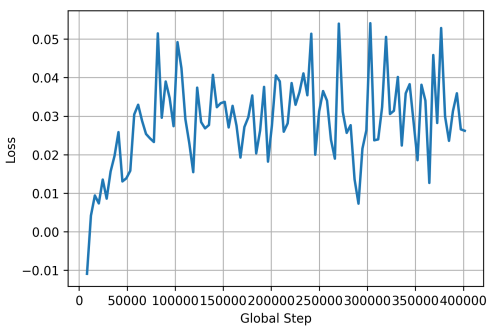
The results for the greedy approach for the large elevator system are shown in Table 8.6, and again we also compare it to the other approaches, displayed in Table 8.7. Our findings

Table 8.4: Greedy normalized AWT comparison of different approaches across configurations for the small elevator system.

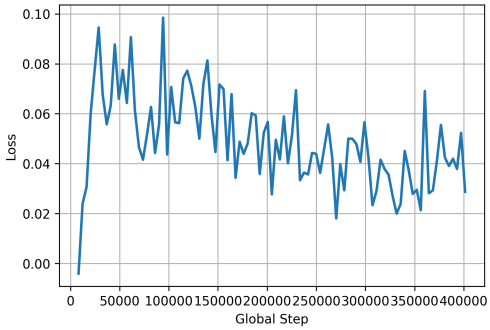
Size	Patt.	Approach	Inst. 1	Inst. 2	Inst. 3	Inst. 4	Inst. 5	Mean	SD	rel. Gap
Short	Up	MILP Off.	2.03	1.47	4.71	13.52	3.68	5.08	5.04	1.00
		MILP On.	**0.92**	**0.98**	**0.92**	**0.85**	**0.78**	**0.89**	0.07	0.41
		RL Up	0.94	1.04	0.94	1.09	0.87	0.98	0.08	-
		RL All	1.01	1.00	1.10	1.25	0.87	1.04	0.14	-
	Down	MILP Off.	4.97	1.57	1.55	2.61	2.64	2.67	1.33	1.00
		MILP On.	**0.46**	**0.61**	**0.54**	**0.62**	**0.47**	**0.54**	0.07	0.29
		RL Down	0.68	0.74	0.59	0.68	0.54	0.65	0.08	-
		RL All	0.58	0.78	0.58	0.66	0.52	0.62	0.10	-
	Inter	MILP Off.	3.13	2.49	2.31	2.95	2.36	2.65	0.37	1.00
		MILP On.	**0.74**	**0.70**	**0.77**	**0.84**	**0.55**	**0.72**	0.10	0.34
		RL Inter	0.97	1.08	0.84	0.91	0.77	0.91	0.12	-
		RL All	1.43	1.11	1.01	1.56	1.07	1.32	0.24	-
Long	Up	MILP Off.	169.74	-	101.73	117.64	99.30	122.60	32.91	1.00
		MILP On.	**0.99**	**1.00**	**1.00**	**1.00**	**0.95**	**0.99**	0.02	$5.40 \cdot 10^{-4}$
		RL Up	1.10	1.12	1.11	1.14	0.98	1.09	0.06	-
		RL All	1.12	1.10	1.14	1.15	1.04	1.11	0.04	-
	Down	MILP Off.	-	-	-	99.37	-	99.37	0.00	1.00
		MILP On.	**0.89**	**0.89**	**0.92**	**0.88**	0.94	**0.90**	0.02	0.00
		RL Down	0.97	1.02	1.02	1.02	0.93	0.99	0.04	-
		RL All	0.96	0.99	1.01	1.02	**0.92**	0.98	0.04	-
	Inter	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	**0.88**	**0.93**	**0.93**	**0.94**	**0.99**	**0.93**	0.04	$1.60 \cdot 10^{-3}$
		RL Inter	1.11	1.13	1.09	1.08	1.14	1.11	0.02	-
		RL All	1.17	1.20	1.14	1.08	1.14	1.15	0.05	-

Table 8.5: Average solving time for the whole scenario and the longest decision point for different approaches for the small elevator system in seconds.

