Redeployment in Convoys of Fleets of Shared Vehicles

Jan-Thierry Wegener

N° d'ordre : D.U 2722
EDSPIC : 767


UNIVERSITÉ BLAISE PASCAL – Clermont-Ferrand II

ECOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

Thèse

présentée par

Jan-Thierry WEGENER

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Specialité : Informatique

Redeployment in Convoys of Fleets of Shared Vehicles

Redéploiement en convois de flottes de véhicules partagés

Soutenue publiquement le 26 juillet 2016 devant le jury :

Présidente :
Prof. Fatiha BENDALI - Université Blaise Pascal, France
Rapporteurs :
Prof. Nabil ABSI - Ecole des Mines de Saint-Etienne, France
Prof. Aziz MOUKRIM - Université de Technologie de Compiègne, France
Directeurs de thèse :
Prof. Alain QUILLOT - Université Blaise Pascal, France
Prof. Annegret K. WAGLER - Université Blaise Pascal, France
Invité :
Prof. Sven O. KRUMKE - TU Kaiserslautern, Germany

Declaration

This dissertation has been completed by Jan-Thierry Wegener under the supervision of Professor Alain QUILLIOT and Professor Annegret K. WAGLER and has not been submitted for any other degree or professional qualification. I declare that the work presented in this dissertation is entirely my own except where indicated by full references.

Jan-Thierry Wegener

Acknowledgment

This work was founded by the French National Research Agency, the European Commission (Feder funds) and the Région Auvergne within the LabEx IMobS3. Without their generous financial support this thesis would never have been possible.

I wish to express my sincere thanks to my supervisor Prof. Annegret Wagler for the continuous support of my PhD study and related research. She always finds a way to make a complex thought understandable for everyone and I hope to have gained a bit of this ability as well. I could not image to have finished this thesis without her immense patience, motivation and guidance throughout the years of research and while writing this thesis.

My sincere thanks also goes to Prof. Alain Quilliot for his great advises during my research and for sharing his expertise.

I also deeply thank Prof. Sven Krumke for giving several thoughtful insights into the topic of Online Optimization and for the many fruitful discussion.

I would like to thank Prof. Louchka Popova who strongly encouraged me to write the thesis.

Furthermore, I would like to thank all of my colleges at the university for their discussions and the good time I spend at the laboratory. Hereby I want to give my special thanks to Sahar Bsaybes, Maxime Bombrun, Abdeslem Belghoul, Kaoutar Ghazi, Lakhdar Akroun, Benoit Bernay, and Romain Pogorelcnik.

I also like to thank my friends, especially, Markus Langer for the support and the great time we spend together.

I would also like to take this opportunity to express my gratitude to my family for their support during the years of study. Especially, I would like to give my wholehearted thanks my mother for her unlimited support and encouragement. My thanks also goes to my half-brother Manuel Möbius who always supported me whenever possible. My deepest gratitude goes to my beloved partner Celine Patin for her loving and support through this venture. Her encouragement when the times got rough are much appreciated.

Abstract

Carsharing is a modern way of car rental, attractive to customers who make only occasional use of a car on demand. In a carsharing system, a fleet of cars is distributed at specified stations in an urban area, customers can take a car at any time and station and return it at any time and station, provided that there is a car available at the start station and a free place at the destination station. To ensure the latter, customers have to book their demands in advance. For operating such a system in a satisfactory way, the stations have to keep a good ratio between the total number of places and the number of cars in each station, in order to serve as many requests as possible. This leads to the problem of balancing the load of the stations, called Relocation Problem: an operator has to monitor the load and to decide when and how to move cars from "overfull" stations to "underfull" ones.

We consider an innovative carsharing system, where the cars are partly autonomous, which allows to build wireless convoys of cars leaded by a special vehicle, such that the whole convoy is moved by only one driver. This setting is similar to bikesharing, where trucks can simultaneously move several bikes during the relocation process.

In this thesis, we address the dynamic and static aspects of the Relocation Problem. The "Dynamic Relocation Problem" describes the situation when cars can be moved between stations during the working hours in order to satisfy the needs of the customers. Hereby, the operator has to make decisions dynamically according to the current situation. In the "Static Relocation Problem" we assume that there is no (or only little) interaction by customers with the system. This situation occurs when the carsharing system is prepared for the next day, i.e., the relocation process is performed during the night.

We model the Relocation Problem in the framework of a metric task system. Afterwards, we theoretically analyze both problems and give strategies to solve them. Finally, we perform some computational experiments to examine the applicability of the presented algorithms in practice.

Keywords: carsharing; partly autonomous cars; Relocation Problem; online optimization; heuristic; network flows

Résumé

L'autopartage est une faọn moderne pour louer une voiture ; l'autopartage est attractive pour des clients qui utilisent une voiture seulement occasionnellement. Dans un système d'autopartage, une flotte de vehicules est distributée dans une area urbaine. Les client peuvent prendre une voiture á tout moment et tout stations et ils peuvent la retourner á tout moment et tout stations, á condition qu'il y a une voiture librée á la station de départ et aussi bien qu'il y a une place de parking librée á la station de destination. Pour assurer le dernier, les clients peuvent rèserver une voiture en avance. Pour opérer tel qu'un système de manière satisfaisante, il faut que le ratio de numéro de vehicules et de numéro de place librée dans les stations est equilibré. Cela conduit au problème de balancer de charge des stations, appelé problème de relocalisation : un opérateur doit surveiller le charge et decider quand et dans quelle manière les voiture doivent etre deplacer d'une station « trop plein » á une station « insuffisante plein ».

Nous considérons un système d'autopartage innovative, où les voitures sont partiellement autonomes. Ceci permit des construire des convois de vehicules, dirigé par une vehicule spécial tel que un convoi est déplacé par seulement un conducteur. Cette configuration est similaire á un système de vélos en libre-service, oú un camion peut déplacer plusieurs vélos simultanément pendant le processus de la relocalisation.

Dans le cadre de cette these, nous étendons les aspects dynamique et statique du problème de relocalisation. Le « problème de relocalisation dynamique » décrit la situation lorsque les voiture sont déplacé pendant les heures de travail afin de satisfaire les besoins des clients. L'operateur doit prendre des décisions dynaimques en fonction de la situation. Dans le cadre du « problème de relocalisation statique », nous supposons qu'il y a aucune (ou seulement un peu) d'interaction par des clients avec le system. Cette situation se produit lorsque le système est préparé pour le lendemain, par example lorsque le processus de la relocalisation est effectuée pendant la nuit.

Nous modélisons le problème de relocalisation dans le framework d'un système de tâches métriques. Ensuite, nous analysons les deux problèmes et nous donnons des stratégies pour les résoudre. Enfin, nous effectuons quelques expériences de calcul pour examiner l'applicabilité des algorithmes présentés en pratique.

Mots clés : autopartage ; voitures partiellement autonomes ; problème de relocalisation ; optimisation en ligne ; rśeau de flot

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Since the first humans came to existence about 200,000 years ago, the world population is growing. In the first half of the 19th century, the world population crossed the 1 billion line [155]. A little more than 100 years later, in the year 1927, 2 billion people lived on the planet [155], and nowadays (in the year 2014), more than 7 billion people inhabit the earth [157]. Within the next 40 years, the world population is expected to grow to more than 9 billion people [157] (see Figure 0.1).

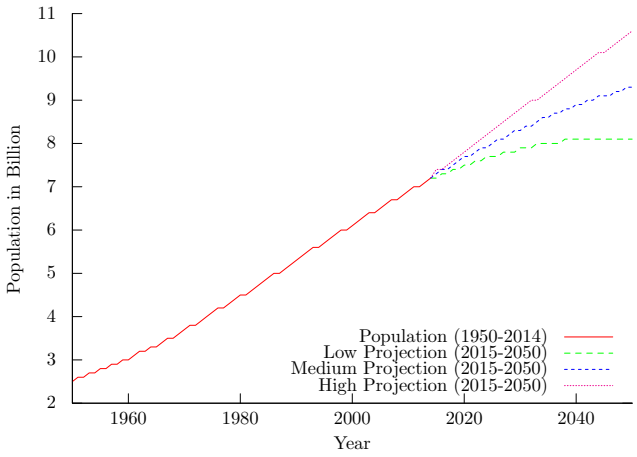


Figure 0.1: This figure shows the world population from the year 1950 to 2014, and a projection of the world population until the year 2050. Hereby, three projections are shown, a low, middle and a high projection. The data set used to plot this image is taken from [69].

In modern history, technological progress and industrialization played a major role in the urbanization process. The more industrialized a region is, the higher is the percentage of people living in urban areas rather than in rural areas. Already in the year 1920, the more industrialized regions, about 30 percent of their population lived in urban areas [158]. This trend continued and already in the year 1950, more than 50 percent of the population of the more developed regions lived in urban areas [156]. The less developed regions are supposed to cross the 50 percent line before the year 2020 [156]. Globally, the world's urban population has already exceeded the world's rural population and is expected to increase up to 70 percent by the year 2050 [156]. Figure 0.2 shows the percentage of people living in urban areas and its estimated percentage until the year 2050.

In the more developed areas, the number of people living in rural areas decreases since several years, while in the less developed regions, the rural population is still increasing. However, the rural population is expected to have its maximum between the years 2020–2025 and to decrease afterwards [156, 158].

The number of megacities, i.e., cities or urban agglomerations with more than 10 million inhabitants, is expected to increase to 25 by the end of next year [158], to 37 by the year 2025 [156], where 22 of these megacities are located on the Asian continent, 9 on the American continent (north and south), and 3 megacities in Africa and Europe as well. Obviously, in bigger cities, the distances the inhabitants have to travel in order to reach their workplaces, shopping facilities, local recreation areas or leisure activities, are much higher than in small cities. Therefore, an efficient transportation system and infrastructure is inevitable for (mega)cities. This becomes even more an issue since the city's economic prosperity is related to the travel times and, thus, to an efficient urban transportation system [164].

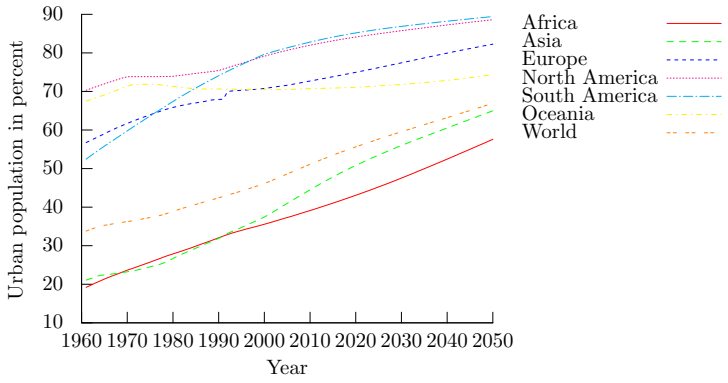


Figure 0.2: This figure shows the percentage of people living in urban areas grouped by continents and world wide and the estimated percentage of the population in urban areas until the year 2050. The data set used to plot this image is taken from [78].

One of the biggest impacts on the travel times are traffic congestions. Therefore, many cities try to reduce the number of cars within the city centers, e.g., Paris, London, Berlin, Stockholm [6]. For that, several strategies have already been introduced or are planned to be installed, e.g., public transportation systems, park and ride opportunities, fees for cars in the city center (congestion pricing), or bike- and carsharing systems.

Several European cities improved their bicycle lanes to attract more people to use their bike instead of their car. However, tourists and commuters usually do not travel to the cities with their bikes, and also not every local inhabitant has a bike. Bikesharing systems are services that provide bikes for shared use for a short time period. Customers can rent a bike at an arbitrary station and they can return it to another station, provided that there is a bike available at the start station and a free place at the destination station.

The first generation of bikesharing systems started in Amsterdam, Netherlands, with the so-called white bikes [129]. These bikes were painted white and distributed over the city for public use without any further security measurements and free of charge. This first generation did not have any stations where the bikes could be picked up or returned. As a result, the bikesharing system suffered highly from vandalism and theft.

In 1991, the second generation started in Farsø and Grenå, Denmark, introducing stations for the bikes [129]. Bikes could be rent by a kind of vending machine for a small fee. Renting a bike was still anonymous and theft could not be prevented.

Nowadays, bikesharing systems are in the 3rd generation. This generation started in Rennes, France, in the year 1998 [129, 140]. In modern bikesharing systems the customer is no longer anonymous due to the need of a customer card or a credit card. Due to the binding of the customer to the vehicle and due to improvements of the secure system at the stations, the possibilities of removing a bike without paying have decreased and, therefore, especially theft could successfully be prevented, in most of the bikesharing systems. However, despite these security measures, especially in Paris, theft and vandalism are still major problems for the operators [103, 127].

Despite the efforts of the governments to reduce the usage of cars, in most industrialized nations, cars are still the most important mode for transporting people [132].

Car rental agencies and carsharing systems are a way of renting a car for a short period of time. While traditional car rental agencies primarily serve customers who require a vehicle for some days or weeks, carsharing systems are attractive to customers who make only occasional

use of a car on demand. Usually, the term of lease is only for a short period.

Carsharing started in Europe, and the first carsharing system was a small cooperative called "Sefage" (Selbstfahrergemeinschaft) which started its services in Zurich in 1948 [141]. More than 20 years later, in the year 1973, a pendant to the white bikes was initiated in Amsterdam, the so-called "Witkars"[1] (white car). The "Witkars" were battery driven, and used the latest technologies of their time, in order to coordinate the trips of the users, giving access to the system, and to bill the users automatically [20]. The original system was running until 1986.

In 1987, two carsharing systems emerged in Switzerland "AutoTeilet-Genossenschaft" and "ShareCom", which then fused to "Mobility Carsharing" in 1997. Nowadays, "Mobility Carsharing" is still running and one of the largest carsharing systems in Switzerland.

In North America, carsharing began in 1983, as a research program from Mobility Enterprise at the Purdue University in West Lafayette, Indiana [141]. This project ended in 1986. Also in the year 1983, a second project started its service under the name Short-Term Auto Rental (STAR) in San Francisco. The project was planned for three years, but had to stop before (in 1985) mainly due to financial issues. Despite many carsharing system failed in the past, nowadays, there are several carsharing systems up and running in all major North American cities, e.g., Communauto (founded 1994), ZipCar (2000), City CarShare (2001), or Car2Go (2008).

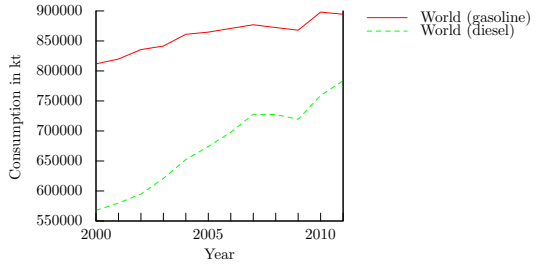
Since one can expect that the costs for buying and operating a vehicle (e.g., the increment of gas prices), increasing pollution, and the demand of parking places over the past years continues in the future (see Figure 0.3), more carsharing systems are likely to be installed in the future [139].

The advantages of installing a carsharing system in urban areas are manifold, for the users of the systems and for the communities. Frequent users of carsharing systems usually do not possess their own car. Therefore, installed carsharing systems in urban areas usually reduce the total number of cars in that area. According to [138], users of a carsharing system that do not possess a car, have a general reduction of the usage of cars by about 33 to 50 percent. Thus, a reduction in the possession of cars translates directly in reduction of the usage of cars in the areas where carsharing systems are installed. Hereby, the authors of [138] note that the impact is greater in Europe than it is in America. Carsharing contributes to sustainable transport and encourages their customers to use public transportation systems or other kinds of environmental friendly transportation. Nowadays, in about one thousand cities all over the world such services are already established [70].

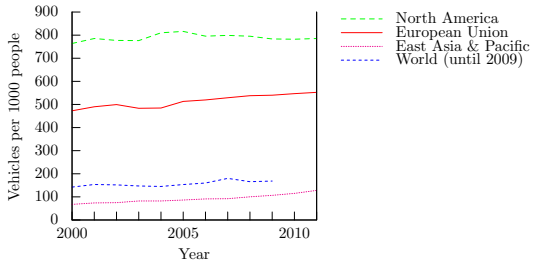
In a carsharing system, a fleet of cars is distributed at specified stations in an urban area and customers can take a car at any time and at any station and return it at any time and at any station, provided that there is a car available at the start station and a free parking place at the destination station. If a customer books his demand sufficiently in advance, he has the guarantee/increases the chance that his request will be satisfied. For operating such a system in a satisfactory way, the stations have to keep a good ratio between the total number of places and the number of cars in each station, in order to refuse as few customer requests as possible. This leads to the problem of balancing the load of the stations, called Relocation Problem: an operator has to monitor the load situations of the stations and to decide when and how to move cars from "overfull" stations to "underfull" ones.

Balancing problems of this type occur for any car- or bikesharing system, but the scale of the instances, possibility and time delay for prebookings and the possibility to move one or more vehicles in balancing steps differ. In traditional carsharing systems, the Relocation Problem is usually solved by transferring drivers in a bus to the overfull stations. Afterwards each driver transfers one car to an underfull station. Due to the higher costs for the operators, returning

[1] An old video about the "Witkars" can be viewed in English at `https://www.youtube.com/watch?v=EItrvudZy4w`.



(a) This image shows the world wide consumption for gasoline and diesel in kilotons.



(b) This image shows the number of cars per 1000 people in selected regions. The data for the global statistics were only available until the year 2009.

Figure 0.3: These charts shows the world wide consumption of gasoline and diesel and the number of registered cars per 1000 people in selected regions of the world. The data sets used to plot these images are taken from [152].

a car to another station than from where the car has originally been taken, may lead to higher rental costs for the client.

This thesis is funded by the LabEx IMobS^3 (Innovative Mobility: Smart and Sustainable Solutions) where more than 300 researchers and engineers, and more than 150 PhD students and post-doctoral fellows work in 7 research laboratories on three main challenges:

(1) Intelligent vehicles and machines: the focus of this challenges is on the conception of ergonomic, safe and intelligent vehicles and machines (autonomous and partly autonomous driving, advanced driver assistance systems, agricultural robotics, ...).

(2) Services and systems for smart mobility: This challenge studies innovative systems for mobility and how they can be integrated into their economical and social environment. The study of the innovative systems also includes the development of new management system supporting an optimized control of fleets of vehicles within these innovative systems for mobility.

(3) Energy production processes for mobility: this challenge focuses on the development of design and optimization of an innovative and efficient processes for the production of biofuel, the storage of biofuel, and the life cycle analysis linked to the production and use of the new forms of energy.

A fleet of autonomous cars, developed by robot experts from the Institute Pascal within Challenge (1), is currently tested on the industrial campus of Michelin in Clermont-Ferrand, France. A tool for managing these vehicles is also developed within Challenge (2).

This thesis contributes to Challenge (2) by considering an innovative carsharing system, where the cars are partly autonomous, allowing to build wireless convoys of cars lead by a special vehicle, such that the whole convoy is moved by only one driver (cf. [71]). This setting is similar to bikesharing systems, where trucks can simultaneously move several bikes during the relocation process [43, 50].

Although, fully autonomous cars may become reality in the near future [75], considering semi-autonomous cars does not make this thesis obsolete. Firstly, there are still legal issues [75, 75] and concerns of the end users and society [39, 135] which must be overcome before fully autonomous

cars populate the roads. Secondly, due to the mathematical similarities to bikesharing systems, this thesis also contributes to the problem of relocating bicycles.

Basically, there are two types of Relocation Problems: when the relocation process is performed during the night in order to prepare the system for the next day (static situation) and when the relocation process is performed during the working hours in order to react on the needs of the customers (dynamic situation).

When the carsharing system is prepared for the next day during the night, cars are transfered between the stations so that there are enough cars (resp. parking spaces) at the stations available at the morning for the customers. For that, usually statistical data are used to compute an appropriate initial state. In this static situation, a given time horizon, usually the time when the system is closed, has to be respected. The objective is usually to relocate the vehicles at minimal costs.

In the dynamic situation (also called online situation), customers come to the stations and take a vehicle and return it either at the same or any other station provided that there is a car and a parking place, respectively. In order to assure that a car and a parking space is available at the requested times, the customers can book a car in advance. The operator can decide to either accept these customer requests or to reject them. If a request is rejected, alternative times or stations may be suggested. However, if a customer request is accepted the operator must ensure that there is a car and a parking place available for that client. This might involve to relocate some cars before the arrival of the client, so that the request can be served. In this reactive scenario, algorithms usually cannot rely purely on statistical data. However, statistical data may help in designing algorithms that can handle situations with uncertainty, and also may help the algorithms for taking decisions.

The objective function depends on the goals of the operator of the carsharing system, e.g., to maximize the profit by minimizing the relocation of vehicles, although if that means that some requests are rejected, or the operator may focus on the quality of service by accepting and serving the maximal number of customer requests.

In this thesis, we consider several variations of the relocation problem, and provide some solutions strategies for each of the considered variations. Furthermore, we provide some theoretical results to support the provided solutions.

Chapter 1

State of the Art

During the last years, car- and bikesharing systems have gained popularity and an increasing number of cities all over the world establish(ed) new systems [70]. Thus, it is canonical that several aspects of bikesharing systems are discussed in the scientific community, e.g., establishing a new system and/or finding the best locations for new stations (see Section 1.1), the behavior of the customers within the system (see Section 1.2), or ensuring a running system and providing a quality of service by transferring vehicles between stations (see Section 1.3).

In general, the stations of a car- or bikesharing systems become imbalance over the time, making relocations necessary during the working hours. Since the customer behavior cannot always be predicted, algorithms must be capable of making good decisions under uncertainty. A mathematical framework for analyzing the performance of such algorithms is the online analysis (see Section 1.4).

1.1 Establishing New Systems

When a new carsharing or bikesharing system is established in an urban area, the planners have to think about the best locations of the stations. Besides the costs of establishing a station at a specific location, several other aspects have to be considered, such as the number of potential customers, the maximal possible number of free parking places at the station, the reachability of the location for the customers, the distance between two stations or the distance to other means of public transport [117, 161]. This problem is usually dealt by considering the so-called hub problem.

An approach to solve this problem is given in [117], where the authors focus on the question of designing a new bikesharing system. Hereby, the authors focus on the costs for constructing the system, as well as providing a high quality service for the customers. An exact optimization problem (as an integer linear program) and a greedy heuristic in order to solve the so-called hub problem is given in the paper.

In order to identify important key nodes, it is important to analyze a given system for the hierarchical structure of the stations. This is done in [162], where hubs and their connecting roads are categorized into different hierarchical levels: region, area, and local. Furthermore, the transit routes between these hierarchical levels are categorized as well. In order to solve this problem, the authors model the problem with a non-linear program, which they reformulate and linearize afterwards.

The attraction of certain stations for customers is modeled and computed in [161], where the authors propose a two-phase optimization approach in order to solve a transit hub location

problem. Unlike many other papers, the authors not only select hub locations from a given set of possible hub locations, but determine candidate nodes in an urban environment in the first phase. Hereby, a passenger attraction is computed for each node which is defined as a product of accessibility (an index which indicates the convenience level for customers to access the node) and of connectivity (an index for the convenience level for customers to arrive at main destinations, e.g., other transport hubs, their work place, places of interest). The authors note that candidate nodes must have both features: a good accessibility and a good connectivity. If a node does not have either a good accessibility or it does not have a good connectivity, it is usually not suitable as a candidate. The second phase of their algorithm deals with the optimization of the distribution of the transit hubs. Hereby, three objects are taken into account: the largest served population, a minimum amount of overlapped served population among the hubs and minimizing the costs for constructing these hubs. The applicability of their two-phase approach is demonstrated in a case study on the transit data of Dalian city, China.

1.2 Customer Behavior

Studying the behavior of the customers of a car- or bikesharing system does not only help the providers of a system to improve their service, but can also help in finding domain specific algorithms. Furthermore, the studies of the behaviors of the customers also help to create realistic test data. With the help of such realistic scenarios, it is easier to test and construct algorithms which can be applied in practice. To achieve this goal, the behavior of the users of the systems has been analyzed under different aspects, e.g., gender studies [124], the motivation of people to use a bikesharing system [9], diffusion theory to find new trends and the social impact of the system [129] and the benefits for cities, environment and the society [138, 141], or the dynamic analysis of the vehicles [29, 30, 73, 125].

Besides general studies about bikesharing system, the larger bikesharing systems, such as Paris [73, 125] or Lyon [29, 30], gained a lot of interest. However, not only specific systems of a city are analyzed, but also differences between the systems of different continents are analyzed, e.g., Europe and North America [129]. The authors provide us with useful data, such as the current and future number of bicycles in several selected systems. Due to rising fuel prices, public health concerns or climate-change considerations, the authors claim that in the future more systems will be installed, and bikesharing system will receive more attraction.

In [30] the authors study Lyon's bikesharing system Vélo'v. Hereby, the authors separately analyze week-days and the week-ends. The paper gives useful information on the number of rides, i.e., how many bicycles are rented per hour. Furthermore, they provide maps where the usage of stations and the number of trips between two stations are shown. The nodes of the graph are grouped into clusters. The clusters of a graph are computed with the Louvain algorithm (e.g., see [80]). Firstly, they are grouped based on the number of bicycles they exchange. Although, grouping nodes into clusters is not taking any geographical information into account, the computed clusters are also grouped on a geographical basis. This approach could be used for clustering and solving the problem on a cluster, see also [137]. In Lyon, comparing the clusters created from snapshots in the morning and in the afternoon of week-days, shows that most of the clusters remain unchanged. Due to a hill, two groups are visible only in the morning. The authors state that more than 90% of the nodes not within these two groups did not change their community. Secondly, the nodes are grouped based on usage patterns taking dynamic behavior into account. Considering the usage patterns, the nodes in a community are no longer grouped considering their geographical positions. However, the dynamic grouping seems to be more of an interest for social studies.

Another paper which also considers Lyon's bikesharing system Vélo'v is [29]. The authors apply techniques from signal processing. Hereby, they consider data collected over two years, namely from May 2005 to the end of 2007. According to the authors, in these two years, there were more than 13 million trips. The collected data not only contains the start and end stations and times of the bikes, but also the duration and lengths of the trips. In the paper, the authors give figures to the following topics: number of rented vehicles, rental duration, lengths of the trips, average speed of the cyclists and computed clusters. Furthermore, the authors give a formula that forecasts the number of rentals in a selected period. Finally, the authors state that their observations are made within other bikesharing systems.

In [138], the authors consider carsharing systems in general, as well as some specific services, in Europe, America and Asia. Besides a historical overview of carsharing systems in these continents, the authors consider the user characteristics (e.g., the average age of the customers, their education level, and gender), and the market potential, as well as the benefits of carsharing systems on the environment and the society. Furthermore, the economic benefits for the users of the services are highlighted.

1.3 Routing and Balancing Problems

Balancing the system over time or during the night, so that the appropriate number of cars and free parking places are available to the users, is one of the major costs. These or similar kinds of optimization problems appear in several other fields, e.g., pizza delivery services (traveling salesperson problem or vehicle routing problem), delivering goods to supermarkets (pickup and delivery problem), paratransit or taxi services (dial-a-ride problem). Thus, it is not astonishing that the problem of balancing car and bikesharing system has gained a lot of interest.

Routing and balancing problems can be modeled within the framework of metric task systems. A metric task system is a tuple $(M, \mathcal{T}, \boldsymbol{z})$, where $M = (V, d)$ is a metric space[1], $\mathcal{T}$ is a set of tasks τ and $\boldsymbol{z}$ is the initial system state. The aim is to serve all tasks, starting from the given initial state.

Balancing algorithms usually include finding shortest paths in the network. This can be done by finding a shortest path in graphs [66], a quickest path in transportation networks [143], or sometimes even by solving the traveling salesperson problem [128].

The shortest path problem in graphs is to find a shortest path in a given weighted graph between two given nodes. Hereby, the graph can be directed or undirected. The weights on the edges usually reflect the traveling times. The most famous algorithm for solving this kind of problem is Dijkstra's algorithm [66] from the year 1956. Since then, Dijkstra's algorithm has been improved and/or generalized, e.g., by using hierarchies [15], the A^*-algorithm [59, 107], or the Bellman-Ford-Moore's algorithm [17, 79, 123].

A well studied basic problem is the traveling salesperson problem (also called traveling salesman problem) [45, 56, 58, 84], where a salesperson has to visit a set of cities, visiting each city exactly once, and then return to its starting city. The traveling salesperson problem has as input a metric space M (usually induced by a complete weighted graph G) and the initial system state $\boldsymbol{z}$ indicating the initial position of the driver. This induces the set of tasks $\mathcal{T}$ containing for each point $v \in M$ a task to visit v and, thus, giving the metric task system $(M, \mathcal{T}, \boldsymbol{z})$. The objective is to find the shortest tour serving all tasks (i.e., visiting all cities).

[1] In this thesis we mainly focus on three metric spaces: intervals of the real line, the uniform metric space and metric spaces induced by weighted graphs where d is induced by the shortest paths distances. The metric space $M = (V, d)$ induced by an interval $[a, b]$ of the real line is denoted by $[a, b]$, and the distance function is $d(v, v') = |v' - v|$. The uniform metric space is a metric space where the distance between two points is always equal to 1. A uniform metric space with n points is denoted by $\mathbb{U}(n)$.

The traveling salesperson problem is a well known $\mathcal{NP}$-hard problem [128]. Thus, several heuristics [91], meta-heuristics [5, 102] and approximation algorithms [7, 37, 87, 90] have been applied in order to solve the problem within a reasonable time.

The nearest neighbor algorithm is one example for a heuristic to solve the traveling salesperson problem which starts in a random city. Afterwards, it selects the nearest not yet visited city as the next city to visit. This process is repeated until all cities are visited.

The nearest neighbor algorithm can easily be implemented and quickly computes a tour, but usually not the optimal solution. Indeed, under certain conditions, the algorithm nearest neighbor may return the worst possible tour [10], i.e., a tour with a maximal tour length.

Probably the most famous approximation algorithm for solving the traveling salesperson problem is the Christofides algorithm [25, 45]. It takes as input a complete weighted graph $G = (V, E, w)$ containing the nodes (cities) to be visited. The output of the Christofides algorithm is a tour visiting all nodes in V. Hereby, the computed tour is within 3/2 of the optimum [45].

The algorithm is performed in three steps:

(i) construct a minimum spanning tree T in G,

(ii) compute a perfect matching M over the complete subgraph of G containing all nodes from T with odd degree,

(iii) combine the edges from T and M, and search an Eulerian circuit in this graph.

The tour can be improved by "shortcutting" which removes repeated nodes from the tours.

This algorithm emerged some other similar approximation algorithms (see, e.g., [37, 60]). These variants usually do not compute a minimum spanning tree but other types of trees. Although, the worst-case analysis of these modifications is larger than 3/2, they tend to show better results in practice.

On a computed tour, the tour is most often improved by the heuristic k-opt [95]. Hereby, k edges from a tour are removed and the resulting fragments are reassembled so that the resulting tour is (hopefully) shorter. This procedure can be repeated arbitrarily often. In practice, 2- and 3-opt are commonly used.

The k-opt algorithm gives also the basis for local search algorithms like simulated annealing [106, 119] or tabu search [14, 121, 122]. Both meta-heuristics have in common that they may accept solutions with a larger tour length than the current solution. The idea behind this is to prevent that the algorithms keeps searching for better tours only within the neighborhood of a local optimum which can be far away from the global optimum.

Besides a tour visiting all cities, a temperature is given as input for the algorithm simulated annealing. Starting from the given tour, simulated annealing searches in a local neighbor of this tour for another tour. Hereby, the newly constructed tour Γ (e.g., by k-opt) is accepted if either Γ is shorter than the previous tour or with a certain probability depending on the current temperature. At the end of each iteration, the temperature decreases, and the algorithm stops when the temperature is below a certain threshold.

Tabu search is a meta-heuristic similar to simulated annealing. However, instead of a temperature, the algorithm has a tabu list of "tabu" optimization steps (e.g., if k-opt is used then the edges removed from the tour can be stored in this list). The number of elements in the tabu list is limited by a given natural number κ; if more than κ elements are added to the list then the oldest elements are removed. A new tour Γ is accepted only if Γ can be reached from the current tour without using any of the "tabu" optimization steps. This (hopefully) prevents the algorithm to visit solutions which have already been visited.

A well-known problem that generalizes the traveling salesperson problem is the vehicle routing problem, where customers have to be served with a fleet of vehicles. Thus, this problem can be

understood as a travelings salesperson problem with several salespersons. The input of the vehicle routing problem (G, v_D, k) is a weighted graph G, a depot v_D and the number of drivers k. This induces the metric task system $(M, \mathcal{T}, \boldsymbol{z})$ where M is induced by G, $\mathcal{T}$ contains the task for visiting each node in G, and the initial state $\boldsymbol{z}$ indicating the depot. The aim is to find k tours (a transportation schedule with k drivers) serving all tasks in $\mathcal{T}$ with a minimal total tour length.

The problem has originally been proposed by Dantzig and Ramser in [57], where the authors modeled a fleet of gasoline delivery trucks that serve numerous gas stations. Since then, several variations of the vehicle routing problem have been studied, e.g., vehicle routing problems with time windows [115], capacitated vehicle routing problems [163], using a heterogeneous or homogeneous fleet of vehicles. Exact approaches for the vehicle routing problem and their variations are usually based on integer linear programmings [46, 49, 65, 77, 82]. Typical applied meta-heuristics range from tabu search [83, 85, 115, 149], simulated annealing [149] and genetic algorithms [83, 150].

Genetic algorithms initialize a set of K transportation schedules (also called the population). Afterwards, they select the best $K' < K$ transportation schedules w.r.t. the total tour length. With these K' the set is repopulated, e.g., by selecting two parents (transportation schedules) and recombine these two schedules to produce an offspring (a new transportation schedule). Finally, the offspring is randomly modified (mutation) with a low probability. This procedure is repeated a given number of times.

The vehicle routing problem is generalized by the dial-a-ride problem. In the dial-a-ride problem a fleet of servers transfer persons between given origin and destination locations. Hereby, different objective functions are considered, e.g., to minimize the costs of transferring the persons, minimizing the waiting or travel time for the persons. A dial-a-ride problem typically occurs for every taxi company, the transportation of elderly or handicapped people. A similar problem is the pickup-and-delivery problem, where goods instead of persons are transferred. This usually leads to some other focuses than in a dial-a-ride problem, since waiting or travel times are usually not as important for goods as they are for people. In the dial-a-ride or pickup-and-delivery problem, the induced tasks indicate the pickup and drop stations of the clients (or goods).

Variations of the dial-a-ride problem [52, 63] include the dial-a-ride problem with time windows [62, 74, 97, 160], capacitated dial-a-ride problems [53], using a heterogeneous [153] or homogeneous fleets [99] of transport vehicles.

Besides tabu search [38, 53], and genetic algorithms [22, 41], also insertion techniques [61, 63] are applied to the dial-a-ride problem. An insertion algorithm initializes all tours for the drivers by the empty tours, and then, in each iteration, it tries to find a good position within a tour to insert and serve the requested transfers. Hereby, capacity and time-window constrains are respected.

In our carsharing system, vehicles have to be transferred between stations, in order to ensure that there is a vehicle available for the customers at their requested pickup station and a parking space at their requested drop station. This induces tasks which indicate the number of cars to be picked up from (resp. dropped at) the stations. Hereby, the tasks to transfer vehicles between the stations are not directly given by customers but are indirectly implied by the customer behavior and their actions as well as the relocation process. Hereby, we consider several objective functions, minimizing the transfer costs, maximizing the gained profit, minimizing the number of rejected customers and maximizing the number of accepted customer requests.

In bikesharing systems, the relocation process is usually done over night time when the system is closed or only few customers change the system [43]. Hereby, the system is prepared for the next day to satisfy the users demands. The online or dynamic counter part, is when the relocation process is performed during the working hours of the system [50].

The relocation problem is often encoded by integer linear programs [50]. Other approaches

use Petri nets to model the system [113], where also a new kind of arcs is introduced. However, since the problem is $\mathcal{NP}$-*hard* [19], it is not surprising that heuristics and meta-heuristics have been applied to the relocation problem [111, 112, 126, 131].

Due to the large number of stations in a bikesharing systems, "clustering" stations (see, e.g., [80, 88, 114]) and solving the relocation problem on these smaller "clusters" can lead to good results, while keeping the computation time low. In the clustering problem nodes are grouped in such a way that nodes within a group are more similar to each other than nodes from another group. In the context of a car- or bikesharing system the stations can be grouped, e.g., by their geographical positions [137], by their usage patterns [81], or by the distances to certain places of interest such as subway stations, parks, or other leisure, business or residential areas [73].

A cluster-first route-second approach is presented in [137]. The authors describe an algorithm for clustering the bikesharing system and solving the rebalancing problem on these clusters. The problem of finding clusters is solved by searching maximum spanning stars in the network. In each of the clusters, there is one driver responsible for the relocation process.

Another idea is to encourage the customers to take or return vehicles at certain stations to reduce the imbalances without the need of drivers [44]. This is done by a pricing strategy where customer pay less or gain free minutes for the usage of the system when a vehicle is taken from or returned to a certain station.

The contrary idea, to encourage customers to cause even more imbalances at certain stations, may be unintuitive at the first glance. However, the idea is to produce only a few highly imbalanced stations which can then be handled with just a few short tours [92]. Since there are less stations where the drivers have to stop, in practice, the total time for the drivers may be reduced as well. Furthermore, due to the reduction of imbalanced stations, the computation times for calculating the tours can be decreased.

1.4 Online Optimization

In real world applications, the input data for an algorithm is not usually not complete in the sense that interactions in the future with the system are unknown. This means that the input data is revealed to an algorithm over time. Online optimization provides a theoretical framework for studying interactions with a system, and helps analyzing solution strategies which make decisions before the complete data is available. A solution strategy or algorithm dealing with data arriving over time is called online strategy/algorithm.

The arrival of the input data over time is most commonly modeled by the "sequence model" and the "time-stamp model", which differ in the way how the information becomes available to the online algorithm. In both cases, the online algorithm ALG is confronted with a finite sequence of requests $R = r_1, r_2, \dots r_\lambda$, and ALG has to serve these requests according to certain rules. Serving requests usually induce costs (or profits) and the overall goal is to minimize the costs (resp. maximize the profits).

In the sequence model the requests must be served in the order of their occurrence. This means that while ALG serves a request r_i, it does not have any knowledge about any of the successive requests r_j with $j > i$. As soon as the request r_i is served, the next request r_{i+1} is released, i.e., becomes visible to the online algorithm. Note that decisions cannot be revoked in the sequence model.

In the time-stamp model the requests are not released one after the other, but over time at their release date: a request r is released at time $t^{rel}(r) \geq 0$. This implies that ALG cannot serve r before time $t^{rel}(r)$. The online algorithm ALG has to determine its behavior at a certain moment t in time based on the requests released up to time t. Hereby, ALG is allowed to wait and

to revoke decisions (as long as they have not yet been executed or communicated to customers as (fixed) appointments).

Due to the lack of information, it is generally impossible to give an exact solution. Furthermore, requests must usually be processed immediately or within a short time horizon. Thus, online algorithms are typically heuristics.

While the general algorithmic scheme for the sequence model is relatively easy by serving each newly released request with a certain RULE_{serve} (see Algorithm 1), online algorithms for the time-stamp model are usually more complex (see Algorithm 2). Hereby, (possibly) several requests are selected due to a certain RULE_{sel} to be served according to a certain RULE_{serve}.

Algorithm 1 Algorithmic Scheme for the Sequence Model

Input: a sequence of requests R
Output: costs for serving all requests in R
1: **for** request $r \in R$ **do**
2: | serve r according to a RULE_{serve}
3: | update costs
4: **return** total costs

Algorithm 2 Algorithmic Scheme for the Time-Stamp Model

Input: a sequence of requests R
Output: costs for serving all requests in R
1: Initialize list σ of released but unserved requests
2: **while** $\sigma \neq \emptyset$ **do**
3: | select one or several request(s) according to RULE_{sel}
4: | serve selected request(s) according to a RULE_{serve}
5: | update costs
6: **return** total costs

Besides deterministic online algorithms, there are also randomized online algorithm ALG, which may produce different solutions when applied several times on the same input sequence. Hereby, RULE_{serve} as well as RULE_{sel} can be randomized.

1.4.1 Online Problems

Online optimization can be at least traced back to a paper from Sleator and Tarjan [144] from the year 1984, in which the authors analyzed the "list update problem". In the list update problem the requests correspond to accessing items in a given list, and accessing an item induces costs proportional to the distance of the head of the list. An online algorithm has to reorder the list so that the total costs are minimized. Hereby, the online algorithm can reorder the list at any time but incurs in additional costs. In [144] the authors proposed the online algorithm "move-to-front" where an accessed item is always moved to the head without changing the order of the other items of the list.

Since then, their analysis techniques have been applied to several online problems, including several transportation problems. A widely analyzed online problem is the so-called "k-server problem" (see, e.g., [11–13, 47, 48, 51, 55, 64, 108, 120]). An instance of the k-server problem consists a metric space (X, d) and a sequence R of request, where a request $\sigma^j = (\boldsymbol{x}, t)$ consists of a point $\boldsymbol{x} \in X$ to visit and its release time t. The requests are released by the rules of the

sequence model and an online algorithm has to compute k tours, such that all requests are served. The goal is to minimize the total path length.

The "online traveling salesperson problem" (see, e.g., [26, 98]) is similar to the k-server problem where instead of the sequence model the time-stamp model is used. Hereby, an online algorithm has to compute a tour for one server visiting all requested points of a metric space (X, d). The server starts and ends in the depot, which is represented by a special point in X. In the online traveling sales person problem, the goal is to minimize the total traveling time, i.e., the time when the server finished serving all requests and returned to the depot.

Obviously, the online version of the traveling salesperson problem differs from the classical offline problem where every city (point in a metric space) has to be visited exactly once. In the online version, some cities may not be visited at all or a city may be requested to be visited several times.

The "online k dial-a-ride problem" (see, e.g., [8, 21, 28, 54, 76, 159]) is a natural generalization of the classical k dial-a-ride problem. In the online k-dial-a-ride problem, k server move in a metric space where they process over time released transportation requests, from an origin to a destination. Obviously, the online k-dial-a-ride problem generalizes the k-server problem, where the origin and the destination of a transportation request are equal.

1.4.2 Online Strategies

There is a large variety of online strategies and online algorithms to solve the different problems. In this section, we present some common online strategies but also some uncommon ones. Most of these strategies can easily modified and applied on very different kind of problems.

The first algorithm we consider is the so-called first-come first-serve algorithm FcFs (see Algorithm 3). The basic idea of FcFs is to serve that requests first which waits for the longest time. In the context of a carsharing system this means that the customers are served in the order in that they arrive at the stations. Since no one is favored nor unfavored by FcFs, this algorithm can be considered to be "fair" to the requests and, thus, fair to the customers.

Algorithm 3 FcFs strategy

Input: a sequence of released requests R
Output: tours for the servers with minimal costs
1: **while** not all requests have been served **do**
2: | select request r that "waits" longest
3: | compute a tour for a server to serve r
4: **return** computed tours

A similar algorithm is the online algorithm "earliest start time" EST (see Algorithm 4). Hereby, the requests are served in the order when the customers want their request to be served (e.g., the time when a customer picks up a car) and not in the order when they announce/release the request.

Example 1.1. In order to illustrate the difference between FcFs and EST we consider the following online problem: customers of a carsharing system are allowed to book cars to pick them up at a station v and return the car to another station v'. Hereby, the customers are allowed to specify the pickup time t_v as well as the drop time $t_{v'}$, inducing a time-window in which a car has to be brought to v, resp. in which a parking space has to be available at v'. A request is given by a tuple $(t, v, t_v, v', t_{v'})$, where t is the time when the customer books the cars. Cars can be transferred by drivers (playing the role as the servers) between the stations. When

the customer cannot pickup a car or cannot return it, then the request must be rejected. The objective is to minimize the number of rejected requests.

Let the carsharing system contain station a station on each integer point of the real line $\mathbb{R}$. Furthermore, we assume that there is one driver and one car in the system, both located in station 0 (the origin) at time 0.