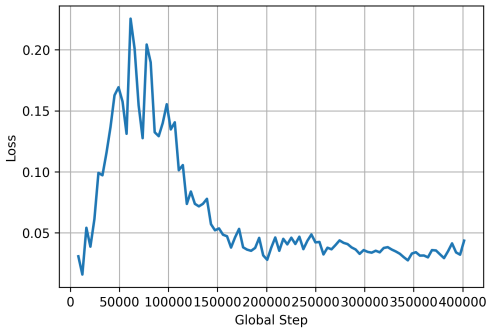
Size	Approach	Up		Down		Inter	
		Scenario	Point	Scenario	Point	Scenario	Point
Short	MILP Off.	10800.00	-	10800.00	-	10800.00	-
	MILP On.	16524.68	600.00	14796.62	600.00	16164.65	600.00
	Greedy	0.01	0.00	0.01	0.00	0.01	0.00
	RL	0.92	0.17	0.69	0.15	0.63	0.15
	RL All	0.95	0.19	0.71	0.13	0.59	0.14
Long	MILP Off.	10800.00	-	10800.00	-	10800.00	-
	MILP On.	1659.27	600.00	1683.04	577.14	1510.15	600.00
	Greedy	0.01	0.00	0.01	0.00	0.01	0.00
	RL	0.97	0.02	1.34	0.03	1.30	0.02
	RL All	0.94	0.02	1.35	0.03	1.36	0.03



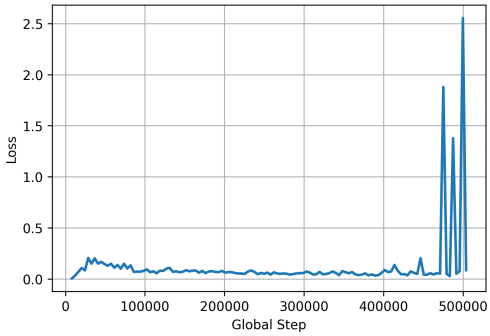
(a) Up Peak



(b) Down Peak



(c) Inter-floor



(d) All Peak

Figure 8.4: Loss of the RL agent during training for the large elevator system.

for the large systems show a similar picture to the findings of the small system. The greedy approach seems to perform generally better on larger systems, especially for the short up peak scenarios, where it is better than the online MILP on every instance. This is likely because, in the larger system, the computational complexity increases, making it harder for the MILP to find an optimal solution within the given time. The large relative gap in the short scenarios further supports this. Interestingly, for the long scenarios the online MILP either did not find a valid solution in time for a decision point or if a solution was found, the relative gap of the solutions was always very small. The RL approach still performs better than the greedy approach in (most) short down peak and inter-floor scenarios but is worse in the other scenarios, indicating that the model may not scale well with the size of the system, as the state space grows with the number of elevators and floors, making it harder for the RL to learn effectively, and maybe also indicating that the flow of information between the elevators may not be as effective in larger systems. In the long up peak scenario the RL approach performed the worst out of all experiments, which was also due to the fact that the learning proved to be difficult and we needed to adjust the training data to longer scenarios in order to get more stable learning curves. When looking at the boxplots in Figure 8.6 we see a greater variance in the performance of the different approaches, but still the specific RL approach is mostly better than the all peak RL approach. For the longer scenarios, the greedy approach