Let us the following two requests be released: $r_1 = (1, 4, 8, 5, 11)$ and $r_2 = (3, 1, 6, 3, 15)$. While FcFs first serves r_1, the algorithm EST starts by serving r_2 since the pickup up time of the customer is 6, while for r_1 it is 8 (and, thus, r_2 is the most urgent released request). ◊

Algorithm 4 EST strategy

Input: a sequence of released requests R
Output: tours for the servers with minimal costs
1: **while** not all requests have been served **do**
2: | select request r which is most urgent
3: | compute a tour for a servers to serve r
4: **return** computed tours

The algorithm RePlan (see Algorithm 5) is less "fair" to the customers. Every time a new request is released, RePlan recomputes tours based on the current situation. Afterwards, RePlan recomputes the tours. Note that the computation of the new tours highly depends on the objective function of the considered online problem. Servers who are not serving any request either wait at their current position or return to the depot (depends on the considered online problem).

Algorithm 5 RePlan strategy

Input: a sequence of released requests R
Output: tours for the servers with minimal costs
1: **when** new request r is released **then**
2: | compute new tours based on $R \cup \{r\}$
3: | servers not needed to serve any request wait at their current position (or return to a depot)
4: **return** computed tours

The Ignore strategy basically ignores all newly released requests and computes a new tour not before a current tour is fully performed. The idea behind this algorithm is to avoid a problem that sometimes occurs in the online algorithm RePlan where drivers may repeatedly change their direction in order to serve newly released requests.

The next online algorithm does not work for every online problem. However, in can be applied in some special cases of the relocation problems. The online algorithm "accept if possible" accepts a request if no server needs to be moved, otherwise the request is rejected. Also one may be tempted to say that the online algorithm AIP is not of any practical use. Especially in bikesharing systems, where users do not book their bike in advance (i.e., when the release time is equal to the pickup time), the online algorithm AIP is one of the most used online algorithm in practice. Hereby, customers arriving at a station take a vehicle if one is available, or search another station if there is none.

The first randomized online algorithm we present in this chapter is a pathological example, the algorithm RandomWalk (see Algorithm 8). The drivers move randomly in the system.

Algorithm 6 IGNORE strategy

Input: a sequence of released requests R
Output: tours for the servers with minimal costs
1: **when** new request r is released **then**
2: | add r to a sequence σ of requests ▷ ignore the new request
3: **when** a tour is fully performed **then**
4: | compute a new tour based on σ
5: | set $\sigma \leftarrow \emptyset$
6: **return** computed tours

Algorithm 7 AIP algorithm (accept if possible)

Input: a sequence of released requests R
Output: (empty) tours for the servers with minimal costs
1: **when** new request r is released **then**
2: | accept r if no server needs to be moved
3: | otherwise reject r
4: **return** computed (empty) tours

Whenever a new request is released, new tours are computed, taking the current positions of the drivers and the current system state into account. Thus, one can see that the basic optimization step of this randomized online algorithm is similar to REPLAN. However, in contrast to REPLAN, when RANDOMWALK is applied, the drivers generally neither wait at their current position nor return to a depot.

Obviously, due to the immense costs induced by randomly moving through the system, this algorithm is rarely applied in practice. However, from a theoretical point of view, the online algorithm RANDOMWALK sometimes outperforms any deterministic online algorithm, which is the reason why it is stated here.

Algorithm 8 RANDOMWALK strategy

Input: a sequence of released requests R
Output: tours for the servers with minimal costs
1: if R is empty, compute random tours for the servers
2: **when** new request r is released **then**
3: | compute new tours based on $R \cup \{r\}$
4: | compute random tours for the idle servers
5: **return** computed tours

The final online algorithm we mention in this section is a randomized version of the deterministic online algorithm REPLAN, the randomized online algorithm RANDOMREPLAN. Hereby, the basic behavior is as REPLAN, but whenever the algorithm has to decide between either serving (first) the request r_i or r_j, then it randomly selects one of them (REPLAN chooses the one with the lower index). Depending on the algorithm to compute a solution for an optimal transportation schedule, such conflicts may be randomly solved by the used solver. Thus, in practice, the randomized online algorithm RANDOMREPLAN may be used instead of the deterministic online algorithm REPLAN.

Finally, note that one can make online algorithms from the strategies of this section by giving

an algorithm for the lines of the form "compute new tours". When this line is called, there are already some customer requests released.

1.4.3 Online Analysis

We next discuss the evaluation of the applied solution strategies. In this thesis, we apply two different methods of online analysis, the "competitive analysis" and the "max/max ratio". Both methods are explained in detail in this section.

The most common method is the so-called competitive analysis, which has been introduced by Sleator and Tarjan in [144]. Hereby, an adversary releases a sequence R of requests which have to be served by the online algorithm. Therefore, the competitive analysis can be viewed as a request answer game: the adversary ADV generates "requests" and the online algorithm ALG has to "answer" them. The costs $\mathsf{ALG}(R)$ induced by the online algorithm for serving the requests in R is then compared against the costs $\mathsf{ADV}(R)$ induced by the adversary (which knows the entire input sequence in advance) for serving R. Then, a deterministic online algorithm ALG is said to be c-competitive against ADV (for a minimization problem), if for every sequence R of requests

$$\mathsf{ALG}(R) \leq c \cdot \mathsf{ADV}(R)$$

holds. The infimum over all c so that ALG is c-competitive is the competitive ratio. Note that measure the worst-case performance with the competitive analysis.

The competitive ratio for a randomized online algorithm ALG is defined analogously, (cf. [109]) where we exhibit a probability distribution over the input sequences such that

$$\mathbb{E}[\mathsf{ALG}(R)] \leq c \cdot \mathbb{E}[\mathsf{ADV}(R)]$$

holds for any deterministic algorithm, where the expectation is taken with respect to the constructed probability distribution, and where $\mathbb{E}(\mathsf{ALG}(R))$ is the expected value of a randomized online algorithm ALG over the input sequence R, and $\mathbb{E}(\mathsf{ADV}(R))$ is the expected value of ADV over R.

In order to prove the non-competitiveness of randomized online algorithms it is useful to apply Yao's Principle (see Theorem 1.2, cf. [31, 32, 109]) which gives a lower bound for randomized online algorithms. From Yao's Principle one can conclude that there does not exist a randomized competitive online algorithm if for some input distribution there is no deterministic competitive online algorithms.

Theorem 1.2 (Yao's Principle)**.** *Let Y be a probability distribution over the set $\{\mathsf{ALG}_y : y \in Y\}$ of deterministic online algorithms for an online minimization problem. If X is a probability distribution of the set $\{R_x : x \in X\}$ of input sequences and $c \geq 1$ is a real number such that*

$$\inf_{y \in Y} \mathbb{E}_X[\mathsf{ALG}_y(R_x)] \geq c\mathbb{E}_X[\mathsf{OPT}(R_x)]$$

then c is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.

Randomized online algorithms may achieve poor results in practice due to their randomness. Therefore, a simulation of the algorithm's behavior on some realistic test instances is typically used to decide which algorithm performs better. Under certain circumstances, randomized online algorithms theoretically outperform deterministic online algorithms, i.e., against certain adversaries, randomizing an algorithm may result in a better competitive ratio. However, some

adversaries (e.g., the adaptive offline adversary[2]) also know the outcome of the random decision of the random decision process and, thus, can react to it.

In the following, we define the adversaries used within this thesis.

An oblivious adversary knows the complete behavior of an online algorithm ALG and chooses a worst-case sequence for ALG as well as the profits for serving the requests. He is allowed to move drivers towards yet unreleased customer requests, but must not serve any customer request before it is released, i.e., before its release time. The oblivious adversary does not know the outcome of a random results of an online algorithm.

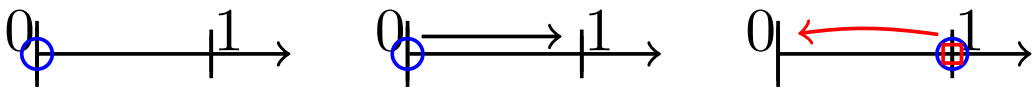
In this thesis, we "weaken" the adversary in two different ways: by limiting the set of requests the adversary can choose from and by limiting the set of algorithms with which the adversary can solve the offline problem.

The non-abusive adversary (see, e.g., [110]) is limited by the algorithms he can choose from. He knows the complete behavior of ALG and can choose a worst case sequence but he is only allowed to move the servers into a direction of a not yet served request.

In the next example we illustrate the difference between the two adversaries.

Example 1.3. Let us consider the carsharing system from Example 1.1 with the request $r = (1, 1, 1, 0, 3)$. In order to serve r, the car has to be moved from 0 to 1 before time 1.

Since the oblivious adversary ADV_{obl} knows the customer request r before it is released, it can start transferring the car to 1 at time 0. Therefore, ADV_{obl} serves the request (see Figure 1.1).



(a) A car and a driver are both positioned in the origin.

(b) A request will be released in one time unit. Therefore, the oblivious adversary transfers the car to station 1. Since the customer request is not released yet, the non-abusive adversary must not move the driver.

(c) The oblivious adversary transfered a car to station 1 and thus, can serve the request r. Since the customer request is now released, the non-abusive adversary is now allowed to transfer cars to position 1. However, he cannot serve r and must reject it.

Figure 1.1: This figure illustrates the behavior of the oblivious adversary in contrast to a non-abusive adversary. At time 0 a driver and a car are positioned in 0 (Figure 1.1a). A customer request $r = (1, 1, 1, 0, 3)$ is released at time 1 which can only be served when the driver starts transferring the car at time 0 to position 1. Since the oblivious adversary knows the complete sequence of customer requests, he can transfer the car to 1 at time 0 (Figure 1.1b), serving the request (Figure 1.1c).

The non-abusive adversary also knows r before it is released. However, he is not allowed to move the driver towards 1 until r is released. Thus, ADV_{non} must reject the request. ◊

[2]The adaptive offline adversary ADV knows (like the oblivious adversary) the behavior of an online algorithm ALG and ADV also knows the outcomes of any randomized number generated. He is allowed to move drivers towards yet unreleased customer requests, but must not serve any customer request before it is released, i.e., before its release time. Since this adversary knows the outcome of any randomly generated number, randomized online algorithms obviously do not help against an adaptive offline adversary.

Since the non-abusive adversary can only move towards released but not yet served request, he might not be able to serve all requests that can be served by an oblivious adversary (or he may serve them much later). If on the other hand, it is better to keep a server in a station, the oblivious adversary is also free to do so. Thus, it is easy to see that the non-abusive adversary is "weaker" than the oblivious adversary, and we can formulate the following theorem.

Theorem 1.4. *If an online algorithm ALG is competitive against the oblivious adversary, then ALG is also competitive against the non-abusive adversary.*

Besides the oblivious and non-abusive adversary, we consider the "δ-adversary", which has a limited set of requests to choose from. A δ-ϵ-adversary is an adversary which chooses only sequences, where the release time of each request is at least δ and at most ϵ time units before it must be served. The requests released by this adversary are called δ-ϵ-requests. If ϵ is allowed to be infinity, then we simply write δ-adversary (resp. δ-request) instead of δ-∞-adversary (resp. δ-∞-request). This adversary corresponds to customers booking their requests in advance, and gives the online adversary a kind of lookahead.

Example 1.5. Let us consider the online problem from Example 1.1. Then, a 5-10-adversary is allowed to release the request $(1, v, 8, v', 27)$ but he can neither release the request $(1, v, 1, v', 3)$ nor $(1, v, 20, v', 27)$. ◊

While the δ-ϵ-adversary $\mathsf{ADV}_{\delta\epsilon}$ is obviously weaker than the oblivious adversary ADV_{obl}, i.e., if an online algorithm is competitive against ADV_{obl} then it is also competitive against $\mathsf{ADV}_{\delta\epsilon}$, the relation between the δ-ϵ-adversary and the non-abusive are unknown.

Finally, we consider the so-called "direct adversary" is an oblivious adversary, which has also only a limited set of requests to choose from. This adversary can be considered in situations where the requests occupy a resource for a certain amount of time, e.g., when customers rent a car for several time units. The direct adversary can only choose from sequences where the resource is not occupied longer than needed. A request released by the direct adversary is called a direct request.

Example 1.6. In the context of the carsharing system from Example 1.1, a direct adversary corresponds to customers who are directly heading towards the requested drop station without making a detour. For example, a direct adversary can release the request $(1, 1, 1, 0, 2)$ but he is not allowed to release the request $(1, 1, 1, 0, 3)$. ◊

The relation between the non-abusive, the direct and the δ-ϵ-adversary, i.e., which adversary is the weakest, is currently unknown to the best of our knowledge. However, it is obvious that the direct adversary is weaker than the oblivious adversary. In this thesis, we mainly consider a combination of the three weaker adversaries.

A downside of the competitive analysis is that some (intuitive) improvements of some algorithms do not necessarily lead to a better competitive ratio (e.g., [1, 2]). A famous example is that online algorithms with a finite lookahead[3] usually do not have a better competitive ratio than their counterparts with no lookahead at all. In [18], Ben-David and Borodin developed the so-called "max/max-ratio" to overcome this counter intuitive problem. Hereby, the authors could give a finer comparison of some online algorithms which have the same competitive ratio.

The basic idea of the max/max ratio comes from the classical definition of amortized complexity. Hereby, the max/max ratio compares the worst case performance of the optimal offline solution and the worst case performance of the online algorithm.

[3] An online algorithm with lookahead does not only know the currently released requests, but has also some knowledge about some future requests.

While the competitive analysis compares the performances of both algorithms on the same sequences, the max/max ratio evaluates the performance (in general) on different sequences both having the same length. This avoids the problem that the competitive analysis already leads to bad competitive ratios when there exists only a single bad sequence for the online algorithm.

Let ALG be an online algorithm with costs $\mathsf{ALG}(\sigma)$ on the input sequence σ. Let

$$M_\lambda(\mathsf{ALG}) := \max_{|\sigma|=\lambda} \frac{\mathsf{ALG}(\sigma)}{\lambda},$$

then the amortized costs $M(\mathsf{ALG})$ are defined as

$$M(\mathsf{ALG}) := \limsup_{\lambda\to\infty} M_\lambda(\mathsf{ALG}).$$

The max/max ratio $w_M(\mathsf{ALG})$ of ALG is then defined as

$$w_M(\mathsf{ALG}) := \limsup_{\lambda\to\infty} \frac{M_\lambda(\mathsf{ALG})}{M_\lambda(\mathsf{OPT})} = \frac{M(\mathsf{ALG})}{M(\mathsf{OPT})},$$

where OPT is the optimal (offline) solution for the input sequence σ. The last equation of the definition of $w_M(\mathsf{ALG})$ is proved in [18, Lemma 4.2].

Besides the competitive analysis and the max/max ratio, there are several other proposed measures for online algorithms [34], e.g., the random-order ratio [104], the bijective analysis [3, 4], the smoothed analysis [16, 27, 134, 146, 147], the probabilistic analysis [96] and the relative worst-order analysis [33, 72]. A good overview of several alternative measures can be found in [67, 68].

Chapter 2

Metric Task System: Modeling Framework for the Relocation Problem

The studied carsharing system can be understood as a discrete event-based system (see, e.g., [112]), where

- the system components are the stations $v_1, \ldots, v_n$, each having an individual capacity $\text{cap}(v_i)$, which correspond to the number of available parking spaces;
- a system state $\boldsymbol{z}^t \in \mathbb{Z}^n$ specifies for each station v the number of parked cars z_v^t at a time point $t \leq T$ within a time horizon $[0, T]$;
- an attribute $\text{att}(v, t)$ of station v at time t reflects the ratio between its capacity $\text{cap}(v_i)$ and the current number of parked cars z_v^t, e.g., "overfull", "balanced" or "underfull";
- states and attributes can be changed by events (customers or convoy drivers take or return cars at a station), or forecasts (customers book at time t requests $r = (t, v, t^v, w, t^w)$ to take a car from the pickup station v at the requested pickup time t^v and return it to the drop station w at the requested drop time t^w); or the operator generates transportation tasks to move cars between stations.

Usually, operators of a carsharing system have statistical information about the behavior of the users of the system. It is possible to use this information by creating and releasing artificial customer requests, which may trigger the relocation process before an imbalance is detected. However, these artificial customer requests have to be replaced by "real" customer requests, as soon as a customer makes a corresponding request.

The system states $\boldsymbol{z}^t$ are influenced by customers and drivers taking or returning cars to the stations. For that we define for every time t an update vector $\mathbf{u}^t \in \mathbb{Z}^n$ where each index corresponds to the number of cars taken from a station v ($u_v^t < 0$) or returned to v ($u_v^t > 0$). Then we can define for a system state $\boldsymbol{z}^t$ at time t the successor state $\boldsymbol{z}^{t+1}$ by

$$\boldsymbol{z}^{t+1} = \boldsymbol{z}^t + \mathbf{u}^t.$$

A system state $\boldsymbol{z}^t$ is feasible if $0 \leq z_v^t \leq \text{cap}(v)$ for every station $v \in V$ and infeasible otherwise. If a system state $\boldsymbol{z}^t$ is infeasible due to a station v, then there is an imbalance in v, and we say that v is underfull if $z_v^t < 0$ and overfull if $\text{cap}(v) < z_v^t$, respectively. The operator can monitor the evolution of system states over time, detecting (future) infeasible system states and create "transportation tasks" to move cars out of overfull stations and into underfull ones

(Relocation Problem). Hereby, the operator can detect future infeasible system states due to the forecasts and, thus, can react before the system becomes infeasible by sending convoys to transfer cars from stations, which will be overfull (with a high probability) in the future, and stations, which will be underfull (with a high probability) in the future. However, if customers arrive spontaneously at a station, an infeasible system state means that the request cannot be immediately served, but has to wait until the operator sends convoys to transfer some cars to the pickup station and from the pickup station, respectively. A transportation task has the form $\tau = (v, t^{ear}, t^{due}, x)$ and requests a driver to pickup $x > 0$ (resp. deliver $x < 0$) cars at station v within the time window $[t^{ear}, t^{due}]$). Hereby, t^{ear} corresponds to the earliest point in time so that the task can be fulfilled without causing an artificial imbalance to the station, and t^{due} corresponds to the latest point in time when the task must be fulfilled, i.e., when the station v becomes imbalanced. We say a transportation task is served (by a driver) if he picks up (resp. delivers) $|x|$ cars at v within $[t^{ear}, t^{due}]$; and we call a task oversatisfied if more than $|x|$ cars are picked up (resp. delivers) at v within $[t^{ear}, t^{due}]$. Furthermore, we call a transportation task with $x > 0$ a pickup (transportation) task, and analogously we call it drop (transportation) task if $x < 0$.

Note that the transportation tasks are induced by the customer requests, but not every customer request induces a transportation task as the following example shows.

Example 2.1. In Figure 2.1 a carsharing system in an urban area is illustrated, and the corresponding graph is shown in Figure 2.2.

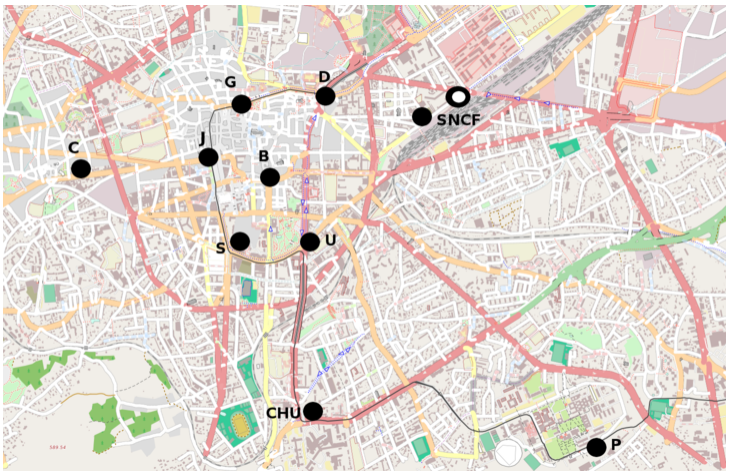


Figure 2.1: This figure shows the stations of the bikesharing system C.vélo in the urban area of Clermont-Ferrand. We use this system as our running example for an imaginary carsharing system. Each station is highlighted by a dot on the map; the name of a station is at the dot. The depot is marked as a white dot within a black dot close to the station $SNCF$.

For this example let us assume that every station has a capacity of 3 and that there are two cars at U, one car at D, and one car at S. Furthermore there are two customers booking a car. The first customer requests to take a car from D to U, and the second customer requests to take a car from S to U. Hereby, the first customer wants to take the car at time 5 and return the car at time 12, while the second customer wants to take the car at time 7 and return the car at time 9.

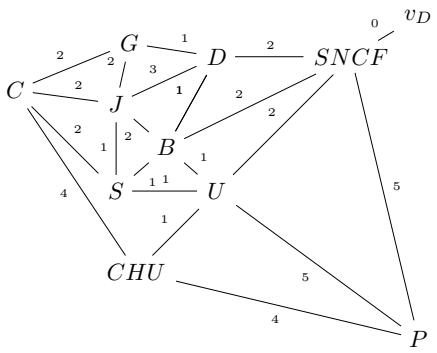


Figure 2.2: This figure shows the graph representing the carsharing system of Figure 2.1. The set of nodes correspond to the stations, the edges to the road connections between the stations. Hereby, the weights of the edges correspond to the distances between the adjacent stations.

Since at time 12 there are more than three cars at station U if no car is transferred from U, the first customer induces the pickup task $(U, 0, 12, +1)$. However, the second customer does not induce any transportation task. ◊

Our aim is defining the notion of a transportation schedule.

The Relocation Problem, can be roughly classified in two situations: the dynamic and the static situation. In our context, the names dynamic and static situation refer to the time when the cars are relocated; in the dynamic situation, the customer interact with the carsharing system during the relocation process, while in the static situation, the customers do not interact with the system during the relocation process. Thus, the system state changes in general during the relocation process in the dynamic situation and the operator has to react dynamically to the changes. Hereby, several relocation steps have to be performed in general. On the other hand, there are no (or only little changes) of the system states in the static situation. This implies, that there is only one relocation step needed in order to balance the system. Due to the missing interaction of the customers with the system, it follows that the transportation tasks of the static situation have a special structure, namely $(\cdot, 0, T, \cdot)$, where T is a given time horizon corresponding to the time difference between the closing and opening of the system (or to the maximal time the relocation step is allowed to last).

In practice, the dynamic situation occurs when the cars are relocated during the operation time of the system during the day, and the static situation is considered when the system is prepared during the night (when the customers do not have any or only a limited access to the system) for the next day by setting up a good start state for the next day.

In the following, we model the dynamic and the static relocation problem within a metric task system. Furthermore, we discuss how the customer behavior and their requests are represented within this model. This implies, the representation of information about the behavior of the customers of the carsharing system (e.g., gained from statistics), and their usage within a metric task system in order to solve the Relocation Problems. When the system is prepared during the night, it is essential to have information about the customer behavior to predict a good start state for the next day. In the dynamic situation, customers can return cars to stations from that other customers take cars. Having information about self-fulfilling[1] customer requests in advance

[1]This means that a customer returns a car to an empty station before another customer takes this car. Therefore, it is not necessary for the operator to relocate cars between the stations. The situation when a customer takes a car from a full station, before another customer returns its car to that station is analog.

can help to avoid unnecessary transports between the stations, resulting in lower costs for the operator. In order to solve the Relocation Problem, an operator computes a "transportation schedule" for the drivers, which we define in the following.

An action for driver j is a 3-tuple $a = (j, v, x)$, where $v \in V$ specifies the location $loc(a)$, and $x \in \mathbb{Z}$ the number of cars $\Delta x(a)$ to be loaded (if $x > 0$) or unloaded (if $x < 0$)[2]. An action is empty if $x = 0$. We say that an action is performed (by a driver) if he loads (resp. unloads) $|x|$ cars at v.

For technical reasons, the vector $\boldsymbol{z}^t \in \mathbb{N}^{|V|}$ represents the number of cars in a location v at time t before any action is performed and before any customer takes or returns a car.

Drivers can transfer cars between stations. We call this a move and denote it by a 6-tuple $m = (j, v, t^v, w, t^w, x_m)$, where $j \in \{1, \dots, k\}$ specifies the driver $driv(m)$ that has to move from the origin station $orig(m) = v \in V$ starting at time $dep(m) = t^v$ to destination station $dest(m) = w \in V$ arriving at time $arr(m) = t^w$, a load of $\ell oad(m) = x_m$ cars in the convoy moving along a shortest path between the stations. A move m with $orig(m) = dest(m)$ is called waiting move. A convoy can transfer a maximal number of cars at one time. This capacity is denoted by L and must not be exceeded, i.e., we have $\ell oad(m) \leq L$ for all moves m. In our situation every convoy has the same capacity and all cars are considered to be the same (single commodity). Additionally, we have the following constraints:

(m.i) from $orig(m) \neq dest(m)$ follows $arr(m) = dep(m) + d(orig(m), dest(m))$,

(m.ii) $0 \leq \ell oad(m) \leq L$.

A tour $\Gamma = (m_1, a_1, m_2, a_2, \dots, a_{n-1}, m_n)$ performed by one driver j is an alternating sequence of moves and actions starting and ending in a depot, with

(t.i) $j = driv(m_1) = driv(a_1) = \cdots = driv(a_{n-1}) = driv(m_n)$,

(t.ii) $dest(m_i) = loc(a_i) = orig(m_{i+1})$,

(t.iii) $arr(m_i) = dep(m_{i+1})$, and,

(t.iv) $\ell oad(m_{i+1}) = \ell oad(m_i) + \Delta x(a_i)$.

Any subsequence of a tour is called a subtour.

Remark 2.2. Note that the number of cars to be loaded ($x > 0$) / unloaded ($x < 0$) is from the view of the driver. Therefore, in (t.iv) we add $\Delta x(a_i)$ to the load of the move m_{i+1}. ♦

If a car is transported in one convoy from its origin to an intermediate location, and from there by another convoy to its destination, then there are dependencies between these two tours. In order to ensure that these dependencies are fulfilled, we probably have to insert empty actions and waiting moves into tours. This leads to the definition of a transportation schedule for a metric task system $(M, \mathcal{T})$, which is a set of tours $\{\Gamma_1, \dots, \Gamma_k\}$, such that

(s.i) every driver has exactly one tour,

(s.ii) each transportation task is served,

(s.iii) all system states are feasible during the whole time horizon.

[2]One can model that each action takes some time depending on the number of cars loaded or unloaded by a function $f : \mathbb{N} \to \mathbb{N}$. In this case, an action can be considered as a 4-tuple $a = (j, v, f(x), x)$.

Hereby, constraint (s.ii) ensures that every transportation task in $\mathcal{T}$ is fulfilled by one or more actions. That the capacity constraints of all stations are respected at any time, is ensured by (s.iii). i.e., at every point in time $t \in [0, T]$ we have $0 \leq z_v^t \leq \text{cap}(v)$ at every station $v \in V$.

Example 2.3. Let us consider the situation described in Example 2.1.

Now, at time 2 a third customer books a car from CHU to U. Hereby, the requested pickup time is 3 and the requested drop time is 4. The system states induced by the customer requests are shown in Table 2.1. Hereby, the future infeasible system states are highlighted by bold numbers.

Table 2.1: This table shows the system states due to the forecasts. Hereby, future infeasible system states are highlighted by bold numbers. For the sake of readability, only the number of cars at the involved stations are listed.

Name / time	1	2	3	4	5	6	7	8	9	10	11	12	13	14
U	2	2	2	3	3	3	3	3	3	3	3	**4**	**4**	**5**
D	1	1	1	1	0	0	0	0	0	0	0	0	0	0
S	1	1	1	1	1	1	0	0	0	0	0	0	0	0
CHU	0	0	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

The third customer request induces two additional transportation tasks: the pickup task $(U, 0, 14, +1)$ and the drop task $(CHU, 0, 3, -1)$. Thus, there are at the moment three transportation tasks: the pickup task $\tau_1 = (U, 0, 12, +1)$ (see Example 2.1), the pickup task $\tau_2 = (U, 0, 14, +1)$ and the drop task $\tau_3 = (CHU, 0, 3, -1)$.

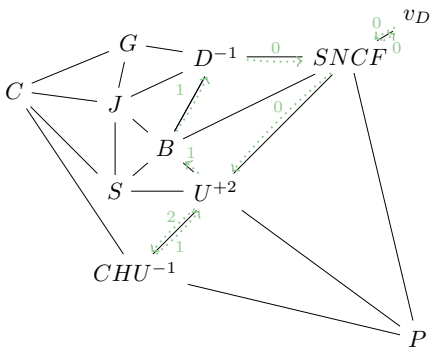


Figure 2.3: This figure illustrates a transportation schedule computed from the transportation tasks $(U, 0, 12, +1)$, $(U, 0, 14, +1)$, $(CHU, 0, 10, -1)$ and $(D, 0, 14, -1)$. The moves of the transportation schedule are highlighted by dotted arcs, the pickup and drop actions by the numbers superscripted at the nodes. The weights of the arcs correspond to the number of cars in the convoy.

Since there are two pickup tasks but only one drop task, it is helpful to release a further artificial drop task $\tau_4 = (D, 0, 14, -1)$, to compute a transportation schedule for one driver with

a capacity of 5

$$
\begin{aligned}
&\{\\
&\quad (1, v_D, 0, U, 2, \{v_D, SNCF, U\}, 0), && \%\ \text{move } v_D \to U\\
&\quad (1, U, 0, 1), && \%\ \text{pickup 2 cars (task } \tau_1 \text{ and } \tau_2)\\
&\quad (1, U, 2, CHU, 3, \{U, CHU\}, 2), && \%\ \text{move } U \to CHU \text{ with 2 cars}\\
&\quad (1, CHU, 0, -1), && \%\ \text{deliver 1 car (task } \tau_3)\\
&\quad (1, CHU, 3, D, 6, \{CHU, U, B, D\}, 1), && \%\ \text{move } CHU \to D \text{ with 1 car}\\
&\quad (1, D, 0, -1), && \%\ \text{deliver 1 car (task } \tau_4)\\
&\quad (1, D, 0, v_D, 2, \{D, SNCF, v_D\}, 0), && \%\ \text{move } D \to v_D\\
&\}.
\end{aligned}
$$

This transportation schedule is illustrated in Figure 2.3. ◊

Next, we explain the notion of preemption between tours, where cars are exchanged by drivers at a station v. Hereby, it is reasonable to assume that cars are only exchanged at stations and not on the street where exchanging cars can possibly cause a traffic jam. In other words, there is a preemption in v if a driver drops cars at v and these cars are picked up by another driver.

Condition (s.iii) of a transportation schedule requires that, besides the canonical precedences between an action $a^i \in \Gamma$ and its successor action $a^{i+1} \in \Gamma$, also dependencies between tours are respected if preemption is used. This causes dependencies between tours, since some moves or actions cannot be performed before others are done without leading to infeasible intermediate states (the reason why tours may contain waiting moves and empty actions). Let $a_1 = (\cdot, v, -x_1)$ and $a_2 = (\cdot, v, x_2)$ be the corresponding drop and pickup actions. It is possible that a_1 and a_2 cannot be performed independently of each other, e.g., if there are less than x_2 cars at v before a_1 is performed. In this case, the system state $\boldsymbol{z}^t$, which would result if a_1 (resp. a_2) is not performed, is infeasible. Then we say that there is a precedence between a_1 and a_2 (avoiding $\boldsymbol{z}^t$). Furthermore, there is a precedence between the preceding move m_1 of a_1 and the successor move m_2 of a_2 and between their corresponding tours.

Example 2.4. Let us continue from the situation of Example 2.3. At time 3, two additional customers arrive in the system, each prebooking a car: one wants to take a car from D at time 6 returning it to station B at time 13; and the second customer books a car from J to S taking it at time 7 and returning it at time 11.

Due to the first customer, the formerly artificially added task τ_4 becomes an obligatory task. However, the time window of this task has to be adjusted accordingly, resulting in $\tau_4 = (D, 0, 6, -1)$. Furthermore, since there are no cars at J, the second customer induces another drop task $\tau_5 = (J, 0, 7, -1)$.

Analogously to before, we have to add an artificial pickup task $\tau_6 = (U, 4, 4, 1)$ so that the number of pickup and drop tasks are equal.

Since one car has to be transferred from U to CHU before time 3, one car from U to D before time 6, and another car to J before 7, it is impossible that only one driver can serve all tasks.

Therefore, we compute a transportation schedule with two drivers. A possible solution in this case includes preemption at the station B. One driver transfers two cars from station U and exchanges one car at B with the other driver. Afterwards, one driver continues to serve τ_4 while the other serves τ_5.

The transportation schedule for both drivers with a capacity of 5 is

$\{$

$(1, v_D, 0, U, 2, 0)$,	% move $v_D \to U$
$(1, U, +2)$,	% pickup 2 cars (task τ_1 and τ_2)
$(1, U, 2, CHU, 3, 2)$,	% move $U \to CHU$ with 2 cars
$(1, CHU, -1)$,	% deliver 1 car (task τ_3)
$(1, CHU, 3, U, 4, 1)$,	% move $CHU \to U$ with 1 car
$(1, U, +1)$,	% pickup 1 car (task τ_6)
$(1, U, 4, B, 5, 2)$,	% move $U \to B$ with 2 cars
(1, B, -1),	% deliver 1 car (preemption, action a_1)
$(1, B, 5, D, 6, 1)$,	% move $B \to D$ with 1 cars
$(1, D, -1)$,	% deliver 1 car (task τ_4)
$(1, D, 6, v_D, 8, 0)$,	% move $D \to v_D$
$(2, v_D, 0, B, 2, 0)$,	% move $v_D \to B$
$(2, B, 0)$,	% empty action preparing to wait
$(2, B, 2, B, 5, 0)$,	% waiting move at B
(2, B, +1),	% pickup 1 car (preemption, action a_2)
$(2, B, 5, J, 7, 1)$,	% move $B \to J$ with 1 car
$(2, J, -1)$,	% deliver 1 car (task τ_5)
$(2, J, 7, v_D, 11, 0)$,	% move $J \to v_D$

$\}$.

This transportation schedule is illustrated in Figure 2.4.

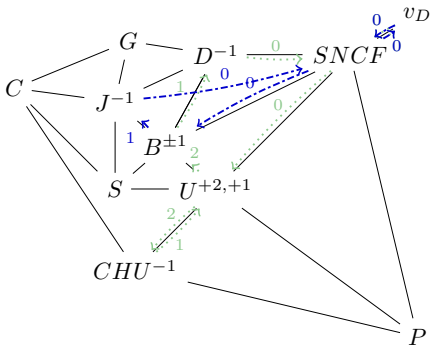


Figure 2.4: This figure illustrates a transportation schedule with preemption. The moves of the transportation schedule are highlighted by dotted (driver 1) and dashdotted arcs (driver 2), the pickup and drop actions by the numbers superscripted at the nodes. The weights of the arcs correspond to the number of cars in the convoy.

Due to the precedence in station B, there is a precedence between the actions a_1 and a_2, i.e., a_1 must be performed before a_2. Furthermore, since driver 2 cannot leave the station B

with a car before driver 1 arrives and exchanges that car, there is also a precedence between the preceding move of a_1 and the successor move of a_2. ◊

It is easy to see that when there is a precedence between two tours, then at least one car is exchanged between the drivers. However, the precedences are not always unique. This means that there may be several possible precedences between several tours, i.e., a precedence can be seen as a precedence between the tours Γ^1 and Γ^2, and it can be seen as a precedence between the tours Γ^1 and Γ^3.

The canonical precedences within a tour and the precedences between tours canonically induce a precedence relation. As mentioned before, this relation is not always unique. For every transportation schedule, there must exist a precedence without directed cycles, but may may contain undirected cycles. Directed cycles in the precedences of a transportation schedule result in a deadlock-like situation and, thus, the transportation schedule could not be performed. Note that a precedence graph without directed cycles induce a partial order. Thus, the precedence relations can be illustrated by a Hasse diagram, the precedence graph (see Figure 2.5 for an illustration).

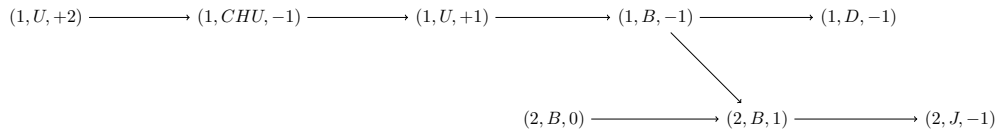


Figure 2.5: This figure illustrates the precedence relation of the actions of the transportation schedule from Example 2.4. The action represented by the start node of an arc must be performed before the action represented by the end node.

The following theorem gives an easy condition to detect preemption.

Theorem 2.5. *Let there be a preemption between tours. Then at least one of the following conditions is true:*

(i) an action drops cars at an overfull station,

(ii) an action picks up cars at an underfull station,

(iii) an action drops cars at a balanced station, or

(iv) an action picks up cars at a balanced station.

Proof. Since there is a preemption between tours, it follows by definition that cars are exchanged between the tours. Furthermore, cars cannot be exchanged on the street but only at a station, i.e, at a balanced, overfull or underfull stations.

If the cars are exchanged at a balanced station v_b, it follows that a car is dropped at v_b and it is later picked up at v_b. Thus, the statements (iii) and (iv) follow.

Let the successor action a^i of m^i drop cars at an underfull station $v^u = loc(a^i)$. If the preceding action a^j of m^j does not pick up cars at v^u, then there is no preemption between the two moves, contradicting the condition of this theorem. Therefore, the statement (ii) follows. The statement (i) can be shown analogously. □

A transportation schedule in that a move possibly depends on another move (in another tour) is called a preemptive transportation schedule, otherwise it is called non-preemptive transportation schedule. If a preemptive transportation schedule is a feasible solution, we say that we

consider the preemptive situation. Whenever there must not be any possible precedence between tours in a transportation schedule, it is the so-called non-preemptive situation. The definition of (non-)preemption we use here coincides with the definition of (non-)preemption often used in the context of vehicle routing problems or dial-a-ride problems, see, e.g., [42, 86, 154].

In the preemptive situation, a transportation schedule can be preemptive or non-preemptive, while in the non-preemptive situation, only non-preemptive transportation schedules are allowed. Thus, the non-preemptive situation is a special case of the preemptive situation.

Besides preemption between tours, there can be "preemption" within a tour. This means that a car is dropped at a station and picked up later again by the same driver. We call a tour with this property a tour with inner preemption[3]. Preemptive transportation schedules can also include tours with inner preemption. If not stated otherwise, a non-preemptive transportation schedule must not contain tours with inner preemption.

In an urban area, there are usually not only the stations where two convoys can meet to exchange vehicles but also other preemption locations, e.g., parking areas or parking decks. From a modeling point of view, the only difference between these locations and a station is that customers are not allowed to take or return cars to a preemption location. Analog to a station, the capacity of a preemption location is set to the number of parking spaces at that location.

Finally, we can formally introduce the Static and Dynamic Relocation Problem.

Problem 2.6 (Dynamic Relocation Problem $(G, \boldsymbol{z}^0, \mathcal{Z}, \gamma, k, \text{cap}, L, R)$)**.** *The Dynamic Relocation Problem has as input the following data:*

- *a weighted graph $G = (V \cup \{v_D\}, E, cap, w)$, where the nodes correspond to stations and the depot v_D, edges to their links, node weights to the station's capacities, and the edge weights $w : E \to \mathbb{R}_+$ determine the driving times between two points $v, v' \in V$ as length of a shortest path from v to v';*
- *the initial quantities $\mathcal{Z}_v$ of drivers at station v at time 0, and a total number of k drivers, the maximum number $L \in \mathbb{N}$ of cars which can be simultaneously moved in one convoy, the initial quantities $z_v^0 \leq cap(v)$ of cars located at v at the start time $t = 0$; and*
- *a sequence R of customer requests, reflecting the interaction of the customers with the system.*

The output is a transportation schedule $\mathcal{S}$ for the induced metric task system.

Note that, due to the interaction of the operator and the customers, the metric task system may change over the time (in particular the transportation tasks).

Finally, we formally define the static version of the Relocation Problem. As already mentioned, the transportation tasks of the static version have a special structure and, thus, it defines a special case of the Dynamic Relocation Problem.

Problem 2.7 (Static Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$)**.** *The Static Relocation Problem consists of the following data:*

- *a weighted graph $G = (V \cup \{v_D\}, E, cap, w)$, where the nodes correspond to stations and the depot v_D, edges to their links, node weights to the station's capacities, and the edge weights $w : E \to \mathbb{R}_+$ determine the driving times between two points $v, v' \in V$ as length of a shortest path from v to v'; and*

[3] A tour that drops a car at one station and picks up this car later again is also called lasso-tour.

- *the initial quantities* $\mathcal{Z}_{v_D}$ *of drivers at depot* v_D *at time* 0 *with* $k = |\mathcal{Z}|$*, the maximum number* $L \in \mathbb{N}$ *of cars which can be simultaneously moved in one convoy, the initial quantities* $z_v^0 \leq cap(v)$ *of cars located at* v *at the start time* $t = 0$*, with* $\gamma = \left|\boldsymbol{z}^0\right| = \left|\boldsymbol{z}^T\right|$.

The output of the Static Relocation Problem is a transportation schedule $\mathcal{S}$ *for a metric task system, whose tasks are directly induced by the start and destination system states, i.e., for every station* v *there exists a transportation task* $(v, 0, T, z_V^T - z_v^0)$.

Chapter 3

Outline

The rest of this thesis is structured as follows. We start in Part II by focusing on a general aspect of the Relocation Problem: the Online Relocation Problem. Hereby, we focus on two main categories of problems: decision and optimization problems (Chapter 5). Decision problems deal with the problem of deciding which customer request shall be accepted and which shall be rejected; while the study of optimization problems addresses the problem of finding cost efficient tours serving all released customer requests.

In Chapter 4 we consider two objective functions for the decision problems. Firstly, we consider the objective to minimize the number of rejected request. Secondly, we try to maximize the number of accepted requests. At first glance, these two objective functions may seem to be equivalent. However, we show that they are indeed very different in terms of a formal analysis. While the first objective does not seem to allow competitive online algorithms, we state some competitive online algorithms for maximizing the number of accepted requests.

In Chapter 5, we consider two optimization problems, where we try to minimize the waiting times for customers who spontaneously arrive at a station, and where we try to minimize the total tour lengths for the drivers while serving all customers. Since the latter itself does not lead to any meaningful results, we focus on a variation of this problem. Hereby, customer requests must be served within a time-window and if it is not possible to serve a customer request then the algorithm is "penalized" by additional costs; the objective of this problem is to find a transportation schedule with minimal costs.

Since for most of the problems, there do not exist meaningful theoretical results, we conclude the part by presenting and discussing computational results (Chapter 6). In this chapter, we also show how an optimal offline solution can be computed exactly and by a flow-based heuristic. This problem is of interest due to two reasons: solving intermediate steps for an online algorithm; and to evaluate the decisions of an online algorithm by comparing it with an optimal solution.

Part III focuses on the problem of preparing the carsharing system during the night for the next morning. Thus, the problem is to reach a destination system state from a given initial system state. Hereby, there is no (or only little) interaction between the customers and the carsharing system, which motivates to call this kind of problem the Static Relocation Problem. In the first chapter of this part, Chapter 7, we consider an "exact" version of this problem, where the destination state must be reached within a given time limit. This problem is already known to be $\mathcal{NP}$-hard, which is why we give several heuristics, besides an exact approach. Furthermore, since the computation time is quite large, and to be able to make some statements about the quality of the heuristics, we give some lower bound for the Static Min-Cost Relocation Problem. Finally, in Chapter 8, we consider a "relaxed" version of this problem, i.e., the aim is to reach a state close to the destination state. Close means in this context, that we minimize the difference

between the reached state and the destination state (with respect to the ℓ_1 norm). Since we consider the ℓ_1 norm as objective function, the problem is not linear. Therefore, we give a reformulation of the problem which is linear.

Afterwards, in Chapter 9, we present computational results and experimentally evaluate the different approaches. Since we have heuristics computing a preemptive and other computing non-preemptive transportation schedules, we give some theoretical results for a worst-case ratio between total tour lengths of preemptive and non-preemptive tours. In our context, preemption means, that a car is transferred from one station to another station by two different drivers. In other words, in a preemptive situation, cars are allowed to be exchanged between convoys, while in a non-preemptive situation, the car must directly be transferred from its origin to its destination station.

Finally, we end this thesis with Part IV by giving a brief conclusion, some final remarks and open problems.

Part II

Dynamic Relocation Problem

In this part, we consider the dynamic situation of the Relocation Problem where customers can take a car from a station and return it at their destination station. The more customers use the system, the more frequently cars are moved between the stations. From this follows that it is also more likely that stations run out of cars or become full. Thus, in bigger systems, it is usually not enough to set up a good system state for the morning, but it is necessary to balance the system by relocating cars during the working hours. For that we apply the theory of online optimization to this problem.

We start by considering the Dynamic Relocation Problem from a theoretical point of view. Unfortunately, it is unlikely that every customer entering the system can be served. In our case, we consider customers which do not wait for their request to be served but immediately search for another way of transportation. Therefore, we consider in Chapter 4 the problem of deciding which customer request should be accepted and which should be rejected in order to maximize the number of accepted customer requests (or to minimize the number of rejected customer requests). Hereby, we demonstrate that whether a competitive online algorithm exists highly depends on the considered objective function. Furthermore, we show that the max/max ratio is not an appropriate tool for evaluating online algorithms for these objective functions.

In Chapter 5, we focus on some optimization problems. Hereby, we assume that customers do not necessarily book their requests but spontaneously arrive in the system. However, if they cannot be immediately served, they wait at the pickup station for a free car and a free parking place at their destination, respectively. Thus, we consider as objective function the minimization of the waiting times: the total waiting time (Online Min-Wait Relocation Problem) and the maximal waiting time of all customers (Online Max-Wait Relocation Problem). While in the Online Min-Wait Relocation Problem it may happen that one customer has to wait for a long time while all other customers are served (nearly) immediately, the latter objective function tries to avoid that customers wait for a highly imbalanced amount of time. In both cases, we show that there does not exist a competitive online algorithm and that the max/max ratio does not lead to applicable theoretical results.

Minimizing the total tour length while allowing customer requests to be rejected turns out to be meaningless for the Relocation Problem since even very basic online algorithms achieve a competitive ratio of 1. However, this changes when customers book their requests in advance and the algorithm is "penalized" for not serving them. Therefore, at the end of the second chapter of this part, we consider a mixed objective function optimizing the total tour length of the resulting transportation schedule while maximizing the number of accepted customer requests. Hereby, customers book their cars in advance as they do in the first chapter of this part and leave the system when their request cannot be served. For this situation, we give a competitive online algorithm for a restricted situation and show when there cannot exists one. As before, we show that the max/max ratio is not an appropriate tool for evaluating the performance of online algorithms.

Finally, in Chapter 6, we evaluate some online algorithms by performing experiments in a simulation.

In this part, if not stated otherwise, we assume that there are no capacities for the drivers ($L = \infty$) as well as for the stations ($\text{cap}(v) = \infty$ for all $v \in V$). Furthermore, throughout this part, the movement speed for the drivers and the cars is 1.

Chapter 4

Decision Problems

In a carsharing system, customers usually can prebook a car which they can pickup at a certain station at a certain time. Depending on the operator of the system, customers may be allowed to return the car at any station at any time or the customer may be required to specify their drop station and time in advance. For the customer this has the advantage that, if the request is accepted, there is a parking place reserved and available for the customer. Furthermore, the time when the booking is confirmed or rejected may differ from carsharing system to carsharing system. Obviously, it is more user-friendly when the confirmation or rejection of a booking is done immediately, and it is more operator-friendly if the operator can confirm or reject a booking in last second.

In this chapter, we consider the situation when customers book their requests in advance. A request can be either accepted or rejected, but once it is accepted it must be served, i.e., there must be a car and a parking place available at the associated stations. Hereby, unless stated otherwise, we consider the operator-friendly (and consumer-unfriendly situation), where the decision whether a request is accepted or rejected can be delayed to the moment the customer arrives at the station. However, when the customer cannot immediately take a car from the pickup station to the drop station, he does not wait but directly searches for another way of transportation.

In the following, we consider two different decision problems, one where the objective is to minimize the number of rejected customer requests (see Section 4.1) and where the objective is to maximize the number of accepted customer requests (see Section 4.2). This means, the problem for the operator is deciding which request to accept and which to reject. Although, at first glance these two problems may seem to be equivalent, we show that these two problems are very different in terms of the theoretical analysis within the framework of the competitive analysis.

This chapter provides more details as well as the omitted proves for the results presented in [93].

4.1 Minimizing the Number of Rejected Customer Requests

In this section, we concentrate on a quality of service aspect of the Dynamic Relocation Problem, where the objective is to minimize the number of rejected requests. Hereby, we do not take the distance traveled by the drivers into account.

The Online Min-Reject Relocation Problem $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \text{cap}, L, R)$ is a Dynamic Relocation Problem where the goal is to compute a transportation schedule, with a minimal number of

rejected customer requests, i.e., where the objective function is

$$\min \sum_{j=1}^{\lambda} \mathrm{rej}(r_j),$$

where

$$\mathrm{rej}(r) = \begin{cases} 1, & \text{if } r \text{ is rejected,} \\ 0, & \text{otherwise.} \end{cases}$$

4.1.1 Competitive Analysis

We start the analysis of the Online Min-Reject Relocation Problem by considering an arbitrary online algorithm (see Section 1.4.2 for some examples) and show that this algorithm, under certain conditions, cannot be competitive against a non-abusive direct δ-adversary. Hereby, we even restrict the metric space to the uniform metric space.

However, we can show that there exists a competitive online algorithm on the uniform metric space, if the number of cars and the number of drivers are equal, which means that the problem basically becomes an online dial-a-ride problem.

Theorem 4.1. *There is no competitive deterministic or randomized online algorithm for the Online Min-Reject Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 3$ *and* $L < \infty$*, against a non-abusive direct* δ*-adversary for any* $\delta \in \mathbb{N}$*, if*

$$\left\lceil \frac{\gamma}{L} \right\rceil \geq k$$

holds.

Proof. We start by considering a deterministic online algorithm ALG and a direct δ-adversary. Afterwards, we show how the result can be generalized for non-abusive adversaries and randomized online algorithms.

From the condition $\lceil \gamma/L \rceil \geq k$ it follows that at least k driver are needed in order to transfer $\gamma - 1$ cars.

Firstly, we assume that $\lceil \gamma/L \rceil = k$ holds. Let the metric space contain the points $\{v_1, v_2, v_3\}$ and let all cars and driver start in v_1.

From time 0 until at least time $2\delta + 1$ and for every second integer point in time $2j$, the adversary releases $\gamma - 1$ customer requests from v_1 to v_2

$$r_{ij} = (2j, v_1, 2j + \delta, v_2, 2j + \delta + 1)$$

for all $1 \leq i \leq \gamma - 1$. In order to serve all these customer requests, one has to move $\gamma - 1$ cars from v_2 to v_1 at every time step after the time δ. For that all drivers have to move between the two stations v_1 and v_2.

At time $\delta + 1$, the "last" car cannot be all three stations. The adversary moves this car to one of the stations where the online algorithm does not have the car at time $\delta + 1$.

Case 1: ALG has the "last" car at station v_1 or v_2 at time $\delta + 1$. Then ADV moves the car to the station v_3. At time $\delta + 1$ he releases the following customer request

$$r = (\delta + 1, v_3, 2\delta + 1, v_1, 2\delta + 2)$$

which can be served by ADV. If ALG serves the request r, at least one of the requests r_{ij} cannot be served. Therefore, the adversary rejects no customer request while the online algorithm rejects at least one.

Case 2: ALG has the "last" car at station v_3. Then ADV moves the car to v_1 and at time $\delta+1$ he releases the following customer request

$$r' = (\delta + 1, v_1, 2\delta + 1, v_3, 2\delta + 2)$$

which can be served by ADV. However, if ALG wants to serve r', then at least one of the requests r_{ij} cannot be served. Therefore, the adversary rejects no customer request while the online algorithm rejects at least one.

Thus, we have in both cases $\mathsf{ADV}(\sigma) = 0$ and $\mathsf{ALG}(\sigma) \geq 1$. This implies that there cannot exist a constant c so that

$$\mathsf{ALG}(\sigma) \leq c \cdot \mathsf{ADV}(\sigma)$$

and the special case of the statement is proved.

Secondly, let there us consider that there are several cars "left" in the system, i.e., the case that $\lceil \gamma/L \rceil > k$ holds. Let $\gamma' = \lceil \gamma/L \rceil - k$. Then, the online algorithm can either place all these γ' cars at the same station which implies the above considered cases; or the cars are placed at different stations. In this case, the adversary positions its cars at station v_3 and releases γ' customer requests of the form r (Case 1). Since the online algorithm does not have positioned γ' cars at v_3, at least one customer request cannot be served and the statement follows also in this case.

Thirdly, we show that the statement holds even if the adversary is non-abusive. However, note that in Case 2, the adversary is already non-abusive, since there are released unserved customer request and, therefore, the driver can transfer cars between v_1 and v_2. However, in Case 1, we "add" a further customer request $r'' = (0, v_3, \delta, v_1, \delta + 1)$ at the beginning of the sequence. The pickup and drop times of all other customer request must be shifted so that they start after the driver serving r'' has returned. The non-abusive adversary can now transfer two (or more) cars to v_3. Therefore, also in Case 1, the online algorithm cannot be competitive against a non-abusive direct δ-adversary.

Finally, we show that randomizing an online algorithm does not help. For that the adversary constructs the sequence R in $\mathcal{C}$ phases. In each of the phases, the adversary randomly chooses to position the "last" car at either v_1, v_2 or v_3 and to release the customer requests accordingly.

A randomized online algorithm also has the "last" car at either of these stations and, therefore, misses at least one customer request with a probability of $2/3$ in each phase. Thus, we have $\mathbb{E}(\mathsf{ADV}(R)) = 0$ and

$$\mathbb{E}(\mathsf{ALG}(R)) = 1/3 \cdot 0 + 2/3 \cdot \mathcal{C} = 2/3 \cdot \mathcal{C},$$

which tends towards infinity when $\mathcal{C}$ tends towards infinity.

Since $\mathcal{C}$ can be chosen arbitrarily large, there cannot exist a constant c so that $\mathbb{E}(\mathsf{ALG}(R)) \leq c \cdot \mathbb{E}(\mathsf{ADV}(R))$ holds, proving the statement. □

The next theorem implies that as long as the number of cars (and the number of stations) is greater than the number of drivers, then one cannot find a competitive online algorithm.

Theorem 4.2. *There is no competitive deterministic or randomized online algorithm for the Online Min-Reject Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$*, against a non-abusive direct* δ*-adversary for any* $\delta \in \mathbb{N}$*, if*

$$\gamma \geq k + 1 \quad \textit{and} \quad n \geq k + 2$$

holds.

Proof. The basic idea of this proof follows the same ideas as the proof of the previous theorem. □

Table 4.1: This table lists several car and bikesharing systems of different sizes. In the first column, the name of the system is stated. The second column gives the number of stations, while the number of vehicles is given in the third column.

Name of the system	# stations	# vehicles
Citiz LPA (Lyon, France)	37	100
Blue cub (Bordeaux, France)	40	90
Autoshare (Toronto, Canada)	175	300
Autolib' (Paris, France)	975	2500
C.vélo (Clermont-Ferrand, France)	22	220
Coca Cola Zero Belfast Bikes (Belfast, UK)	30	300
nextbike (Berlin, Germany)	50	300
V'Lille (Lille, France)	220	1100
Vélo'v (Lyon, France)	348	4000
Vélib' (Paris, France)	1230	18000
Hangzhou Public Bicycle (Hangzhou, China)	2400	61000

The conditions of previous theorem are pretty realistic. In Table 4.1 there are several car and bikesharing systems listed with their number of cars/bicycles and stations. Unfortunately, we could not find the number of drivers/trucks within the system. However, due to the high costs of trucks and drivers it is unlikely that the number of driver/trucks is greater than or equal to the number of stations, and even less likely that the number of drivers is equal to the number of cars (resp. bikes). However, since the given sequences are more of a theoretical nature, a low number of drivers should be irrelevant in practice.

Furthermore, note that the two previous results also imply that there does not exist a deterministic or randomized online algorithm for the Online Min-Reject Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \text{cap}, L, R)$ for any graph G with at least three points and a minimal distance of 1 between two stations when the stated conditions from Theorem 4.1 or 4.2 hold.

Finally, we restrict the situation so that we can give a competitive online algorithm. Hereby, the number of cars and drivers are equal, leading more to a dial-a-ride or pickup-and-delivery problem than a Relocation Problem. This becomes even more clear, when the driver always "follows" the customer in order to be able to directly pickup the car at the destination station of the customer request. This behavior can theoretically be justified since we consider in our objective function only the number of rejected customer requests, but the additional costs for traveling are not taken into account. However, in practice, the additional costs (for the additional number of drivers and for moving the driver) can usually not be justified within the scope of a carsharing system, and even less within the scope of a bikesharing system.

In the next theorem, we consider a slightly modified version of RePlan (Algorithm 5) where to each driver a car is assigned. Whenever, a customer takes a car, the driver follows the customer to the drop station.

Theorem 4.3. *If the number of cars γ is equal to the number of drivers k (i.e., $\gamma = k$), then the online algorithm* RePlan *(where drivers follow its car) is at most 3-competitive for the Online Min-Reject Relocation Problem $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ with $n \geq 2$, against a direct δ-adversary for all $\delta \geq 2$.*

Proof. W.l.o.g. let the driver and its assigned car start at the same station. Since we consider the uniform metric space, the time needed to serve a customer request is at most 2 (at most one time unit is needed to move the driver to a station with a car, and another time unit is needed

to move to the pickup station of the customer request). Thus, it is possible to serve a customer request every second time step.

Since the number of cars is equal to the number of drivers, and we consider a discrete time and since all direct customer requests are direct, it follows from $\delta \geq 2$ that every newly released customer request can also be served. Furthermore, the algorithm can decide until the pickup time of a customer request whether to serve the request or to reject it.

Additionally, when a new customer request r_j is released at time t and when the driver and the car are positioned at the pickup station of a customer request r at the pickup time $t_p = t$ of r, the two requests r and r_j can be served.

Finally, let $\sigma = (r_1, \ldots, r_\lambda)$ be an arbitrary sequence of customer requests. W.l.o.g. let the customer requests be ordered so that the requested pickup times are increasing, i.e., for two customer request $r_i, r_j \in \sigma$, with $r_\ell = (t_\ell, v_\ell, t_{v_\ell}, v'_\ell, t_{v'_\ell})$, it holds $t_{v_i} \leq t_{v_j}$ if $i < j$. Next, we compare the requests which are served by ADV and the requests which are served by RePlan. From this we can conclude the number of rejected customer requests.

Firstly, we make some general observations, and secondly, we consider a worst-case scenario. Hereby, we show that RePlan rejects at most three times more customer requests than the adversary.

Let $\sigma_{\mathsf{ADV}} \subseteq \sigma$ be a sub-sequence of all customer requests which are served by ADV. If the pickup time of two succeeding customer requests of σ_{ADV} is at least 2, then it follows from above that RePlan can also serve the same number of customer requests. Note, the requests served by ALG and by RePlan may be different. Furthermore, the time when ALG and RePlan start serving the requests may be shifted by 1 and, thus, it may be possible that the number of requests rejected by ALG is one less than the number of requests rejected by RePlan. Let $x \in \mathbb{N}$ be the number of customer requests rejected by ADV then it follows in this case

$$\frac{\text{RePlan}(\sigma)}{\mathsf{ADV}(\sigma)} \leq \frac{x + 2x}{x} = 3.$$

Next, we consider a worst-case scenario for RePlan. Let ADV serve one customer request at every time step (within a time window). We show that then there are no three successive time steps where RePlan does not serve any customer request. For that let us assume the contrary, i.e., that there are three successive time steps $t, t+1$ and $t+2$ where RePlan does not serve any customer request. Hereby, let $t > 0$ (the case $t = 0$ is handled afterwards). Afterwards, let RePlan serve the customer request r (with a pickup time of at least $t+3$). Let ADV serve the customer requests $r_1, r_2, r_3 \in \sigma_{\mathsf{ADV}}$ with $\hat{t}(r_1) = t$, $\hat{t}(r_2) = t+1$ and $\hat{t}(r_3) = t+2$. The existence of these customer requests is ensured, since ADV serves one customer request at every time step. Since $t > 0$ it follows that the car and the driver are at the same station. Thus, no matter where RePlan has positioned the driver (with the car), RePlan can serve r_2 and afterwards r. Since, RePlan always computes an optimal transportation schedule when a new customer request is released, RePlan would have served r_2 (or another customer request with a pickup time at $t+1$).

In the case of $t = 0$, it may happen that the car and the driver are at different stations. However, if this is the case, then the car and the driver of ADV are also in different station. Since r_1 can be served by ADV, it follows that r_1 can also be served by RePlan, and again, there cannot be three successive time steps where RePlan does not serve any customer request.

Thus, in the worst-case, there are two successive time steps where RePlan does not serve any customer request. In the following, we examine this case in detail.

Let ADV serve the customer requests $r_1, r_2, r_3, r_4 \in \sigma_{\mathsf{ADV}}$ with $\hat{t}(r_1) = t$, $\hat{t}(r_2) = t+1$, $\hat{t}(r_3) = t+2$ and $\hat{t}(r_4) = t+3$. Furthermore, let RePlan serve the customer requests r'_1, r'_4 with $\hat{t}(r'_1) = \hat{t}(r_1)$ and $\hat{t}(r'_4) = \hat{t}(r_4)$. We prove that $r_1 \neq r'_1$ or $r_4 \neq r'_4$ follows. For that let us assume

that $r_1 = r_1'$ or $r_4 = r_4'$. Then, RePlan could serve the two customer requests r_2 and r_3, and since RePlan always computes an optimal transportation schedule as soon as a new customer request is released, it also would serve these two requests. Thus, it follows that at least one customer request must be different.

Since we have $r_1 \neq r_1'$ or $r_4 \neq r_4'$, it follows that ADV rejects at least one request (two if both are different). The online algorithm RePlan rejects at least 3 requests (four if both are different). Additionally, both may reject another x customer requests. Thus, we have

$$\frac{\text{RePlan}(\sigma)}{\text{ADV}(\sigma)} = \frac{3+x}{1+x} \leq \frac{3+3x}{1+x} = 3,$$

and if both customer requests are different

$$\frac{\text{RePlan}(\sigma)}{\text{ADV}(\sigma)} = \frac{4+x}{2+x} \leq \frac{4+2x}{2+x} = 2.$$

In all cases, the number of rejected customer requests of RePlan is at most three times the number of rejected customer requests of ADV, and thus, the competitive ratio is at most 3. □

In the proof of previous theorem, we mainly used the possibility that the driver and the car are always at the same position. Therefore, the online algorithm can react in time to serve a new released request. As we have seen before, e.g., in Theorem 4.2, this is not possible when the number of cars is greater than the number of drivers. That $\delta \geq 2$ is insofar important, that every newly released customer request can theoretically be served by an online algorithm. Finally, note that solely from the number of accepted customer requests we cannot conclude the number of rejected customer requests. Here it is necessary to show that the adversary also rejects some customer requests.

Next, we generalize the result by generalizing the notion of direct customer requests.

Theorem 4.4. *If the number of cars γ is equal to the number of drivers k (i.e., $\gamma = k$), then the online algorithm* RePlan *is at most $(2+\lceil b/a \rceil)$-competitive for the Online Min-Reject Relocation Problem $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ with $n \geq 2$, against a direct $[a,b]$-δ-adversary for all $\delta \geq 2$, $1 \leq a \leq b < \infty$*

Proof. Since most of the arguments from the proof of Theorem 4.3 hold also in this case, we concentrate on the differences to the proof of that theorem.

When RePlan serves a customer request r, the adversary can serve at most $\lceil b/a \rceil$ customer requests at the same time. This ratio can be seen as follows: when RePlan serves a customer request, the difference between the pickup and drop time is at most b. Thus, it follows that during this time, the adversary can then serve at most $\lceil b/a \rceil$ customer requests when all the differences between the pickup and drop time of these customer requests is a. In order to serve r, RePlan may reject another customer request r' with $\hat{t}(r') = \hat{t}(r) - 1$. Furthermore, RePlan may not be able to serve a customer request r'' with $\check{t}(r) = \hat{t}(r'')$. However, both r' and r'' may be served by the adversary. When the adversary serves $\lceil b/a \rceil$ customer requests while RePlan serves r, it follows that the adversary must reject r. Additionally, both may reject x customer requests. Thus, we have in this case

$$\frac{\text{RePlan}(\sigma)}{\text{ADV}(\sigma)} = \frac{1 + \lceil \frac{b}{a} \rceil + 1 + x}{1+x} \leq 2 + \left\lceil \frac{b}{a} \right\rceil. \tag{4.1}$$

If RePlan is not serving another request, the driver and the car are always at the same position. Therefore, it follows from $\delta \geq 2$ that RePlan can decide to serve any newly released customer request. Thus, the statement follows with Estimate (4.1). □

4.1.2 Max/Max Ratio

In this section, we apply the max/max ratio to the Online Min-Reject Relocation Problem and show that there does not exist a meaningful ratio for this problem on the uniform metric space with at least two points.

Theorem 4.5. *For the Online Min-Reject Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 2$*, any online algorithm* ALG *has a max/max ratio of* 1*. This result even holds if only direct customer requests are released.*

Proof. Let $\{v_1, v_2\}$ be the points of the metric space, and let all cars driver and drivers be located at v_1. Then let us consider the sequence of customer requests $\sigma = (r_1, \ldots, r_\lambda)$ with

$$r_j = (0, v_2, 0, v_1, 1),$$

for all $1 \leq j \leq \lambda$. Since the time to transfer the cars to v_2 takes at least one time unit, none of these customer requests can neither be served by any online algorithm nor by the optimal solution. Obviously, the maximal number of rejected requests is λ and, therefore, we have for any online algorithm ALG the max/max ratio

$$w_M(\mathsf{ALG}) = \limsup_{\lambda \to \infty} \frac{M_\lambda(\mathsf{ALG})}{M_\lambda(\mathsf{OPT})} = \limsup_{\lambda \to \infty} \frac{\lambda/\lambda}{\lambda/\lambda} = 1$$

which proves the statement. □

In the next theorem, we show that the same result holds for every reasonable algorithm even for direct δ-customer requests. The idea is the similar to the idea used in the proof of the previous result. All customer requests are released at the same time. Since there are only a limited number of cars, but the considered sequence grows, not every request can be served. This implies:

Theorem 4.6. *For the Online Min-Reject Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 2$*, any online algorithm* ALG *which does not unnecessarily reject customer requests has a max/max ratio of* 1*. This result even holds if only direct* δ*-customer requests are released for any* $\delta \in \mathbb{R}$*.*

From previous theorem, it follows that the max/max ratio is not a meaningful tool to evaluate the performance of online algorithms for the Online Min-Reject Relocation Problem.

4.2 Maximizing the Number of Accepted Customer Requests

In this section, we consider another quality of service aspect of the Online Relocation Problem, where the aim is to maximize the number of accepted customer requests.

At first glance, the two problems Online Min-Reject Relocation Problem (see Section 4.1) and the Online Max-Accept Relocation Problem may seem to be equivalent. However, we show in this section, that there are some fundamental differences between the two problems. One major difference is that, in order to prove the non-existence of a competitive online algorithm, for the Online Min-Reject Relocation Problem it is sufficient to find a sequence so that the adversary can serve all released customer request, while the online algorithm has to reject at least one. For the Online Max-Accept Relocation Problem, this is not sufficient. This is also the main reason why it is hard to find a competitive online algorithm (under certain but unrealistic conditions) for the

Online Min-Reject Relocation Problem (Section 4.1.1) but more easy to find one (under certain but more realistic conditions) for the Online Max-Accept Relocation Problem (Section 4.2.1).

Note, only the number of accepted customer requests are considered, but the costs for transferring cars between stations are not taken into account.

In this chapter, if not stated otherwise, we assume that there are no capacities for the drivers ($L = \infty$) as well as for the stations ($\text{cap}(v) = \infty$ for all $v \in V$). Furthermore, the movement speed for the drivers and the cars is 1.

The Online Max-Accept Relocation Problem $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \text{cap}, L, R)$ is a Dynamic Relocation Problem where the goal is to compute a transportation schedule, with a maximal number of accepted customer requests, i.e., where the objective function is

$$\max \sum_{j=1}^{\lambda} \text{acc}(r_j),$$

where

$$\text{acc}(r) = \begin{cases} 1, & \text{if } r \text{ is accepted,} \\ 0, & \text{otherwise.} \end{cases}$$

4.2.1 Competitive Analysis

In this section, we start by considering an arbitrary online algorithm (see Section 1.4.2 for some examples) and show that this algorithm cannot be competitive against the oblivious adversary. Hereby, we show this statement when the metric space is the real line, when it is an interval of the real line, and for the uniform metric space. Afterwards, we consider weaker adversaries like the direct δ-adversaries. Under certain conditions, we give some competitive online algorithms for the Online Max-Accept Relocation Problem against these weaker adversaries.

However, the proofs for the non-existence of competitive online algorithms for the Online Max-Accept Relocation Problem in the "basic situations" are similar to the proofs for the non-existence of competitive online algorithms for the Online Min-Reject Relocation Problem. Therefore, in this section, we concentrate on the differences between these two problems as well as the different results.

In the Online Max-Accept Relocation Problem, deterministic online algorithms are only competitive under certain circumstances. However, we can give some examples where there does not exist a competitive online algorithm for the corresponding Online Min-Reject Relocation Problem but there exists one for the Online Max-Accept Relocation Problem.

Finally, we consider the randomized online algorithm RandomWalk and show that it is competitive even against the oblivious adversary. The basic idea is that the driver every now and then positions a car at the "correct" station. Due to its randomness, and since the oblivious adversary does not know the outcome of random events, the adversary cannot give a worst-case scenario so that RandomWalk cannot accept any released request.

Theorem 4.7. *Let $G = (V, E, w)$ be a weighted graph and M the metric space induced by G. Then the randomized online algorithm* RandomWalk *with a uniform distribution is c-competitive for the Online Max-Accept Relocation Problem $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ against the oblivious adversary with*

$$c = 2 \cdot \sum_{e \in E} w(e) + |V| \,.$$

Proof. Firstly, let us consider the situation when there is only one car and only one driver in the system. Secondly, we show that the general situation with more cars and drivers is even more in

favor of RandomWalk, thus, the situation with only one driver and one car gives already an upper bound.

Let $r = (t, v, t_v, v', t_{v'})$ be an arbitrary customer request. When the driver picks up a car, the driver moves in the system with the car. Thus, we can show that the driver has a car at v at time t_w with a bounded probability (see Equation (4.3)). From that we can conclude the expected values for RandomWalk and ADV which then proves the statement.

We apply the theory of random walks on graphs theory. For that, we construct an unweighted directed graph $G' = (V', A')$ from G with

(i) V' contains all nodes V and for every edge $vv' \in E$ with $w(v, v') \geq 1$, an additional $2(w(v, v') - 1)$ nodes are added to V', hereby, $w(v, v') - 1$ nodes represent the direction from v to v', and the other $w(v, v') - 1$ nodes the direction from v' to v,

(ii) for every edge $vv' \in E$ there is an arc between the additionally added nodes (once to the direction v to v' and once for the other direction); if $w(v, v') = 1$, the arcs (v, v') and (v', v) are added to A', and

(iii) for every node $v \in V$ a loop (v, v) is added to A.

Note that by construction, it holds $\delta^+(v) = \delta^-(v)$ for all $v \in V$.

Note, G' represents the possible positions of the driver and the possible paths the driver can take (see Figure 4.1 for an illustration of the construction of G').

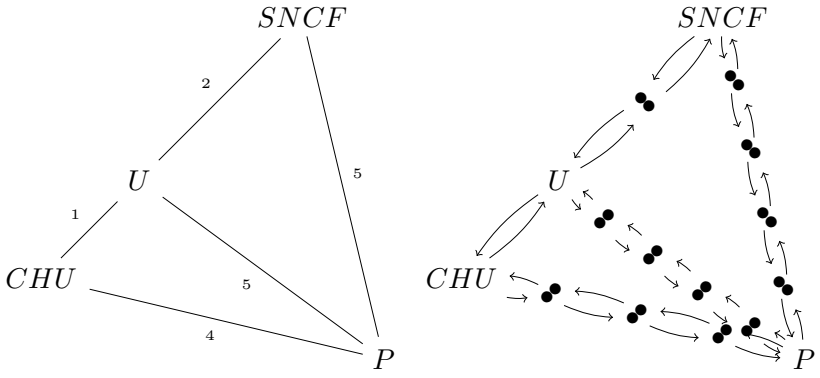


Figure 4.1: This figure illustrates the construction of the directed graph G'. Each node represents a possible position of the drivers. The graph on the left side is a subgraph of the graph from Figure 2.2, the graph on the right represents the corresponding directed graph G'.

Due to the loops, G' is non-bipartite and, therefore, there exists a stationary distribution for every node $v \in V'$ (see, e.g., [118]), i.e., the probability for the driver being at time t at node $v \in V'$ tends towards a constant as $t \to \infty$. This probability is given by

$$\frac{|\delta^+(v)|}{\sum_{v' \in V'} |\delta^+(v')|}. \tag{4.2}$$

In order to get the "real" position of the driver, we have to translate this value to G. For that, we concentrate on the bottom of the fraction of Equation (4.2).

By translating the value $\sum_{v' \in V'} |\delta^+(v')|$ to G, the additionally added nodes have to be exchanged by a suitable value. For an arc, there are $w(v, v') - 1$ additional nodes (for each

direction) and, thus, $w(v, v')$ arcs in each direction. Finally, there are $|V|$ loops. This leads to the probability for the driver being at time t at station $\tilde{v}$ (see, e.g., lecture notes on spectral graph theory [145])

$$\pi(\tilde{v}) = \frac{|\delta^+(\tilde{v})|}{\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e) + |V|}. \tag{4.3}$$

Thus, if the adversary ADV releases a customer request $r = (t_v, v, t_v, v', t_{v'})$, the online algorithm RandomWalk serves r with a probability of $\pi(v, t_v)$. In the case that ADV releases δ-customer requests for any $\delta > 0$, this probability is increased, since RandomWalk has to position the driver only within a radius of δ of v. This means, whenever ADV serves a customer request, RandomWalk serves this request with a probability of at most $\pi(v, t_v)$.

Obviously, the value $\pi(v)$ is minimal for a station with only one outgoing edge. Thus, for a sequence of customer requests σ, with $\lambda = |\sigma|$, we can estimate the expected value for RandomWalk by

$$\mathbb{E}(\text{RandomWalk}) \geq \frac{\lambda}{\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e) + |V|} \quad \text{and} \quad \mathbb{E}(\text{ADV}) \leq \lambda.$$

Thus, we have

$$\frac{\mathbb{E}(\text{RandomWalk})}{\mathbb{E}(\text{ADV})} \geq \frac{\lambda / \sum_{v \in V} \sum_{e \in \delta^+(v)} w(e) + |V|}{\lambda} = \frac{1}{\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e) + |V|}. \tag{4.4}$$

Due to the first sum in $\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e)$, each edge is counted twice. Therefore, the sum $\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e)$ can be rewritten as $2 \sum_{e \in E} w(e)$, proving the statement for one car and one driver.

Finally, we show that this ratio is already an upper bound for the general situation with k drivers. For that let the number of cars be greater or equal to the number of drivers (otherwise, not all drivers are used). To each driver, we assign one car. The probability that one driver with a car is at time t at station $\tilde{v}$ is

$$\pi_k(\tilde{v}) = \frac{k \cdot |\delta^+(\tilde{v})|}{\sum_{v \in V} \sum_{e \in \delta^+(v)} w(e) + |V|}.$$

Now, the statement follows from $\pi(\tilde{v}) \leq \pi_k(\tilde{v})$ and Estimate (4.4). □

Next, we consider the situation for deterministic online algorithms. Hereby, one can show that there does not exist a competitive deterministic online algorithm ALG for the Online Max-Accept Relocation Problem $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \text{cap}, L, R)$ with $n \geq 2$ against a non-abusive δ-adversary for any $\delta \in \mathbb{N}$. For that the adversary releases a customer request r with a sufficiently large occupation time. If ALG rejects r, the adversary serves this request and ends the sequence. However, if ALG serves r, the adversary releases a sequence of c customer requests with a small occupation time, which all can be served by the adversary (see Figure 4.2 for an illustration of this sequence on $[-x, x]$). Since c can be arbitrarily large, there cannot exist a competitive deterministic online algorithm against a non-abusive δ-adversary.

One may argue that the behavior of the adversary is not very realistic and more of a theoretical nature. Unfortunately, it may happen that users borrow their vehicle in the morning, keep it during the day close to their place of work and return it in the evening. This scenario is described by Kidd in [105] where the author writes in a blog about a bikesharing system in Japan: “Unfortunately locals have been using the bicycles for commuting and leaving them parked outside schools, offices or at train stations for the entire day rather than returning them

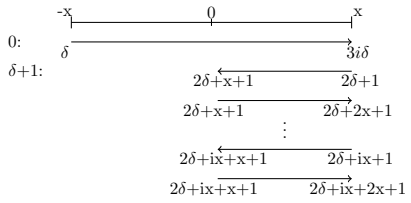


Figure 4.2: This figure illustrates the released customer requests released by a δ-adversary with $\delta \in \mathbb{N}$. On the top of the image, the interval $[-x, x]$ is shown. The number on the left corresponds to the release time of the customer requests, which are illustrated by arrows. The numbers below the arrows correspond to the pickup and drop times of the customer request.

to docking stations making them unavailable to all, in particular tourists to whom the scheme was originally targeted". However, the pricing structure of this system encouraged a behavior like this, namely the price for renting a bike has been payed only once per day and one can use the bicycle as long as one wants. In most systems one pays for the time the vehicle is used. This encourages the customers to return the car or bike as soon as one reaches the destination. In order to reflect this behavior, we consider an adversary who selects only from those customer requests where the difference between the pickup time and the drop time is bounded from above.

It is easy to find a sequence of customer requests which prove that there cannot exists a competitive deterministic online algorithm on $\mathbb{U}(n)$, $n \geq 2$, against a non-abusive direct δ-adversary with $\delta = 0$. The next result shows that increasing δ to one already enables us to find a competitive online algorithm on the uniform metric space.

Theorem 4.8. *Let the online algorithm* RePlan *send each waiting driver to a station which has a car available for the driver (if possible). Then* RePlan *is* $2\gamma/k$*-competitive for the Online Max-Accept Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $\gamma \geq k$ *against a direct* δ*-adversary with* $\delta \geq 1$*.*

Proof. Let us assume that the adversary serves the maximum of γ customer requests at each time step.

Since we have $\delta \geq 1$ a customer request released at time t can be served whenever a driver is at time t at a station with a free car. Due to the modification of RePlan it is ensured that this is the case in at least every second point of time. Thus, it follows with $\gamma \geq k$ that RePlan serves at least k customer requests every second time step.

From that we compute

$$\frac{\mathsf{ADV}(R)}{\textsc{RePlan}(R)} \leq \frac{2\gamma}{k}$$

proving the stated competitive ratio. □

Note that the modified RePlan becomes 2-competitive in the case that $\gamma = k$ holds. Recall that under similar conditions, we could only prove a competitive ratio of 3 for the Online Min-Reject Relocation Problem (Theorem 4.3).

Furthermore, note that basically the same arguments show the same competitive ratio for the online algorithm EST.

That the modification to RePlan in the previous theorem is indeed necessary can be easily shown. In fact, if the drivers wait at their current station when they cannot serve any released request, one can show that RePlan is not competitive against a direct δ-adversary with $\delta = 1$.

However, for $\delta \geq 2$, the arguments from the proof of the previous theorem can be used to prove competitiveness of RePlan.

Next, we consider a generalized situation, namely when the metric space is the real line and an interval of the real line, respectively.

Before we show the competitive ratio for some algorithms, we give some lower bounds for the competitive ratio. These lower bounds, help us not only to evaluate the competitive online algorithms which we present afterwards, but also to know the limit when we cannot find a competitive online algorithm.

Theorem 4.9. *For any competitive deterministic online algorithm for the Max-Accept Online Relocation Problem* $([-x, x], \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $x \geq 3$ *the competitive ratio against a direct* δ*-adversary with* $\delta \in \mathbb{N}$ *is at least* $1 + (2x-1)/\delta$*. The result holds even in the case that* $\gamma = 2$ *and* $k = 1$*.*

Proof. Let σ be a sequence of customer requests and let $\mathsf{ALG}(\sigma)$ be the number of accepted customer requests of an online algorithm ALG with respect to σ. Furthermore, let $\mathsf{ADV}(\sigma)$ the number of accepted customer requests of the adversary ADV with respect to σ. We show that for any arbitrary $c \leq 1 + (2x-1)/\delta$ there exists a sequence of customer requests σ so that

$$c \cdot \mathsf{ALG}(\sigma) \leq \mathsf{ADV}(\sigma)$$

holds, proving a lower bound for a competitive ratio against ADV for the Max-Accept Online Relocation Problem.

W.l.o.g. let $3x \leq \delta$.

Let there be only one driver and two cars and let them start all in the origin 0. We construct a sequence of customer requests σ as follows. At time 0 the adversary releases δ-customer requests, where each customer request is of the form $\tilde{r}_j = (0, -x, \delta + 2j, -x + 1, \delta + 2j + 1)$, $0 \leq j < \delta$ (see Figure 4.3 for an illustration). Let $\tilde{\sigma} = (\tilde{r}_0, \ldots, \tilde{r}_\delta)$.

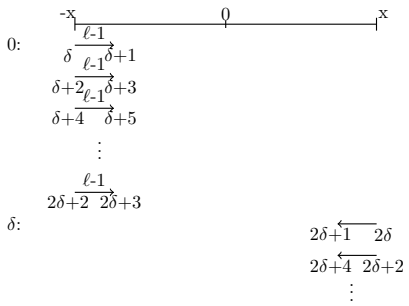


Figure 4.3: This image illustrates the construction of a sequence of customer requests to prove a lower bound of $1 + (2x-1)/\delta$ against a direct δ-adversary with $3x \leq \delta$.

If ALG rejects at least one customer request then the adversaries does not release any more requests and serves all. Otherwise, the adversary transfers one car to x (or $x-1$ if ALG transfers a car to x as well). W.l.o.g. let ALG have no car at x or $x-1$ (otherwise, adjust the following customer requests accordingly). Furthermore, at time δ the adversary releases $2x-1$ customer requests

$$\hat{r}_i = \begin{cases} (\delta, x, 2\delta + i, x - 1, 2\delta + i + 1) & \text{if } i \text{ is even,} \\ (\delta, x - 1, 2\delta + i, x, 2\delta + i + 1) & \text{otherwise,} \end{cases}$$

and let $\hat{\sigma} = \hat{r}_1, \ldots, \hat{r}_{2x-1}$, and let $\sigma = (\tilde{\sigma}, \hat{\sigma})$. Since ADV has positioned a car at x, these requests can all be accepted by ADV. If ALG transfers a car to x, it misses at least $(l-1)$ requests from $\tilde{\sigma}$. This can be seen as follows. In order to move a car to $x-1$ (resp. x) and return to $-x$, at least $4x-2$ time units are needed. Therefore, $2x-1$ customer requests are rejected from $\tilde{\sigma}$. If ALG does not move to x, then all $2x-1$ customer requests from $\hat{\sigma}$ are rejected.

Therefore, ALG can accept at most δ customer requests, and ADV accepts $\delta+2x-1$ customer requests. Thus, we have

$$\frac{\mathsf{ADV}(\sigma)}{\mathsf{ALG}(\sigma)} \geq \frac{\delta+2x-1}{\delta} = 1 + \frac{2x-1}{\delta},$$

proving the statement of the theorem. □

Considering smaller values for δ one can proof some larger lower bounds, e.g., if $\delta \leq 3x-1$, the competitive ratio against a non-abusive direct δ-adversary is at least $4x-\delta-1$. Hereby, the basic idea of previous proof can be reused. In the case that $\delta \leq x-1$ one can even show that there does not exist a competitive deterministic online algorithm against an abusive direct δ-adversary.

Combined with Theorem 4.9 we have a lower bound lb for a competitive ratio on the interval $[-x, x]$ against a direct δ-adversary for the Max-Accept Online Relocation Problem

$$lb = \begin{cases} 4x-\delta-1, & \text{if } \delta \leq 3x-1, \\ 1+\frac{4x-1}{\delta}, & \text{if } 3x \leq \delta. \end{cases}$$

From that one can easily verify that the lower bound is within the range

$$lb \in \begin{cases} [1, x], & \text{if } \delta \leq 3x-1, \\ \left[1, 2\frac{1}{3}\right], & \text{if } 3x \leq \delta. \end{cases}$$

According to Theorem 4.1, there does not exist a competitive online algorithm for the Online Min-Reject Relocation Problem on the uniform metric space against a direct δ-adversary for any $\delta \in \mathbb{N}$ if there are more cars than drivers in the system. However, for the Online Max-Accept Relocation Problem, we can show that there exist competitive online algorithms (for sufficiently large δ).

Next, we consider the algorithm "longest occupation time" LOT (see Algorithm 9). The basic idea of LOT is to select the released but not served customer request with the longest difference between the pickup time and the drop time. When the driver and a car are free, the next customer requests to serve are selected. Obviously, one can expect that this algorithm performs poorly in practice and in theory as well. However, it serves as a good pathological example to illustrate the difference between the Online Min-Reject Relocation Problem and the Online Max-Accept Relocation Problem, since in the latter also algorithms can be competitive which make obviously "bad" decisions.

The algorithm LOT is not competitive against a direct δ-adversary for any δ. This is due to a specific behavior of the algorithm, namely since the algorithm assigns a car to every customer request it is serving and waits until he can serve this request. When there are no unassigned cars, LOT does not serve any other request in between, and the adversary can abuse this behavior by letting the online algorithm assign all of its cars before releasing "its" customer requests. In other words, in contrast to before, the non-competitiveness is not a result of not being able to react in time, but results of having "too much time".

However, we show that LOT is competitive against a direct δ-ϵ-adversary. Before we can prove this, we have to introduce another notion.

Algorithm 9 LOT algorithm for the Online Max-Accept Relocation Problem (longest occupation time)

Input: a sequence of released customer requests σ
Output: a transportation schedule $\mathcal{S}$ for the Online Min-Reject Relocation Problem
1: **when** a driver is not transferring a car to a pickup station **and** a car is not serving a customer request **then**
2: deterministically select r which can be served by non-assigned car with $\check{t} - \hat{t}$ maximal
3: compute an optimal tour for the driver so that r is served and assign r to the used car
4: all drivers without a tour, and all unassigned cars wait at their current station
5: **return** $\mathcal{S}$

Definition 4.10. *Let ALG be an online algorithm (or an adversary), and let $r = (t, v, t_v, v', t_{v'})$ be a customer request. We say that r fills a time interval $[t, T]$ (w.r.t. ALG), with $[t_v, t_{v'}] \subseteq [t, T]$, if r is accepted by ALG and one of the following conditions is true*

- *the interval $[t, T]$ is equal to $[t, t_{v'}]$, i.e., we have $T = t_{v'}$, or*
- *there exists a direct δ-ϵ-customer request $\hat{r} = (\hat{t}, \hat{v}, T-1, \hat{v}', T)$ which is rejected by ALG.*

Let $\sigma = (r_1, \ldots, r_\ell)$ be a sequence of customer requests. We say that σ fills a time interval $[T, T']$ if there exists a partition of $[T, T']$ into ℓ subintervals $[T_1, T_1'], \ldots, [T_\ell, T_\ell']$, so that r_j fills the interval $[T_j, T_j']$, for each $1 \leq j \leq \ell$.

The definition when a sequence or a customer request fills a time interval, depends on the considered online algorithm. Note, the definition only ensures that there exists at least one customer request which is rejected by the online algorithm ALG, but there may also exist several other which may be accepted by ALG. If a customer request r does not fill a time interval $[t, T]$ then it follows that every customer request $\hat{r} = (\hat{t}, \hat{v}, T-1, \hat{v}', T)$ is served by ALG. Therefore, by finding a maximal T so that r fills $[t, T]$ can help in constructing a worst-case sequence for ALG. In the next lemma, we show such a maximal T for LOT.

Lemma 4.11. *Let $x \in \mathbb{N}$ be arbitrarily selected with $x \geq 1$. Let $r = (t, v, t_v, v', t_{v'})$ be a direct δ-ϵ-customer request on $[-x, x]$ with $4x \leq \delta \leq \epsilon$. Then the two statements hold for* LOT

(i) there does not exist a time interval $[t, T]$ with $T > t + \epsilon + 4x$ so that r fills $[t, T]$,

(ii) if r fills $[t, T]$ with $T = t + \epsilon + 4x$, then $t_v - t = \epsilon$ and $t_{v'} - t_v = 2x$ follow.