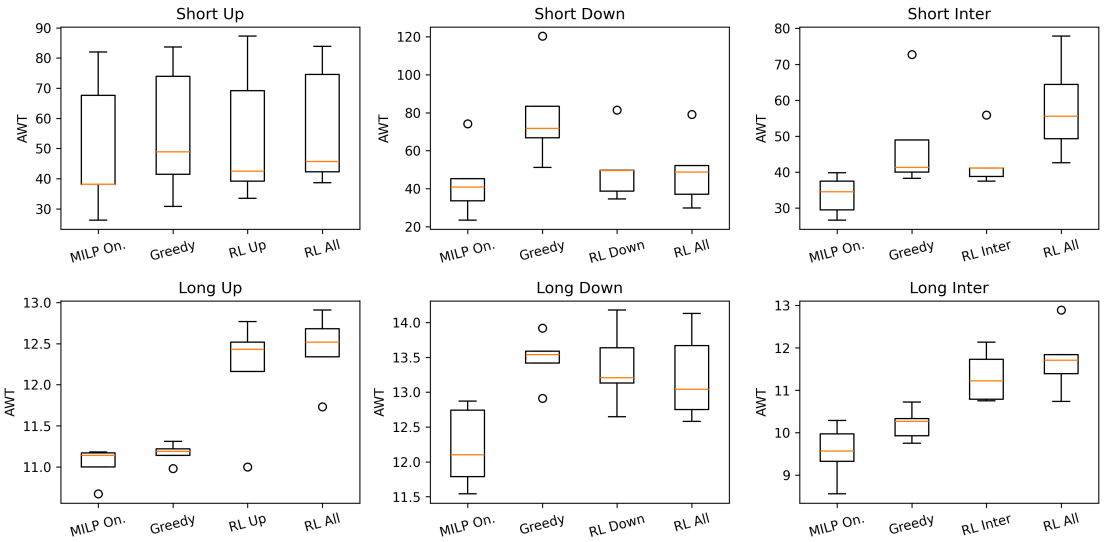


Figure 8.5: Boxplots of AWTs of different approaches for the small elevator system.

outperforms the RL approaches clearly, but it seems that the online MILP approach is still the best performing approach for these scenarios, if it finds a solution in time, although we only have one result, so it is hard to make a general statement. The offline MILP can only solve one out of the 30 instances and does not find any valid solution for the others in three hours time, and even the online MILP approach often does not find a solution for a decision point, as especially in the longer scenarios the network graph grows too large to handle in the given time. In Table 8.8 we can also see that the average solving time for an instance increased for the RL approach, while the greedy approach still performs very efficiently, but the maximal time for solving a decision point is still very fast and shows that the RL approach could be used in real-world applications.

Table 8.6: Average waiting time of the greedy approach across configurations for the large elevator system in seconds.

Size	**Patt.**	**Inst. 1**	**Inst. 2**	**Inst. 3**	**Inst. 4**	**Inst. 5**	**Mean**	**SD**
Short	Up	37.47	34.38	34.63	33.78	52.63	38.58	7.98
	Down	34.99	48.12	53.47	42.31	71.97	50.17	13.99
	Inter	22.66	23.85	27.79	22.70	28.20	25.04	2.75
Long	Up	16.66	19.52	17.48	17.36	18.24	17.85	1.09
	Down	18.29	15.73	20.44	18.19	18.68	18.27	1.68
	Inter	11.28	11.78	11.34	12.98	10.16	11.51	1.02

Because the examples of the two scenarios both did not show the real lower bound the offline MILP can achieve, we decided to generate a very small down peak example with only six orders within one minute on the small elevator system. The results, presented in Table 8.9, show that the offline MILP, when not cut off by the time limit and the relative

Table 8.7: Greedy normalized AWT comparison of different approaches across configurations for the large elevator system.

Size	Patt.	Approach	Inst. 1	Inst. 2	Inst. 3	Inst. 4	Inst. 5	Mean	SD	rel. Gap
Short	Up	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	1.09	1.27	1.25	**1.05**	-	1.16	0.10	0.47
		RL Up	1.07	**1.04**	1.03	1.14	**1.11**	**1.08**	0.04	-
		RL All	**1.06**	1.25	**0.94**	1.15	1.13	1.10	0.11	-
	Down	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	**0.61**	**0.91**	**0.74**	**0.75**	-	**0.75**	0.12	0.28
		RL Down	0.89	1.06	0.95	0.86	0.92	0.94	0.07	-
		RL All	0.95	1.01	0.79	0.98	**0.90**	0.92	0.09	-
	Inter	MILP Off.	-	15.54	-	-	-	15.54	0.00	1.00
		MILP On.	**0.92**	-	-	0.98	**0.93**	**0.94**	0.03	0.27
		RL Inter	0.93	**0.91**	**0.85**	1.04	1.02	0.95	0.08	-
		RL All	1.10	0.94	1.01	**0.96**	1.11	1.02	0.08	-
Long	Up	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	-	-	-	-	-	-	-	-
		RL Up	1.20	**1.02**	**1.10**	**1.11**	**1.04**	**1.09**	0.07	-
		RL All	**1.15**	1.06	1.16	1.14	1.06	1.12	0.05	-
	Down	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	**0.91**	-	-	-	-	0.91	0.00	0.02
		RL Down	1.07	**1.11**	**1.08**	**1.05**	**1.06**	**1.07**	0.02	-
		RL All	1.08	1.12	1.13	1.08	1.09	1.10	0.02	-
	Inter	MILP Off.	-	-	-	-	-	-	-	-
		MILP On.	-	-	-	-	-	-	-	-
		RL Inter	1.51	1.37	1.63	1.39	**1.56**	1.49	0.10	-
		RL All	**1.37**	**1.36**	**1.59**	**1.37**	1.67	**1.47**	0.15	-