Proof. "(i)": In order to prove the statement, we show that for every δ-ϵ-customer request r on $[-x, x]$, that r does not fill the time interval $[t, T]$ with $T = t + \epsilon + 4x + 1$. For that we show that every direct δ-ϵ-customer request $\hat{r} = (\hat{t}, \hat{v}, T-1, \hat{v}', T)$ is accepted by LOT. Afterwards, we show that there exists a direct δ-ϵ-customer request r on $[-x, x]$ which fills the interval $[t, T]$ with $T = t + \epsilon + 4x$.

Let $r = (t, v, t_v, v', t_{v'})$ be an arbitrary δ-ϵ-customer request on $[-x, x]$ which is served by LOT. We show that r does not fill the time interval $[t, T]$ with $T = t + \epsilon + 4x + 1$. Then the difference between the release time and the pickup time is at most ϵ. Since r is direct and on the interval $[-x, x]$, it follows that the difference between the pickup and drop time of r is at most $2x$. Thus, it follows that r is finished serving latest at $t + \epsilon + 2x$. Since the pickup time of $\hat{r}$ is $T - 1 = t + \epsilon + 4x$ and $\delta \geq 4x$ it follows that the release time of $\hat{r}$ is less than or equal to $t + \epsilon$. From that it follows that the driver can transfer the car to any position $y \in [-x, x]$, the car reaches y latest at time $t + \epsilon + 4x$. Since the pickup time of $\hat{r}$ is $T - 1 = t + \epsilon + 4x$, it follows

that $\hat{r}$ can be served and is accepted since there is no other released customer request. The customer requests r and $\hat{r}$ are selected arbitrary, and, thus, r cannot fill the time interval $[t, T]$ with $T = t + \epsilon + 4x + 1$.

"(ii)": Firstly, we show that there exists a direct δ-ϵ-customer request on $[-x, x]$ which fills the interval $[t, T]$ with $T = t + \epsilon + 4x$, and $t_v - t = \epsilon$ and $t_{v'} - t_v = 2x$ hold. For that we consider $r = (t, x, t + \epsilon, -x, t + \epsilon + 2x)$. It is easy to see that r is a direct δ-ϵ-customer request on $[-x, x]$. Let r be served by LOT, then $\hat{r} = (T - \delta, x, T - 1, x - 1, T)$ cannot be served by LOT. By following the same steps as above, one can see that the car cannot reach x before time $t + \epsilon + 4x$. However, since the pickup time is $T - 1 = t + \epsilon + 4x - 1$, the car cannot be transferred to x in time and thus, $\hat{r}$ is rejected. Therefore, r fills the time interval $[t, T]$ with $T \leq t + \epsilon + 4x$.

Secondly, we prove that if either $t_v - t < \epsilon$ or $t_{v'} - t_v < 2x$ hold, then r does not fill $[t, T]$ with $T = t + \epsilon + 4x$. It holds for the drop time of r

$$t_{v'} < t + (t_v - t) + d(v, v') \leq t + \epsilon + 2x.$$

Since all values are integers it follows

$$t_{v'} \leq t + \epsilon + 2x - 1.$$

Next, we show that every direct δ-ϵ-customer request $\hat{r} = (\hat{t}, \hat{v}, T - 1, \hat{v}', T)$ is accepted by LOT. For the pickup time of $\hat{r}$ we have, $T - 1 = t + \epsilon + 4x - 1$. Thus, the difference between the drop time of r and the pickup time of $\hat{r}$ is at most $2x$, and the car can be transferred to any arbitrary position in $[-x, x]$ within $2x$ time units. Therefore, $\hat{r}$ is accepted by LOT and r does not fill $[-x, x]$, and the statement is proved. □

With the help of the previous lemma, we next prove a competitive ratio for the online algorithm LOT for the Online Max-Accept Relocation Problem on the interval $[-x, x]$ against a direct δ-ϵ-adversary for all $4x \leq \delta \leq \epsilon$. Hereby, we construct a sequence of customer requests, and prove that it is a worst-case sequence for LOT. From that we then conclude the competitive ratio $(4x + \epsilon)$.

Theorem 4.12. *If $kL \geq \gamma$ then the online algorithm* LOT *is $(4x+\epsilon)$-competitive for the Online Max-Accept Relocation Problem $([-x, x], \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ with $x \geq 1$ against a direct δ-ϵ-adversary for all $4x \leq \delta \leq \epsilon$. This result holds even in the case that $\gamma = k = 1$.*

Proof. Since $\delta \geq 4x$ it follows that a newly released customer request can be served (unless all cars are currently serving a customer request), independently of the position of the driver and the cars. Therefore, it is ensured that LOT can serve at least one customer request.

Let there be only one driver and two cars and let them start all in the origin 0. In the following, we fix a time horizon $[0, T]$ and construct two sequences σ_1^* and σ_2^* for this time horizon. Hereby, each sequence σ_j^* fills the time horizon $[0, T]$. Afterwards, we show that these sequences are a best-case sequence for ADV and a worst-case sequence for LOT, i.e., for any other sequence σ filling this time horizon it follows

$$\mathsf{ADV}(\sigma_1^*) \geq \mathsf{ADV}(\sigma) \text{ and } \mathrm{LOT}(\sigma_2^*) \leq \mathrm{LOT}(\sigma).$$

Finally, these two sequences are "merged" to one sequence σ^*, which then lead to

$$\frac{\mathsf{ADV}(\sigma^*)}{\mathrm{LOT}(\sigma^*)} \geq \frac{\mathsf{ADV}(\sigma)}{\mathrm{LOT}(\sigma)} \tag{4.5}$$

for any σ which fills the time horizon $[0, T]$. Thus, it follows from Equation (4.5) we can compute the competitive ratio by using σ^* as a representative.

Firstly, we define the customer requests of $\sigma_1^* = (r_{10}, r_{11}, \ldots, r_{1\ell})$, and show that they fill a given time horizon. The customer requests of σ_1^* are defined as follows

$$r_{1j} = \begin{cases} (j, x, \delta + j, x - 1, \delta + j + 1), & \text{if } i \text{ is even,} \\ (j, x - 1, \delta + j, x, \delta + j + 1), & \text{if } i \text{ is odd,} \end{cases}$$

and the last customer request of σ_1^* has as drop time T. The customer requests r_{1j} are defined in such a way that they allow the adversary to serve all customer requests without moving the car. Furthermore, starting from time δ, the adversary can serve at each time step one customer request. Since there is only one car in the system, this value cannot be increased. Therefore, any arbitrary customer request which is added to the sequence will be rejected by the adversary, showing that the sequence fills the time horizon $[0, T]$. For σ_1^* we have $\mathsf{ADV}(\sigma_1^*) = T - \delta$.

Secondly, we consider the sequence $\sigma_2^* = (r_{20}, r_{21}, \ldots, r_{2\ell'})$. Hereby, let us assume that T is a multiple of $\epsilon + 4x$. The customer requests of σ_1^* are defined by

$$r_{2j} = (j(\epsilon + 4x), x, \epsilon + j(\epsilon + 4x), x, \epsilon + j(\epsilon + 4x) + 2x).$$

One can easily verify that these customer requests are indeed direct δ-ϵ-customer requests. Furthermore, we have

$$\check{t}(r_{2i}) + 2x = \epsilon + j(\epsilon + 4x) + 4x = (j + 1)(\epsilon + 4x) = t^{rel}(r_{2i+1}) \tag{4.6}$$

which means that the customer request r_{2i+1} is released $2x$ time units after the drop time of the customer request r_{2i}. Additionally, let ℓ' be chosen so that the drop time of $r_{2\ell'}$ is T (this is possible since T is a multiple of $\epsilon + 4x$ by assumption).

The difference between the drop time of r_{2i} and the pickup time of r_{2i+1} is

$$\begin{aligned} \epsilon + (j + 1)(\epsilon + 4x) - (\epsilon + j(\epsilon + 4x) + 2x) &= \epsilon + (j + 1)(\epsilon + 4x) - \epsilon - j(\epsilon + 4x) - 2x \\ &= \epsilon + 2x \\ &\geq \delta + 2x \\ &\geq 6x, \end{aligned}$$

and, therefore, every customer request of σ_2^* can be served by LOT.

From Lemma 4.11 (i) it follows that the sequence σ_2^* fills the time horizon $[0, T]$. Furthermore, one can conclude from Lemma 4.11 (ii), that $\mathrm{LOT}(\sigma_2^*) \leq \mathrm{LOT}(\sigma)$ holds for any arbitrary sequence of direct δ-ϵ-customer requests σ which fills $[0, T]$.

Finally, note that if σ does not fill $[0, T]$, then a customer request with a difference between the pickup and drop time of 1 can be added to σ. Furthermore, these added customer requests are all served by LOT (and by ADV). Hereby, it follows from above that after adding customer request until σ fills $[0, T]$, we have $|\sigma_2^*| \leq |\sigma|$.

For σ_2^* we can conclude $\mathrm{LOT}(\sigma_2^*) = \ell'$, with $T = \ell'(\epsilon + 4x)$.

Thirdly, we "merge" the two sequences σ_1^* and σ_2^*, i.e., we consider $\sigma^* = \sigma_1^* \cup \sigma_2^*$. From above it follows that Equation (4.5) holds, and that we can compute the competitive ratio by using σ^* as a representative. Furthermore, we already showed that any additional customer request added to σ_1^* is rejected by ADV, and any customer request added to σ_2^* is rejected by LOT. Thus, with $T = \ell(\epsilon + 4x)$, it follows

$$ADV(\sigma^*) = T - \delta = \ell(\epsilon + 4x) - \delta$$

and

$$\mathrm{LOT}(\sigma^*) = \ell.$$

Therefore, we have

$$\begin{aligned}\frac{\mathsf{ADV}(\sigma^*)}{\mathrm{LOT}(\sigma^*)} &= \frac{\ell(\epsilon+4x)-\delta}{\ell} \\ &= \epsilon + 4x - \frac{\delta}{\ell} \\ &\leq \epsilon + 4x\end{aligned}$$

and with (4.5) this proves the stated competitive ratio. □

An obvious improvement of LOT is to select the released but not served customer request with the shortest difference between the pickup time and the drop time. However, one can show that their competitive ratios differ only by $\delta + 1$, i.e., this improved version of LOT is $(4x + \epsilon - \delta - 1)$-competitive for the Online Max-Accept Relocation Problem on the interval $[-x, x]$ with $x \geq 1$ against a direct δ-ϵ-adversary for all $4x \leq \delta \leq \epsilon$.

Finally, we consider a more serious online algorithm for solving the Online Max-Accept Relocation Problem, namely the algorithm RePlan, and give a competitive ratio for this online algorithm.

Theorem 4.13. *The online algorithm* RePlan *is* $(4x\gamma/k)$*-competitive for the Online Max-Accept Relocation Problem* $([-x, x], \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $x \geq 1$ *and* $\gamma \geq k$ *against a direct* δ*-adversary for all* $4x \leq \delta$*. This result holds even in the case that* $\gamma = k = 1$*.*

Proof. Since we consider the interval $[-x, x]$, the time needed to serve a customer request is at most $4x$ (at most $2x$ time units are needed to move the driver to a station with a car, and another $2x$ time units are needed to move to the pickup station of the customer request). If the driver follows the car picked up, it is possible to serve a customer request every $4x$ time step, since the adversary only releases direct customer request. Thus, from $\delta \geq 4x$ it follows that every newly released customer request can also be served. Furthermore, the algorithm can decide until the pickup time of a customer request whether to serve the request or to reject it.

Assuming the adversary serves γ request every time step, and every driver of RePlan only one every $4x$ time steps, we get

$$\frac{\mathsf{ADV}(R)}{\textsc{RePlan}(R)} \leq \frac{4x\gamma}{k}$$

proving the stated competitive ratio. □

The arguments in the proof of the previous theorem can be applied to weighted graphs $G = (V, E, w)$. Hereby, one can show that RePlan is $(2x\gamma/k)$-competitive against a direct δ-adversary with $\delta \geq 2x$, where $x = \mathrm{diam}(G)$. Note that the conditions of the previous theorem deal with the "radius" and not the diameter of G resulting in a competitive value twice the value for graphs. Furthermore, note that the arguments also apply for the online algorithm EST.

We applied a fairly simple analysis on the online algorithm RePlan, resulting in probably not the best possible competitive ratio. To the best of our knowledge, a good lower bound for the Online Max-Accept Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \mathrm{cap}, L, R)$ is currently unknown (and, especially for larger δ, the lower bound given in Theorem 4.9 is far below the proven competitive ratio).

4.2.2 Max/Max Ratio

At the end of this section, we briefly consider the max/max ratio on the Online Max-Accept Relocation Problem and show that there does not exist meaningful ratios for this online problem. Hereby, we directly concentrate on the uniform metric space. After considering general customer requests, we directly focus on direct δ-customer requests.

In the first theorem, we consider general customer requests and show that every online algorithm has an undeterminate max/max ratio. This result is also true if only direct customer requests are considered.

Theorem 4.14. *For the Online Min-Reject Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 2$*, any online algorithm* ALG *has an indeterminate max/max ratio. This result also holds if only direct customer requests are considered.*

Proof. Let σ be a sequence of customer requests. We denote by $\mathsf{ALG}(\sigma)$ be the number of accepted customer requests of an online algorithm ALG with respect to σ. Furthermore, by $\mathsf{OPT}(\sigma)$ the number of accepted customer requests of an optimal offline solution OPT with respect to σ.

We show that for every $\lambda > 0$ there exists a sequence of customer requests σ with $\lambda = |\sigma|$ and so that $\mathsf{OPT}(\sigma) = 0$ holds. Since $\mathsf{ALG}(\sigma) \leq \mathsf{OPT}(\sigma)$ holds for all σ, it follows $\mathsf{ALG}(\sigma) = 0$. Therefore, it follows by the definition of the max/max ratio

$$w_M(\mathsf{ALG}) = \limsup_{\lambda \to \infty} \frac{M_\lambda(\mathsf{ALG})}{M_\lambda(\mathsf{ALG})} = \frac{0}{0},$$

showing that the value is indeterminate.

Let the metric space contain the two points v_1 and v_2. Let the cars all be positioned at v_1 and let $\lambda > 0$ be arbitrary. Then we consider the sequence $\sigma = (r_1, \ldots, r_\lambda)$ with

$$r_j = (0, v_2, 0, v_1, 1)$$

for all $1 \leq j \leq \lambda$.

Since all cars are positioned at v_1, also any optimal offline algorithm cannot serve any of these customer requests. Thus, we have $\mathsf{OPT}(\sigma) = 0$ and the statement follows. □

As in previous section, the max/max ratio also does not give a meaningful insight to the Online Max-Accept Relocation Problem. Therefore, in the next and final result for this section, we directly consider the situation when only direct δ-customer requests are released.

Theorem 4.15. *For the Online Max-Accept Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 2$*, with only direct* δ*-customer requests with* $\delta \geq 1$*, any online algorithm* ALG *that does not unnecessarily reject any customer request has a max/max ratio of* 1*.*

Proof. Let σ be a sequence of customer requests. We denote by $\mathsf{ALG}(\sigma)$ be the number of accepted customer requests of an online algorithm ALG with respect to σ. Furthermore, by $\mathsf{OPT}(\sigma)$ the number of accepted customer requests of an optimal offline solution OPT with respect to σ.

Let the metric space contain the two points v_1 and v_2. In the case that $\delta \geq 2$ holds, the cars can be transferred to any station. Therefore, let all γ cars be the positioned at v_1. Then we consider the following sequence of customer requests $\sigma = (r_1, \ldots, r_\lambda)$ with $\lambda \geq \gamma$ and

$$r_j = (0, v_1, \delta, v_2, \delta + 1)$$

for all $1 \leq j \leq \lambda$.

Since $\delta \geq 1$, there is always enough time to transfer the cars to any arbitrary station, and thus, this proves that σ is a worst-case sequence for OPT and for ALG. Obviously, at most γ customer requests can be served by an optimal offline solution. Since ALG does not reject any customer request if it can serve it, ALG also accepts γ customer requests.

Finally, it follows by the definition of the max/max ratio

$$w_M(\mathsf{ALG}) = \limsup_{\lambda \to \infty} \frac{M_\lambda(\mathsf{ALG})}{M_\lambda(\mathsf{ALG})} = \frac{\gamma}{\gamma} = 1,$$

showing that the max/max value for ALG is 1. □

Chapter 5

Optimization Problems

In this chapter, we focus on dynamic optimization problems. Hereby, the aim is to minimize the total tour length of the transportation schedule or so that the waiting times for the customers is minimized.

Usually, not every customer books its request in advance but may ask for a car when they want one. In this case, the customer cannot expect a free car at the pickup station or a free parking place at the destination, and the customer has to wait for a car to be delivered (resp. a car to be moved to free a parking place). In Section 5.1, we aim at minimizing the waiting times for the customers. However, we show that there cannot exist a competitive online algorithm even in very restricted situations.

Afterwards, in Section 5.2, we first show that aiming at minimizing the total tour length of a transportation schedule is meaningless. Therefore, we focus on a mixture of an optimization and a decision problem, where we aim at minimizing the total tour lengths of the transportation schedules while having to decide which customer can be served and which request has to be rejected. In contrast to the previous section, customers book their request in advance and leave the system if their request cannot be served, resulting in some losses (financial and reputation) for the company.

5.1 Minimizing the Waiting Time

In this section, we consider the situation when customers spontaneously arrive at a station to take a car without booking their car in advance. When a customer cannot immediately take a car from the pickup station to the drop station, he waits at the pickup station until his request is served. In order to ensure a certain quality of the service, the objective is to minimize the waiting times. Hereby, we consider two different objective functions: minimize the total waiting time and minimize the maximal waiting time. In particular the second objective function ensures that no customer has to wait longer than a certain amount of time at a station until its request can be served.

The waiting time τ_j of a floating customer request $r_j = (t_j, v_j, w_j, \delta_j)$ is the difference between the release time and the time when the customer can pickup its car.

The Online Min-Wait Relocation Problem and the Online Max-Wait Relocation Problem $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \text{cap}, L, R)$ are Dynamic Relocation Problems. The output of the Online Min-Wait Relocation Problem is a transportation schedule, with a minimal total waiting time for the

sequence of floating customer requests $R = \{r_1, \ldots, r_\lambda\}$, i.e., where the objective function

$$\sum_{j=1}^{\lambda} \tau_j - t_j,$$

is minimal, while the Online Max-Wait Relocation Problem aims at computing a transportation schedule where the maximal waiting time is minimized, i.e., where

$$\max_{j=1,\ldots,\lambda} \{\tau_j - t_j\},$$

is minimal.

Note, in this section, only the waiting times are considered, but the costs for transferring cars between stations are not taken into account.

This section gives the proves for the results already presented in [94].

5.1.1 Competitive Analysis

In this section, we show that there does not exist a competitive online algorithm neither for the Online Min-Wait Relocation Problem nor for the Online Max-Wait Relocation Problem even in the case when there are several restrictions to the problem: a restricted metric space, and weaker adversaries than the oblivious adversary. For that, we show that there does not exists a competitive online algorithm for the Online Max-Wait Relocation Problem. Afterwards, we can directly conclude this negative result for the Online Min-Wait Relocation Problem.

Theorem 5.1. *There is no competitive deterministic online algorithm for the Online Max-Wait Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 3$ *against a direct non-abusive adversary. The result holds even in the case that* $\gamma = k = 1$.

Proof. Let ALG be an online algorithm and let ADV be a non-abusive adversary. W.l.o.g. let $L = 1$, otherwise, i.e., if $L > 1$, the adversary releases in the following instead of one floating customer request, L floating customer requests.

In order to prove the statement, we show that for every $c \in \mathbb{N}$ there exists a sequence of floating customer requests σ so that

$$\mathsf{ALG}(\sigma) \geq c \quad \text{and} \quad \mathsf{ADV}(\sigma) \leq 3. \tag{5.1}$$

In other words, we show that the maximal waiting time for the online algorithm grows, while the maximal waiting time remains equal for the adversary.

The adversary releases the floating customer requests in several phases. Hereby, we show that the maximal waiting time for ALG increases by one in every phase while the maximal waiting time for ADV is at most 3. More precisely, we show that in phase j there are j released floating customer requests which have not been served by ALG. Since there is only one driver and one car, at every point in time at most one floating customer request can be served and, thus, the number of released but not served floating customer requests gives a lower bound for the maximal waiting time.

We start by describing the initial phase (i.e., phase 1). Hereby, the driver and car from the adversary and from the online algorithm start from the same station. At the end of the initial phase, the driver and car of ADV and ALG are positioned at different stations. Afterwards, we continue with phase j, where the driver and car from the adversary and from the online algorithm start from different stations. Furthermore, we start every phase (except for phase 1) with two

floating customer requests which are released but not yet served by ADV. In this phase, we show that the online algorithm has to make an additional move which can be avoided by the adversary. Due to this additional move, the maximal waiting time (resp. the number of released but not yet served floating customer requests) increases by 1. At the end of each phase, the driver and car of ADV and ALG are ensured to be positioned at different stations.

Let $V = \{v_1, v_2, v_3\}$ be the set of stations.

Phase 1: W.l.o.g. let the car and driver of ADV and of ALG be located in v_1. At time 0, the adversary starts releasing floating customer requests $r = (0, v_2, v_3, 1)$ and afterwards $r' = (1, v_3, v_2, 1)$. For an illustration see Figure 5.1.

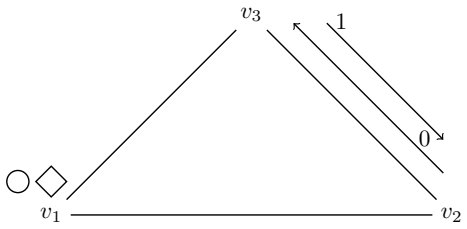


Figure 5.1: This figure illustrates the positions of the car of the oblivious adversary (circle) and the car of an online algorithm (diamond) at time 0. Furthermore, the released floating customer requests are illustrated as arcs. The number close to the start node of an arc corresponds to the release time of the floating customer request. All floating customer requests are direct, and therefore, the occupation time is 1 for every request.

Either r or r' is served first by ALG. The adversary serves the requests in the opposite order. Therefore, the car and driver of ALG are positioned at time 1 at v_2 and of ADV at v_3 (or vice versa).

This ends phase 1 at time 1. Both, the adversary and the online algorithm have two released but not yet served floating customer requests between v_2 and v_3. Furthermore, the position of the driver and of the car of ADV and of ALG are at different stations.

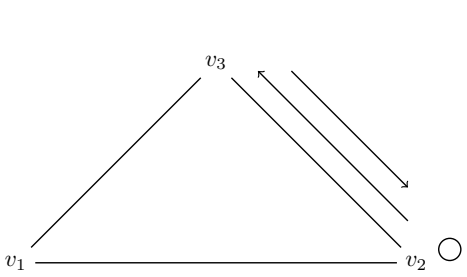
Phase j: Let the phase j start at time t. W.l.o.g. let the car and driver of ADV be located at v_2. Let there be two released but not yet served floating customer requests σ_j^{ADV} between v_2 and v_3 for ADV (one pickup station is v_2 the other is v_3), and (at least) j released but not yet served floating customer requests σ_j^{ALG} between v_2 and v_3 for ALG. If the number of released but not yet served floating customer requests for ALG is even, then let the position of the driver and of the car of ADV and of ALG be at different stations (w.l.o.g. at v_3). Otherwise, let them be located at the same station for both, the adversary and the online algorithm (i.e., both are at v_2).

Due to the position of the driver and car of ALG and due the number of requests in σ_j^{ALG}, at the time when ALG has finished serving all requests from σ_j^{ALG}, the car of ALG is located in v_3. However, the car of ADV is positioned at v_2 after serving all floating customer requests from σ_j^{ADV}.

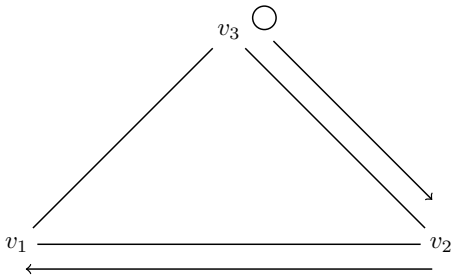
W.l.o.g. let t be odd. From time $t+1$, the adversary continues to release requests at integer points in time t' in the following form

$$r_{t'} = \begin{cases} (t', v_2, v_1, 1) & \text{if } t' \text{ is even,} \\ (t', v_1, v_2, 1) & \text{if } t' \text{ is odd.} \end{cases}$$

If t is even, the two cases are swapped. Obviously, the adversary does not need to transfer any car in order to serve these requests (see Figure 5.2 for an illustration). The adversary releases the floating customer requests $r_{t'}$ until ALG has served all requests from σ_j^{ALG}.



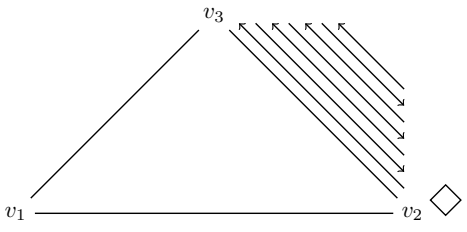
(a) This figure illustrates the positions of the car of the oblivious adversary at time t, i.e., one time step before releasing a new form of floating customer requests.



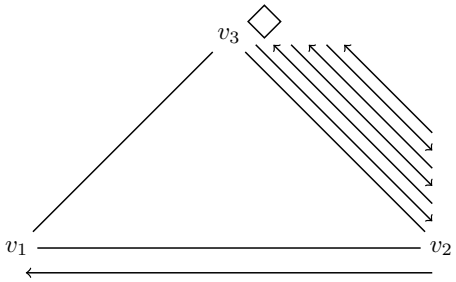
(b) At time $t+1$ the adversary releases requests between v_1 and v_2. Since at time $t+2$, the car of the adversary is positioned at v_2, he can directly start serving these requests.

Figure 5.2: This figure illustrates the positions of the car of the adversary (circle) at time t and at time $t+1$. Furthermore, the released floating customer request are illustrated as arcs.

Since the car of ALG is, at the time he finished serving all requests from σ_j^{ALG}, positioned at v_3, he ALG must transfer the car to v_2 (or v_1). Thus, it follows that the number of released but not yet served floating customer requests increases by 1 (see Figure 5.3 for an illustration).



(a) This figure illustrates the positions of the car of the online algorithm at time t, i.e., one time step before releasing a new form of floating customer requests.

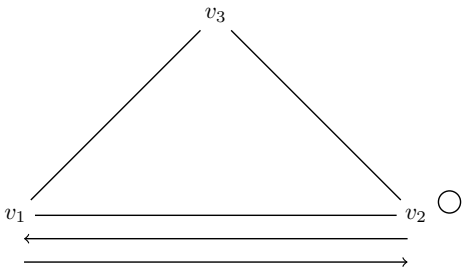


(b) At time $t+1$ the adversary starts releasing requests between v_1 and v_2. Since at the time when the online algorithm finished the floating customer requests between v_1 and v_3, the car is positioned at v_3, and therefore, the online algorithm has to transfer the car in order to serve the newly released requests.

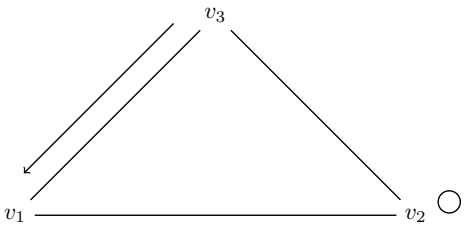
Figure 5.3: This figure illustrates the positions of the car of the online algorithm (diamond) at time t and at time $t+1$. Furthermore, the released floating customer request are illustrated as arcs.

Let the car of ALG arrive at time t'' at v_2 (if the car arrives at v_1, this phase is over and the next phase starts). Then, when the number of released floating customer requests is even,

the adversary stops releasing the floating customer requests $r_{t'}$ and continues releasing floating customer request between v_1 and v_3 after a pause of one time unit (we denote these floating customer requests by $r_{t''}$). Since the number of released floating customer requests is even, it is ensured that the car of ADV ends at v_2. Thus, the adversary has to transfer the car to v_1 or v_3, and so has ALG when he has finished serving the requests $r_{t'}$ (see Figure 5.4).



(a) This figure illustrates the positions of the car of the oblivious adversary at time t''.



(b) At time $t+1$ no floating customer request has been released by the adversary. However, at time $t''+2$ he releases requests between v_1 and v_3. Since at time $t+2$, the car of the adversary is positioned at v_2, he must transfer the car to v_2 before he can start serving the newly released requests.

Figure 5.4: This figure illustrates the positions of the car of the adversary (circle) at time t'' and at time $t+2$. Furthermore, released floating customer request are illustrated as arcs.

Since the adversary knows to which station the online algorithm will transfer his car, after serving the floating customer requests $r_{t'}$, he can position his car so that the conditions stated at the beginning of this phase are fulfilled for the next phase and for the floating customer requests $r_{t''}$. The phase ends when the online algorithm starts serving the requests of the form $r_{t''}$.

Furthermore, there is one time unit when the adversary does not release any new request and so the number of served but not yet released floating customer requests does not increase for the adversary (and for the online algorithm in this case as well). Therefore, and since the number of released but not served floating customer requests for ALG is increased by 1, and the car of ALG and of ADV are positioned in different (resp. the same) stations, the conditions stated at the beginning of phase j are fulfilled at the end of phase j for phase $j+1$. Thus, the next phase can start.

The number of phases can be arbitrarily high, and at each phase the number of released but not yet served floating customer requests increases by 1 for ALG but remains below 3 for ADV, the equations (5.1) hold, proving the statement for deterministic online algorithms.

Finally, we prove the statement for randomized online algorithms. For that we can assume that whenever the algorithm has to make a decision which request has to be served first is equally distributed. Otherwise, the adversary adjusts its probability distribution accordingly. Therefore, at the end of a phase, the probability that the randomized online algorithm has positioned the car at the same station as the adversary is 0.5. Whenever the online algorithm has positioned the car at the same station as the adversary, the number of released but not yet served floating customer requests is not increased. However, when the online algorithm positions the car at a different station, the number of released but not yet served floating customer requests is increased by 1 in the next phase.

From that it follows that the expected value is increased by 1/2 in each phase. Let σ_j be the sequence constructed until the jth-phase. Then it follows

$$\mathbb{E}(\mathsf{ALG}(\sigma_j)) = \frac{j}{2}.$$

Since the expected value for the adversary remains constant (as the maximal waiting time remains less or equal to 3), there cannot exist a $c \in \mathbb{N}$ so that

$$c \cdot \mathbb{E}[\mathsf{ADV}(\sigma)] \leq \mathbb{E}[\mathsf{ALG}(\sigma)].$$

holds, which proves that the statement of this theorem holds even for randomized online algorithms. □

When the maximal waiting time increases, it follows directly that the total waiting time also increases. Thus, we can directly conclude the next negative result from the previous theorem.

Corollary 5.2. *There is no competitive deterministic or randomized online algorithm for the Online Min-Wait Relocation Problem* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R)$ *with* $n \geq 3$ *against a non-abusive direct adversary. The result holds even in the case that* $\gamma = k = 1$.

5.1.2 Max/Max Ratio

Finally, we briefly summarize the results for the max/max ratio, which show that also the max/max ratio does not lead to meaningful results neither for the Online Max-Wait Relocation Problem nor for the Online Min-Wait Relocation Problem.

In both problems, every online algorithm ALG has an undeterminate max/max ratio in the general case, i.e., if any floating customer request can be released, and ALG has a ratio of 1 if only direct floating customer requests are considered. However, for the latter one must assume that ALG does not unnecessarily wait before serving a request.

5.2 Minimizing the Total Tour Length

In this section, we consider the situation when customers book their cars in advance. Hereby, it may be possible that not every request can be served, and it may be necessary to reject some requests. An online algorithm is allowed to wait until the pickup time before deciding whether the customer request is accepted or rejected. The objective is to minimize the total tour length while rejecting as few customer requests as possible. This is insofar different to the Online Min-Reject Relocation Problem as the total tour length is taken into account.

For that, we consider an online problem with a penalty function: the Online Min-Distance Relocation Problem with Penalty $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, \mathrm{cap}, L, R, p)$, where a penalty function $p : R \to \mathbb{R}_0^+$ is given as additional input. The output is a transportation schedule $\mathcal{S} = (\Gamma_1, \ldots, \Gamma_k)$, where the objective function is to minimize the total tour length and the costs induced by the penalty function for not serving a customer request,

$$\min_{\Gamma \in \mathcal{S}} \ell(\Gamma) + \sum_{r \in R^-_{\mathsf{ALG}}} p(r),$$

where $R^-_{\mathsf{ALG}} \subseteq R$ is the set of all rejected customer requests by the online algorithm ALG.

5.2.1 Competitive Analysis

In the special of the Online Min-Distance Relocation Problem with Penalty where the penalty function always returns a value of 0 for each rejected customer request, the online algorithm AIP achieves a competitive ratio of 1. This is due to the fact that any customer request can be rejected without any consequence and, thus, the value of the objective function is never increased but remains 0. Since, in this context, many online algorithms achieve a competitive ratio of 1, we assume for the rest of this section that $p(r) > 0$ for every customer request r.

In this section, we show that there exist competitive online algorithms in some restricted cases. Hereby, the penalty function must be bounded from above, as the next result suggests.

Theorem 5.3. *If $n \geq 4$, $\gamma \geq 2$ and $k \geq 1$ holds, then there does not exist a deterministic competitive online algorithm for the Online Min-Distance Relocation Problem with Penalty $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R, p)$ with an unbounded penalty function p against a direct δ-adversary for any $\delta \in \mathbb{N}$.*

Proof. W.l.o.g. let $\delta \geq n + 1$. Otherwise, we consider $\delta' = n + 1$ instead.

Let the uniform metric space contain the four nodes v_1, v_2, v_3, v_4. Furthermore, let there be one driver and two cars in the system, all starting in v_1. Let ALG be a customer request.

We construct a sequence of customer requests in phases. Hereby, we consider the penalty function $p(r) = j$ for all customer requests which are released in phase j. At the beginning of each phase, the driver and the two cars are located in v_1.

Phase j: Let the phase start at time t. Then the adversary releases $2\delta + 1$ customer requests of the following form

$$r_i = (t + 2i, v_1, t + \delta + 2i, v_2, t + \delta + 2i + 1),$$

for all $0 \leq i \leq 2\delta$. All of these customer requests can be served by ALG by using one car only. At time $t + \delta + 2$, the second car cannot be at v_3 and at v_4. If one car is at v_3 and the other at v_4 then at least one customer request has already been rejected in this phase. W.l.o.g. let the second car not be at v_3. Then the adversary releases a customer request

$$r = (t + \delta + 2, v_3, t + 2\delta + 2, v_4, t + 2\delta + 3).$$

Since there is no car at v_3, and the time needed to serve r and $r_{\lceil \delta/2 \rceil}$ is at least 3. Therefore, ALG has to reject at least one customer request, resulting in $\mathsf{ALG}(R_j) \geq j$ for $R_j = (r_1, \ldots, r_{2\delta}, r)$.

The adversary however, can move a car to v_3 before time $t + \delta$ and, therefore, ADV can serve all customer requests. This results in $\mathsf{ADV}(R_j) = 4\delta + 2$.

In total we have after $\mathcal{K}$ phases, the following ratio for $R = R_1 \cup \cdots \cup R_{\mathcal{K}}$

$$\frac{\mathsf{ALG}(\sigma)}{\mathsf{ADV}(\sigma)} \geq \frac{\sum_{j=1}^{\mathcal{K}} j}{\mathcal{K}(4\delta + 2)} = \frac{\mathcal{K}^2 + \mathcal{K}}{2\mathcal{K}(4\delta + 2)} = \frac{\mathcal{K} + 1}{8\delta + 4}.$$

Since $\mathcal{K}$ can be selected arbitrarily large, it follows that there does not exist a $c \in \mathbb{N}$ with $c \cdot \mathsf{ADV}(R) \geq \mathsf{ALG}(R)$, proving the statement. □

The previous result indicates that it is necessary to bound the penalty function. However, even if the penalty function is bounded, one still has to restrict the power of the adversary in order to prove competitiveness for an online algorithm. Furthermore, not every online algorithm is competitive also in restricted cases as the next theorem shows.

Theorem 5.4. *The online algorithm* RePlan *is not competitive for the Online Min-Distance Relocation Problem with Penalty $(M, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R, p)$ with a bounded penalty function $0 < p(r) \leq \alpha$ for all customer requests $r \in R$ against a non-abusive direct δ-adversary ADV for all $\delta \in \mathbb{N}$, if $n \geq 3$, $\gamma \geq 2$ and $k = 1$.*

Proof. We show that for any $c \in \mathbb{R}$ there exists a sequence of customer requests $\sigma = (r_1, \ldots, r_\lambda)$ so that

$$c \cdot \mathsf{ADV}(\sigma) \leq \textsc{RePlan}(\sigma)$$

holds for every online algorithm RePlan, where $\mathsf{ADV}(\sigma)$ is the solution value of a solution computed by the adversary ADV and $\textsc{RePlan}(\sigma)$ is the solution value of a solution computed by RePlan.

W.l.o.g. let $\delta \geq 4$. Otherwise, we consider $\delta' = 4$ instead. Furthermore, let M contain the nodes $\{v_1, v_2, v_3\}$, and let the driver and the cars all start in v_1.

Then the adversary releases two customer requests at time 0

$$r_1 = (0, v_2, \delta, v_3, \delta + 1) \text{ and } r_2 = (0, v_3, \delta + 1, v_2, \delta + 2).$$

The online algorithm RePlan transfers one car to v_2, and so does ADV.

At time δ, the adversary the adversary releases two further customer requests

$$r_3 = (\delta, v_3, 2\delta, v_2, 2\delta + 1) \text{ and } r_4 = (\delta, v_2, 2\delta + 1, v_3, 2\delta + 2).$$

The driver controlled by RePlan is at time δ at v_2. Furthermore, a car is at v_2. Therefore, the optimal tour to for the driver is to transfer the car from v_2 to v_3. On the other hand, ADV transfers the other car from v_1 to v_3.

Until now, we have for $\sigma' = (r_1, r_2, r_3, r_4)$

$$\textsc{RePlan}(\sigma') = 2 \text{ and } \mathsf{ADV}(\sigma') = 3.$$

However, ADV repeats releasing customer requests as r_1, r_2, r_3, r_4 (with a proper shift of the release time as well as the pickup and drop time), the adversary does not move the driver, while RePlan moves the driver whenever another subsequence is released. Since this procedure can be repeated $2c$ times, we have for $\sigma = (r_1, \ldots, r_{8c})$

$$\textsc{RePlan}(\sigma) = 4c \text{ and } \mathsf{ADV}(\sigma) = 3,$$

and thus,

$$\frac{\mathsf{ADV}(\sigma)}{\textsc{RePlan}(\sigma)} = \frac{3}{4c} \leq \frac{4}{4c} = \frac{1}{c}$$

proving the statement. □

Note, the sequence in the proof of previous theorem can also be used to show that several other standard online algorithm, e.g., FcFs or Ignore, are not competitive.

Although, the standard RePlan is not competitive, we prove in the next theorem that applying only a small change results in a competitive online algorithm MarkRePlan (see Algorithm 10). Hereby, we mark each car that has been used by a customer and only move those cars by a driver which are not marked. One can easily see that the transportation schedule computed by MarkRePlan on the sequence from the proof of Theorem 5.4 results in the same transportation schedule as the non-abusive adversary computes.

However, before we can prove a competitive ratio for MarkRePlan, we have to define the notion of "essential stations". Let σ be a sequence of customer requests, and let γ be the number of cars in the system. A vector $\boldsymbol{z}^{[t,T]} \in \mathbb{N}^{|V|}$ with $\left|\boldsymbol{z}^{[t,T]}\right| \leq \gamma$ is called <u>essential</u> for the time-interval $[t, T]$ and for σ, if all $r \in \sigma$ with a pickup time $\hat{t}(r) \in [t, T]$ are served without moving a driver by every feasible system state $\boldsymbol{z}^t$ with $\boldsymbol{z}^{[t,T]} \leq \boldsymbol{z}^t$ (component wise, i.e., $z_v^{[t,T]} \leq z_v^t$ for every $v \in V$). A station $v \in V$ is called <u>essential</u> for the time-interval $[t, T]$ and for σ if there

Algorithm 10 MarkRePlan algorithm (replan marking algorithm)

Input: a sequence of released customer requests R
Output: a transportation schedule $\mathcal{S}$ with a minimal total tour length

```
1: when car serves a customer request then
2: |  mark that car
3: |  if all cars are marked then
4: |  |  remove mark from all cars
5: when new customer request is released then
6: |  compute a new optimal transportation schedule S based on R only moving unmarked
   |      cars
7: |  hereby, cars are moved in the last moment
8: |  drivers not needed to serve any request wait at their current station
9: return S
```

must be at least one car at v at time t, i.e., if $0 < z_v^{[t,T]}$ holds, and a station $v' \in V$ with $0 = z_v^{[t,T]}$ is called non-essential.

Note that the online algorithm RePlan transfers one car from one essential station to another. One of the basic ideas of MarkRePlan is to transfer cars to essential stations and once a car is positioned in an essential station, the goal is to keep the cars in it.

Theorem 5.5. *The online algorithm* MarkRePlan *for the Online Min-Distance Relocation Problem with Penalty* $(\mathbb{U}(n), \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, cap, L, R, p)$ *with a penalty function* $1 \leq p(r) \leq \alpha$ *for all customer requests* $r \in R$ *for an* $\alpha \in \mathbb{R}$ *is at most* $8\gamma + \alpha(4\gamma^2 - 2\gamma) + 1$ *competitive against a* δ*-adversary with* $\delta \geq n + 1$.

Proof. Firstly, we prove that, if at a point in time MarkRePlan and the adversary both have all cars in the same stations, then MarkRePlan moves a driver or rejects a request only if the adversary moves a driver or rejects a request.

Secondly, we prove a general statement on essential stations for a time-interval. This result ensures that cars are not moved out of essential stations.

Thirdly, we partition the time-interval $[0, \overline{T}]$, where $\overline{T}$ is the maximal drop time of all released customer requests, into subintervals where the adversary moves a driver or rejects one or more requests, and where all requests are served without moving a driver. For each of these intervals, we give an upper bound on the solution value of ADV and of MarkRePlan. Finally, we conclude the upper bound of the competitive ratio.

Claim 5.5.1. Furthermore, let at time t be all cars of MarkRePlan at the same station as the cars of a δ-adversary ADV with $\delta \geq n$. If MarkRePlan rejects a customer r with $t^{rel}(r) > t$ or moves a driver, then ADV rejects a customer r' with $t^{rel}(r') > t$ or moves a driver as well.

Proof. We prove the statement by contradiction. Therefore, let us assume that there exists a sequence of customer requests σ so that ADV does neither move a driver after time t nor does ADV reject any customer request after time t, but MarkRePlan either rejects a customer request r or moves a driver.

By assumption, ADV does not reject any customer request after time t, and therefore, every customer request with a release time greater than or equal to t can be served. Furthermore, since ADV does not move any driver after t, it follows that every customer request (released after time t) can be served without moving a driver. Since all cars of MarkRePlan are at time t at the same station as the cars of ADV, the online algorithm can accept all customer requests without moving a driver. Since MarkRePlan does not unnecessarily move a driver and waits

until the last moment before a driver is moved in order to serve a customer request r, i.e., the online algorithm waits at least until $t' = t^{rel}(r) - n = t^{rel}(r) - \delta$. If there is no car available for r at the pickup station of r, then the δ-adversary ADV cannot release a customer request after t' so that r can served by the car transferred by a customer. Furthermore, MarkRePlan accepts a customer request if he can. Thus, it follows that MarkRePlan accepts all customer requests with a release time greater than or equal to t without moving a driver, contradicting our assumption. ◊

Claim 5.5.2. Let car i be at time t' in an essential station $v \in V$ for the time-interval $[t, T]$, with $t' \in [t, T]$. If i is moved by MarkRePlan arriving at time t'', with $t'' \in [t', T]$, at station v' then v' is an essential station for $[t'', T]$.

Proof. The statement follows simply by the fact that the cars are moved in the last moment, i.e., not before n time units before the pickup time of r. Since $\delta \geq n + 1$, it is ensured that ADV cannot release a customer request r' after MarkRePlan started to transfer the car to v', so that r can be served without moving a car. Therefore, MarkRePlan must transfer a car to v' in order to serve r. Since r is served by the car i, it follows that v' is an essential station for $[t'', T]$. ◊

Next, we partition the time horizon $[0, \overline{T}]$ into sub-intervals. Let $\sigma = (r_1, \ldots, r_m)$ be an arbitrary sequence of customer requests. For the sake of simplicity, let us assume that σ contains only customer requests which are either served by ADV or by MarkRePlan (or by both). This can be justified as follows.

Let $\sigma' = (r_1, \ldots, r_{m'})$ be a sequence of customer requests also containing those customer requests which are neither served by ADV nor by MarkRePlan. Since there are requests in σ' which are neither served by ADV nor by MarkRePlan, there exists a positive number $z > 0$ equal to the penalty for not serving these requests. Let $x + z := \mathsf{ADV}(\sigma')$ be the solution value of ADV, and let $y + z :=$ MarkRePlan(σ') be the solution value of MarkRePlan. Then we have

$$\frac{\textsc{MarkRePlan}(\sigma')}{\mathsf{ADV}(\sigma')} = \frac{y+z}{x+z} = \frac{y}{x+z} + \frac{z}{x+z} \leq \frac{y}{x} + 1. \tag{5.2}$$

This means that we have to adjust the computed competitive ratio by 1, but we can assume that there are no customer requests in σ which are neither served by ADV nor by MarkRePlan.

By partitioning the time horizon into sub-intervals, it follows that for each sub-interval $[t_i, t_{i+1}]$ one of the following cases is true

(i) ADV does not move a driver, and

  (a) serves all released customer requests r with a pickup time $\hat{t}(r) \in [t_i, t_{i+1}]$, or

  (b) rejects at least one customer requests r with a pickup time $\hat{t}(r) \in [t_i, t_{i+1}]$;

(ii) ADV moves a driver, and

  (a) serves all released customer requests r with a pickup time $\hat{t}(r) \in [t_i, t_{i+1}]$, or

  (b) rejects at least one customer requests r with a pickup time $\hat{t}(r) \in [t_i, t_{i+1}]$.

Obviously, if the adversary rejects some customer requests, it follows that the solution value is increased. Therefore, it is sufficient to give an upper bound for the cases (i)(a) and (ii)(a). Furthermore, moving a driver increases the solution value as well. Thus, it follows that it is sufficient to handle Case (i)(a) and to give a lower bound for the solution value of ADV and an upper bound for the solution value of MarkRePlan. From this we can then conclude an upper bound for the competitive ratio.

A sub-interval $[t_i, t_{i+1}] \subseteq [0, \overline{T}]$ is essential if Case (i)(a) holds for $[t_i, t_{i+1}]$, and it is maximal if in addition, in the intervals $[t_i - 1, t_{i+1}]$ and $[t_i, t_{i+1} + 1]$ the adversary moves a driver or rejects a customer request. Subintervals where Case (i)(a) does not hold are called non-essential sub-intervals.

Let $[t_1, t_2] \cup \cdots \cup [t_{\eta-1}, t_\eta] = [0, \overline{T}]$ be a partition of the time horizon $[0, \overline{T}]$ so that

- there is no non-essential sub-interval containing an essential sub-interval,
- each essential sub-interval is maximal, and
- the partitions alternate between maximal essential and non-essential sub-intervals.

Let $[t_i, t_{i+1}]$ be a maximal essential sub-interval. Our next aim is to show that, if the interval $[t_i, t_{i+1}]$ is large enough, MarkRePlan transfers a car to every essential station. Otherwise, if $[t_i, t_{i+1}]$ is not large enough, the number of rejected requests and movements of a driver is smaller.

Before we show which value is large enough for a maximal essential sub-interval, we prove that a car is transferred by a driver between the stations by MarkRePlan at most twice.

Claim 5.5.3. Each car is transferred by a driver at most twice within $[t_i, t_{i+1}]$.

Proof. A car is marked by MarkRePlan when a car serves a customer requests, i.e., when it is used by a customer. Since cars are transferred in the last moment, and due to $\delta \geq n+1$, it follows that each car transferred by a driver is "immediately" used by a customer. Thus, a car becomes marked after it is transferred by a driver. When all cars are marked, the markings are removed from all cars. This may happen before all cars are at an essential station. However, if all cars are marked for the second time within $[t_i, t_{i+1}]$, it follows from Claim 5.5.2 and by definition that every cars is in an essential station. Therefore, a car does not need to be transferred anymore after it has been marked for the second time, and the statement follows. ◊

Claim 5.5.4. The length of a (sub)tour transferring all cars within $[t_i, t_{i+1}]$ is at most 4γ.

Proof. Note that in the static situation, if the capacity L is large enough, all cars can be transferred within $n+1$ time units by visiting all overfull stations and afterwards all underfull stations. The total number of overfull and underfull stations is at most n. Since the station from that a driver starts can also be an underfull station, a driver may need to return to its starting station, resulting in $n + 1$ time units, in a uniform metric space with n nodes.

In the dynamic situation, due to the time-window when a car must arrive at a station, it may be necessary to visit a station several times. However, a car can be transferred to another station within 2 time-units, one in order to move a driver to the station the car is located at, and one to move the car to the pickup station.

Due to Claim 5.5.3, each car is moved at most twice, and therefore, the length of a (sub)tour which transfers all cars is at most 4γ units. ◊

From previous claim and Claim 5.5.2 we can already conclude that MarkRePlan moves a driver at most 4γ units and rejects at most $4\gamma^2$ customer requests within $[t_i, t_{i+1}]$. Hereby, recall that we assume that σ does not contain customer requests which are rejected by ADV and by MarkRePlan.

The upper bound of $4\gamma^2$ customer requests for MarkRePlan is only justifiable if ADV serves γ customer requests at each time step. However, in this worst-case, we can do a better analysis on the maximal number of rejected requests by MarkRePlan.

The maximal distance a driver has to traverse, before a customer request is served, is 2, and therefore, MarkRePlan can serve a "new" customer request at least every two time-units. Since each car of ADV serves a customer request at each time step, it follows that a car serving a

customer request also continues serving further customer requests without the need to be further transferred to another station. This implies that at every second time step, MarkRePlan rejects one customer request less than before, i.e., the number of rejected customer requests is at most

$$\begin{aligned} 2\gamma + (2\gamma - 1) + (2\gamma - 2) + \cdots + (2\gamma - 2\gamma) &= 4\gamma^2 - \sum_{i=1}^{2\gamma} i \\ &= 4\gamma^2 - \frac{4\gamma^2 + 2\gamma}{2} \\ &= 2\gamma^2 - \gamma. \end{aligned}$$

Thus, we can conclude that the solution value of MarkRePlan is at most $4\gamma + \alpha(2\gamma^2 - \gamma)$ for every sub-interval. Due to the definition of the partition of the time horizon, it follows that ADV moves a driver or rejects a customer request, between two maximal essential sub-intervals. Since the initial system states are equal for ADV and MarkRePlan, it follows from Claim 5.5.1 that ADV moves a driver or rejects a customer requests at least once, otherwise MarkRePlan serves all customer requests without moving a driver. In other words, we can assume that there is at least one non-essential sub-interval in $[0, \overline{T}]$ and, thus, if $\textsc{MarkRePlan}(\sigma) > 0$ then $\mathsf{ADV}(\sigma) > 0$ follows. If there are x maximal essential sub-intervals then there are at least $x - 1$ non-essential subintervals, and due to $1 \leq p(r) \leq \alpha$ for all customer requests r we have

$$\mathsf{ADV}(\sigma) \geq x - 1 \quad \text{and} \quad \textsc{MarkRePlan}(\sigma) \leq (2x - 1)(4\gamma + \alpha(2\gamma^2 - \gamma)).$$

Therefore, it follows

$$\begin{aligned} \frac{\textsc{MarkRePlan}(\sigma)}{\mathsf{ADV}(\sigma)} \leq \frac{(2x - 1)(4\gamma + \alpha(2\gamma^2 - \gamma))}{x - 1} &\leq \frac{2(x - 1)(4\gamma + \alpha(2\gamma^2 - \gamma))}{x - 1} \\ = 2(4\gamma + \alpha(2\gamma^2 - \gamma)) &= 8\gamma + \alpha(4\gamma^2 - 2\gamma) \end{aligned}$$

proving the stated maximal competitive ratio with the help of Equation (5.2). □

We could show that there does not exist a competitive online algorithm for the Online Min-Dist Relocation Problem with Penalty against a direct δ-adversary for any $\delta \leq n - 2$. In the case that $\delta \geq n + 1$, previous result shows that the online algorithm MarkRePlan is competitive. The remaining two cases, i.e., when $n - 1 \leq \delta \leq n$ is still an open question.

5.2.2 Max/Max Ratio

At the end of this section, we briefly discuss the max/max ratio of the Online Min-Distance Relocation Problem with Penalty. As before one can show that the max/max ratio does not lead to meaningful results by applying the techniques from the previous sections about the max/max ratio.

For the Online Min-Distance Relocation Problem with Penalty we could give a competitive online algorithm when the penalty function is bounded. Furthermore, we could show that not every online algorithm is competitive. However, the max/max ratio does not confirm this result. If the penalty function is bounded and the online algorithm does not unnecessarily move a driver, the max/max ratio becomes 1. However, if the penalty function is not bounded, then there does not exist an online algorithm with a determinate max/max ratio.

Chapter 6

Computational Results

In this chapter, we present the computational results for some online problems. Hereby, we concentrate on the Online Max-Accept Relocation Problem. The Online Min-Wait Relocation Problem is not experimentally considered within this thesis due to the enormous runtimes needed for solving even small instances. Hereby, proving infeasability due to a too short time horizon or solving even the relaxed version using continuous variables instead of natural numbers can take more than four hours. The enormous runtimes can be easily explained due to the number of variables in the corresponding integer linear program (see Chapter B in the appendix). For that we begin by giving an integer linear program with which we compute an optimal offline solution, "simulating" the adversary. This optimal offline solution is then used to evaluate the performance of the online algorithms computationally.

In Section 6.1, we propose a way to solve the Online Min-Reject Relocation Problem and the Online Max-Accept Relocation Problem optimally in one step by defining a time-expanded network G_T with two coupled flows: a car and a driver flow. Hereby, note that the online versions of these problems are very different in their nature (within the framework of the competitive analysis), but they become equivalent when it comes to compute an optimal solution. This is mainly due to the fact that the number of customer requests is fixed in the offline situation, but they may vary when the adversary releases new requests dynamically over time.

In order to solve the two offline problems, we consider a max-profit flow problem in G_T where moving the drivers in the system induces some small costs and a high profit is given for each accepted request (Offline Max-Profit Relocation Problem). This construction is theoretically irrelevant but has some practical implications. Due to the high profit on an accepted request, the solver emphasizes on serving these, while the small fee on moving the drivers ensures that drivers are not send out unnecessarily. This results in more natural transportation schedules than if there are no costs for the drivers. Furthermore, the model can be used to compute an optimal solution for the Online Min-Distance Relocation Problem with Penalty Function by slightly modifying the objective function.

Due to the long computation time of finding a good/optimal solution, we propose a flow-based heuristic (see Section 6.2). This flow-based heuristic computes a solution in two steps: a preprocessing phase where we aim at computing the arcs in a time-expanded network which are likely to be used in an optimal solution, afterwards, in the second phase, we compute a transportation schedule on a reduced time-expanded network.

Both approaches for solving the offline instances have been presented in [111].

Since some online algorithms (e.g., RePlan) require that an instance of the offline problem are solved repeatedly, we give some hints in Section 6.3 how flow-based approaches have to be modified so that they can be used within the context of online algorithms.

Finally, in Section 6.4, we present the computational results and evaluate the different online algorithms.

6.1 Computing an Optimal Offline Solution

In this section, we explain the construction of the time-expanded network G_T as well as the car and driver flows within G_T. A solution for the flows can be computed with the help of an integer linear program which we state at the end of this section.

Flows in Time-Expanded Networks We consider a max-profit flow problem which rejects customer requests from R whose profit is smaller than the relocation cost to satisfy them. For that, we build a directed graph $G_T = (V_T, A_T)$, with $A_T = A_H \cup A_L \cup A_R$ as a time-expanded version of the original network G which includes arcs A_R corresponding to the customer requests in R (see Section 6.1). The cars and drivers form two flows f and F through G_T which are coupled in the sense that on arcs $a \in A_L$ (the relocation arcs) we have the condition $f(a) \leq L \cdot F(a)$ reflecting the dependencies between the two flows. Note that the tasks are directly derived from the sequence R of customer requests (if an accepted request causes an infeasible system state, the task to balance this state is implicitly generated, see Section 6.1).

Time-Expanded Networks We build a time-expanded version $G_T = (V_T, A_T)$ of the original network G.

The node set V_T is constructed as follows. For each station $v \in V$ and each time point $t \in [0, T]$ in the time horizon $[0, T]$, there is a node $(v, t) \in V_T$ which represents station v at time t as a capacitated station where convoys can simply pass or cars can be picked up, delivered and exchanged by drivers.

The arc set $A_T = A_H \cup A_L \cup A_R$ of G_T is composed of several subsets:

- A_H contains, for each station $v \in V$ of the original network and each $t \in \{0, 1, \ldots, T-1\}$, the holdover arc connecting (v, t) to $(v, t+1)$;
- A_L contains, for each edge (v, v') of G and each point in time $t \in T$ such that $t + d(v, v') \leq T$, the relocation arc from (v, t) to $(v', t + d(v, v'))$;
- A_R contains, for each customer request $r = (v, t, v', t') \in R$ a request arc from (v, t) to (v', t').

Note that the time-expanded network G_T is acyclic by construction.

Furthermore, for the construction (and later for solving the resulting integer linear program) of a time-expanded network, it is not necessary that the distances between two stations are symmetric, i.e., that $w(v, v') = w(v', v)$ holds.

Finally, the time-expanded network can be constructed in such a way that the travel times between two stations are different depending on the time a driver leaves a station.

A Max-Profit Flow Problem On the time-expanded network G_T, we define two different flows, the car flow f and the driver flow F, to encode the relocation of cars in convoys.

Note that a flow on a relocation arc corresponds to a (sub)move in a tour, i.e., some cars are moved by drivers in a convoy from a station u to another station v. Thus, a relocation arc from (u, t) to $(v, t + d(u, v))$ has infinite capacity for the drivers, but to ensure that cars can be moved only in convoys and at most L cars per driver, we require that

$$f(a) \leq L \cdot F(a) \text{ for all } a \in A_L$$

holds (see (6.1g)). Thus, the capacities for f on the relocation arcs are not given by constants but by a function. Note that due to these flow coupling constraints, the constraint matrix of the network is not totally unimodular (as in the case of uncoupled flows) and therefore integrality constraints for both flows are required (6.1j)), reflecting that solving the problem is $\mathcal{NP}$-hard.

Flows on holdover arcs correspond to cars/drivers remaining at the station in the time interval $[t, t+1]$. Thus, the capacity of all holdover arcs for flow f is set to $\text{cap}(v)$ (see (6.1f)), whereas there is no capacity constraint for F on such arcs. Moreover, a car flow on a customer request arc corresponds to an accepted request (see (6.1h)), whereas driver flow is forbidden on such arcs (see (6.1i)).

We consider a max-profit flow problem to decide which customer requests can be satisfied without spending more costs in the relocation process than gaining profit by satisfying them. Accordingly, our objective function (6.1a) considers profits $p(a)$ for the car flow f on all $a \in A_R$ and costs $d(a) := d(u,v)$ for the drivers on all relocation arcs $a = ((u,t),(v,t+d(u,v)))$ corresponding to an edge (u,v) in G, whereas all other arcs have zero profits and costs.

To correctly initialize the system, we use the nodes $(v,0) \in V_T$ as sources for both flows and set their balances accordingly to the initial numbers of cars and drivers at station v and time 0 in z_v^0 and z_v^d (see (6.1b) and (6.1c)). For all internal nodes $(v,t) \in V_T$ with $t > 0$, we use normal flow conservation constraints[1] (which is possible due to the fact that the entire flow of cars is modeled by taking parked cars, convoy moves and customer actions into account), see (6.1d) and (6.1e).

This leads to a Max-Profit Relocation Problem, whose output is a subset of accepted customer requests $R' \subseteq R$ and a transportation schedule for a metric task system, whose tasks are induced by the decision which customer requests are accepted. The corresponding integer linear program is as follows:

$$\max \sum_{a \in A_R} p(a)f(a) - \sum_{a \in A_L} d(a)F(a) \tag{6.1a}$$

$$\sum_{a \in \delta^-(v,0)} f(a) = z_v^0 \qquad \forall\ (v,0) \in V_T \tag{6.1b}$$

$$\sum_{a \in \delta^-(v,0)} F(a) = z_v^d \qquad \forall\ (v,0) \in V_T \tag{6.1c}$$

$$\sum_{a \in \delta^-(v,t)} f(a) = \sum_{a \in \delta^+(v,t)} f(a) \qquad \forall\ (v,t) \in V_T, t > 0 \tag{6.1d}$$

$$\sum_{a \in \delta^-(v,t)} F(a) = \sum_{a \in \delta^+(v,t)} F(a) \qquad \forall\ (v,t) \in V_T, t > 0 \tag{6.1e}$$

$$0 \le f(a) \le \text{cap}(v) \qquad \forall\ a = [(v,t),(v,t+1)] \in A_H \tag{6.1f}$$

$$f(a) \le L \cdot F(a) \qquad \forall\ a \in A_L \tag{6.1g}$$

$$f(a) \le 1 \qquad \forall\ a \in A_R \tag{6.1h}$$

$$F(a) = 0 \qquad \forall\ a \in A_R \tag{6.1i}$$

$$f, F \text{ integer}, \tag{6.1j}$$

where $\delta^-(v,t)$ denotes the set of outgoing arcs of (v,t), and $\delta^+(v,t)$ denotes the set of incoming arcs of (v,t).

[1] The flow conservation constraints and the finite number of drivers induce capacities for the drivers on holdover and relocation arcs. However, some solvers have problems with infinite capacities. Therefore, when implementing the model, one should always give at least k as upper bound for the driver flow on every arc.

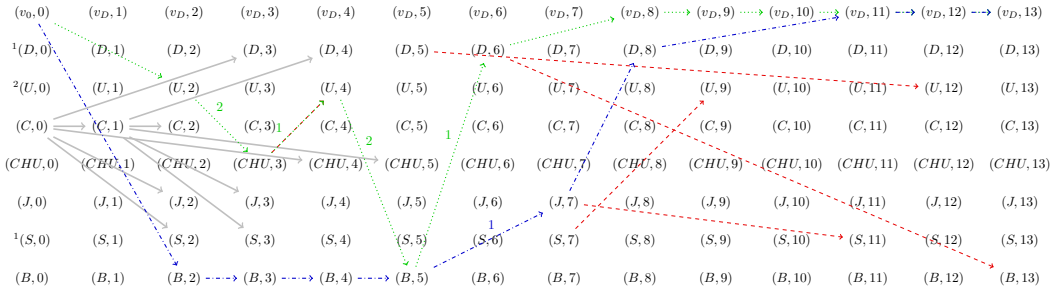


Figure 6.1: This figure illustrates an example of a time-expanded network G_T. For the sake of readability, we give a subnetwork of G_T, where only some arcs (solid and dashed arcs) of the time-expanded network and only the driver flows (dash-dotted and dotted arcs) are shown. Furthermore, only those nodes are in the figure which are important for the solution and the customer requests. Every node of the form (v, t) represents a station v at time t. Customer request arcs are shown as dashed arcs, the tour of driver 1 as dash-dotted arcs, and the tour of driver 2 as dotted arcs. Positive values for $b(v, 0)$ are superscripted before the nodes $(v, 0)$. The numbers at the arcs correspond to the number of cars transferred by the corresponding move.

The next example illustrates a time-expanded network with capacities on the arcs as well as the balances for the nodes $(v, 0)$ and D.

Example 6.1. Let us consider the Online Max-Accept Relocation Problem from Example 2.4. The requests and a solution of the integer linear program 6.1 is illustrated as flows in a time-expanded network in Figure 6.1. ◊

The next theorem shows that one can construct a transportation schedule from a solution of this integer linear program.

Theorem 6.2. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, L, R)$ be an Offline Max-Profit Relocation Problem, and let f be a car flow and F be a driver flow respecting the constraints* (6.1a)–(6.1j) *then there exists a transportation schedule $\mathcal{S}$ solving $(G, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, L, R)$ so that the profit for $\mathcal{S}$ is $\sum_{a \in A_R} p(a) f(a) - \sum_{a \in A_L} d(a) F(a)$.*

Proof. "⇒" Let $\mathcal{S} = \{\Gamma^1, \ldots, \Gamma^k\}$ be a transportation schedule solving $(G, \boldsymbol{z}^0, \boldsymbol{z}^d, \gamma, k, L, R)$. Furthermore, let T be the time when all tours of $\mathcal{S}$ end. We construct flows F and f from these tours and show that they satisfy (in)equalities (6.1b) to (6.1j).

Due to (s.i) it is ensured that there are exactly k drivers in the system. Each driver has its starting location v, represented in G_T by $B(v)$. By starting the flows F from the starting locations, it is ensured that constraints (6.1c) are fulfilled. Analogously, each parking location v has an initial number of cars, represented in G_T by $b(v)$. Since cars are only moved in convoys from one location to another or by customers taking cars and returning them at another station, it follows, by starting the flows f at the starting locations, that equalities (6.1b) are fulfilled.

Performing actions does not consume time, and thus, each move $m \in \Gamma^j$ directly corresponds to flow F on the corresponding arcs including the path $path(m)$. The flow values $F(a)$ for holdover arcs $a = ((v, t), (v, t+1)) \in A_H$ are set to the number of drivers waiting at station v (during t to $t+1$). Then it follows from the definition of a tour (t.i)–(t.iv), that the flow conservation constraints (6.1e) hold ($dur(a) = 0$ for all actions a) for all nodes $V_T \setminus \{(v, 0), (v, T)\}$.

Next we construct the flows f. We initialize the flows by $f(a) = 0$ for all arcs $a \in A_T$. When a convoy transports $\ell oad(m)$ cars in a move m, we increase the flow f by $\ell oad(m)$ along the path

in the time-expanded network corresponding to the path $path(m)$. Due to (m.i) of the definition of a move, it is ensured that drivers do not wait at a station between the origin and destination station of m. In the case that cars are not moved, i.e., waiting at a station, we increase the flow f on the corresponding holdover-arcs. For every served customer request, we set the flow value $f(a) = 1$ for the corresponding arc $a \in A_R$, and, analogously, for every rejected customer request, we set the flow value $f(a) = 0$, fulfilling equalities (6.1h). Due to (s.iii) of the definition of a transportation schedule, it is ensured that there are enough cars (resp. enough space) at the start node (resp. end node) of the corresponding customer request arc.

Furthermore, (s.iii) ensures that not more than $\text{cap}(v)$ cars are waiting at station v. Therefore, constraints (6.1f) are satisfied. Furthermore, when cars are moved in a convoy, there are never more than L cars in that convoy (due to (m.ii)), ensuring that inequalities (6.1g) hold.

Cars are only moved by convoys or by customers. Otherwise, they wait at a parking space. This ensures the flow conservation constraints (6.1d) for f.

Finally, since only an integer amount of cars and drivers are moved in a transportation schedule, conditions (6.1j) hold.

"$\Leftarrow$" We show that we can create a transportation schedule from flows f and F which are constrained by (6.1b) to (6.1j).

For that we define temporary flows f_t and F_t and initialize them with $f_t(a) = F_t(a) = 0$ for all arcs $a \in A_T$. For each driver j the tour is created as follows.

We search a path P in G_T from a node $(v, 0)$ to D with $F(a) - F_t(a) > 0$ for all $a \in P$. The tour for driver j then starts at v. If there does not exists such a path, we stop.

First, we construct a sequence of moves $\mathcal{M}$. Then we insert actions between the moves, resulting in an alternating sequence of moves and actions. Finally, we prove that this defines a tour and that all tours together are a transportation schedule.

Depending on the kind of arc on the path we do the following:

Case 1 ($a \in A_H$): this means that the driver is waiting at the station corresponding to the holdover arc. Let $a = (v, t, v, t + 1)$ then we add the move $(j, v, t, v, t + 1, \emptyset, 0)$ to $\mathcal{M}$.

Case 2 ($a \in A_L$): now the driver moves from one location to another (possibly) transferring cars. For that let $a = (v, t^v, w, t^w)$ and let $x := \min\{f(a) - f_t(a), L\}$. Then we add the move $(j, v, t^v, w, t^w, \{(v, w)\}, x)$ to $\mathcal{M}$. Furthermore, the value $f_t(a)$ is increased by x.

After all moves have been added to $\mathcal{M}$, we insert actions between the moves. For that let $m^\ell, m^{\ell+1} \in \mathcal{M}$ be two successive moves. Then we insert the action $(j, dest(m^\ell), 0, \ell load(m^{\ell+1}) - \ell load(m^\ell))$ between these moves. By adding actions between all moves, we receive an alternating sequence of moves and actions.

We still need to prove that this constructed alternating sequence is a tour. By construction all moves and all actions belong to one driver j, ensuring (t.i). Condition (t.ii) and (t.iii) are obviously fulfilled, since the sequence is created from paths in the time-expanded version of the graph and the duration of an action is always 0. Hereby, the moves are created on basis of a path from a node $(v, 0)$ to a node (v', T), and constraints (6.1i) ensure that only paths, which exist in the graph, are taken. Condition (t.iv) follows by construction of the actions.

The condition (m.i) is ensured by construction of the moves. Furthermore, the moves are constructed so that at most L cars are transferred by a move, and condition (m.ii) directly follows.

Therefore, the constructed alternating sequence is indeed a tour.

It follows from the flow conservation constraints for the drivers (6.1e) and from (6.1c) that there exist k paths from $(v, 0)$ to some (v', T). Hereby note that $k = \sum_{v \in V} B(v, 0) = B(D)$. Since the time-expanded network is cycle-free and since the flow conservation constraints hold, it is ensured that no additional drivers are created during the construction. Furthermore, it follows that after applying the steps above k times, it holds $F(a) = F_t(a)$.

Analogously, it follows from the flow conservation constraints for the cars (6.1b), and from the fact that the time-expanded network is cycle-free, that no additional cars are created during the construction. Furthermore, (6.1g) and the construction of the moves ensures that $f(a) = f_t(a)$ for all relocation arcs $a \in A_L$ when the algorithm stops. Since cars waiting at a station, the $f(a) = f_t(a)$ is not important for the holdover arcs $a \in A_H$ (and it also does not hold).

Finally, we need to prove that all tours together are a transportation schedule. Hereby, we show that each condition of the definition of a transportation schedule holds. Condition (s.i) is obviously fulfilled by construction, i.e., every driver has exactly one tour.

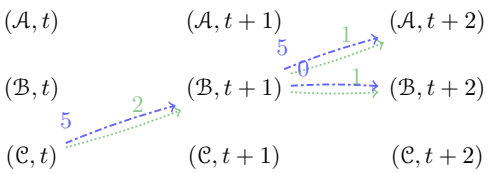
Since all task are induced by the accepted customer requests, and customer requests can be either accepted or rejected (due to inequalities (6.1h)), it follows that condition (s.ii) holds.

Finally, condition (s.iii) holds due to inequalities (6.1f). Thus, it is ensured that all tours together are a transportation schedule, proving the statement. □

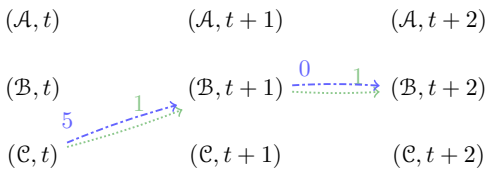
Remark 6.3. The tours constructed from the flows in the proof of Theorem 6.2 are not the only possible tours. In fact, they are not necessarily the best possible in the sense that unnecessary exchanges are performed by the drivers. This is illustrated in Figure 6.2.

Additionally, it is likely that there are unneeded empty actions in the tours due to the artificial construction of the actions, e.g., actions between two "waiting moves". These actions can be safely removed and the waiting time can be expanded using a single move only.

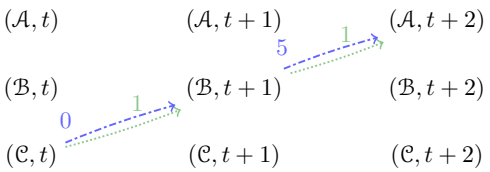
Finally, we can merge paths along relocation arcs, when there are empty actions between the two corresponding moves. ⧫



(a) This figure illustrates the "original" output of the flows.



(b) This figure illustrates the subtour for driver 1.



(c) This figure illustrates the subtour for driver 2.

Figure 6.2: This figures demonstrates a situation that can occur during the proof of Theorem 6.2. In this example, the capacity of a convoy is 5. Hereby, two drivers drive from station $\mathcal{C}$ to station $\mathcal{B}$, transferring 5 cars. After the drivers arrive at $\mathcal{B}$ driver 1 continues to station $\mathcal{A}$, while the other (driver 2) stays at $\mathcal{B}$. Since driver 2 transports the cars from $\mathcal{C}$ to $\mathcal{B}$, but driver 1 continues to transfer the cars to $\mathcal{A}$ the convoys have to exchange cars. A better solution would be if driver 2 picks up the cars already at $\mathcal{C}$.

Note, for the Offline Max-Profit Relocation Problem there exists always a feasible transportation schedule (since rejecting all customer requests is a solution).

6.2 A Flow-Based Heuristic

Computing an optimal solution is generally very slow and, thus, not applicable in practice. However, the runtime can be improved by reducing the number of arcs and nodes in the time-expanded network, which corresponds to a reduction of variables in the corresponding integer linear program. As experiments have shown that only a small percentage of arcs in G_T is used in the optimal solution, the idea is to reduce G_T to a network containing only arcs which are taken in the optimal solution with high probability.

Therefore, we present a heuristic that solves the Offline Max-Profit Relocation Problem in two phases. In the first phase, we compute those arcs which are likely to be used in an optimal solution. In the second phase, we construct a reduced time-expanded network G'_T, where we keep only the previously computed arcs and discard all others; afterwards, we compute an optimal solution on this reduced network G'_T. This does not lead to a globally optimal solution on G_T, but provides reasonable solutions in short time.

6.2.1 Preprocessing (Phase 1)

This phase itself is performed in two steps. Firstly we compute only a car flow and, secondly, a driver flow that "covers" the car flow.

Car Flow Model and its Linear Program. In this paragraph, we specify the capacities as well as the profits and the costs for each arc with respect to the car flow f_1 on the original time-expanded network G_T. Hereby, we mention only the differences to the time-expanded network from Section 6.1. Finally, we give a linear program in order to compute this car flow.

Unlike to the exact approach, in the preprocessing step, the car flow on a relocation arc is not coupled to another flow but has infinite capacity. The costs with respect to the flow f_1 are set to 1 for each relocation arc.

On customer request arcs we set an upper bound of 1. Customer request arcs have no costs but a profit. In order to ensure that there is positive car flow on a customer request arc $a \in A_R$ whenever possible, the profit must be selected high enough, compared to the costs of the relocation arcs, e.g., $p(a) = |V|$.

In order to compute the car flow f_1 we consider

$$\max \sum_{a \in A_R} p(a) f_1(a) - \sum_{a \in A_L} d(a) f_1(a),$$

with subject to the constraints (6.1b), (6.1f), (6.1h) and f_1 real.

Constraining the flow to be integer is not necessary since the constraint matrix is totally unimodular. Thus, the solution contains integer values only.

Note, from the solution of the car flow, it is possible to directly compute an upper bound on the maximal profit as well as the maximal number of customer requests which can be theoretically served.

Driver Flow Model and its Linear Program. Next, we define a driver flow F_1 on the time-expanded network G_T. We specify the capacities as well as the profits and the costs for each arc with respect to F_1. Finally, we give a linear program in order to compute the driver flow F_1.

The driver flow is influenced by the car flow in such a way that we try to "cover" the car flow on the relocation arcs by giving a profit p on the relocation arcs having a positive car flow.

All other relocation arcs have costs of 1. Note, that the profits for the relocation arcs must be chosen high enough, e.g., $p(a) = |V|$, $a \in A_L$. The capacity for a relocation arc is infinity.

In order to compute the driver flow F_1 in $G_T = (V_T, A_T)$, we consider

$$\max \sum_{a \in A_L^+} p(a)F_1(a) - \sum_{a \in A_L^-} d(a)F_1(a)$$

where $A_L^- = \{a \in A_L \mid f_1(a) = 0\}$ and $A_L^+ = \{a \in A_L \mid f_1(a) > 0\}$, with subject to the (in)equalities (6.1c), (6.1e), (6.1i), and F_1 real.

Like in the previous step, the flow does not need to be constrained to integer values, since the constraint matrix is totally unimodular.

6.2.2 Computing a Transportation Schedule (Phase 2)

Unlike in the exact approach, the two flows are not coupled in the first phase. This implies that each flow can be rapidly computed, but the computed solution is in general not a feasible transportation schedule. In this section, we describe the construction of a reduced version $G'_T = (V'_T, A'_H \cup A'_L \cup A_R)$ of the original time-expanded network $G_T = (V_T, A_H \cup A_L \cup A_R)$ based on the flows computed in the preprocessing (Phase 1). Hereby, we reduce the total number of nodes as well as of holdover and relocation arcs. The set of customer request arcs is not changed. Afterwards, we compute an optimal solution on G'_T providing a feasible transportation schedule.

Constructing the Reduced Time-Expanded Network. The reduced network G'_T is constructed as follows. First, we add for each station $v \in V$ the nodes $(v, 0), (v, T)$ to V'_T, and for all customer request arcs $[(v, t), (w, t')] \in A_R$ we add the nodes $(v, t), (w, t')$ to V'_T.

Only the relocation arcs $a = [(v, t), (w, t')] \in A_L$ with $f_1(a) > 0$ or $F_1(a) > 0$ remain in A'_L, and we add the nodes $(v, t), (w, t')$ to V'_T.

Next, for each station $v \in V$ we add holdover arcs between two successive nodes on the time line of v. The set of request arcs is taken unchanged from G_T.

Computing a Feasible Transportation Schedule. A feasible transportation schedule is computed by solving the integer linear program from Section 6.1 on the reduced time-expanded network G'_T.

The problem is always feasible due to the following reason: In the Relocation Problem, every customer request can be rejected. Due to the holdover arcs, for every station $v \in V$ there is a path from the source node $(v, 0)$ to the node (v, T) for the car and driver flows. Thus, we can directly conclude that the flow-based heuristic always computes a feasible solution for the Relocation Problem.

6.2.3 Improving the Solution

Although, the flow-based heuristic already computes a feasible solution for the Relocation Problem, the solution might be suboptimal. Due to the reduction of the relocation arcs, it might be impossible to find a tour for the drivers on the reduced network $G'_T = (V'_T, A'_H \cup A'_L \cup A_R)$ so that they can serve (some) customer request. Adding more relocation arcs to A'_L than described in Section 6.2.2 can increase the number of served customer requests. However, this generally also results in higher computation times. Therefore, one has to carefully select which relocation arcs shall be added.

We now briefly describe a variation where further relocation arcs are added to A'_L in order to improve the solution. For that, let f_1 be the car flow and let F_1 be the driver flow from the preprocessing step. When the car flow f_1 on a relocation arc $a = [(v,t),(w,t')] \in A_L$ is greater than 0, but the driver flow F_1 on this arc is 0, it is likely that it is impossible to find tours for the drivers in G'_T which transfer cars on the corresponding relocation arc $a' \in A'_L$. In order to increase the probability of the existence of a tour which transfers cars on a, we add all relocation arcs from $a' \in A_L$ where (v,t) is the end node of a' to A'_L.

6.3 Online Flow-Based Approaches

Since customer requests are released over time, the sequence of already released customer requests until a point in time can be used as input for the exact approach or the flow-based heuristic. Therefore, it is natural to use algorithms originally designed for offline problems in the context of the Dynamic Relocation Problem. However, it is still necessary to apply some modifications. For that we give some hints on how to change the network, the flow model, and the resulting integer linear program when flows in time-expanded networks are used to compute the transportation schedule (e.g., when using the RePlan strategy). In this section, we describe how the exact approach from Section 6.1 can be modified so that it can be used within the online situation.

Dynamically Computing a new Transportation Schedule. To take the dynamic evolution of the customer requests into account, we partition R into two subsets R^A of previously accepted customer requests and R^N of customer requests released within the new time interval. Accordingly, we refine the arc set A_R of G_T by considering two subsets $A_{R^A} \cup A_{R^N}$ for the two request subsets R^A and R^N and set $f(a) = 1$ for all $a \in A_{R^A}$, to ensure that previously accepted customer requests are fulfilled. For every newly released customer request, we bound the car flow on the corresponding arc $a \in A_{R^N}$ by $f(a) \leq 1$ to allow that new customer requests can be rejected.

Note that in the case when the online algorithm can wait until the pickup time to accept or reject a customer requests, the set R^A may be always empty while the set R^N contains all released customer requests.

Example 6.4. Let us consider the customer requests and the computed transportation schedule from Example 6.1. At time 7 another customer wants to take a car from U to J, taking it at time 10 and returning it three time units later, i.e., at time 13.

Since there are already two customers with their rented cars on the roads, the set R^A contains at least the customer requests $(0, D, 5, U, 12)$ and $(3, D, 6, B, 13)$. The newly released customer request $(7, U, 10, J, 13)$ is in R^N. The other customer request may be in R^A or in R^N, depending on when the operator has to accept and reject the requests, respectively, and depending on its previous decisions. ◊

Due to newly released customer requests it is usually advisable to update the current transportation schedule. In general, not all drivers are at a station at the time t when one recomputes the transportation schedule, but some may be between two stations transferring cars. A driver transferring cars x at time t to a station v arriving at time t_v, can be modeled by adding a source S and a source arc $a = (S, (v, t_v))$. This arc has capacity 1 w.r.t. the driver flow and a capacity of x w.r.t. the car flow, i.e., we set $F(a) = 1$ and $f(a) = x$. Since the driver always drives to the station v, the costs for the source arcs can be neglected.

The same applies to customers that are still using their rented car while the transportation schedule is recomputed, i.e., we add a customer source arc from S to (v, t_v), where v is the destination station of the corresponding customer request and t_v the destination time, respectively.

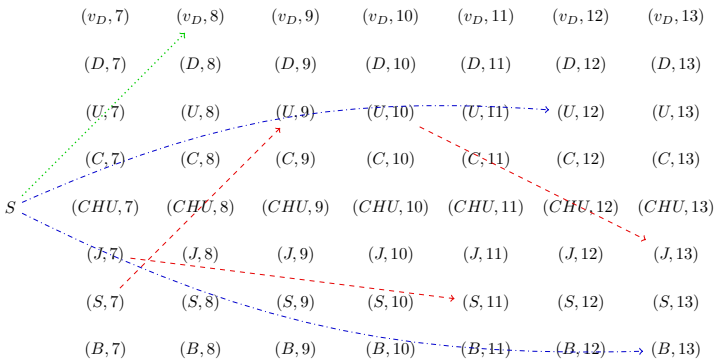


Figure 6.3: This figure illustrates an example of a dynamically computed time-expanded network G_T, where, for the sake of readability, only the (customer) source arcs and the customer request arcs of the time-expanded network are shown. Every node of the form (v, t) represents a station v at time t. Customer request arcs are shown as dashed arcs, customer source arcs are shown as dash-dotted arcs, and source arcs are indicated by dotted arcs.

Hereby, the capacity w.r.t. the driver flow is 0 and the capacity w.r.t. the car flow is 1, i.e., we set $F(a) = 0$ and $f(a) = 1$. Note, due to the capacities of the stations the car flow induced by a customer request cannot be ignored. Furthermore, since the customer request corresponding to the customer source arc can neither be accepted nor rejected, the profit for the arc can be neglected.

Example 6.5. Let us consider the situation from previous example. A recomputation of the transportation schedule at time 7 has to take into account that one driver in on the road at this point of time. Furthermore, there are two customers with their rented cars on the roads.

Thus, we add the source S to the network G_T and a source arc representing the driver on the road and two customer source arcs representing the customers with their cars (see Figure 6.3 for an illustration). Note, since the other driver is located at a station at time 7, there is only one source arc for the drivers. Since there is a positive flow on these arcs, the flow conservation constraints ensure that these drivers and customers are correctly taken into account when a new transportation schedule is computed. ◊

Testing if a New Transportation Schedule is Needed. When a new customer request r is released, one usually wants to check if r can be served without changing the current transportation schedule (but possibly changing the number of cars transfered by the drivers). This problem can be resolved efficiently by using flows in a time-expanded network G_T as we show in the following.

For that let $\mathcal{S}$ be a transportation schedule serving all customer requests of the sequence R^A. Furthermore, let R^N be the sequence of newly released customer request (containing at least r). On G_T we define only car flows f. The "driver flows" in G_T induced by $\mathcal{S}$ define the capacities for f on the relocation arcs. This means, that if there are $\kappa(a)$ drivers moving from station v to v' (starting at time t_v and arriving at time $t_{v'}$), then we remove the Constraints (6.1g) and require for the relocation arc $a = ((v, t_v), (v', t_{v'}))$ that

$$f(a) \leq L \cdot \kappa(a)$$

holds.

Since there is only one flow, one can easily see that the coefficient matrix of this integer linear program is totally unimodular, and it follows that the program can be solved efficiently. Furthermore, it follows that f does not need to be restricted to be integer.

6.4 Test Instances

The tests have been run on a server with Intel Xeon E7-8870 processors clocked at 2.40 GHz. In total there are 160 kernels, and in total 1 TB RAM available. The operating system is CentOS 6.6 with the Linux kernel 2.6.32.

The framework for the metric task system is implemented in Java 6, and so are the combinatorial approaches. Optimization problems (i.e., the integer linear programs) are solved with Gurobi 6.0 using the Python interface. In order to run several instances at the same time we use the gnu tool "parallel" [151].

Besides the optimal offline solution, we tested the following online algorithms: AIP, FcFs, EST, and RePlan. Hereby, we implemented the algorithm AIP in two different ways: once a simple version (AIP (aic)) of the algorithm where only the current system state is taken into account whether a request is accepted or rejected, and once where a minimal flow problem is solved (AIP (aic-flow)) where only the car flows are considered. Furthermore, we implemented two versions for RePlan, one using the exact approach (replan) and one using the flow-based heuristic (replan-fbh). The approaches are not implemented in their "pure" form but mixed with an Ignore-like strategy, i.e., all customer requests which are released at time x are handled as a bundle of customer requests which are released at the same time. Hereby, all release times are natural numbers.

For the online algorithms AIP, FcFs and EST we did not set any time limit. For the two versions of RePlan we set a time limit of 2 minutes for solving an intermediate offline instance. For computing an exact offline solution, we set a time limit of 1 hour, for the flow-based heuristic the time limit is set to 10 minutes. The runtimes for the different approaches are summarized in Table 6.7.

All approaches have been tested on three sets of test instances, a small, a medium and a big set. Hereby, the "small" instances have 15 stations; the number of cars is 150, the number of drivers 2 and 5, convoy capacities of 3 and 5, time horizons are set to 120 time units, and there are 500, 800, 1000 and 1600 customer requests. Furthermore, there are 30 instances per permutation, this means that there are, e.g., 30 instances with 2 drivers with a convoy capacity of 3 and 1000 customer requests. This results in 480 randomly generated instances. Note that the sizes of these instances corresponds to small car- or bikesharing systems (e.g., C.Vélo in Clermont-Ferrand, France) or to clusters of larger systems.

The "medium" instances have 50 stations; the number of cars is 500, the number of drivers 10, the convoy capacities are 5, the time horizons are set to 120 and 240 time units, and there are 500, 800, and 1600 customer requests. As in the small instances, there are 30 instances per permutation, resulting in a total of 180 test instances. The number of stations of these instances correspond to intermediate sized car- or bikesharing systems (e.g., Velodi in Dijon, France).

The "big" instances have 250 stations with a total number of 2500 cars. Five drivers relocate the cars in convoys of a length of maximal 5 cars. The time horizons are set to 400 time units, and there are 1600 and 3200 customer requests. As before, there are 30 instances per permutation, resulting in a total of 60 test instances. The number of stations of these instances correspond to larger car- or bikesharing systems (e.g., V'Lille in Lille, France).

The stations are uniformly distributed over a plane with a width and height of 10 units for the small and medium sized test instances and a with and height of 25 units for the big instances, respectively, and added distances between two stations correspond to the rounded Euclidean

distances. Hereby, not all roads are added to the system, but it is ensured during the generation of the carsharing system that the resulting graphs are connected.

All customer requests are uniformly distributed, i.e., the requested pickup and the drop stations are uniformly distributed. The profits attached to serving a request is between 100–120 units, while the costs for moving a driver for one unit is set to 1; there are no costs for moving a car. Furthermore, the maximal difference between the pickup and release time of a customer request is 20 time units.