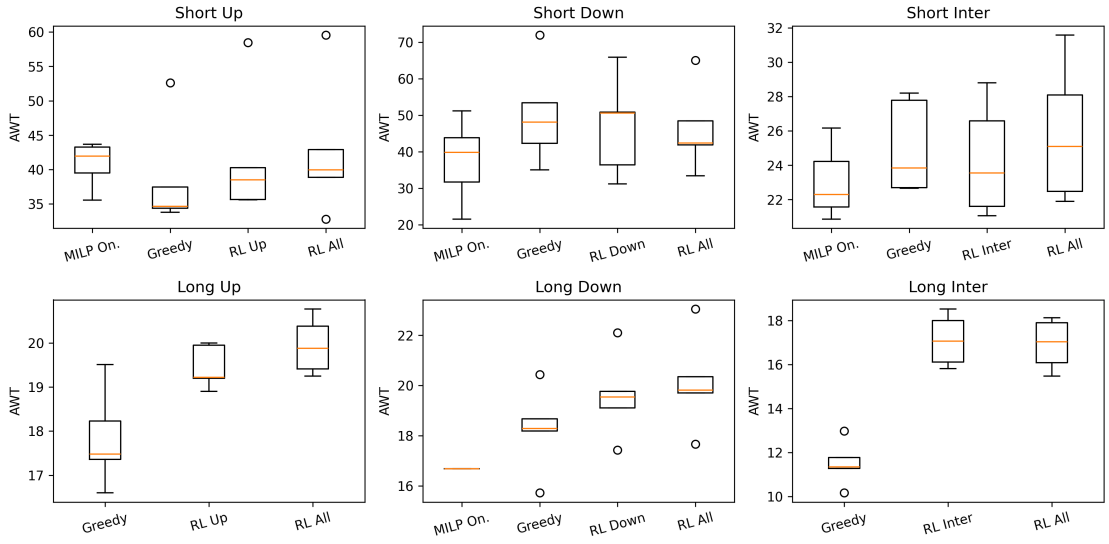


Figure 8.6: Boxplots of AWTs of different approaches for the large elevator system.

Table 8.8: Average solving time for the whole scenario and the longest decision point for different approaches for the large elevator system in seconds.

Size	Approach	Up		Down		Inter	
		Scenario	Point	Scenario	Point	Scenario	Point
Short	MILP Off.	10800.00	-	10800.00	-	10800.00	-
	MILP On.	22050.87	600.00	12572.71	600.00	14461.85	600.00
	Greedy	0.06	0.04	0.01	0.00	0.01	0.00
	RL	2.25	0.17	2.28	0.15	2.0	0.12
	RL All	3.32	0.57	3.05	0.49	2.86	0.67
Long	MILP Off.	10800.00	-	10800.00	-	10800.00	-
	MILP On.	-	-	11938.46	600.00	-	-
	Greedy	0.01	0.00	0.01	0.00	0.01	0.00
	RL	7.10	0.33	6.66	0.21	6.90	0.21
	RL All	8.02	0.42	6.92	0.22	6.73	0.20

gap being zero, can achieve a much lower AWT than the other approaches. While this is only possible because the offline MILP knows all information in advance, it is still interesting to see how large the gap between the online approaches and the theoretical lower bound can be, for some instances it reduces the AWT by around 70% compared to the online approaches. Note that here the greedy approach sometimes performs better than the online MILP approach, which shows that even if we make suboptimal decisions at a decision point it may lead to better results for the whole scenario.

Table 8.9: Greedy normalized AWT comparison of different approaches on a very small down peak example to show the lower bound with the offline MILP.

Approach	Inst. 1	Inst. 2	Inst. 3	Inst. 4	Inst. 5	Mean	SD	rel. Gap
MILP Off.	**0.42**	**0.45**	**0.42**	**0.54**	**0.32**	**0.43**	0.07	0.00
MILP On.	1.16	0.95	0.86	0.95	0.79	0.94	0.14	0.00
RL Down	1.16	0.97	0.84	0.96	0.93	1.01	0.21	-
RL All	1.36	1.23	0.87	0.95	0.93	1.07	0.22	-

Table 8.10: Results of the RL all peak approach compared to the greedy approach for two very large elevator systems with 50 and 100 floors and six and four elevators, respectively.

System Size		Pattern	Mean ratio	SD
k	m			
4	100	Up	1.05	0.09
		Down	0.92	0.07
		Inter	1.38	0.19
6	50	Up	1.43	0.23
		Down	1.31	0.22
		Inter	1.42	0.23

To also show some results for even larger systems we use an elevator system with six elevators and 50 floors and an elevator system with four elevators and 100 floors, where we only compare the RL all peak approach with the greedy approach, because the MILP approaches take too long to find any feasible solution, as we have seen in previous examples. We used the same hyperparameters as for the large elevator system and trained for 700000 steps because we need more time to learn as our observation space increases and scenarios vary more. We generate 50 up peak, down peak and inter-floor scenarios with 60 orders with the latest arrival time being 600, and again show the mean ratio of the RL approach to the greedy approach as well as the standard deviation. The results are shown in Table 8.10, where we can see that the RL approach performs generally worse than for smaller systems. For the four elevator and 100 floor system the RL approach is a bit worse than the greedy compared to the smaller systems, but still manages to outperform the greedy in the down peak scenarios. For the six elevator and 50 floor system the RL approach is worse in every scenario, which shows that the RL approach does not scale as well with the size of the system, especially when the number of elevators increases, indicating that the information flow across the elevators may not be as effective and too compressed for the RL to learn effectively.

CHAPTER 9

Conclusion and Future Work

In this thesis, we have presented a comprehensive overview of the elevator assignment and control problem for elevator group control systems with a destination registration system and three different approaches tackling it, with the goal of minimizing the average waiting time of passengers. We looked at different traffic patterns, i.e., up peak, down peak and inter-floor, which define the distribution of arrival floors and destination floors of a given instance, to simulate different real-world scenarios. We looked at the problem from an offline perspective, where all the information about a scenario and the arriving passengers is known in advance, as well as an online perspective, where the information about the passengers is not known in advance and a real-life scenario can be simulated as decisions have to be made only on current information about the system. We used the online perspective as the main focus of this thesis, as it is more realistic and challenging, while the offline perspective was used for comparison to show the lower bound on objective values of the problem and what can be achieved with optimal solutions.

The first approach was a mixed integer linear Program, which was the only approach applied to both perspectives. Both variants used MTZ formulations and a network flow approach to model the movement of the elevator cars across time. The offline version of the MILP can optimally solve the problem and generate a complete elevator schedule, determining which orders to pick up and which floors to travel to – providing sufficient time is given. The online version deals with the dynamic nature of the problem by re-solving the MILP every time a new passenger arrives or an elevator reaches its destination, which is generally less computationally expensive than the offline version, but still does not scale well with the number of elevators and passengers.

The second approach was a greedy algorithm, which is a simple and fast heuristic that assigns passengers to elevators based on a greedy strategy. The main goal of the greedy approach is to assign elevators to pick up the nearest orders first and also try to deliver orders which have nearer destination floors first. The algorithm also needs to handle the coordination of the elevators in a way that they do not pick up the same orders, which

was achieved iteratively by finding, for each elevator, the order it can pick up in the nearest future, and the elevator with the earliest pick up time is then assigned to the order. The process is repeated until all elevators have been assigned an order or no more orders are available.