For the "small" instances, the exact approach did not find any optimal solution within the given time limit (the duality gap is on average 1.82% and the median is 1.51%), while the flow-based heuristic did not find one in 106 instances (the duality gap for these instances in on average 0.45% and the median is 0.40%).

As before, for the "medium" instances, the exact approach did not find any optimal solution. In three cases, the exact approach did not even find any feasible solution, four times the number of accepted customer requests is 0, and in 14 instances the number of accepted customer requests is below 25. The duality gap is on average 5.22% and the median is 0.55%, when the above 14 instances are not taken into account.

The exact offline approach found a feasible solution for two "big" test instance: once accepting 5 customer requests and once 0. On all other instances, no feasible solution could be found within the time limit of 1 hour.

The duality gaps of the computational results are summarized in the Tables 6.1, 6.3 and 6.5. In the following we give a brief discussion on the results.

Small instances. As one can see, the online algorithm RePlan (replan) accepts in most cases more customer requests than the other three online algorithms. Especially, when there are only a small number of drivers, the algorithm EST is nearly as good as RePlan (and in some cases it achieves even better results). As expected, in most cases, EST performs better than the strategy FcFs. However, the gap between these two online algorithms is not very large. There is only a minor difference between the two version of AIP; in most of the cases, the flow-based version is only slightly better.

In Table 6.1 we summarize the percentage of accepted customer requests from an algorithm in comparison with the maximum number of accepted customer requests. This upper bound is computed during the first step of the flow-based heuristic. A comparison between the number of accepted customer requests computed by an (online) algorithm and the exact approach is shown in Table 6.2. Hereby, a value in a cell (of column 4–10 and 4–9, respectively) corresponds to the average of 30 instances.

As one can see, the number of accepted customer requests increases with the number of drivers and the capacity, but the percentage of compared to the exact approach decreases. This can be easily explained since the number of accepted customer increases faster in the exact approach than with the online algorithms.

Medium instances. On the medium sized test instances, the results are similar to the results on the smaller instances. However, some further effects become visible which we discuss in the following.

Unlike before, the online algorithm RePlan (replan) does not necessarily accept more customer requests than the other online algorithms. The results imply that there seems to be a correlation between the number of customer requests which can be accepted by an optimal offline solution and the performance of RePlan. Hereby, RePlan seems to profit if many customer requests cannot be served even by an optimal offline solution. Another explanation may also be the fact, that an optimal intermediate solution cannot always be computed. However, since the

Table 6.1: This table summarizes the percentage of accepted customer requests by an algorithm (in comparison with the number of accepted customer requests by the upper bound computed in step 1 of the flow-based heuristic) applied on the "small" instances. In columns one to three, the number of drivers, the capacity of the convoys and the number of requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FcFs (fcfs), EST (est), RePlan (replan and replan-fbh), the flow-based heuristic (fbh) and the exact approach (exact) with a time limit of 1 hour.

k	L	$\|R\|$	aic	aic-flow	fcfs	est	replan-fbh	replan	fbh	exact
2	3	500	50.57	49.99	53.47	54.52	53.86	54.85	74.64	81.25
2	5	500	50.57	49.99	53.47	54.52	55.28	56.81	80.52	88.05
5	3	500	50.24	49.74	57.12	58.96	58.46	61.46	87.27	95.91
5	5	500	50.24	49.74	57.12	58.96	59.61	64.15	94.23	98.84
10	3	500	50.38	49.92	62.63	64.81	64.00	68.16	97.38	99.93
10	5	500	50.38	49.92	62.63	64.81	67.27	71.90	99.28	100.00
2	3	800	41.64	42.05	44.42	45.40	45.52	47.10	66.67	73.58
2	5	800	41.64	42.05	44.42	45.40	46.24	48.49	72.49	80.40
5	3	800	41.67	42.04	47.57	49.98	50.55	53.98	79.76	89.16
5	5	800	41.67	42.04	47.57	49.98	52.55	56.70	87.57	96.21
10	3	800	41.66	42.04	52.99	56.36	57.52	61.78	91.50	98.35
10	5	800	41.66	42.04	52.99	56.36	60.59	65.05	96.36	99.63
2	3	1000	37.15	37.61	39.73	41.43	41.20	42.69	61.84	68.97
2	5	1000	37.15	37.61	39.73	41.43	42.30	44.33	67.09	75.57
5	3	1000	37.02	37.48	42.87	46.68	46.45	49.23	73.96	84.51
5	5	1000	37.02	37.48	42.87	46.68	48.56	52.29	81.70	92.27
10	3	1000	37.08	37.58	47.96	51.95	53.61	57.84	85.25	96.51
10	5	1000	37.08	37.58	47.96	51.95	56.97	61.33	91.59	98.73
2	3	1600	37.15	37.61	39.73	41.43	41.20	42.69	61.84	68.97
2	5	1600	37.15	37.61	39.73	41.43	42.30	44.33	67.09	75.57
5	3	1600	37.02	37.48	42.87	46.68	46.45	49.23	73.96	84.51
5	5	1600	37.02	37.48	42.87	46.68	48.56	52.29	81.70	92.27
10	3	1600	37.08	37.58	47.96	51.95	53.61	57.84	85.25	96.51
10	5	1600	37.08	37.58	47.96	51.95	56.97	61.33	91.59	98.73

Table 6.2: This table summarizes the percentage of accepted customer requests by an algorithm (in comparison with the number of accepted customer requests by the exact approach) applied on the "small" instances. In columns one to three, the number of drivers, the capacity of the convoys and the number of requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FcFs (fcfs), EST (est), RePlan (replan and replan-fbh), and the flow-based heuristic (fbh).

k	L	$\|R\|$	aic	aic-flow	fcfs	est	replan-fbh	replan	fbh
2	3	500	62.36	61.64	65.92	67.19	66.36	67.60	91.93
2	5	500	57.54	56.87	60.82	61.99	62.79	64.52	91.47
5	3	500	52.39	51.88	59.55	61.47	60.93	64.05	90.94
5	5	500	50.82	50.32	57.78	59.64	60.29	64.88	95.31
2	3	800	56.67	57.23	60.45	61.77	61.93	64.05	90.64
2	5	800	51.87	52.38	55.32	56.53	57.57	60.34	90.20
5	3	800	46.85	47.26	53.42	56.10	56.65	60.51	89.51
5	5	800	43.36	43.74	49.46	51.96	54.58	58.89	91.01
2	3	1000	53.91	54.60	57.64	60.09	59.75	61.92	89.66
2	5	1000	49.23	49.85	52.63	54.86	56.00	58.67	88.79
5	3	1000	43.91	44.45	50.77	55.25	54.95	58.21	87.54
5	5	1000	40.19	40.68	46.49	50.59	52.59	56.61	88.53
2	3	1600	47.43	48.62	50.90	53.64	54.76	56.26	86.03
2	5	1600	43.01	44.09	46.15	48.63	51.05	53.53	84.50
5	3	1600	38.73	39.66	45.32	49.74	50.48	54.03	83.71
5	5	1600	35.16	36.01	41.16	45.18	48.94	52.37	83.70

version of RePlan using the flow-based heuristic does not perform any better, this argument may seems to have only a small impact on the bad behavior of RePlan within this set of test instances. Furthermore, the flow-based version of AIP accepts less customer requests than the naive implementation of AIP despite solving all intermediate problems optimally. Therefore, it seems to be more likely that these instances show that having locally optimal solutions does not necessarily lead to a globally optimal solution.

Especially, when the number of released customer requests is small (500 customer requests), the algorithm EST outperforms the other online customer requests. In the other two cases, the performance seems to be good.

In Table 6.4 we summarize the percentage of accepted customer requests from an algorithm in comparison with the number of accepted customer requests computed by the exact approach, and in Table 6.3 we present the results compared to the upper bound computed by the first step of the flow-based heuristic.

As one can see, the number of accepted customer requests increases with the number of drivers and the capacity, but the percentage of compared to the exact approach decreases. This can be easily explained since the number of accepted customer increases faster in the exact approach than with the online algorithms.

Big instances. The trend from the medium sized test instances continues within the set of instances with 250 stations. While the algorithm EST serves most of the released customer requests, both version of RePlan serve even less customer requests than the naive implementation

Table 6.3: This table summarizes the percentage of accepted customer requests by an algorithm (in comparison with the number of accepted customer requests by the upper bound computed in step 1 of the flow-based heuristic) applied on the "medium" instances. In columns one and two, the time horizon T and the total number of released customer requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FCFS (fcfs), EST (est), REPLAN (replan and replan-fbh), the flow-based heuristic (fbh) and the exact approach (exact) with a time limit of 1 hour.

T	$\|R\|$	aic	aic-flow	fcfs	est	replan-fbh	replan	fbh	exact
120	500	79.99	75.53	85.74	86.30	84.75	81.61	100.00	100.00
240	500	80.87	76.39	87.65	88.10	86.53	83.56	100.00	99.93
120	800	64.83	62.24	69.01	70.22	71.86	67.95	97.95	99.55
240	800	65.96	63.71	71.63	73.09	75.30	71.28	100.00	97.69
120	1600	52.57	51.88	56.49	58.33	60.92	57.77	85.90	87.84
240	1600	48.40	47.84	53.19	55.09	59.15	55.55	92.02	55.73

Table 6.4: This table summarizes the percentage of accepted customer requests by an algorithm applied on the "medium" instances. Since the exact approach does not always lead to good results due to the given time limit, we compare the algorithms with the number of accepted customer requests by the maximum of the exact approach and the flow-based heuristic. In columns one and two, the time horizon T and the total number of released customer requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FCFS (fcfs), EST (est), and REPLAN (replan and replan-fbh).

T	$\|R\|$	aic	aic-flow	fcfs	est	replan	replan-fbh
120	500	79.99	75.53	85.74	86.30	84.75	81.61
120	800	65.10	62.50	69.30	70.51	72.15	68.23
120	1600	57.97	57.21	62.30	64.33	67.18	63.70
240	500	80.87	76.39	87.65	88.10	86.53	83.56
240	800	65.96	63.71	71.63	73.09	75.30	71.28
240	1600	52.62	52.02	57.82	59.90	64.30	60.39

of AIP. Only the flow-based version of AIP serves less customer requests.

Thus, it seems that looking further into the future does not lead to better but to worse results. Since only the flow-based approaches perform poorly, the problem may also lie within the solver or the framework for handling online problems.

In Table 6.6 we summarize the percentage of accepted customer requests from an algorithm in comparison with the number of accepted customer requests computed by the flow-based heuristic, while in Table 6.5 the values are compared to the upper bound.

Summary. The results of the small and medium sized test instances clearly point to two approaches leading to the best results: REPLAN and EST. Especially, REPLAN gives the bests results of the considered online algorithms (with the exception of two sets). In this case, the results computed by EST are within an acceptable range of REPLAN (see Table 6.8).

Table 6.5: This table summarizes the percentage of accepted customer requests by an algorithm (in comparison with the number of accepted customer requests by the upper bound computed in step 1 of the flow-based heuristic) applied on the "big" instances. In columns one and two, the time horizon T and the total number of released customer requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FcFs (fcfs), EST (est), RePlan (replan and replan-fbh), the flow-based heuristic (fbh) and the exact approach (exact) with a time limit of 1 hour.

T	$\|R\|$	aic	aic-flow	fcfs	est	replan-fbh	replan	fbh	exact
400	1600	96.73	88.11	97.22	97.24	88.69	88.63	99.93	-
400	3200	88.50	81.67	89.13	89.20	82.48	82.40	93.37	-

Table 6.6: This table summarizes the percentage of accepted customer requests by an algorithm applied on the "big" instances. Since the exact approach does not lead to any results due to the given time limit, we compare the algorithms with the number of accepted customer requests by the flow-based heuristic. In columns one and two, the time horizon T and the total number of released customer requests are shown. The other columns show the number of accepted customer requests by the corresponding (online) algorithm in percent. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FcFs (fcfs), EST (est), and RePlan (replan and replan-fbh).

T	$\|R\|$	aic	aic-flow	fcfs	est	replan	replan-fbh
400	1600	96.80	88.17	97.29	97.30	88.75	88.69
400	3200	94.80	87.48	95.47	95.55	88.35	88.26

On the big sized test instances, EST clearly outperforms RePlan. However, that there is not only one online algorithm performing best on all test instances, reflects the theoretical results that there does not exist a competitive online algorithm for the considered parameters.

Furthermore, the computational time differ tremendously as Table 6.7 shows. That solving the medium sized test instances are slower in average is due to a few instances with a large runtime of more than 9000 seconds.

The version of RePlan based on the flow-based heuristic solves the test instances much faster than using the exact approach to solve the intermediate offline problems: 10.2 seconds for the small instances, 58 seconds for the medium and 615.33 for the big instances. Note, in addition EST is single threaded, while RePlan uses up to 16 threads to compute the flows within a time-expanded network.

Thus, one can see that EST scales best of these three online algorithms when the number of stations or the time horizons increase. Combined with the great results on the big test instances, one can say that EST is the online algorithm to use (at least for our considered test instances).

Table 6.7: This table summarizes the runtimes in seconds of the different algorithms. In columns one to five, the different variables for the test instances are shown. The other columns show the runtimes of the corresponding (online) algorithm in seconds. Hereby, the values correspond to the average of 30 instances. We consider the following algorithms: AIP (aic), AIP computed with flows in a time-expanded network (aic-flow), FcFs (fcfs), EST (est), RePlan (replan) and RePlan based on the flow-based heuristic (replan-fbh), the flow-based heuristic (offline-fbh), and the exact approach (exact) with a time limit of 1 hour.

$\|V\|$	T	k	L	$\|R\|$	aic	aic-flow	fcfs	est	replan	replan-fbh	fbh	exact
15	120	2	3	500	0.40	20.39	0.46	0.77	58.19	7.34	14.37	3600.41
15	120	2	5	500	0.39	19.91	0.47	0.79	108.82	8.71	48.63	3600.44
15	120	5	3	500	0.39	19.67	0.50	0.67	199.61	7.64	234.33	3600.35
15	120	5	5	500	0.39	19.46	0.51	0.79	383.74	7.56	471.03	3600.25
15	120	2	3	800	0.46	21.33	0.56	0.93	85.29	7.90	12.70	3601.00
15	120	2	5	800	0.46	20.85	0.55	0.90	105.49	7.69	54.28	3600.59
15	120	5	3	800	0.46	21.15	0.63	0.84	128.91	7.85	161.50	3600.28
15	120	5	5	800	0.47	20.83	0.59	1.02	561.83	8.64	578.84	3600.55
15	120	2	3	1000	0.50	21.35	0.61	1.10	80.41	9.28	15.17	3600.33
15	120	2	5	1000	0.50	21.20	0.60	0.88	157.20	8.19	42.85	3600.39
15	120	5	3	1000	0.50	21.15	0.65	0.90	275.71	8.76	123.75	3600.34
15	120	5	5	1000	0.51	21.41	0.66	1.06	336.67	8.98	568.37	3600.44
15	120	2	3	1600	0.59	22.36	0.73	1.38	131.87	9.66	9.04	3600.68
15	120	2	5	1600	0.59	23.76	0.73	1.07	192.97	12.31	21.35	3600.64
15	120	5	3	1600	0.60	22.85	0.77	1.24	333.03	9.89	80.18	3600.59
15	120	5	5	1600	0.60	22.57	0.77	1.31	503.45	10.05	505.35	3600.47
50	120	10	5	500	0.50	35.10	0.68	0.74	255.59	33.43	602.11	3601.55
50	120	10	5	800	0.58	36.53	0.81	0.90	1782.91	40.08	602.10	3600.07
50	120	10	5	1600	0.71	38.54	1.09	1.46	3540.57	48.88	602.90	3600.05
50	240	10	5	500	0.51	61.32	0.67	1.06	569.62	50.15	605.77	3601.00
50	240	10	5	800	0.59	68.75	0.83	1.00	2694.62	55.48	605.21	3600.30
50	240	10	5	1600	0.73	73.90	1.03	1.25	4941.90	83.07	606.75	3603.93
250	400	5	5	1600	0.88	410.91	3.88	5.062	1630.80	567.79	670.21	3608.71
250	400	5	5	3200	1.04	453.68	4.19	4.407	2084.27	662.88	677.68	3617.20

Table 6.8: This table summarizes the average relative gaps between the online algorithms EST and RePlan (resp. the flow-based heuristic version of RePlan with RePlan using the exact approach). In columns one to five, show the different variables of the test instances. The other two columns show the average relative gap in percent between the corresponding algorithm and RePlan, i.e., let $\mathsf{ALG}(R)$ be the number of accepted customer requests computed by the online algorithm ALG on the sequence of customer requests R and let $\textsc{RePlan}(R)$ be the number of accepted customer requests computed by RePlan on R, then the relative gap is computed by $(\textsc{RePlan}(R) - \mathsf{ALG}(R))/\textsc{RePlan}(R)$. We consider EST (est) and RePlan with the flow-based heuristic (replan-fbh). Each cell in the latter two columns corresponds to the average computed from 30 test instances. A negative value indicates that the corresponding online algorithm achieved better results than RePlan.

T	$\lvert V\rvert$	k	L	$\lvert R\rvert$	est	replan-fbh
120	15	2	3	500	0.57	1.78
120	15	2	5	500	3.87	2.63
120	15	5	3	500	3.94	4.87
120	15	5	5	500	7.84	6.96
120	15	10	3	500	4.78	6.16
120	15	10	5	500	9.76	6.46
120	15	2	3	800	3.59	3.32
120	15	2	5	800	6.31	4.55
120	15	5	3	800	7.14	6.33
120	15	5	5	800	11.52	7.26
120	15	10	3	800	8.50	6.91
120	15	10	5	800	12.99	6.83
120	15	2	3	1000	2.93	3.45
120	15	2	5	1000	6.48	4.53
120	15	5	3	1000	4.98	5.59
120	15	5	5	1000	10.36	6.98
120	15	10	3	1000	9.78	7.17
120	15	10	5	1000	14.99	7.13
120	15	2	3	1600	4.77	2.67
120	15	2	5	1600	9.13	4.57
120	15	5	3	1600	7.95	6.56
120	15	5	5	1600	13.55	6.47
120	15	10	3	1600	11.89	6.47
120	15	10	5	1600	18.14	8.19
120	50	10	5	500	-1.85	3.71
120	50	10	5	800	2.25	5.42
120	50	10	5	1600	4.24	5.17
240	50	10	5	500	-1.83	3.44
240	50	10	5	800	2.91	5.33
240	50	10	5	1600	6.85	6.09
400	250	5	5	1600	-9.65	0.06
400	250	5	5	3200	-8.15	0.11

Part III

Static Relocation Problem

In this part, we consider the Static Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$, where the system, outgoing from the current system state $\boldsymbol{z}^0$, has to be set into a certain system state $\boldsymbol{z}^T$ within a given time horizon T. The Static Relocation Problem occurs, e.g., when the relocation step is performed only once during the night in order to prepare the system for the next morning. We consider two different aspects of the Static Relocation Problem, a quality of service aspect (Static Min-Cost Relocation Problem) and a profit oriented aspect (Static Max-Profit Relocation Problem). For each of the stated problems, we give exact and/or heuristic approaches in order to solve the problem.

The Static Relocation Problem is a special case of the Dynamic Relocation Problem where all transportation tasks are known in advance and consist of the special form $\tau = (v_i, 0, T, z_i^0 - z_i^T)$ for all v_i with $z_i^0 \neq z_i^T$. These tasks are induced by the initial and the target system state. Although the target system state is indirectly induced by the behavior of the customers, customer requests cannot be accepted or rejected. Thus, in the static version of the Relocation Problem, it does not make much sense to consider decision problems as in the Dynamic Relocation Problem, which is why we consider only optimization problems in this part.

In the first chapter of this part (Chapter 7), we consider the Static Min-Cost Relocation Problem, which aims at transferring all cars from the overfull stations to the underfull stations with minimal costs. Hereby, at the end of the time horizon, the target system state must be reached. Since the problem of finding an optimal solution is at least $\mathcal{NP}$-hard and, in general, computing an optimal solution needs enormous computation times. Furthermore, in the worst-case, the only feasible solutions for the problem are also the optimal ones. Therefore, finding a feasible solution can become already $\mathcal{NP}$-hard. Thus, we give several heuristic approaches to solve the Static Min-Cost Relocation Problem.

In Chapter 8, we consider the Static Max-Profit Relocation Problem, the profit oriented aspect of the Static Relocation Problem. Hereby, we aim at transferring as many cars as possible from the overfull stations to the underfull stations, while ensuring that the costs of the relocation step do not exceed the expected profits. Thus, the target system state does not need to be fully reached, i.e., at the end of the time horizon, some overfull stations may remain overfull, and some underfull stations may remain underfull. As the Static Min-Cost Relocation Problem, solving the max-profit version optimally is also $\mathcal{NP}$-hard. However, reaching the initial system state at the end of the time horizon is also feasible. Thus, a feasible solution can be found in linear time w.r.t. the number k of drivers, by simply initializing k empty tours.

The main problem of giving an exact approach for the Static Max-Profit Relocation Problem is that the natural formulation of an integer program is non-linear. Therefore, we show at the end of the chapter how the model can be linearized.

Finally, we present the computational results for the stated algorithms in Chapter 9. In order to evaluate the performance of the algorithms, we try to compute the optimal solution for the test instances and compare the computed results. However, even on very small instances, we were rarely able to get an optimal solution within the given time limit.

Chapter 7

Static Min-Cost Relocation Problem

In this chapter, we consider the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$, where the objective is to compute a transportation schedule with a minimal total tour length for transporting the cars between the stations. Hereby, the target system state must be reached from the initial system.

In order to solve the Static Min-Cost Relocation Problem, we first state an exact approach based on flows in a time-expanded network (Section 7.1). Since the computation of an exact solution needs a huge amount of time, we give two heuristics.

Firstly, we give a greedy heuristic, where cars are transferred from overfull stations to the closest underfull station (Section 7.2).

Secondly, we give a randomized metaheuristic based on the flows in the time-expanded network (Section 7.3). Hereby, we randomly remove arcs from the time-expanded network and compute a solution on this reduced network.

Afterwards, we present a combinatorial algorithm in Section 7.4. Hereby, we firstly compute a set of transportation tasks, which serve as input for a dial-a-ride problem. This dial-a-ride problem is then solved in the second step of the algorithm. For this combinatorial algorithm, there exists an approximation factor which we state and prove within the section.

Finally, we present a heuristic approach that computes flows in an aggregated network, and computes from these solutions a transportation schedule using a combinatorial algorithm (Section 7.5). Under certain conditions, this approach gives an optimal solution for the Static Min-Cost Relocation Problem.

The last two sections of this chapter cover some general topics. In practice, some stated approaches do not necessarily find a solution within the given time horizon, e.g., the greedy heuristic or the algorithm LiftFlow. Therefore, in Section 7.6 we show how such infeasible transportation schedules can be used in order to compute feasible ones.

It is necessary (or at least desirable) to have an exact solution in order to be able to measure the quality of a solution given by a heuristic. However, due to the tremendous amount of time needed to compute an exact solution, we are rarely able to get a proven optimal solution. In order to be able to give at least a duality gap, we describe several approaches to compute a lower bount for an optimal solution in Section 7.7.

7.1 Exact Approach: Min-Cost Flows in Time-Expanded Networks

In this section, we give an exact approach for the Static Min-Cost Relocation Problem. This approach is based on the time-expanded network with two coupled flows: a car and a driver flow.

We start by considering the computation of a preemptive transportation schedule, and the case when there is only one depot in the system. Afterwards, we briefly handle the case when there are several depots. At the end of this section, we show how the model can be modified to compute a non-preemptive transportation schedule.

The construction of the time-expanded network G_T as well as the flows basically follow the same steps as in Section 6.1. Therefore, we concentrate on the differences.

Since the customer requests are not directly involved in the Static Relocation Problem, there are no customer request arcs in G_T. Furthermore, the constraints concerning these arcs have to be removed.

In the Dynamic Relocation Problem, the system state finally reached is not known apriori. Therefore, the driver and car flows have only been initialized in Section 6.1. However, in the static version, we must ensure that every driver returns to the depot and that the target system state is reached. For that, we use as sinks the nodes (v, T), $v \in V$, for the car flow and the node (v_D, T) for the driver flow, and set their balances accordingly to $\boldsymbol{z}^T$ resp. to k for the driver flow (see Equations (7.1b) and (7.1c)).

Example 7.1. Let us consider the carsharing system from Figure 2.1 where currently the number of cars at the stations are

- C: 2, D: 0, S: 0, CHU: 2,
- U: 3, J: 2, and 0 at the rest of the stations.

The operator decides that cars shall be transferred within T time units between the stations so that the following number of cars are located at the stations:

- C: 3, D: 1, S: 1, CHU: 3,
- U: 1, J: 0, and
- at all other stations 0 cars.

This induces the following transportation tasks:

$$\tau_1 = (C, 0, T, -1), \tau_2 = (D, 0, T, -1), \tau_3 = (S, 0, T, -1), \tau_4 = (CHU, 0, T, -1),$$
$$\tau_5 = (U, 0, T, 2), \text{ and } \tau_6 = (J, 0, T, 2),$$

which are illustrated in Figure 7.1.

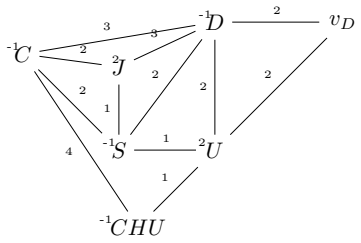


Figure 7.1: This figure illustrates transportation tasks in the static situation. The numbers at the stations correspond to the number x of cars to be picked up (if $x > 0$) and to be dropped (if $x < 0$).

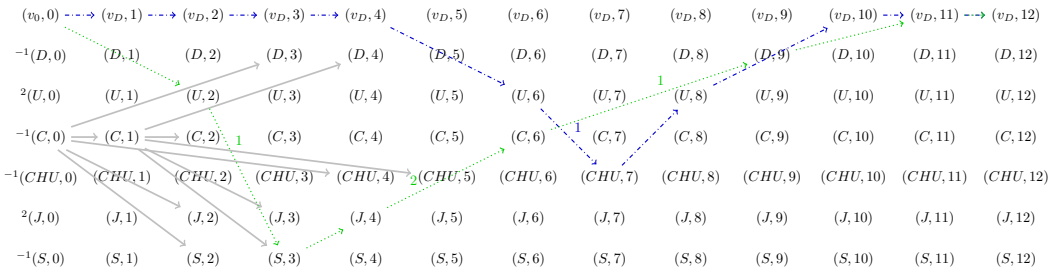


Figure 7.2: This figure illustrates the time-expanded network G_T for the static situation. For the sake of readability, only some arcs (solid arcs) of the time-expanded network and only the driver flows (dash-dotted and dotted arcs) are shown. Furthermore, only those nodes are in the figure which are important for the solution. Every node of the form (v,t) represents a station v at time t. The tour of driver 1 is illustrated as dash-dotted arcs, and the tour of driver 2 as dotted arcs. The number x of cars to be picked up ($x > 0$) or dropped ($x < 0$) at a station v are superscripted before the nodes $(v,0)$. The numbers at the arcs correspond to the number of cars transferred by the corresponding move.

From the graph G (see Figure 2.2) and the tasks τ_1 to τ_6 we construct the time-expanded network G_T and compute the car and driver flows (see Figure 7.2 for an illustration). For $T = 12$, the total tour length of an optimal solution is 17. ◊

An integer linear programming formulation for a min-cost flow problem in the time-expanded network $G_T = (V_T, A_T)$ as follows:

$$\min \sum_{a \in A_L} d(a)F(a) \tag{7.1a}$$

$$\sum_{a \in \delta^-(v,0)} f(a) = z_v^0, \sum_{a \in \delta^-(v_D,0)} F(a) = k \qquad \forall (v,0) \in V_T \tag{7.1b}$$

$$\sum_{a \in \delta^+(v,T)} f(a) = z_v^T, \sum_{a \in \delta^+(v_D,T)} F(a) = k \qquad \forall (v,T) \in V_T \tag{7.1c}$$

$$\sum_{a \in \delta^-(v,t)} f(a) = \sum_{a \in \delta^+(v,t)} f(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{7.1d}$$

$$\sum_{a \in \delta^-(v,t)} F(a) = \sum_{a \in \delta^+(v,t)} F(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{7.1e}$$

$$0 \le f(a) \le \text{cap}(v) \qquad \forall a = [(v,t),(v,t+1)] \in A_H \tag{7.1f}$$

$$f(a) \le L \cdot F(a) \qquad \forall a \in A_L \tag{7.1g}$$

$$f, F \text{ integer.} \tag{7.1h}$$

The following result can be proven in a similar way as Theorem 6.2.

Theorem 7.2. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ be a Static Relocation Problem. Then an optimal solution of system (7.1) corresponds to a preemptive transportation schedule without backhaul with minimal total tour length for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$.*

Computing a Non-Preemptive Transportation Schedule. Finally, we highlight the differences to the model when the transportation schedule must be non-preemptive.

Since we do neither allow precedences between tours nor inner preemption, a car is dropped only at underfull stations and only picked up from overfull stations; at balanced stations, cars are neither picked up nor dropped. Thus, we can consider a weighted "subgraph" $G' = (V', E', w')$ of G, where the node set contains only the overfull and underfull stations of V. The edge set E' contains all edges $(v, v') \in E$ where v, v' are overfull or underfull stations. Additionally, an edge (v, v') is added to E', if all intermediate stations of all shortest paths between v and v' are balanced stations. This ensures that the graph G' is connected and contains all shortest paths between the stations. The weight of an artificially added edge (v, v') is set to $d(v, v')$.

Then the time-expanded network G_T can be constructed from G' as before.

In the non-preemptive case, we must ensure that cars are not exchanged between the convoys. For that we distinguish between the drivers and between their loads, resulting in one driver and one car flow for each driver i on the relocation arcs: F_i and f_i. On the holdover arcs, we define a single car flow, representing the number of cars at the station.

Inner preemption and preemption between tours can be avoided by ensuring that cars are only picked up at overfull stations and dropped at underfull stations. This means that there are not more cars in a convoy leaving an overfull station than entering the station. Analogously, one can state such conditions for underfull stations. Thus, one has to ensure that the following constraints hold for every $1 \leq i \leq k$

$$\sum_{a \in \delta_L^+(v_o,t)} f_i(a) \geq \sum_{a \in \delta_L^-(v_o,t)} f_i(a), \qquad \text{for all overfull stations } v_o \in V_O \text{ and all } 1 \leq t < T, \quad (7.2)$$

$$\sum_{a \in \delta_L^+(v_u,t)} f_i(a) \leq \sum_{a \in \delta_L^-(v_u,t)} f_i(a), \quad \text{for all underfull stations } v_u \in V_U \text{ and all } 1 \leq t < T. \quad (7.3)$$

These constraints are given in the linear integer program by the constraints (7.4h) and (7.4i).

A flow on a holdover arc corresponds to cars/drivers remaining at the station in the time interval $[t, t+1]$. Since we do not allow precedences between tours, a car is dropped only at underfull stations and only picked up from overfull stations (see constraints (7.2) and (7.3)). This is the motivation that there is only one car flow f defined on the holdover arcs. Furthermore, it follows from (7.2) and (7.3) that at an overfull station the number of cars is non-increasing over time, at an underfull station the number of cars is non-decreasing over time. Thus, it holds for every reachable system state $\boldsymbol{z}$

$$\begin{aligned} z_v^0 \geq z_v \geq z_v^T, &\qquad \text{if } v \text{ is an overfull station,} \\ z_v^0 \leq z_v \leq z_v^T, &\qquad \text{if } v \text{ is an underfull station.} \end{aligned}$$

Since $\boldsymbol{z}^0$ and $\boldsymbol{z}^T$ are feasible system states by definition, this implies that every reachable system state $\boldsymbol{z}$ is feasible as well. Thus, there are no capacities needed for holdover arcs with respect to the car flow f. For the driver flows, the initialization of the flows (7.4d) and the flow conservation constraints (7.4g) imply the capacity of 1 on each arc. Moreover, the cost for all flows on such arcs are zero.

Note that the coupling constraints (7.4j) simplify (w.r.t. previous paragraph) to

$$f(a) \leq L \cdot \sum_{i=1}^{k} F_i(a).$$

To solve the Static Relocation Problem exactly, we aim at determining convoy tours with a minimal total tour length. For that, we present an integer linear programming formulation for a

min-cost flow problem in the time-expanded network $G_T = (V_T, A_T)$ as follows:

$$\min \sum_{a \in A_L} d(a) \sum_{i=1}^{k} F_i(a) \tag{7.4a}$$

$$f((v,0),(v,1)) + \sum_{a \in \delta_L^-(v,0)} \sum_{i=1}^{k} f_i(a) = z_v^0 \qquad \forall (v,0) \in V_T \tag{7.4b}$$

$$f((v,T-1),(v,T)) + \sum_{a \in \delta^+(v,T)} \sum_{i=1}^{k} f_i(a) = z_v^T \qquad \forall (v,T) \in V_T \tag{7.4c}$$

$$\sum_{a \in \delta^-(v_D,0)} F_i(a) = 1 \qquad \forall 1 \le i \le k \tag{7.4d}$$

$$\sum_{a \in \delta^+(v_D,T)} F_i(a) = 1 \qquad \forall 1 \le i \le k \tag{7.4e}$$

$$\begin{aligned} &f((v,t-1),(v,t)) + \sum_{a \in \delta_L^+(v,t)} \sum_{i=1}^{k} f_i(a) \\ &\qquad = f((v,t),(v,t+1)) + \sum_{a \in \delta_L^-(v,t)} \sum_{i=1}^{k} f_i(a) \end{aligned} \qquad \forall v \in V, 0 < t < T \tag{7.4f}$$

$$\sum_{a \in \delta^-(v,t)} F_i(a) = \sum_{a \in \delta^+(v,t)} F_i(a) \qquad \forall v \in V, 0 < t < T, \forall 1 \le i \le k \tag{7.4g}$$

$$\sum_{a \in \delta_L^+(v_o,t)} f_i(a) \ge \sum_{a \in \delta_L^-(v_o,t)} f_i(a), \qquad \forall v_o \in V_O, \forall 0 < t < T, \forall 1 \le i \le k \tag{7.4h}$$

$$\sum_{a \in \delta_L^+(v_u,t)} f_i(a) \le \sum_{a \in \delta_L^-(v_u,t)} f_i(a), \qquad \forall v_u \in V_U, \forall 0 < t < T, \forall 1 \le i \le k \tag{7.4i}$$

$$f_i(a) \le L \cdot F_i(a) \qquad \forall a \in A_L \tag{7.4j}$$

$$f, f_i \text{ integer}, F_i \text{ binary}, \tag{7.4k}$$

Note, the number of variables in the integer linear program (7.4) drastically increase compared to the number of variables in the integer linear program (7.1), resulting in an even slower computation of a solution in general.

7.2 Greedy Heuristic

In this section, we describe a simple greedy algorithm Greedy which computes a non-preemptive transportation schedule. It is widely known that a greedy can lead to a worst case for the Traveling Salesperson Problem [91], which is generalized by the Static Relocation Problem. Thus, greedy algorithms generally cannot be expected to lead to good results. However, the runtime of greedy algorithms is in general extremely fast, and thus, the computed solution can be used as a warm start for other algorithms.

We consider a metric space $M = (V, d)$ induced by a weighted graph $G = (V \cup \{v_D\}, E, w)$ representing the set of stations V, the depot v_D, the road (or logical) connections E between them, driving times $w\colon E \to \mathbb{N}$, and the metric d induced by the shortest path distances in G.

The task set $\mathcal{T}$ consists of the tasks $\tau = (v_i, 0, T, z_i^0 - z_i^T)$ for all v_i with $z_i^0 \neq z_i^T$. The output is a non-preemptive transportation schedule for the metric task system $(M, \mathcal{T})$, and the objective is to minimize its total tour length.

Basically, this heuristic approach takes the view points of the drivers, and decides to move the driver whose costs of moving to the next overfull or underfull are lowest (thus, it is basically like the nearest neighborhood algorithm for the Travelings Salesperson Problem [100, 101]).

For that we keep track of the driver movement, and search for the cheapest move for a driver. In order to be able to test a move for feasibility, we assign to each driver j some variables:

(i) the current position p^j,

(ii) the tour length t^j, and

(iii) the current load x^j.

Furthermore, every driver has a tour Γ^j.

The algorithm Greedy initializes the position for all the drivers at their starting depots, i.e., $p^j = v_D$ for a depot v_D. This implies that the other variables are initialized as follows: the tour $\Gamma^j = (v_D)$, the tour length $t^j = 0$ and the current load $x^j = 0$.

Next, we iterate through all tasks and compute the distance to the station for each driver who can serve the task. Hereby, it is tested whether the convoy capacities are violated (when the task is a pickup task), if there are cars in the convoy (when the task is a drop task) and whether traveling to the station of the task is within the time limit. If no driver can be found fulfilling all these conditions, then the algorithms stops. Otherwise, the sequence of stations Γ^j for the selected driver is updated, as well as all of its other variables. When all tasks are served, a transportation schedule is computed from the sequences Γ^j. The algorithm Greedy is summarized in Algorithm 11.

Algorithm 11 Greedy

Input: a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$
Output: a non-preemptive transportation schedule **or** false if no feasible transportation schedule can be found within the time horizon

1: for every driver j initialize an empty tour Γ^j
2: for every driver j set the current position $p^j \leftarrow v_D$
3: for every driver j set the current time $t^j \leftarrow 0$
4: for every driver j set the current load $x^j \leftarrow 0$
5: **while** not every task is served **do**
6: | search over- or underfull station s and driver j so that $d = d(p^j, s) + d(s, v_D)$ minimal **and** $t^j + d \leq T$ **and** $x^j < L$ if s overfull **and** $x^j > 0$ if s underfull
7: | **if** no such station or driver exists **then**
8: | | **return** false
9: | update current time $t^j \leftarrow t^j + d(p^j, s)$
10: | update current position $p^j \leftarrow s$
11: | append induced action and move to Γ^j
12: | update x^j by maximal load that can be picked up resp. dropped
13: add all tours to the transportation schedule $\mathcal{S}$
14: **return** $\mathcal{S}$

Theorem 7.3. *The greedy algorithm* GREEDY *(Algorithm 11) computes a feasible solution for the Static Min-Cost Relocation Problem* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$ *or returns* false *if it cannot find a solution within the given time horizon.*

7.3 Randomized Approach

In this section, we present a randomized algorithm RANDTEN to solve the Static Relocation Problem. This approach is based on the time-expanded network from Section 7.1 and, thus, can be used to compute a preemptive as well as a non-preemptive transportation schedule. The basic idea is to construct a smaller time-expanded network, and to compute the flows within this smaller network. This can be justified, since generally only a small fraction of arcs in the "complete" time-expanded network have positive flow values. Afterwards, in each round, we construct another time-expanded network and recompute the flows within this modified time-expanded network. This procedure is repeated several times.

Besides the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$, we have as additional input a feasible transportation schedule $\mathcal{S}^i = (\Gamma_1^i, \ldots, \Gamma_k^i)$ and the following parameters:

- a computation time limit $t^{\ell im}$ (for each round),
- a repetition limit Ω (the number of rounds),
- an initial acceptance probability $\rho \in [0, 1]$ (the probability for keeping an arc),
- an initial acceptance duality gap $\gamma \in [0, 1]$ (the duality gap for accepting a non-optimal solution),
- a success probability factor $\rho_a \in (1, \infty)$ and a failure probability factor $\rho_f \in (0, 1]$ (adjusts the acceptance probability depending on whether a solution is found within the time limit or not), and
- a success gap factor $\gamma_a \in (1, \infty)$ and a failure gap factor $\gamma_f \in (0, 1]$ (adjusts the acceptance duality gap depending on whether a solution is found within the time limit or not).

In each round i, we build a directed graph $G_T^i = (V_T^i, A_T^i)$, with $A_T^i = A_H^i \cup A_L^i$, similar to the time-expanded network of previous section. In this approach, we do not construct the complete time-expanded network, but a smaller version of the network. Hereby, a relocation arc is added to A_T^i with a probability of γ. Afterwards, all relocation arcs traversed by $\mathcal{S}^i$ are added to A_T^i. It is easy to see that the parameter γ influences the size of the network G_T^i. As before, the cars and drivers will form two flows f and F through G_T^i. The integer linear program (7.1) can then be used without any modifications.

Note, due to the "missing" nodes and arcs in the time-expanded network, an optimal solution in the reduced time-expanded network does not necessarily correspond to an optimal solution for the given Static Min-Cost Relocation Problem.

The algorithm RANDTEN is performed in several rounds. Hereby, there are Ω rounds.

Each round, has the following basic steps:

(i) construct time-expanded network G_T^i,

(ii) compute flows f and F in G_T^i,

(iii) adjust the acceptance probability ρ and the acceptance duality gap γ.

Hereby, the solution computed in (ii) is used to construct the time-expanded network in step (i), and the computation is stopped with the solution f^* and F^*, when the duality gap is less than or equal to γ, or if the runtime of solving the integer linear program is greater than or equal to the time limit t^{lim}. If no solution is found within the time-limit, then f^* and F^* are set to the initial solution f^i and F^i. Similarly, if only solutions with a duality gap greater than γ are found, then f^* and F^* are set to the best solution which has been found during the computation.

At the end of each step, we adjust the acceptance probability ρ and the acceptance duality gap γ (see (iii)). Hereby, if a solution with a duality gap of at most γ has been found within the time-limit t^{lim}, we set

$$\rho := \rho_a \cdot \rho \text{ and } \gamma := \gamma_a \cdot \gamma.$$

If no solution or only solutions with a duality gap of more than γ could be found within the time-limit, we set

$$\rho := \rho_f \cdot \rho \text{ and } \gamma := \gamma_f \cdot \gamma.$$

Since $\rho_a \in (1, \infty)$ and $\rho_f \in (0, 1]$ it follows that

- if an "optimal" solution is found within the time-limit, we construct a network with more arcs (with a high probability),
- if no "optimal" solution is found within the time-limit, we construct a network with less arcs (with a high probability).

Analogously, we can conclude that the acceptable duality gap decreases if a solution is found, and increases if no solution is found. The idea behind this strategy is to try to find even better solutions after a successful run, while if we could not find a better solution in the current round then we try to improve the solution with relaxed conditions. Since each round has a time-limit t^{lim}, it follows that the runtime of RandTEN is limited by $t^{lim} \cdot \Omega$. The algorithm RandTEN is summarized in Algorithm 12.

Algorithm 12 RandTEN

Input: a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$, a transportation schedule $\mathcal{S}$, a time limit t^{lim}, a repetition limit Ω, an initial acceptance probability ρ, an initial acceptance duality gap γ, a success probability factor ρ_a and failure probability factor ρ_f, a success gap factor γ_a and failure gap factor γ_f

Output: a non-preemptive transportation schedule

1: **for** $i = 1 \dots \Omega$ **do**
2: construct time expanded network G_T from $\mathcal{S}$, ρ and γ
3: compute flows in G_T^i with a time limit of t^{lim} time units
4: update $\mathcal{S}$, ρ and γ
5: **return** $\mathcal{S}$

Since a transportation schedule is given as input and this solution is used to construct the time-expanded network, and since this solution is kept if no better solution is found, the next result follows with above and Theorem 7.2.

Theorem 7.4. *Algorithm* RandTEN *always returns a feasible solution for the Static Min-Cost Relocation Problem* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$.

Finally, we give some remarks on the approach RandTEN. Since the arcs are added randomly, it may not even possible to find a feasible solution at all, if no initial solution would be given.

Thus, it is crucial for this approach that one can find an initial solution fast (e.g., with the greedy heuristic GREEDY from Section 7.2).

Instead of a success gap factor $\gamma_a \in (1, \infty)$, a failure gap factor $\gamma_f \in (0, 1]$, a success probability factor $\rho_a \in (1, \infty)$ and a failure probability factor $\rho_f \in (0, 1]$ one can use success and failure functions to adjust the gap and probability factors.

If the repetition limit Ω and time limit t^{lim} are large enough, it follows from the infinite monkey theorem that an optimal solution is found almost surely. However, this approach does not give any lower bound. Thus, with this approach alone, one never knows when an optimal solution is found.

In order to compute a non-preemptive transportation schedule, one can apply this method with the modified time-expanded network from Section 7.1 (see Paragraph 'Computing a Non-Preemptive Transportation Schedule').