The third and last approach to solve the elevator assignment problem was a reinforcement learning approach, which uses a PPO algorithm to learn a policy that assigns actions to elevators for each decision point. The observation and the neural network model were designed to be able to handle the generally large state space of the problem by dividing the state information into parts containing only information for a specific action, which was then used to predict the value of the action. To be able to handle coordination between the elevators and their actions, aggregation of the action values was used to ensure that we do not predict the value in isolation but rather in the context of the other elevators. The training of the model was either done specifically for a given traffic pattern or with a general model that was trained on all traffic patterns. The reward of an episode summed up to be the negative of the average waiting time of all passengers normalized by the solution of the greedy approach to give the model a baseline to distinguish between good and bad partial solutions.

The experiments showed that the offline MILP approach was only able to generate optimal solutions for small instances. For larger instances, the computational complexity was too high to be practical and the solutions after three hours of computation were far from optimal and even worse than solutions from the online approaches. The online MILP approach on the other hand provided for most of the tested instances the best solutions, but the computational complexity was still too high to be practical for real-time applications. On some instances, the online MILP approach also ran into the given time limit of ten minutes per decision point and thus could not find valid solutions for the scenario, further showing that the MILP does not scale well with increasing size of the system. The greedy approach showed to be the most efficient approach in terms of computation time and was able to provide good solutions for most of the tested instances and even outperformed the online MILP approach in some cases. The reinforcement learning approach was not easy to tune, as it showed to be quite sensitive to the hyperparameters, and needed fine-tuning for each traffic pattern and elevator system to be able to provide good and competitive solutions. For most models, we found suitable settings that were able to generate reasonable solutions, especially for smaller elevator systems and for scenarios where many passengers arrive in a short time, as well as scenarios with down peak traffic. The RL approach generally produced better solutions when the model was trained on a specific traffic scenario rather than on all traffic patterns combined. Increasing the size of the system, especially the number of elevators, showed that the performance of the RL model, in comparison to the greedy approach, decreased, indicating that the RL approach is not able to scale as well as the greedy approach, w.r.t. the solution quality and computation time.

Future work may focus on improving the MILP by finding a different approach to model the problem or improving our approach, where a focus point could be the network flow

graph as the maximum number of trips is often the bottleneck of the computation time and finding tighter bounds could increase the efficiency.

Another interesting approach would be to add a neural network for the pickup decision to the RL approach, as in our approach this was done by a greedy strategy. This could improve the overall performance as the pickup process can be crucial for the whole process, especially in scenarios where many orders arrive within a short time on the same floor, i.e., in up peak traffic scenarios. New approaches could also focus on the communication between the elevators, as this was a crucial part of the problem and the performance of the RL approach decreased with increasing number of elevators for most scenarios, indicating that improvements can be made in this area.

The recognition of the traffic pattern is another crucial part, which was not touched on in this work, but could improve performance in real-world scenarios. A new approach could focus on recognizing the traffic pattern and adjusting the model to the specific pattern.

Overview of Generative AI Tools Used

No generative AI tools were used while working on this thesis.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[2] T. Beielstein, C. Ewald, and S. Markon. Optimal elevator group control by evolution strategies. In *Genetic and Evolutionary Computation — GECCO*, pages 1963–1974, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. URL https://doi.org/10.1007/3-540-45110-2_95.

[3] D. Bertsimas and J.N. Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, 1997.

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, abs/1411.1607, 2014. URL http://arxiv.org/abs/1411.1607.

[5] P. Cortés, J. Larrañeta, and L. Onieva. Genetic algorithm for controllers in elevator groups: analysis and simulation during lunchpeak traffic. *Applied Soft Computing*, 4 (2):159–174, 2004. URL https://doi.org/10.1016/j.asoc.2003.11.002.

[6] R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998. URL https://doi.org/10.1023/A:1007518724497.

[7] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. URL https://doi.org/10.1137/15M1020575.

[8] J. R. Fernandez and P. Cortes. A survey of elevator group control systems for vertical transportation: A look at recent literature. *IEEE Control Systems Magazine*, 35(4):38–55, 2015. URL https://doi.org/10.1109/MCS.2015.2427045.

[9] J. Fernández, P. Cortés, J. Muñuzuri, and J. Guadix. Dynamic fuzzy logic elevator group control system with relative waiting time consideration. *IEEE Transactions on Industrial Electronics*, 61(9):4912–4919, 2014. URL https://doi.org/10.1109/TIE.2013.2289867.

[10] Python Software Foundation. Python language reference, 2025. URL https://www.python.org/.

[11] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A. Togawa. An elevator group control system with floor-attribute control method and system optimization using genetic algorithms. *IEEE Transactions on Industrial Electronics*, 44(4):546–552, 1997. URL https://doi.org/10.1109/41.605632.

[12] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 2025. Available at: https://www.gurobi.com/documentation/.

[13] J. B. Halpern. Elevator operation and control. In *The Vertical Transportation Handbook*, pages 131–180. John Wiley & Sons, Ltd, 2010. URL https://doi.org/10.1002/9780470949818.ch7.

[14] M. Hamdi and D.J. Mulvaney. Prioritised A* search in real-time elevator dispatching. *Control Engineering Practice*, 15(2):219–230, 2007. URL https://doi.org/10.1016/j.conengprac.2006.06.005.

[15] K. Ikeda, H. Suzuki, H. Kita, and S. Markon. Exemplar-based control of multi-car elevators and its multi-objective optimization using genetic algorithm. In *The 23rd International Technical Conference on Circuits/Systems, Computers and Communications*, pages 701–704, 2008. URL https://doi.org/10.34385/proc.39.H5-4.

[16] M. Ikuta, K. Takahashi, and M. Inaba. Strategy selection by reinforcement learning for multi-car elevator systems. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 2479–2484, 2013. URL https://doi.org/10.1109/SMC.2013.423.

[17] N. Imasaki, S. Kubo, S. Nakai, T. Yoshitsugu, J. Kiji, and T. Endo. Elevator group control system tuned by a fuzzy neural network applied method. In *Proceedings of IEEE International Conference on Fuzzy Systems.*, volume 4, pages 1735–1740, 1995. URL https://doi.org/10.1109/FUZZY.1995.409916.

[18] J. Jamaludin, N. A. Rahim, and W. P. Hew. An elevator group control system with a self-tuning fuzzy logic group controller. *IEEE Transactions on Industrial Electronics*, 57(12):4188–4198, 2010. URL https://doi.org/10.1109/TIE.2010.2044117.

[19] Steven G. Johnson. Pycall.jl: Calling python from julia. GitHub repository, 2014. URL https://github.com/JuliaPy/PyCall.jl.

[20] J. Koehler and D. Ottiger. An AI-based approach to destination control in elevators. *AI Magazine*, 23(3):59, 2002. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1657.

[21] S. Kuzunuki, Y. Morita, K. Yoneda, T. Ueshima, and T. Tobita. Group-control method and apparatus for an elevator system with plural cages, U.S. Patent 4947965, 1990. URL https://patents.google.com/patent/US4947965A/ko.

[22] D. Levy, M. Yadin, and A. Alexandrovitz. Optimal control of elevators. *International Journal of Systems Science*, 8(3):301–320, 1977.

[23] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, 1960. URL https://doi.org/10.1145/321043.321046.

[24] V. Novak, I. Perfiljeva, and J. Mockor. *Mathematical Principles of Fuzzy Logic*, pages 9–14. Springer New York, 1999. URL https://doi.org/10.1007/978-1-4615-5217-8.

[25] D.L. Pepyne and C.G. Cassandras. Optimal dispatching control for elevator systems during uppeak traffic. *IEEE Transactions on Control Systems Technology*, 5(6): 629–643, 1997. URL https://doi.org/10.1109/87.641406.

[26] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[27] M. Ruokokoski, H. Ehtamo, and P. M. Pardalos. Elevator dispatching problem: a mixed integer linear programming formulation and polyhedral results. *Journal of Combinatorial Optimization*, 29(4):750–780, 2015. URL https://doi.org/10.1007/s10878-013-9620-1.

[28] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[30] Z. Shen and Q. Zhao. A branch and bound method to the continuous time model elevator system with full information. In *SICE Annual Conference 2007*, pages 327–330, 2007. URL https://doi.org/10.1109/SICE.2007.4421001.