7.4 The combinatorial algorithm ReOpt

In this section, we describe in detail the strategy REOPT proposed in [112] and in [35] to solve a Static Min-Cost Relocation Problem. The combinatorial algorithm REOPT computes a non-preemptive transportation schedule in three steps. Firstly, we construct a weighted complete bipartite graph and find a matching between overfull and underfull stations with minimal edge weight. Each edge in this matching corresponds to a transport request between two stations. Secondly, tours for all convoys are constructed (using a heuristic insertion technique) serving each transport request. Since the transport requests stemming from the minimum matching do not necessarily lead to optimal convoy tours, the final step is to iteratively augment the tours by "rematching" certain origin/destination pairs, and by reinserting accordingly adapted moves in such a way that the total tour length decreases.

7.4.1 First step: Compute transport requests

In the first step, we compute transport requests of the form (v_o, v_u, x), where v_o is an overfull station, v_u an underfull station and x is the number of cars to be transported from v_o to v_u. For that, we construct a weighted complete bipartite graph $B = (V_O \cup V_U, A, \boldsymbol{w}, \boldsymbol{p})$, where V_O is the set of overfull stations and V_U the set of underfull stations, and consider a restricted vector $\boldsymbol{w} \in \mathbb{R}^{|A|}$ of edge weights (reflecting the distance between the two adjacent stations) and a vector $\boldsymbol{p} \in \mathbb{N}^{|V_O \cup V_U|}$ of node weights reflecting the number $p_v = |z_v^0 - z_v^T|$ of cars which have to be moved in or out the corresponding station v.

Define a perfect p-matching in B to be a multiset $x\colon A \to \mathbb{N}$ of the edges such that for each node $v \in V_O \cup V_U$, exactly p_v incident edges are selected, counted with multiplicities x_a. Note that by construction of $\boldsymbol{p} \in \mathbb{N}^{|V_O \cup V_U|}$, the existence of such a perfect p-matching is ensured by $\sum_{v_o \in V_O} p_o = \sum_{v_u \in V_U} p_u$ since $\sum_{v \in V} z_v^0 = \sum_{v \in V} z_v^T$. The goal is to find a perfect p-matching x with minimal edge weight $\sum_{a \in A} w_a x_a$, including multiplicities. The problem can be formulated by the following integer linear program

$$\min \sum_{a \in A} w_a x_a, \tag{7.5a}$$

$$\sum_{a \in \delta^+(v_o)} x_a = p_{v_o}, \qquad \forall v_o \in V_O \tag{7.5b}$$

$$\sum_{a \in \delta^+(v_u)} x_a = p_{v_u}, \qquad \forall v_u \in V_U \tag{7.5c}$$

x_a integer. (7.5d)

Note that the constraint matrix is totally unimodular and, thus, the problem can be solved efficiently [136].

Each selected matching edge $a = v_o v_u$, with $v_o \in V_O$ and $v_u \in V_U$, corresponds to a transport request for x_a cars from station v_o to station v_u. The set TR of all such transport requests provides the input for a Pickup and Delivery Problem (PDP) which has to be solved subsequently in order to construct tours for all convoys serving each transport request.

7.4.2 Second step: Serving the transport requests

In this section, we give an algorithm Pdp-Insert which solves the PDP using heuristic insertion techniques. The input for Pdp-Insert is the complete weighted graph $G = (V_O \cup V_U \cup V_D, E, w)$, where V_O is the set of overfull stations, V_U the set of underfull stations and V_D is the set of the depots, the total number k of drivers, the convoy capacity L, and the set of transport requests TR. The output of Pdp-Insert is a non-preemptive transportation schedule for the drivers, which serves all transport requests in TR within the time horizon $[0, T]$.

For that, we define some further notions. Let Γ_j be a tour for a driver j and let $tr = (v_o, v_u, x)$ be a transport request. We say that tr is served by Γ_j if there exists a pickup action $a^o = (\cdot, v_o, y)$ and a drop action $a^u = (\cdot, v_u, -y)$ in Γ_j, so that $t(a^o) < t(a^u)$. Hereby, tr is fully served if $y = x$ and partially served if $y < x$. By $y^{tr}(\Gamma_j) := y$ we denote the number of cars served from tr by Γ_j. A transport request $tr = (v_o, v_u, x)$ can be (partially) inserted into a tour Γ_j, serving y cars, as follows:

- select a move $m = (j, v, t^v, w, t^w, x_m)$ where v_o shall be inserted,
- remove m from Γ_j,
- add move $(j, v, t^v, v_o, t^v + d(v, v_o), x_m)$, action (j, v_o, y), move $(j, v_o, t^v + d(v, v_o), w, t^v + d(v, v_o) + d(v_o, w), x_m + y)$ to the tour,
- update departure and arrival times of all successive moves and actions,
- do analogous steps for u.

This yields a new tour Γ'_j which (partially) serves tr. By applying the opposite steps, tr can be removed from Γ'_j, which yields Γ_j.

Now let tr be (fully or partially) served by Γ, and Γ_j^w be the tour derived from Γ_j without serving tr. We denote the marginal costs per load unit $CM(tr, \Gamma_j)$ by

$$CM(tr, \Gamma_j) = \frac{len(\Gamma_j) - len(\Gamma_j^w)}{y^{tr}(\Gamma_j)}.$$

Now the algorithm Pdp-Insert can be described as follows:

(i) For each driver j initialize the tour so that it starts and ends in the drivers depot $v_D \in V_D$, i.e., initialize the tour with the move $(j, v_D, 0, v_D, 0, 0)$.

(ii) Choose a transport request tr that has an origin or a destination already in a tour, else randomly select one.

(iii) Calculate the marginal cost per load unit for inserting this transport request to each possible tour. Hereby, take the number of cars to be transported into account (i.e., respect the convoy capacity) as well as the time. Select the tour with the minimum marginal cost per load unit and insert the transport request into this tour.

(iv) If the transport request tr is fully served, remove it from TR. Otherwise, it is partially served by a tour Γ. Then subtract the number of cars inserted from the load of the transport request, i.e., remove tr from TR and add a new transport request $tr' = (v_o, v_u, x - y^{tr}(\Gamma))$ to TR.

(v) Repeat these steps until all the transport requests are fully served.

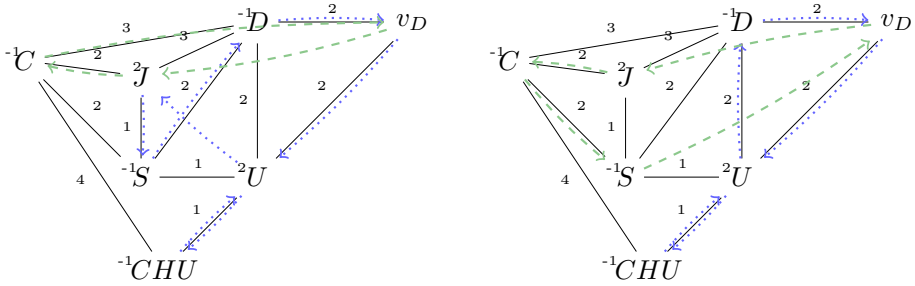


Figure 7.3: In each figure (left and right side), a subgraph G of the graph of the carsharing system from Figure 2.2 is shown. There are two tours (dashed and dotted), one for each of the two drivers (server capacity $L = 2$), giving a non-preemptive transportation schedule. The numbers at the stations show the amount of cars to be moved from (> 0) or to the station (< 0). On the left side, the transportation schedule before the reoptimization step is shown with a total tour length is 22; on the right side, the transportation schedule after this step has a total tour length of 19 which is only 2 units away from the optimal solution (see Example 7.1 for a proof).

The algorithm Pdp-Insert computes a non-preemptive transportation schedule. However, the transportation schedule created from the transport requests stemming from the minimal perfect p-matching does not necessarily lead to optimal tours. Therefore, the final step is to iteratively augment the tours by "rematching" certain origin/destination pairs, i.e., to reinsert accordingly adapted moves in such a way that the total tour length decreases.

7.4.3 Third step: Reoptimization

The algorithm defined here involves the two previous steps: computation of transport requests and the algorithm Pdp-Insert. The input for the reoptimization step is a transportation schedule $\mathcal{S}$ serving all transport requests in TR and two natural numbers $\Delta, N \in \mathbb{N}$. The output is a transportation schedule having a total tour length less or equal to the total tour length of $\mathcal{S}$. The algorithmic scheme of the reoptimization step is as follows:

(i) From $\mathcal{S}$ we withdraw the N transport requests with highest marginal cost per load: $\mathrm{TR}' = \{(v_o^1, v_u^1, x^1), \ldots, (v_o^N, v_u^N, x^N)\}$.

(ii) From the withdrawn transport requests we compute sets of over- and underfull stations, i.e., the set $V_{O^*} = \{v_o \in V_O \mid (v_o, \cdot, \cdot) \in \mathrm{TR}'\}$, the set $V_{U^*} = \{v_u \in V_U \mid (\cdot, v_u, \cdot) \in \mathrm{TR}'\}$ and the vector $\boldsymbol{x} \in \mathbb{N}^{|V_{O^*}| \cdot |V_{U^*}|}$ by $x_{ou} = \min\{p_o, p_u\}$, where $p_o = \sum_{(v_o, \cdot, x) \in \mathrm{TR}'} x$ and $p_u = \sum_{(\cdot, v_u, x) \in \mathrm{TR}'} x$.

(iii) For every pair $(v_o, v_u) \in V_{O^*} \times V_{U^*}$ and every tour Γ we compute the additional marginal cost $CM^+(\Gamma, (v_o, v_u))$, where $CM^+(\Gamma, (v_o, v_u)) = \Gamma^{(v_o, v_u)} - \Gamma$ and $\Gamma^{(v_o, v_u)}$ is the tour after inserting a transport request $(v_o, v_u, 1)$. Let $w_{ou} = \min_{\Gamma \in \mathcal{S}} CM^+(\Gamma, (v_o, v_u))$ be the minimal additional marginal cost.

(iv) Next we generate a weighted complete bipartite graph $B = (V_{O^*} \cup V_{U^*}, A, \boldsymbol{w}, \boldsymbol{p})$ and compute a minimal perfect p-matching (as in Step 1). From the minimal perfect p-matching, transport requests are generated, which serve as input for the algorithm Pdp-Insert (as in Step 2).

(v) Redo these steps Δ times.

(vi) Finally, we return the best found transportation schedule, i.e., the one with the smallest total tour length.

Figure 7.3 shows an example for the reoptimization step improving a transportation schedule stemming from the minimal perfect p-matching of Step 1.

The algorithm ReOpt is summarized in Algorithm 13.

Algorithm 13 ReOpt

Input: a Static Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$, integers N, Δ
Output: a non-preemptive transportation schedule

```
Find a minimal perfect p-matching (Step 1)
while counter < Δ do
    update counter
    construct k tours by Pdp-Insert serving all transport requests (Step 2)
    rematch after withdrawing the N requests that have highest additional marginal costs in
        their tours (Step 3)
return transportation schedule of smallest found total tour length
```

Finally, we give some comments about the complexity of the algorithm ReOpt. The minimal perfect p-matching of Step 1 of the algorithm ReOpt (Section 7.4.1), can be computed in polynomial time. Since constructing an optimal transportation schedule from the minimal perfect p-matching (Section 7.4.2) results in a dial-a-ride problem, this step is at least $\mathcal{NP}$-hard. Thus, the second step cannot be solved in polynomial time unless $\mathcal{P} = \mathcal{NP}$ holds. Therefore, the total complexity of ReOpt is at least in $\mathcal{NP}$.

7.4.4 Approximation factor

In this section, we show that the algorithm ReOpt achieves a finite approximation factor based on the capacity of the convoys. That ReOpt computes a non-preemptive transportation schedule for the Static Relocation Problem has already been observed in the previous section. In order to prove the approximation factor, we first introduce some definitions, as well as state and prove some lemmas.

Firstly, from a given tour, we construct a new tour where each action picks up (resp. drops) exactly one car. Considering such tours only simplifies several technical issues, like estimating the number of consecutive pickup actions. Secondly, we construct a new transportation schedule from an optimal transportation schedule and a minimal perfect p-matching. Finally, we compare

the lengths of an optimal transportation schedule $\mathcal{S}^*$, a transportation schedule $\mathcal{S}^p$ derived from a minimal perfect p-matching, and the constructed transportation schedule $\mathcal{S}^t$ and show that

$$\ell(\mathcal{S}^p) \leq \ell(\mathcal{S}^t) \leq (L+1)\ell(\mathcal{S}^*)$$

holds, which proves the stated approximation factor. Hereby, we construct the transportation schedule $\mathcal{S}^p$ by taking moves and actions of the optimal transportation schedule $\mathcal{S}^*$ and by constructing moves from the transport requests of the minimal perfect p-matching. Then, the approximation factor $(L+1)$ emerges from the maximal number of moves corresponding to moves through the system in order to pickup cars and from moves which are serving a transport requests.

We start by showing how to construct a tour $\overline{\Gamma}$ from a given tour $\Gamma = (m_1, a_1, \ldots, a_{n-1}, m_n)$, where in each action exactly one car is picked up or dropped.

For every $i \in \{1, \ldots, n\}$ do

- add move m_i to $\overline{\Gamma}$,
- for every action $a_i = (j, v, x)$ where more than one car is picked up from a station v at time t_v, we "replace" the action by actions each picking one car and waiting moves (with 0 waiting time) between these actions, i.e., if $x > 1$ then add the following x actions and $x-1$ moves $((j, v, 1), (j, v, t_v, v, t_v, x_i + 1), \ldots, (j, v, t_v, v, t_v, x_i + x - 1), (j, v, 1))$ are added to $\overline{\Gamma}$,
- for every action $a_i = (j, v, x)$ where more than one car is dropped at a station w at time t_w, we "replace" the action by actions each picking one car and waiting moves (with 0 waiting time) between these actions, i.e., if $x < -1$ then add the following x actions and $x-1$ moves $((j, w, -1), (j, w, t_w, w, t_w, x_i - 1), \ldots, (j, w, t_w, w, t_w, x_i - x + 1), (j, w, -1))$ are added to $\overline{\Gamma}$,
- every action $a^i = (j, v, x)$ with $-1 \leq x \leq 1$ is added unchanged to $\overline{\Gamma}$.

The tour $\overline{\Gamma}$ is called a uniform tour corresponding to Γ. A transportation schedule containing only uniform tours is called uniform transportation schedule.

Note that a uniform tour is indeed a tour. Furthermore, note that there exists exactly one uniform tour corresponding to a tour (if no unnecessary empty actions and waiting moves are added), but from a uniform tour, one can generally derive several non-uniform tours.

Since we consider non-preemptive transportation schedules, i.e., there does not exist a tour depending on another tour, empty actions can be safely removed from any tour in a transportation schedule (some moves may need to be adjusted accordingly). For the rest of this section, we assume that no action is empty.

Example 7.5. Let us consider the graph and the dashed tour for driver 1 from the right side of Figure 7.3. The tour is then given by

$$\begin{aligned}
\Gamma = \{ & \\
& (1, v_D, 0, J, 4, 0), \\
& (\mathbf{1}, \mathbf{J}, \mathbf{2}), \\
& (1, J, 4, C, 6, 2), \\
& (1, C, -1), \\
& (1, C, 6, S, 8, 1), \\
& (1, S, -1), \\
& (1, S, 8, v_D, 11, 0) \\
\}, &
\end{aligned}$$

and the corresponding uniform tour is then

$$\begin{aligned}\overline{\Gamma} = \{ & \\ & (1, v_D, 0, J, 4, 0), \\ & (\mathbf{1}, \mathbf{J}, \mathbf{1}), \\ & (\mathbf{1}, \mathbf{J}, \mathbf{4}, \mathbf{J}, \mathbf{4}, \mathbf{1}), \\ & (\mathbf{1}, \mathbf{J}, \mathbf{1}), \\ & (1, J, 4, C, 6, 2), \\ & (1, C, -1), \\ & (1, C, 6, S, 8, 1), \\ & (1, S, -1), \\ & (1, S, 8, v_D, 11, 0) \\ \}. &\end{aligned}$$

The "replaced" action is highlighted with bold fonts, all other actions $a \in act(\Gamma)$ have already the form $\Delta x(a) = \pm 1$. ◊

The following lemma is a direct conclusion from the construction of an uniform tour.

Lemma 7.6. *Let $\overline{\Gamma} = (m_1, a_1, \ldots, m_{n-1}, a_{n-1}, m_n)$ be a uniform tour for driver j. Then there are at most L consecutive pickup (resp. drop) actions in the sequence $act(\overline{\Gamma})$.*

Next, we construct a graph G from a given tour, where the set of nodes corresponds to the actions, and the set of arcs to the moves of the tour. Afterwards, we combine this graph with transport requests (leading to a graph G^t), which then helps us to construct another tour (from this tour we finally gain the transportation schedule $\mathcal{S}^t$). This constructed tour has some nice properties with respect to the number of traverses of each arc of G^t, which finally helps us to prove our main result (Theorem 7.15).

From a given tour $\Gamma = (m_1, a_1, m_2, \ldots, a_{n-1}, m_n)$, we construct a directed weighted graph $G = (V^+ \cup V^- \cup V^=, A, w)$, where

(i) the set of nodes $V = V^+ \cup V^- \cup V^=$ corresponds to the actions in Γ, V^+ corresponds to the set of pickup actions, V^- to the set of drop actions, and $V^=$ to an artificially added empty action representing the depot v_D, i.e., $V^= = \{(\cdot, v_D, 0)\}$;

(ii) there is an arc from $v \in V$ to $v' \in V$ if $v = a_j$ and $v' = a_{j+1}$ for a $1 \leq j \leq n$, furthermore there is an arc from $(\cdot, v_D, 0)$ to a_1 and from a_{n-1} to $(\cdot, v_D, 0)$;

(iii) the weight function w corresponds to the distances between the origin and destination stations of the corresponding moves, i.e., we set $w(a_j, a_{j+1}) = d(orig(m_{j+1}), dest(m_{j+1}))$.

We call such a graph a tour graph for Γ, the set A is called the set of tour arcs.

Note that one can assign to every arc $a \in A$ of a tour graph a move $m \in \Gamma$. Then m is called the corresponding move to a.

Example 7.7. The tour graph for the tour $\overline{\Gamma}$ from Example 7.5 is illustrated in Figure 7.4. ◊

Analogously, to a uniform tour we now define a set of uniform transport requests. A transport request (v_i, w_i, x_i) is called uniform if $x_i = 1$. Obviously, every set of transport requests can be transformed into a set of uniform transport requests by splitting every transport request

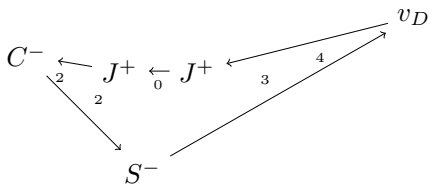


Figure 7.4: This figure shows the tour graph for the uniform tour $\overline{\Gamma}$ from Example 7.5. The weights of the arcs correspond to the shortest distance between the stations. In the tour graph, a pickup action at station $v \in V$ is denoted by v^+ and a drop action at v by v^-.

(v_i, w_i, x_i) into x_i uniform transport requests. A set of transport requests TR is called set of uniform transport requests if every transport request $r \in \text{TR}$ is uniform.

To each uniform transport request $(v, w, 1) \in \text{TR}$, we can now assign two actions for a driver j, one pickup $(j, v, 1)$ and one drop action $(j, w, -1)$. Hereby, every action is assigned to exactly one transport request.

Let $G = (V, A, w)$ be a tour graph and let TR be a set of transport requests. Then we construct a directed weighted graph $G^t = (V, A \cup A^t, w)$, where A^t is the set of transport request arcs, which consist of arcs corresponding to the transport requests in TR, i.e., for a transport request $r \in \text{TR}$ there is an arc between $v, v' \in V$ if v is the assigned pickup action of r and v' the assigned drop action of r. The weight of a transport request arc is equal to the distance between the locations of the two assigned actions. The graph G^t is called a transport graph (see Figure 7.5 for an illustration).

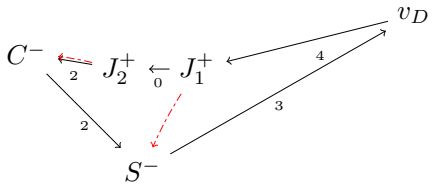


Figure 7.5: This figure shows a transport graph for the uniform tour $\overline{\Gamma}$ from Example 7.5 and the set of transport requests $\text{TR} = \{(J, C, 1), (J, S, 1)\}$. The weights of the arcs correspond to the shortest distance between the connected two stations. In the tour graph, a pickup action at station $v \in V$ is of the form v^+ and a drop action is of the form v^-. The dash-dotted arcs correspond to transport request arcs.

Remark 7.8. Let $G = (V, A, w)$ be a tour graph for a tour $\overline{\Gamma}$ for driver j and let TR be a set of transport requests. From a transport graph $G^t = (V, A \cup A^t, w)$ for G and TR, one can construct a new tour $\overline{\Gamma}^t$ that serves all transport requests in TR, as follows:

- Start in the depot v_D (resp. the node corresponding to the depot).
- Consider the next tour arc $a \in A$ or non-traversed transport request arc $a^t \in A^t$.
- If a tour arc $a = (v, w)$ is selected, we add a corresponding move m to $\overline{\Gamma}^t$ from $loc(v)$ to $loc(w)$ with $x_m(m) = 0$.
- If necessary, add empty actions (or merge the moves).

- If a transport request arc $a^t = (v, w)$ is selected, we add the pickup action v, a move $(j, loc(v), \cdot, loc(w), \cdot, 1)$ and a drop action w to $\overline{\Gamma}^t$.
- When all transport requests are served, return to the depot by following tour arcs until the depot is reached.

The departure and arrival times of a move m_j are directly induces by the departure and arrival times of the moves preceding $m_1, \ldots, m_{j-1}$.

Note that following this construction, we always construct a tour. However, without further restrictions this easy construction does not ensure an upper bound on the number of traverses of an arc (later in Algorithm 14 we give a refined construction which ensures an upper bound on the number of traverses of an arc). ⧫

When we speak about a constructed tour (from a transport graph G^t and a set of transport requests TR), we mean a tour Γ which is constructed using only the arcs from G^t and which serves all transport requests from TR.

Example 7.9. Let us consider the transport graph from Figure 7.5. A possible new tour serving all transport requests TR constructed from the transport graph, is then given by

$$\begin{aligned}
\overline{\Gamma}^t = \{ & \\
& (1, v_D, 0, J_1, 4, 0), \\
& (1, J_1, 0), \\
& (1, J_1, 0, J_2, 4, 0), \\
& (1, J_2, 1), \\
& (1, J_2, 4, C, 6, 1), \\
& (1, C, -1), \\
& (1, C, 6, S, 8, 0), \\
& (1, S, 0), \\
& (1, S, 8, v_D, 11, 0), \\
& (1, v_D, 0), \\
& (1, v_D, 11, J_1, 15, 0), \\
& (1, J_1, 1), \\
& (1, J_1, 15, S, 17, 1), \\
& (1, S, -1), \\
& (1, S, 17, v_D, 19, 0), \\
\} &
\end{aligned}$$

(see Figure 7.6 for an illustration). Hereby, the stations b_1 and b_2 both correspond to the station b. However, for the sake of readability, we use b_1 and b_2 instead of b. ◊

Our goal is to construct a new tour from a transport graph, constructed from an optimal tour, and from a set of transport requests, which is generated from a minimal perfect p-matching. Then, we show that this constructed tour has an approximation factor based on the capacity of the convoys. For that we define a function which returns for each arc of a transport graph the number of traverses of the arc during the construction of the tour.

Let Γ be a uniform tour, TR be a set of transport requests and $G^t = (V, A \cup A^t, w)$ be a transport graph for Γ and TR. Furthermore, let Γ^t be a constructed tour from G^t and TR.

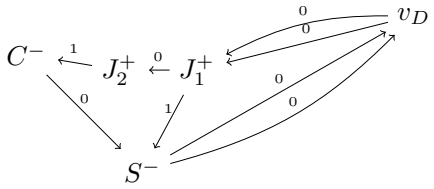


Figure 7.6: This figure shows a possible tour constructed from the transport graph of Figure 7.5 serving the transport requests $\mathrm{TR} = \{(J, C, 1), (J, S, 1)\}$.

Then, we consider a so-called traverse counter function, which is a function $f^a : A \to \mathbb{N}$ which reflects how often a tour arc $a \in A$ is traversed during the construction of Γ^t.

Since the tour arcs of a transport graph emerge from moves, the tour arcs can be directly translated back to a move when a tour is constructed from a transport graph. With the function f^a, we count the traverses of the tour arcs in a constructed tour, which enables us to compute the distance traveled by the driver when no car is transferred from one station to another. However, the transport request arcs correspond to performing the actions to serve the underlying transportation tasks. Unlike to the tour arcs, the distance traveled by a move corresponding to a transport request arc is not given and, thus, (at the moment) unknown. Therefore, we handle the transport request arcs differently from the tour arcs and in a later step.

Example 7.10. Let us consider the transport graph from Figure 7.5 and the tour $\overline{\Gamma}^t$ from Example 7.9. The traverse counter function f^a is then

$$\begin{aligned} f^a(v_D, J_1^+) &= 2, \\ f^a(J_1^+, J_2^+) &= 1, \\ f^a(J_2^+, C^-) &= 0, \\ f^a(C^-, S^-) &= 1, \\ f^a(S^-, v_D) &= 2. \end{aligned}$$

Note that there are two arcs in the transport graph from Figure 7.5 between J_2^+ and C^-, one tour arc and one request arc. Hereby, the tour $\overline{\Gamma}^t$ is constructed not by traversing the tour arc (J_2^+, C^-) but instead by traversing the transport request arc from J_2^+ to C^- is traversed. Thus, we have $f^a(J_2^+, C^-) = 0$. ◊

Lemma 7.11. *Let Γ be a uniform tour starting and ending in depot v_D, TR a set of transport requests, and $G^t = (V^+ \cup V^- \cup V^=, A \cup A^t, w)$ a transport graph for Γ and TR. Furthermore, let $\overline{\Gamma}$ be the constructed tour from Algorithm 14 and let f^a be a traverse counter function for $\overline{\Gamma}$. Then $f^a(a) \leq L + 1$ holds for all tour arcs $a \in A$. More specifically, we have*

(i) $f^a(a) \leq L + 1$ holds for all tour arcs $a = (v, w) \in A$ with $v \in V^-$,

(ii) $f^a(a') \leq L$ holds for all tour arcs $a' = (v', w') \in A$ with $v' \in V^+$.

Proof. The first statement we prove is that the nodes corresponding to a pickup action increase the number of traverses, and nodes corresponding to a drop action decrease the number of traverses.

Claim 7.11.1. Let $a_1, a_2 \in A$ be two tour arcs with $a_1 = (v, w)$ and $a_2 = (w, u)$ (i.e., a_2 is the successive tour arc of a_1). Then $|f^a(a_1) - f^a(a_2)| = 1$ holds. More specifically, it holds

(i) $f^a(a_1) - f^a(a_2) = 1$ if $w \in V^+$,

(ii) $f^a(a_1) - f^a(a_2) = -1$ if $w \in V^-$.

Proof. The movement of a driver can be modeled by a flow $f : A \cup A^t \to \mathbb{N}$. Then the flow conservation constraint $\sum_{a \in \delta^+(v)} f(a) = \sum_{a \in \delta^-(v)} f(a)$ must hold. Hereby, the transport request arcs must be taken into consideration as well. It is easy to see that there always exists a flow f with $f(a) = f^a(a)$ for all tour arcs $a \in A$. Since every node corresponding to a pickup action has exactly one outgoing transport request arc and every drop action has exactly one incoming transport request arc, and due to the flow conservation constraint it follows that

(i) $f^a(a_1) - f^a(a_2) = 1$ if $w \in V^+$,

(ii) $f^a(a_1) - f^a(a_2) = -1$ if $w \in V^-$

holds and, thus, proves the statement. ◊

There are several consequences from this claim. Firstly, we show in the next corollary that the difference of the number of traverses of two different tour arcs can be bounded from above. Secondly, we show a relation between the number of traverses of an arc and the number of cars that are transfered in the corresponding move within the constructed tour.

Claim 7.11.2. Let $a_1, \ldots, a_\tau \in A$ be successive tour arcs (i.e, $a_1 = (v_1, v_2), a_2 = (v_2, v_3), \ldots, a_\tau = (v_\tau, v_{\tau+1})$), and let $t : A \to \mathbb{N}$ be a function that returns the number of cars transfered in the corresponding moves in $\overline{\Gamma}$. Then

(i) $|f^a(a_1) - f^a(a_\tau)| \leq L$, and

(ii) $f^a(a) + t(a) = const$ holds for all tour arcs $a \in A$. Especially it holds $f^a(a) + t(a) = f^a(a') + t(a')$ for every pair of tour arcs $a, a' \in A$.

Proof. "(i)" Lemma 7.6 shows that in every tour there are maximal L consecutive pickup actions and maximal L consecutive drop actions. Furthermore, the number of cars in a convoy must not exceed the capacity L. Since $|f^a(a_j) - f^a(a_{j+1})| = 1$ holds, the difference cannot be greater than L.

"(ii)" Let $a \in A$ be a tour arc and let $a' \in A$ be the successive tour arc. We proof the statement with two cases, when the end node of a is a node in V^+ and when it is a node in V^-.

Case i (the end node of a is a node in V^+): From Claim 7.11.1 (i) it follows that $f^a(a) - f^a(a') = 1$. Since the end node of a corresponds to a pickup action, the number of cars transported in $\overline{\Gamma}$ is increased by 1, i.e., we have $t(a) - t(a') = -1$. Thus, it follows $f^a(a) - f^a(a') + t(a) - t(a') = 0$ and, therefore, $f^a(a) + t(a) = f^a(a') + t(a')$.

Case ii (the end node of a is a node in V^-): From Claim 7.11.1 (ii) it follows that $f^a(a) - f^a(a') = -1$. Since the end node of a corresponds to a drop action, the number of cars transported in $\overline{\Gamma}$ is decreased by 1, i.e., we have $t(a) - t(a') = 1$. Thus, it follows $f^a(a) - f^a(a') + t(a) - t(a') = 0$ and, therefore, $f^a(a) + t(a) = f^a(a') + t(a')$.

We have shown, that the statement holds for two successive arcs. The statement for arbitrary pairs now follows by iteratively applying the two cases. ◊

Next, we show that the maximum value of a traverse counting function is always on the incoming and outgoing tour arcs of the depot.

Claim 7.11.3. Let $V^= = \{v_D^=\}$. Then

(i) $f^a((v_D^=, \cdot)) = f^a((\cdot, v_D^=)) \geq f^a(a')$ for all tour arcs $a' \in A$,

(ii) $f^a((v_D^=, \cdot)) = f^a((\cdot, v_D^=)) > f^a(a')$ for all tour arcs $a'' = (v, w) \in A$, with $v \in V^+$.

Proof. Let $t : A \to \mathbb{N}$ be a function that returns the number of cars transfered in the corresponding moves in $\overline{\Gamma}$. From Lemma 7.11 (ii) we know that $f^a(a) + t(a) = f^a(a') + t(a')$ for $a, a' \in A$. At the beginning and end of a tour, the number of cars in a convoy is always 0 and, thus, we have $f^a((v_D^=, \cdot)) = f^a((\cdot, v_D^=)) = f^a(a) + t(a)$ for all $a \in A$. The statements now follows since $t(a') \geq 0$ for all $a' \in A$ and $t(a'') \geq 1$ for all $a'' = (v, w) \in A$, with $v \in V^+$. ◊

In Algorithm 14 we describe a specific construction for new tours from a given tour and a given set of transport requests. For this new tour $f^a(a) \leq L + 1$ holds for all $a \in A$. This can be seen as follows.

Algorithm 14 Construct new tour

Input: a uniform tour $\overline{\Gamma} = (m_1, a_1, \ldots, a_{\nu-1}, m_\nu)$ for driver j, the depot v_D, a set of transport requests TR
Output: a tour Γ serving all transport requests of TR

```
1: construct transport graph G^t = (V^+ ∪ V^- ∪ V^=, A ∪ A^t, w)
2: initialize Γ ← ∅
3: initialize currNode ← v_D^=
4: while not every node in V^+ has been visited do
5:     if currNode ∈ V^+ and has not been visited then
6:         mark currNode as visited
7:         follow transport request arc and add corresponding moves and actions to Γ
8:     else
9:         follow tour arc and add corresponding move to Γ
10:    update currNode
11: follow tour Γ̄ until arriving in depot node
12: if necessary insert empty actions between two successive moves in Γ
13:
14: return Γ
```

It is fairly easy to see that Algorithm 14 follows basically the same steps as in Remark 7.8. Furthermore, one can easily see that really a tour is constructed, serving all transport requests. In contrast to the construction in Remark 7.8, Algorithm 14 "follows" the given tour only until it arrives at a non-visited node corresponding to a pickup action and directly serve the transport request.

This ensures that the number of traverses of an arc is not artificially increased. Thus, the result follows with the help of Lemma 7.6, Claim 7.11.1, Claim 7.11.2 and Claim 7.11.3. □

Remark 7.12. Although Algorithm 14 starts the construction of the tour from the depot, any arbitrary node could be used as a starting node. That could be done by removing line 3 and giving *currNode* as a parameter. When the construction is started with another node, Lemma 7.11 still holds, and so does Lemma 7.14. ⧫

Let $\overline{\Gamma}$ be a uniform tour, and let TR be a set of uniform transport requests so that $\overline{\Gamma}$ serves all transport requests in TR. Furthermore, let $G^t = (V^+ \cup V^- \cup V^=, A \cup A^t)$ be a transport graph for $\overline{\Gamma}$ and TR. Since every transport request $r = (v, v', 1) \in \text{TR}$ is served by $\overline{\Gamma}$, there exists a path of tour arcs in G^t from the action a_1 corresponding to v to the action a_l corresponding to v'. Let this path of tour arcs be $p(a_1, a_l) = (a_1, a_2), (a_2, a_3), \ldots, (a_{l-1}, a_l)$. Due to the triangle inequality we can estimate the length $w(a_1, a_l)$ of the transport request arc by

$$w(a_1, a_l) \leq w(a_1, a_2) + \cdots + w(a_{l-1}, a_l).$$

Therefore, the length of all transport request arcs can be estimated by applying the above formula iteratively on all transport request arcs,

$$\sum_{a \in A^t} w(a) \leq \sum_{a \in A^t} \sum_{(a,a') \in p(a)} w(a, a'), \tag{7.6}$$

where $p(a)$ is the minimal path of tour arcs in G^t between the corresponding actions. Now we can consider a function $f^{\mathrm{TR}} : A \to \mathbb{N}$, called transport estimate function, where $f^{\mathrm{TR}}(a)$ shows how often the tour arc a is used in the right hand side of Equation (7.6). With F^{TR} can reformulate Equation (7.6) as

$$\sum_{a \in A^t} w(a) \leq \sum_{a \in A^t} \sum_{(a,a') \in p(a)} w(a, a') = \sum_{a \in A} f^{\mathrm{TR}}(a) w(a). \tag{7.7}$$

It is easy to see, that the values of a transport estimate function depends on the choice of the set of transport requests TR. In order to prove the main theorem, we consider a specific set of transport requests. For that let $\overline{\Gamma}$ be a uniform tour for a driver starting and ending in depot v_D. A set of uniform transport requests TR so that

(i) $\overline{\Gamma}$ serves all transport requests in TR,

(ii) for every transport request $r \in \mathrm{TR}$ there are at most L actions between two corresponding actions for r, and

(iii) there does not exist a transport request $(loc(v), loc(w), 1) \in \mathrm{TR}$ so that the minimal path from $loc(v)$ to $loc(w)$ of tour arcs traverses the tour arcs connecting the depot,

holds is called a set of close distance uniform transport requests for $\overline{\Gamma}$.

A close distance uniform transport request r can be constructed as follows: to each pickup action a_p the "next" unperformed drop action a_d (following the tour graph) is selected. Then, r is the transport request from the station of a_p to the station of a_d. The "next" unperformed drop action, is the closest one from a_p, motivating the choice of the name. In the next lemma, we show that this construction can be applied to any uniform tour, leading to a set of close distance uniform transport requests.

Lemma 7.13. *Let $\overline{\Gamma}$ be a uniform tour for a driver starting and ending in depot v_D. Then there exists a set of close distance uniform transport requests for $\overline{\Gamma}$.*

Proof. Let $\overline{\Gamma} = (m_1, a_1, \ldots, m_{n-1}, a_{n-1}, m_n)$. Since a tour starts and ends in the depot and there must be no car in the depot it follows that

$$\sum_{a=(j,v,x) \in \Gamma} x = 0 \tag{7.8}$$

holds for every tour $\Gamma = (m_1, a_1, \ldots, m_{n-1}, a_{n-1}, m_n)$. Furthermore, if Γ contains actions, it follows that the first action is a pickup action and the last action a drop action. Otherwise, there exists a move $m = (j, \cdot, \cdot, \cdot, \cdot, x)$ with $x < 0$ or $x > L$ or the driver transfers vehicles into the depot, contradicting the definition of a tour.

We construct a set of transportation requests TR by assigning the station of the first pickup action in $\overline{\Gamma}$ to the station of the first drop action in $\overline{\Gamma}$, the station of the second pickup action in $\overline{\Gamma}$ to the station of the second drop action in $\overline{\Gamma}$, and so forth until all actions are assigned. Since the number of cars picked up or dropped in an action in $\overline{\Gamma}$ is exactly one and due to Equation (7.8),

there exists one pickup action for each drop action. Thus, it also follows that the number of actions is even in $\overline{\Gamma}$.

Let $r \in \text{TR}$ be a transportation request and let a be the corresponding pickup and a' the corresponding drop action for r. Using this construction, we show that there are at most L actions between a and a'.

We prove the statement by induction over the number of actions n in $\overline{\Gamma}$. For $n = 0$ and $n = 2$ there are $0 < L$ actions between a and a' proving the base case.

Let us assume that the induction hypothesis holds, i.e., there are at most L actions between a and a' for all tours with n actions.

Next, we prove the inductive step. For that, let $\overline{\Gamma}$ have $n + 2$ actions.

We construct a sequence of n actions and $n + 1$ moves from $\overline{\Gamma}$ by removing the first pickup and the first drop action from $\overline{\Gamma}$. Afterwards, we show that this sequence is a tour. Then the statement follows from the induction hypothesis.

Since every action is non-empty, a^1 is the first pickup action. Let a_ℓ be the first drop action. Furthermore, let $m_1 = (j, v_1, t_{v_1}, w_1, t_{w_1}, x_1)$, $m_2 = (j, v_2, t_{v_2}, w_2, t_{w_2}, x_2)$ and $m_\ell = (j, v_\ell, t_{v_\ell}, w_\ell, t_{w_\ell}, x_\ell)$, $m_{\ell+1} = (j, v_{\ell+1}, t_{v_{\ell+1}}, w_{\ell+1}, t_{w_{\ell+1}}, x_{\ell+1})$.

First, let us consider the following new moves $\hat{m}_{1,2} = (j, v_1, t_{v_1}, w_2, t_{w_2}, x_1)$ and $\hat{m}_{\ell,\ell+1} = (j, v_\ell, t_{v_\ell}, w_{\ell+1}, t_{w_{\ell+1}}, x_{\ell+1})$ and let $\hat{m}_i = (j, v_i, t_{v_i}, w_i, t_{w_i}, x_i - 1)$, for all $3 \leq i \leq \ell - 1$, be a move constructed from the move $m_i = (j, v_i, t_{v_i}, w_i, t_{w_i}, x_i)$. Finally, let $\hat{m}^\iota = m^\iota$ for all $\ell + 2 \leq \iota \leq n + 2$.

Then $\Gamma' = (\hat{m}_{1,2}, a_2, \hat{m}_3, \ldots, a_{\ell-1}, \hat{m}_{\ell,\ell+1}, a_{\ell+1}, \hat{m}_{\ell+2}, \ldots, a_{n-1}, \hat{m}_n)$ is an alternative sequence of $n+1$ moves and n actions. We show that Γ' is indeed a tour. It is sufficient to show for every move $\hat{m}_i = (j, \cdot, \cdot, \cdot, \cdot, x_i)$ in Γ' that $0 \leq x_i \leq L$ holds.

For that, we consider the number n^+ of consecutive pickup actions before the move $m_{\ell,\ell+1}$ and number n^- of consecutive drop actions directly after $m_{\ell,\ell+1}$ in Γ, as well as the number $\hat{n}^+$ of consecutive pickup actions before the move $\hat{m}_{\ell,\ell+1}$ and number $\hat{n}^-$ of consecutive drop actions directly after $\hat{m}_{\ell,\ell+1}$ in $\hat{\Gamma}$. Since $\overline{\Gamma}$ is a tour, it follows from Lemma 7.6 that $0 \leq n^+ - n^- \leq L$ holds. From the construction of Γ' it follows

$$n^+ - n^- = (\hat{n}^+ + 1) - (\hat{n}^- + 1) = \hat{n}^+ - \hat{n}^-$$

and, thus, we have $0 \leq \hat{n}^+ - \hat{n}^- \leq L$. Furthermore, it follows that the number of cars in the convoy of driver j are equal in both sequences in and after the move $m_{\ell+1}$ and $\hat{m}_{\ell,\ell+1}$, respectively, i.e., for every $\ell + 2 \leq i \leq n + 2$ we have $0 \leq x_i \leq L$, where x_i is the number of cars in the move $\hat{m}_i$.

Then, it follows for every move $\hat{m}_i = (j, \cdot, \cdot, \cdot, \cdot, x_i)$ in Γ' that $0 \leq x_i \leq L$ holds, and, thus, that Γ' is a tour with n actions. Therefore, the induction hypothesis can be applied to Γ' and it follows that there are at most L actions between the corresponding pickup and drop actions of a transportation request. Since there are at most L consecutive pickup actions at the start of a tour (Lemma 7.6), it follows that there are at most L actions between the first pickup action and the first drop action. This proves the inductive step and the statement follows.

Since we assign a transport request to every pickup and drop action, the tour serves all transport requests in TR.

Finally, it follows by construction that there does not exist a transport request $(loc(v), loc(w), 1)$ so that the minimal path from $loc(v)$ to $loc(w)$ of tour arcs traverses the tour arcs connecting the depot. □

Lemma 7.14. *Let $\overline{\Gamma} = (m_1, a_1, \ldots, m_{n-1}, a_{n-1}, m_n)$ be a uniform tour starting and ending in depot v_D, and let $G = (V, A, w)$ be a tour graph for $\overline{\Gamma}$. Furthermore, let $t : A \to \mathbb{N}$ be a function that returns the number of cars transfered in the corresponding moves in $\overline{\Gamma}$, and let* TR *be a set*

of close distance uniform transport requests for $\overline{\Gamma}$. *Then for a transport estimate function* f^{TR} *for* TR *it holds*

(i) $f^{\mathrm{TR}}(a) = t(a)$ *for all* $a \in A$, *and*

(ii) $f^{a}(a) + f^{\mathrm{TR}}(a) = f^{a}(a') + f^{\mathrm{TR}}(a')$ *for all* $a, a' \in A$.

Proof. Firstly, we show that for a transport graph $G^t = (V^+ \cup V^- \cup V^=, A \cup A^t, w)$ for $\overline{\Gamma}$ and TR and a transport estimate function f^{TR} for G^t

(a) $f^{\mathrm{TR}}(a) - f^{\mathrm{TR}}(a') = 1$ for all $a = (v, w) \in A$, $a' = (w, u) \in A$ with $w \in V^-$, and

(b) $f^{\mathrm{TR}}(a) - f^{\mathrm{TR}}(a') = -1$ for all $a = (v, w) \in A$, $a' = (w, u) \in A$ with $w \in V^+$,

holds. Within the proof, we also show that (i) holds. Secondly, the Statement (ii) follows from Claim 7.11.2 (ii).

For that, we show that $f^{\mathrm{TR}}(a) = x_a$, where $m_a = (j, \cdot, \cdot, \cdot, \cdot, x_a)$ is the move corresponding to the tour arc $a = (v, w) \in A$. This can be seen as follows. Since TR is a set of close distance transport requests, there does not exist a transport request $(loc(v), loc(w), 1)$ so that the minimal path of tour arcs traverses the tour arcs connecting the depot. Thus, we have $f^{\mathrm{TR}}(a) = f^{\mathrm{TR}}(a') = 0$ where $a = (v_D, v_1) \in A$ and $a' = (\cdot, v_D) \in A$. Since $\overline{\Gamma}$ is a tour, the node $v \in V$ corresponds to a pickup action. Thus, for the move $m_{a_1} = (j, \cdot, \cdot, \cdot, \cdot, x_{a_1})$ corresponding to the tour arc $a_1 = (v_1, v_2)$, we have $x_{a_1} = 1$. Furthermore, the tour arc a_1 appears once on the right hand side of Equation (7.6), i.e., $f^{\mathrm{TR}}(a_1) = 1 = x_{a_1}$.