[31] J. Sorsa, H. Hakonen, and M. L. Siikonen. Elevator selection with destination control system. *Elevator World*, 54(1):148, 2006. URL https://api.semanticscholar.org/CorpusID:63721246.

[32] J. Sun, Q. Zhao, P.B. Luh, and M.J. Atalla. Estimation of optimal elevator scheduling performance. In *Proceedings of IEEE International Conference on Robotics and*

Automation, ICRA., pages 1078–1083, 2006. URL https://doi.org/10.1109/ROBOT.2006.1641853.

[33] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[34] S. Tanaka, Y. Uraguchi, and M. Araki. Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part I. formulation and simulations. *European Journal of Operational Research*, 167(2):550–573, 2005. URL https://doi.org/10.1016/j.ejor.2004.04.038.

[35] S. Tanaka, Y. Uraguchi, and M. Araki. Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part II. the solution algorithm. *European Journal of Operational Research*, 167(2):574–587, 2005. URL https://doi.org/10.1016/j.ejor.2004.04.039.

[36] S. Tsuji, M. Amano, and S. Hikita. Application of the expert system to elevator group-supervisory control. In *Proceedings. The Fifth Conference on Artificial Intelligence Applications*, pages 287–294, 1989. URL https://doi.org/10.1109/CAIA.1989.49165.

[37] T. Tyni and J. Ylinen. Method and apparatus for allocating landing calls in an elevator group, U.S. Patent 5932852, 1999.

[38] A. Valdivielso, T. Miyamoto, and S. Kumagai. Multi-car elevator group control: Schedule completion time optimization algorithm with synchronized schedule direction and service zone coverage oriented parking strategies. In *Proceedings of the 23rd International Technical Conference on Circuits/Systems, Computers and Communications*, pages 689–692, Osaka, Japan, 2008. URL https://api.semanticscholar.org/CorpusID:15803300.

[39] Q. Wei, L. Wang, Y. Liu, and M. M. Polycarpou. Optimal elevator group control via deep asynchronous actor–critic learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(12):5245–5256, 2020. URL https://doi.org/10.1109/TNNLS.2020.2965208.

[40] B. L. Whitehall, T. M. Christy, and B. A. Powell. Method for continuous learning by a neural network used in an elevator dispatching system, U.S. Patent 5923004, 1999.

[41] L. A. Wolsey. *Integer Programming: 2nd Edition*. Wiley, 2020. URL https://books.google.at/books?id=x7RvQgAACAAJ.

[42] J. Xu and T. Feng. Single elevator scheduling problem with complete information: An exact model using mixed integer linear programming. In *2016 American Control Conference (ACC)*, pages 2894–2899, 2016. URL https://doi.org/10.1109/ACC.2016.7525358.

[43] L. Yu, J. Zhou, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. Double-deck elevator group supervisory control system using genetic network programming with ant colony optimization. In *IEEE Congress on Evolutionary Computation*, pages 1015–1022, 2007. URL https://doi.org/10.1109/CEC.2007.4424581.

[44] L. Yu, S. Mabu, and K. Hirasawa. Multi-car elevator system using genetic network programming for high-rise building. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1216–1222, 2010. URL https://doi.org/10.1109/ICSMC.2010.5642411.

[45] F. Zheng, H. Yao, R. Liang, and M. Tu. Research on elevator group control algorithms. In *6th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI)*, pages 563–567, 2024. URL https://doi.org/10.1109/IoTAAI62601.2024.10692730.

[46] J. Zhou, T. Eguchi, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. A study of applying genetic network programming with reinforcement learning to elevator group supervisory control system. In *IEEE International Conference on Evolutionary Computation*, pages 3035–3041, 2006. URL https://doi.org/10.1109/CEC.2006.1688692.

[47] J. Zhou, L. Yu, S. Mabu, K. Hirasawa, J. Hu, and S. Markon. Double-deck elevator systems using genetic network programming with reinforcement learning. In *IEEE Congress on Evolutionary Computation*, pages 2025–2031, 2007. URL https://doi.org/10.1109/CEC.2007.4424722.