If $v_2 \in V^+$ corresponds to a pickup action then the number of cars transfered from v_2 to the next station is increased by one. Since the destination of the transport request that started in v_1 does not correspond to v_2, the corresponding tour arc a_2 appears twice on the right hand side of Equation (7.6): once due to the transport request $(v_1, \cdot, 1)$ and once due to $(v_2, \cdot, 1)$.

Analogously, if $v_2 \in V^-$ corresponds to a drop action, the number of cars transfered from v_2 to the next station is decreased by one. Since the transport request arc corresponding to the transport request $(v_1, v_2, 1)$ ends in v_2, the number of appearances of the tour arc $a_2 = (v_2, \cdot) \in A$ on the right hand side of Equation (7.6) is decreased as well.

In the first case we have $f^{\mathrm{TR}}(a_1) - f^{\mathrm{TR}}(a_2) = -1$ and in the second case $f^{\mathrm{TR}}(a_1) - f^{\mathrm{TR}}(a_2) = 1$.

The above arguments can be applied iteratively to all nodes in $V^+ \cup V^-$, showing that (a) and (b) hold.

Since the values of the transport estimate function f^{TR} corresponds to the number of cars transfered in the corresponding move, the stament follows directly from Claim 7.11.2 (ii). □

Finally, we prove the main theorem of this section.

Theorem 7.15. *For the Static Relocation Problem* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$ *with one depot, the algorithm* ReOpt *achieves an approximation factor of* $L + 1$ *for all* $L \in \mathbb{N}$.

Proof. We start by proving a special case when there is only one driver in the system. Afterwards, we generalize this special case to the general situation when there are k drivers in the system.

Let Γ^* be an optimal tour for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, 1, L)$. Let TR^p be a set of transportation requests induced by a minimal perfect p-matching, and let Γ^p an optimal tour serving all transport requests in TR^p, i.e., a tour with a minimal total tour length serving all transport requests in TR^p. Finally, let $\overline{\Gamma}$ be the constructed tour from Algorithm 14. Then we have

$$\ell(\Gamma^*) \leq \ell(\Gamma^p) \leq \ell(\overline{\Gamma})$$

where $\ell(\Gamma)$ is the total tour length of the tour Γ. Thus, we only need to show that

$$\ell(\overline{\Gamma}) \leq (L+1)\ell(\Gamma^*) \tag{7.9}$$

holds.

Let TR be a set of close distance uniform transport requests. Since TR^p is induced by a minimal perfect p-matching, it holds

$$\sum_{(v,w,1)\in \mathrm{TR}^p} d(v,w) \leq \sum_{(v,w,1)\in \mathrm{TR}} d(v,w)$$

Let f^a be a traverse counter function for Γ^* and let f^{TR} be a transport estimate function. Let $t : A \to \mathbb{N}$ be a function that returns the number of cars transfered in the corresponding moves in Γ^*. By definition, a convoy is empty at the beginning of a tour, and therefore, $t(a_0) = 0$ for $a_0 = (v_D, \cdot) \in A$. Since we have $t(a) = f^{\mathrm{TR}}(a)$ for all $a \in A$ (Lemma 7.14 (i)), it follows $f^a(a_0) + f^{\mathrm{TR}}(a_0) = f^a(a_0)$. From Lemma 7.11 we know that $f^a(a) \leq L+1$ for all $a \in A$, and thus, it follows from Lemma 7.14 (ii) that $f^a(a) + f^{\mathrm{TR}}(a) \leq L+1$ holds for all $a \in A$.

With above and Equation (7.7) we can estimate the total tour length $\ell(\overline{\Gamma})$ by

$$\begin{aligned}
\ell(\overline{\Gamma}) &= \sum_{a\in A} f^a(a)w(a) + \sum_{a^t\in A^t} w(a^t) \\
&\leq \sum_{a\in A} f^a(a)w(a) + \sum_{a\in A} f^{\mathrm{TR}}(a)w(a) \\
&= \sum_{a\in A} \left(f^a(a) + f^{\mathrm{TR}}(a)\right) w(a) \\
&\leq (L+1) \sum_{a\in A} w(a) \\
&= (L+1) \cdot \ell(\Gamma^*)
\end{aligned}$$

proving the statement of the theorem if there is only one driver in the system.

Next we consider the general case, i.e., there are $k \in \mathbb{N}$ drivers. Since there is only one depot, and every tour starts and ends in the depot, all tours can be "merged" to one tour. For that let $sched = (\Gamma^1, \ldots, \Gamma^k)$ be a transportation schedule with k tours and with $\Gamma^j = (m_1^j, a_1^j, \ldots, a_{\lambda_j-1}^j, m_{\lambda_j}^j)$. Since by assumption T is large enough, we can construct a new tour $\Gamma = \Gamma^1\Gamma^2\ldots\Gamma^k$, where (except for the first and last tour) the moves to and from the depot are replaced by one move between the succeeding and preceding station, respectively (see Figure 7.7 for an illustration). Thus, all k tours are performed by only one driver. Due to the triangle inequality it follows that the length of the tour Γ is at most the total tour length of the transportation schedule *sched*. Therefore, we can generalize the statement when there are k drivers in the system. □

7.5 Lifted Flows in Aggregated Networks

We describe in this section a heuristic approach of lifted flows in aggregated networks (LiftFlow) to solve the Static Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$, where G is the complete weighted graph $G = (V_O \cup V_U \cup V_D, E, d)$ containing the overfull stations V_O (with $z_i^0 > z_i^T$), the underfull stations V_U (with $z_i^0 < z_i^T$), a set of depots V_D, all connections E between them and distances $d\colon E \to \mathbb{N}$.

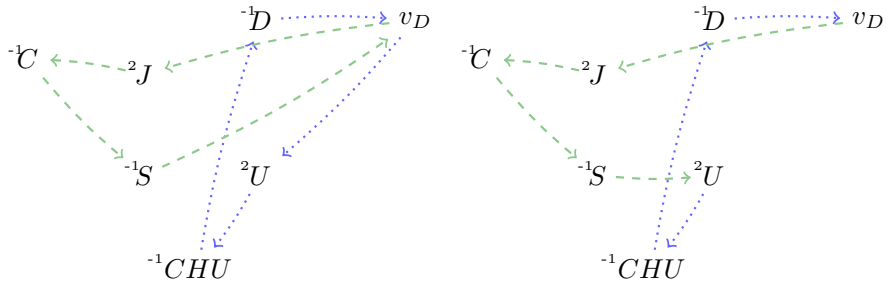


Figure 7.7: This figure illustrates how one driver can perform k tours if there is only one depot. On the left side, the figure illustrates a transportation schedule with 2 drivers, On the right side, a combined transportation schedule with only one driver.

The approach LiftFlow computes a preemptive transportation schedule and is performed in two steps. Firstly, we construct a weighted complete bipartite graph and find a flow that starts and ends in the depot, while passing overfull and underfull stations, minimizing the costs. Each arc carrying a flow corresponds to a move between two stations. Secondly, we compute possible precedences between moves and from that we construct tours (with preemption) for all convoys.

This section gives the proves and further details for the approach already presented in [36].

7.5.1 First step: Flows in aggregated networks

In the first step, we construct an aggregated network and solve a min-cost flow problem on this network. Arcs carrying positive flows then correspond to the movement of drivers and cars in the network. The aggregated network is a directed, weighted graph $G^A = (V_A, A_A, w)$, based on the graph G.

The node set $V_A = V_S \cup V_O \cup V_U \cup V_D$ is composed by the following nodes: V_S and V_D contain the depot, V_O contains the set of overfull stations, V_U contains the set of underfull stations.

The arc set $A_A = A_{SO} \cup A_O \cup A_U \cup A_{OU} \cup A_D \cup A_{SD}$ is composed of several subsets:

- the set of start arcs $A_{SO} = \{(v_D, v_o) : v_D \in V_S, v_o \in V_O\}$ connecting the depot to the overfull stations,
- the set of overfull arcs $A_O = \{(v_o, v'_o), (v'_o, v_o) : v_o, v'_o \in V_O\}$ connecting all overfull stations,
- the set of connection arcs $A_{OU} = \{(v_o, v_u), (v_u, v_o) : v_o \in V_O, v_u \in V_U\}$ connecting overfull and underfull stations,
- the set of underfull arcs $A_U = \{(v_u, v'_u), (v'_u, v_u) : v_u, v'_u \in V_U\}$ connecting all underfull stations,
- the set of sink arcs $A_D = \{(v_u, v_D) : v_u \in V_U, v_D \in V_D\}$, and
- the depot arc $A_{SD} = \{(v_D, v_D) \in V_S \times V_D\}$.

The set containing all overfull, connection and underfull arcs is denoted by $A_L := A_O \cup A_{OU} \cup A_U$. For an arc $a = (v, v') \in A_A$ the arc weights $w(a) := d(v, v')$ correspond to the distances.

An illustration of an aggregated network is given in Figure 7.8.

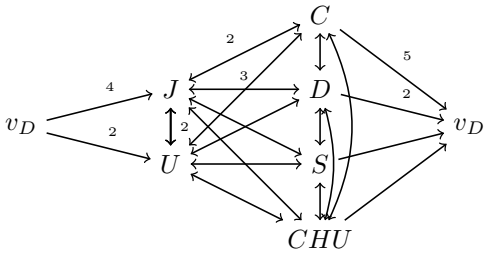


Figure 7.8: This figure illustrates the aggregated network for the carsharing system from Figure 7.1. The set of overfull stations contains the stations J and U, the set of underfull stations the stations C, D, S and CHU. For the sake of readability, only some weights of the arcs are given in this figure. Furthermore, not all underfull arcs are shown.

Note, the subgraphs (V_O, A_O) and (V_U, A_U) are complete graphs, and the subgraph $(V_O \cup V_U, A_{OU})$ is a complete bipartite graph.

On the aggregated network G^A, we define two different flows, the car flow f and the driver flow F, and specify the capacities as well as the costs for each arc with respect to both flows. Flow on an overfull, connection or underfull arc corresponds to a move in a tour, i.e., some cars are moved by drivers in a convoy from station v to another station v'.

To correctly initialize the system, we use the depot $v_D \in V_S$ as source for the driver flow, see (7.10d), and the nodes of overfull stations $v \in V_O$ as sources for the car flow and set their balances accordingly to the number of cars that have to be picked up at station v, i.e., $b^+(v) := \max\{\boldsymbol{z}_v^0 - \boldsymbol{z}_v^T, 0\} \geq 0$, see (7.10b). Flow on the depot arc corresponds to drivers waiting at the depot. Equalities (7.10e) are the flow conservation constraints for the driver flows. The nodes of underfull stations $v \in V_U$ are the destinations of the cars. So we set their balances accordingly to the number of cars that have to be dropped at station v, i.e., $b^-(v) := \min\{\boldsymbol{z}_v^0 - \boldsymbol{z}_v^T, 0\} \leq 0$, see (7.10c). Finally, the sink $v_D \in V_D$ is the destination of the k drivers, see (7.10d). We consider a min-cost flow problem where we intend to balance all stations with minimal costs (7.10a).

$$\min \sum_{a \in A_A} w(a)F(a) \tag{7.10a}$$

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = b^+(v) \qquad \text{for all } v \in V_O \tag{7.10b}$$

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = b^-(v) \qquad \text{for all } v \in V_U \tag{7.10c}$$

$$\sum_{a \in \delta^-(v_D)} F(a) = \sum_{a \in \delta^+(v_D)} F(a) = k \qquad v_D \in V_S, v_D \in V_D \tag{7.10d}$$

$$\sum_{a \in \delta^-(v)} F(a) = \sum_{a \in \delta^+(v)} F(a) \qquad \text{for all } v \in V_O \cup V_U \tag{7.10e}$$

$$0 \leq f(a) \leq L \cdot F(a) \qquad \text{for all } a \in A_L \tag{7.10f}$$

$$f, F \text{ integer}, \tag{7.10g}$$

Note, due to constraints (7.10f), the above constraint matrix is again not totally unimodular.

Figure 7.9 gives an example for flows in an aggregated network.

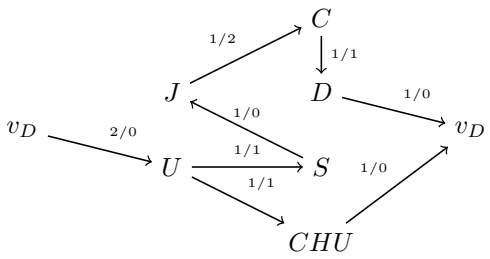


Figure 7.9: This figure illustrates possible flows in the aggregated network G_A from Figure 7.8. The flows are drawn as solid arcs, the numbers x/y above the arcs correspond to x drivers with y cars in their convoy.

7.5.2 Second step: Compute transportation schedule

Next, we describe how to compute a transportation schedule from the solution of the first step.

For that, we first compute "pre-moves", which have the origin and destination stations as well as the load of a move, but no times. With these pre-moves we compute "pre-tours" as sequences of "pre-moves". Formally, a pre-move $\check{m} = (v, v', x)$ is a 3-tuple, where $v = orig(\check{m})$ is the origin station, $v' = dest(\check{m})$ the destination station and $x = \ell oad(\check{m})$ the load of the pre-move; a pre-tour is a sequence of pre-moves.

Let f^A be the car flow and F^A the driver flow computed by the integer linear program (7.10) in the first step. Then every arc $a = (v, v') \in A_A$ with $F^A(a) > 0$ corresponds to a pre-move $\check{m}_a = (v, v', x)$, with $x \leq f^A(a)$. All pre-moves have to be assigned to a pre-tour in a feasible order, i.e., the destination station of each pre-move has to be equal to the origin station of the successor pre-move. This can be done by searching for paths within the aggregated network. However, in general, there is not only a unique path within the aggregated network leading to several possible pre-tours. Note that the aggregated network is not cycle-free. Thus, there can be isolated cycles in the solution. For this section, let us assume that there are no such isolated cycles; a strategy to detect and handle isolated cycles is presented in the next section.

Since we consider the preemptive situation, it is possible that there are precedences between different tours. We define a precedence relation between pre-moves in an analog way to the definition of precedences between moves. For two different pre-tours $\check{\Gamma}$ and $\check{\Gamma}'$ there exists a potential precedence between two pre-moves $\check{m}_i \in \check{\Gamma}$ and $\check{m}'_j \in \check{\Gamma}'$ ($\check{m}_i$ precedes $\check{m}'_j$) if

- the destination station v_o of $\check{m}_i$ is an overfull station, $\check{m}_i$ drops cars at v_o, and $\check{m}'_j$ picks up cars at v_o; or
- the origin station v_u of $\check{m}'_j$ is an underfull station, $\check{m}'_j$ picks up cars at v_u, and $\check{m}_i$ drops cars at v_u.

A pre-tour without precedences to another tour, can be directly transformed to a tour, by computing the departure and arrival times of each (pre-)move, and then assigning all moves to a driver. Otherwise, there is a preemptive situation which means that one convoy transports cars to a station and another convoy picks up these cars afterwards. In order to ensure that the cars are dropped before they are picked up, we possibly have to add additional waiting moves to the final tour. For that we construct a precedence relation between pre-moves.

In some rare cases, the precedence relation is not acyclic. In this situation, we add an additional constraint to the integer linear program (7.10a)–(7.10g), to receive a new solution.

For that we add

$$\sum_{a \in A_A} w(a)F(a) > \sum_{a \in A_A} w(a)F^A(a)$$

to the constraints (7.10b)–(7.10g) and recompute the steps above. In fact, in our set of randomly generated test-instances, this case never occurred, and we do not expect it to occur often in practice.

Thus, let us consider an acyclic precedence relation containing precedences between different pre-tours. The arrival and departure times of the moves are derived from the distances between the origin and destination stations of the pre-moves. When the departure time of a move is computed from a pre-move, having a precedence relation to a pre-move in another pre-tour then we have to compute the arrival time of the preceding pre-move before, in order to be able to compute the waiting time. If several pre-moves $\check{m}_1, \ldots, \check{m}_\lambda$ precede a pre-move $\check{m}$ then, in general, $\check{m}$ does not need to wait for all preceding pre-moves but only until there are enough cars at $orig(\check{m})$. For that, we compute a minimal set $S \subseteq \{\check{m}_1, \ldots, \check{m}_\lambda\}$ by solving a minimum matching problem on the complete bipartite graph $(\{\check{m}_1, \ldots, \check{m}_\lambda\} \cup \{\check{m}\}, E, w)$, where the arc weight $w(a)$ of $a = (\check{m}_j, \check{m})$ corresponds to $\ell oad(\check{m}_j)$. Hereby, the sum of the weights of the selected arcs and the number of cars at $orig(\check{m})$ must be at least $\ell oad(\check{m})$. The waiting time for $\check{m}$ is then induced by the latest arrival time of all $\check{m}_j \in S$.

After all waiting times have been computed, we construct a transportation schedule from the (waiting) moves. Hereby, we have to insert a pickup action between two moves m, m', if we have $\ell oad(m') - \ell oad(m) > 0$, and we have to insert a pickup action between m and m', if $\ell oad(m') - \ell oad(m) < 0$ holds. Otherwise, we insert an empty action.

7.5.3 Handling cycles in the lifted flows

In this section, we describe how isolated cycles in the lifted flows can be handled. The basic idea is to "break" the cycle and include the remaining path into a tour. The next lemma ensures that every cycle can be "broken".

Lemma 7.16. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L, cost^c, cost^d)$ be a Static Relocation Problem. Let f^A (car flow) and F^A (driver flow) be an optimal solution in an aggregated network G^A, so that there exists an isolated cycle C. Then it is true:*

(i) C contains at least one overfull and one underfull station, and

(ii) there exists an arc $(u, o) \in C$, where u is an underfull station, o is an overfull station, and so that $f(u, o) = 0$.

Proof. Let $C = (v^1, \ldots, v^\ell, v^1)$ be an isolated cycle.

"(i)": let us assume that there are only overfull stations in C. Then due to constraints 7.10b, it follows that

$$f(v^1, v^2) - f(v^\ell, v^1) = b^+(v^1)$$

holds. Since $b^+(v^1) > 0$ by definition, it follows

$$f(v^1, v^2) > f(v^\ell, v^1).$$

Analogously, we get

$$f(v^2, v^3) > f(v^1, v^2) \text{ and, generally, } f(v^j, v^{j+1}) > f(v^{j-1}, v^j).$$

Thus, finally it follows that

$$f(v^j, v^{j+1}) > f(v^j, v^{j+1})$$

which is a contradiction.

The case that there are not only underfull stations in C can be proved analogously.

"(ii)": let us assume that there is no arc $(u, o) \in C$ with $f(a) = 0$. Then there exists a minimal value $x = \min_{a \in C^{UO}} f(a)$, where C^{UO} contains the arcs in C from underfull to overfull stations.

Then we subtract x from the car flow on every arc in C, i.e., we consider a car flow f' with $f'(a) = f(a) - x$ for any $a \in C$. Obviously, for f' there exists an arc $a' \in C^{UO}$ with $f'(a') = 0$.

Therefore, we have only to show that for every arc $a \in C$ the flow remains non-negative, i.e., $f'(a) \geq 0$.

Since x is the minimal value for every arc from an underfull to an overfull station, it directly follows that $f'(a) \geq 0$ for every $a \in C^{UO}$. From constraints 7.10b it follows that the flow value on an outgoing arc of an overfull station o is greater than the flow value on the incoming arc, i.e.,

$$f(o, v) > f(v', o)$$

holds. Combined with above, this implies $f(o, v) > x$ for every outgoing arc of an overfull station.

Analogously, it follows from constraints 7.10c that the flow value on an outgoing arc of an underfull station is greater than the flow value on the incoming arc. Thus, the minimal value is on an arc from an underfull to an overfull station. This implies that $f(u^1, u^2) > x$ for every underfull stations u^1, u^2.

Putting all cases together yields that $f'(a) \geq 0$ for every $a \in C$, proving the statement. □

Now we describe in detail how to handle isolated cycles. For that, let $v^1 \cdots v^\lambda v^1$ be an isolated cycle with $F(v^1, v^2) = \cdots = F(v^{\lambda-1}, v^\lambda) = F(v^\lambda, v^1)$. W.l.o.g. let the cycle be ordered so that v^1 is an overfull station and v^λ is an underfull station and so that $f(v^\lambda, v^1) = 0$. That isolated cycles contain overfull and underfull stations, as well as the existence of an arc from an underfull to an overfull station with $f(v^\lambda, v^1) = 0$ is ensured by Lemma 7.16. Since the cycle is isolated, it holds by definition $F(v, v^j) = F(v^j, v) = 0$ for all $v \in V_A \setminus \{v^1, \ldots, v^\lambda\}$.

Since v^1 is an overfull station and v^λ an underfull station, we can consider the path $v^1 \cdots v^\lambda$ as a subsequence of pre-moves. Precedences within this subsequence are then handled as described in the previous paragraph.

Let Γ be a tour from the transportation schedule computed in Step 2, and let m be the last move from Γ. Since there are no cars transfered from v^λ to v^1, we can modify Γ by removing m and adding a move from $orig(m)$ to v^1, as well as the moves and actions induced from the path $v^1 \cdots v^\lambda$. Finally, a move from v^λ to the depot $dest(m)$ is added.

Since we can handle precedences between pre-moves and the case of isolated cycles in the two flows, we obtain:

Theorem 7.17. *The approach* LiftFlow *computes a feasible (possibly preemptive) transportation schedule for the Static Relocation Problem.*

7.5.4 Optimal Solution

In this section, we show that, under certain conditions, a computed solution by LiftFlow is the optimal solution. Hereby, we consider and compare a solution computed from the flows in aggregated networks (Section 7.5.1) with a solution computed by flows in time-expanded networks (Section 7.1). The considered restrictions are partly of general nature (large enough time horizon, and enough parking places at the stations), partly conditions on the quality of

an optimal solution (balanced stations are not used as preemption stations), and conditions on the solution of the flows in the aggregated network (cycle-free precedence graph, and no isolated cycles in the flows).

The first intermediate result is that from a solution in time-expanded networks we can construct a solution in the aggregated network. Hereby, the objective function value remains equal.

Lemma 7.18. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$ be a Static Relocation Problem. Let f^T (car flow) and F^T (driver flow) be an optimal solution in a time-expanded network G^T, so that no balanced station is used as a preemption station. Then there exist feasible flows f^A (car flow) and F^A (driver flow) in the aggregated network G^A so that the total tour lengths are equal, i.e.,*

$$\sum_{a \in A_L} d(a) F^T(a) = \sum_{a \in A_A} w(a) F^A(a). \tag{7.11}$$

Proof. In order to show the statement, we first construct a solution in the aggregated network from the flows f^T and F^T. Afterwards, we prove the correctness of the construction.

Let $a = (u, v) \in A_L$ be a relocation arc in G^T, then

(i) u and v are overfull or underfull stations or depots, i.e., $u, v \in V_S \cup V_O \cup V_U \cup V_D$; or

(ii) either u and/or v are balanced stations.

By assumption there are no balanced stations used as preemption station. Thus, it follows in Case (ii) that there exists a path from an overfull station (or underfull station or depot) to an overfull station (or underfull station or depot), i.e., $v_1, \dots, v_\ell$ where $u = v_j, v = v_{j+1}$ for a $1 \leq j \leq \ell$, and $v_1, v_\ell \in V_S \cup V_O \cup V_U \cup V_D$, and all other stations are balanced stations $v_j \in V_B$, $1 < j < \ell$. Additionally, it follows that all car flows and driver flows on all these arcs are equal, i.e., $f^T(v_1, v_2) = \cdots = f^T(v_{\ell-1}, v_\ell)$ and $F^T(v_1, v_2) = \cdots = F^T(v_{\ell-1}, v_\ell)$.

Since f^T and F^T are optimal, it follows in Case (ii) that no cars are picked up or dropped between v_1 and v_2, and we have

$$d(v_1, v_\ell) = d(v_1, v_2) + \cdots + d(v_{\ell-1}, v_\ell),$$

and thus, by definition

$$w(v_1, v_\ell) = w(v_1, v_2) + \cdots + w(v_{\ell-1}, v_\ell).$$

Therefore, and for the sake of simplicity, we can assume that Case (i) holds for all relocation arcs.

Next, we construct the flows f^A and F^A in G^A from f^T and F^T. For every arc $a' = (u, v) \in A_A$ we set the driver flow

$$F^A(a') := \sum_{a \in A_{uv}} F^T(a) \tag{7.12}$$

where $A_{uv} = \{(u, \cdot, v, \cdot) \in A_L\}$. Analogously, we set

$$f^A(a') := \sum_{a \in A_{uv}} f^T(a) \tag{7.13}$$

for the car flows. By construction we have

$$\sum_{a \in A_L} d(a) F^T(a) = \sum_{a \in A_A} w(a) F^A(a).$$

Finally, we show that these constructed flows are indeed a solution of (7.10) for the aggregated network G_A by showing that all constraints of the integer linear program (7.10a)–(7.10g) are respected.

Let v_o be an overfull station and v_u an underfull station. By definition we have $b^+(v_o) = z_v^0 - z_v^T$ and $b^-(v_u) = z_v^0 - z_v^T$. Thus, subtracting (7.1c) from (7.1b) we gain

$$\sum_{a \in \delta^-(v_o,0)} f(a) - \sum_{a \in \delta^+(v_o,T)} f(a) = z_{v_o}^0 - z_{v_o}^T = b^+(v_o)$$

and

$$\sum_{a \in \delta^-(v_u,0)} f(a) - \sum_{a \in \delta^-(v_u,T)} f(a) = z_{v_u}^0 - z_{v_u}^T = b^-(v_u),$$

proving that (7.10b) and (7.10c) hold. Since all drivers start and end their tours in the depot, the equations for the drivers (7.10d) follow directly from the driver flow equations in (7.1c) and (7.1b).

From the flow conservation constraints (7.1e) follow the flow conservation constraints (7.10e). Drivers waiting at the depot correspond to flow on the depot arc. The coupling constraints (7.10f) hold due to the coupling constraints (7.1h) and due to (7.13) and (7.12). Finally, the constraints (7.10g) follow directly from (7.1h). Thus, the statement is proved. □

Next, we show the main result of this section, namely that, under the stated precondictions at the beginning of this section, a solution computed from the aggregated network is optimal. This means that the objective function values from a solution computed by the integer linear program (7.1) and by the aggregated network (7.10) are equal.

Theorem 7.19. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$ be a Static Relocation Problem. Let f^A (car flow) and F^A (driver flow) be an optimal solution in an aggregated network G^A, so that*

(i) there are no cycles in the precedence graph,

(ii) there are no isolated cycles in the flows,

(iii) the capacities of the stations are sufficiently large,

(iv) the time horizon is sufficiently large, and

(v) there exists an optimal solution so that no balanced station is used as a preemption station.

Then there exist feasible flows f^T (car flow) and F^T (driver flow) in the time-expanded network G^T so that

$$\sum_{a \in A_L} d(a) F^T(a) = \sum_{a \in A_A} w(a) F^A(a).$$

Furthermore, this solution is an optimal solution for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$.

Proof. First, we show that there exist k paths from the depot node in V_S to the depot node in V_D. Driver flow on the depot arc, i.e., drivers waiting at their initial depot, obviously form a path from V_S to V_D. Thus, we concentrate only on those drivers which leave the depot. For that we construct a directed graph $\overline{G} = (\overline{V}, \overline{A})$, where $\overline{V} = V$ and for every flow value $F(a) > 0$ there is an arc in $\overline{A}$ counting multiplicities. Furthermore, all nodes v_T in V_D are connected with $v_0 \in V_S$ (by adding an arc (v_T, v_0) to $\overline{A}$), if there exists an arc $a \in A_{SO}$ with $F(a) > 0$ and there exists an arc $a \in A_{UD}$ with $F(a) > 0$. Then, from (7.10e) it follows that the number of incoming and outgoing arcs in the nodes in $\overline{V}$ which correspond to $V_O \cup V_U$ are equal. Since

there is an arc between the nodes in V_D to V_S when there is an outgoing arc of V_S with flow and there is an incoming arc of V_D with flow, the number of incoming arcs and outgoing arcs of all nodes in $V_S \cup V_D$ are equal. Altogether, and from the condition that there are no isolated cycles (ii) it follows from Euler's Theorem that there exist Euler paths in $\overline{G}$. The subpaths of these Euler paths from the nodes from V_S to V_D correspond to k paths from the depot nodes V_S to the depot nodes V_D in G^A.

By condition (iv) the time horizon is large enough so that the car flow F^A and driver flow F^A can be transformed to flows f^T and F^T in the time-expanded network, including any probable waiting times. Furthermore, due to conditions (i) and (iii) it is ensured that all preemptive situations can be resolved and there is always enough space for the cars at the stations. As usual, flow on an arc $a \in A_A$ corresponds to flow on a corresponding relocation arc, and waiting times which may be induced by preemption to flow on holdover arcs. Note, whenever for an arc $(u, v) \in A_A$ no relocation arc (u, t_u, v, t_v) exists then, by construction of the aggregated network, there exists a shortest path from u to v in G, which is then used instead. Furthermore, it is then easy to see that the costs of both solutions are equal.

Next, we show that all constraints of the integer linear program (7.1) are fulfilled. That the flows f^T and F^T are correctly initialized (Equations (7.1b) and (7.1c)) is ensured by (7.10b), (7.10c), and (7.10d). The flow conservation constraints (7.1d) and (7.1e) are ensured by construction. The upper bound of Estimates (7.1f) hold due to the condition (iii) that the capacities of the stations are sufficiently large. That the number of cars at a station is always non-negative is ensured by the start system state $\boldsymbol{z}^0$ and the target system state $\boldsymbol{z}^T$, which are both feasible system states. The coupling constraints (7.1g) are ensured due to the coupling constraints (7.10f) and due to additionally added waiting times if there are preemptions. Since the flows in the aggregated network are all integral (7.10g) it follows that the flows in the time-expanded network can be set integral as well, ensuring conditions (7.1h).

Finally, we prove that an optimal solution of the flows in the aggregated network lead to an optimal solution for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$ (under the assumption of the stated conditions (i)–(v)).

Let us assume that the flows f^T and F^T are not optimal. Then there exist flows $\hat{f}^T$ and $\hat{F}^T$ in the time expanded network so that

$$\sum_{a \in A_L} d(a)\hat{F}^T(a) < \sum_{a \in A_L} d(a)F^T(a)$$

holds and so that there are no balanced station is used as a preemption station (Condition (v)). From Lemma 7.18 it follows that there exist flows $\hat{f}^A$ and $\hat{F}^A$ in the aggregated network with the same costs as $\hat{f}^T$ and $\hat{F}^T$, i.e.,

$$\sum_{a \in A_A} w(a)\hat{F}^A(a) < \sum_{a \in A_A} w(a)F^A(a).$$

But this contradicts the condition that f^A and F^A are optimal. □

Finally, we give a small discussion about the conditions (i) to (iv) from the previous theorem. Unfortunately, none of these conditions is in general known before we have computed the flows in the aggregated network and a solution from these flows. However, we show that most of them can be concluded from a solution. This gives us at least in several cases an optimal solution.

One of the main reasons for conditions (iii) and (iv) is to ensure that a feasible transportation schedule can be computed from the flows within the aggregated network.

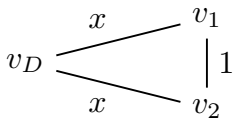
Since the system states $\boldsymbol{z}^0$ and $\boldsymbol{z}^T$ must be feasible system states, it follows that enough cars can be picked up and dropped, respectively. Thus, if there is no preemption within the

solution then condition (iii) automatically follows. If additionally the first two conditions are satisfied, also Condition (v) is satisfied. Whether or not conditions (i) and (ii) hold, is always tested by the algorithm. If the computed solution contains preemption, one can still easily test if the capacities of the stations are violated by computing the intermediate system states, but one cannot conclude whether condition (v) holds or not.

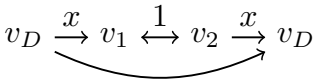
Condition (iv) can also be tested easily by computing the makespan of the computed transportation schedule. However, in practice, it may be acceptable if the time horizon is slightly exceeded.

Finally, note that if condition (i) does not hold then the flows in the aggregated network have to be recomputed with some additional constraints (see Section 7.5). Then, the computed solution might no longer be optimal. Furthermore, if there are isolated cycles (i.e., if condition (ii) is violated), the ratio between the optimal solution value and the solution value computed from flows with isolated cycles cannot be bound from above as the next example shows.

Example 7.20. Let us consider a complete weighted graph $G = (V \cup \{v_0\}, E, w)$ with the node set $V = \{v_1, v_2\}$, and $w(v_0, v_1) = w(v_0, v_2) = x$ and $w(v_1, v_2) = 1$ (see Figure 7.10a for an illustration). Furthermore, let us consider the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, 1, 1, L, 0, 1)$ with $z^0_{v_1} = 1$, $z^0_{v_2} = 0$ and $z^T_{v_1} = 0$, $z^T_{v_2} = 1$, and $T \geq 2x + 1$. This means that there is only one driver and only one car in the system. This induces the set of task $\{(v_1, 0, T, +1), (v_2, 0, T, -1)\}$.



(a) This figure illustrates the complete weighted graph G.



(b) This figure illustrates the aggregated network based on G.

Figure 7.10: This figure illustrates that the ratio between the lower bound computed by flows in the aggregated network and the total tour length of the optimal transportation schedule can be arbitrarily large.

Then an optimal transportation schedule for this relocation problem is given by

$(1, v_0, 0, v_1, x, 0)$, % move from the depot to v_1
$(1, v_1, 1)$, % pickup one car at v_1
$(1, v_1, x, v_2, x + 1, 1)$, % transfer a car from v_1 to v_2
$(1, v_2, -1)$, % drop one car at v_2
$(1, v_2, x + 1, v_0, 2x + 1, 0)$ % return to the depot

and, thus, the total tour length is $\mathsf{OPT} = 2x + 1$.

However, if we compute the flows in the corresponding aggregated network, if x is large enough, there is an isolated cycle between v_1 and v_2, resulting in a lower bound of $\mathsf{LB} = 2$ (see Figure 7.10b for an illustration of the aggregated network).

Thus, we have

$$\frac{\mathsf{LB}}{\mathsf{OPT}} = \frac{2x + 1}{2} > x,$$

and since x can be selected arbitrarily large, it follows that the ratio between the lower bound computed by flows in the aggregated network and the total tour length of the optimal transportation schedule can be arbitrarily large. ◊

7.6 Handling Solutions Exceeding the Time Horizon

Especially, heuristic approaches may not find a feasible solution within the given time horizon. In practice, this problem may occur with some approaches described in the previous sections: Greedy from Section 7.2, ReOpt from Section 7.4, and LiftFlow from Section 7.5. Therefore, we describe in this section how one can transform a collection of tours $\mathcal{S}'$ with makespan T' which exceeds the time horizon T into a transportation schedule. Hereby, $\mathcal{S}'$ is a feasible solution for the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^{T'}, \gamma, k, L)$.

For that, we construct a variation $G'_{T,T'} = (V'_T, A'_T)$ of the time-expanded network G_T from Section 7.1 which concatenates the time-expanded network G_T with a smaller version $G_{[T,T']}$ of the time-expanded network within the time window $[T, T']$. Hereby, $G_{[T,T']}$ contains only those relocation arcs which are needed to represent $\mathcal{S}'$ in the time-expanded network within the time window $[T, T']$, i.e., for every (sub)move between two stations v and v' with an arrival time greater than T', we add a relocation $((v, t_v), (v', t_{v'}))$ to A'_L, where t_v is the departure time and $t_{v'}$ the arrival time of the (sub)move.

Note that this approach can be used to compute a preemptive and a non-preemptive transportation schedule, depending on $\mathcal{S}'$, and the construction of $G_{[T,T']}$ and their flows (see Section 7.1).

Each relocation arc $a = ((v, t), (v', t + d(v, v')))$, with $t \leq T$, corresponding to edge (v, v') has the original cost

$$C(a) := d(v, v'),$$

and each relocation arc $a' = ((v, t), (v', t + d(v, v')))$, with $t \leq T$, corresponding to edge (v, v') has increased cost

$$C(a) := \mathrm{cost}^{\mathrm{d}'} \cdot d(v, v'),$$

where $\mathrm{cost}^{\mathrm{d}'} > 1$ are penalty costs.

Then the integer linear program (7.1) (or (7.4)) can then be used by only modifying the objective function to

$$\min \sum_{a \in A_L} C(a) F(a)$$

and

$$\min \sum_{a \in A_L} C(a) \sum_{i=1}^{k} F_i(a),$$

respectively.

Finally, we give a small discussion about this approach. By definition of C and due to the higher penalty costs $\mathrm{cost}^{\mathrm{d}'} > 1$, it follows that the costs for moving cars or drivers on a relocation arc within the original time horizon induces smaller costs than moving cars or drivers within $[T, T']$. Thus, by solving the integer linear program (7.1) results (hopefully) in a feasible solution.

However, if there does not exist a solution with a makespan within the time horizon T, this approach, in general, one can hardly prove the non-existence of a solution with a makespan of maximal T. A way to prove that there does not exist a solution with a makespan of maximal T with the approach of this section is to set penalty unit costs $\mathrm{cost}^{\mathrm{d}'}$ high enough, i.e., let $d = \max_{v,v' \in V} d(v, v')$ be the maximal distance within G, then set $\mathrm{cost}^{\mathrm{d}'} > T \cdot k$. Then an optimal solution of the integer linear program (7.1) is a feasible solution $\mathcal{S}$, if and only if the makespan of $\mathcal{S}$ is less than or equal to T. This result is stated and proved in the next theorem.

Theorem 7.21. *Let $P = (G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ be a Static Relocation Problem. Furthermore, let the penalty costs $\text{cost}^{\text{d}'} > T \cdot k$. Then an optimal solution of the integer linear program* (7.1) *on $G'_{T,T'}$ is a solution $\mathcal{S}$ for P, if and only if the makespan of $\mathcal{S}$ is less than or equal to T.*

Proof. "⇐" Since by pre-condition the solution $\mathcal{S}$ has a makespan less than or equal to T. Then the statement follows from Theorem 7.2.

"⇒" Let us assume that there exists a solution $\mathcal{S}'$ with a makespan less than or equal to T. Since $\mathcal{S}$ is optimal, it follows that the total costs c of $\mathcal{S}'$ are greater than or equal to the total costs c' of $\mathcal{S}$.

By assumption, the makespan of $\mathcal{S}$ is greater than T, it follows that there exists a relocation arc $a = ((v, t_v), (v', t_{v'}))$ with $F(a) > 0$, and $T < t_{v'}$. Thus, due to the choice of $\text{cost}^{\text{d}'}$, the costs for traversing a is greater than $T \cdot k$.

If every driver moved by one unit at each time then the costs for the drivers are

$$T \cdot k.$$

Thus, the total costs for $\mathcal{S}'$ are bounded from above by $T \cdot k$. But this means that $c \leq T \cdot k < c'$, contradicting the assumption that $\mathcal{S}$ is an optimal solution. □

Theoretically, one can select the penalty costs $\text{cost}^{\text{d}'}$ arbitrarily high, and as the previous theorem shows this also might be necessary to choose high unit costs. In practice, one usually sets a time limit on solving the integer linear program, and thus, does not necessarily get an optimal solution. Hereby, one wants to gain a best possible result within the time limit. If the penalty costs are selected too high, i.e., with $\text{cost}^{\text{d}'} \gg 1$, then any feasible solution within the time horizon T may be chosen. However, this computed solution might be far away from an optimal solution. If one selects moderately penalty costs, the chances to receive a solution with costs close to the costs of the initial solution may increase. In our experiments, we usually set $\text{cost}^{\text{d}'} = 2$ and quickly gained a feasible solution with a smaller objective function.

The approach we described in this section, aims at finding a feasible solution with a makespan of less than or equal to T or returns the initial solution. If there does not exist a solution within the time horizon, it may be intended to have a solution with a smaller makespan. For that, the whole time-expanded network can be constructed, i.e., to add all nodes, relocation and holdover even in the time interval $[T, T']$. In this case, we set increasing costs for the relocation arcs, e.g., for $a = ((v, t), (v', t_{v'}))$ with $T \leq t_{v'}$ we set $C(a) = \text{cost}^{\text{d}'} \cdot (t_{v'} - T') \cdot d(v, v')$. Then, the solver is rewarded in finding a solution with minimal makespan. However, since there are (in general) more arcs within the time-expanded network when the whole network is constructed than when only the arcs corresponding to the initial solution are added, the runtime may increase drastically.

7.7 Lower Bounds

Normally, heuristics are tested against known solutions in order to evaluate their efficiency. However, the computation of the optimal value takes a tremendous amount of time even for small systems. On the tested instances, the gap between a lower bound computed by an integer linear program solver[1] and the computed solution can be quite large (see Section 9). This leads to the problem of finding a way to compute better lower bounds, so that we are at least able to compare the solutions computed by a heuristic approach against a lower bound.

[1] Usually, solver for integer linear programs use a relaxed version of the given program to compute a lower bound. This bound is then iteratively improved, e.g., by a branch and bound algorithm.

In this section, we discuss some approaches to compute a lower bound for the Static Min-Cost Relocation Problem.

7.7.1 Reduction to the Vehicle Routing Problem

Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ be a Static Min-Cost Relocation Problem. Then we receive a lower bound by solving a vehicle routing problem (G^{tsp}, v^0, k) on the weighted complete graph $G^{tsp} = (V_{OU}, E, w)$, where $V_{OU} \subseteq V$ is the set of overfull and underfull stations, E contains all edges between the stations, and w are the shortest path distances between the stations. That a solution to the vehicle routing problem indeed gives a lower bound can be seen as follows.

Let $\mathcal{S}$ be an optimal transportation schedule for the Static Min-Cost Relocation Problem, and let $\mathcal{S}^{tsp}$ be an optimal schedule for the vehicle routing problem. Let us assume that $\ell(\mathcal{S}) < \ell(\mathcal{S}^{tsp})$ holds. Since all stations in V^{OU} must be visited by $\mathcal{S}$ it follows that $\mathcal{S}$ is a (generally non-optimal) solution for a traveling salesperson problem on the set of stations V^{OU}. Since by assumption we have $\ell(\mathcal{S}) < \ell(\mathcal{S}^{tsp})$, and thus, $\mathcal{S}^{tsp}$ could not be an optimal solution for the vehicle routing problem, contradicting our precondition.

Under certain circumstances this lower bound cannot be improved, as we show in the next lemma.

Lemma 7.22. *Let $\mathcal{S}^{tsp}$ be an optimal schedule for a vehicle routing problem (G^{tsp}, v_D, k). Then there exists a Static Min-Cost Relocation Problem P so that for every optimal solution $\mathcal{S}$ for P*

$$\ell(\mathcal{S}) = \ell(\mathcal{S}^{tsp})$$

holds.

Proof. We show that from any vehicle routing problem (G^{tsp}, v_D, k) with $G^{tsp} = (V^{OU}, E, w)$, we can construct a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ so that the costs for an optimal solution for each of the problem are equal. For that we keep the graph, i.e., $G = G^{tsp} \cup \{v_D\}$ and their weights, set the convoy capacity $L = 2$, and define the start state $\boldsymbol{z}^0$ and destination state $\boldsymbol{z}^T$ as follows.

Let $\mathcal{S}^{tsp} = (\Gamma_1^{tsp}, \ldots, \Gamma_k^{tsp})$ be a schedule for (G^{tsp}, v_D, k), with $\Gamma_j^{tsp} = (v_D, v_{j1}, \ldots, v_{jj_n}, v_D)$. We consider each tour:

Case 1 (j_n is even): then we alternatively set the visited stations in Γ_j^{tsp} to overfull and underfull stations. Hereby, each overfull station has one excess and each underfull station one deficit, i.e., we set

$$\boldsymbol{z}^0_{v_{jl}} = \begin{cases} 1, & \text{if } l \text{ is odd} \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\boldsymbol{z}^T_{v_{jl}} = \begin{cases} 0, & \text{if } l \text{ is odd} \\ 1, & \text{otherwise.} \end{cases}$$

Case 2 (j_n is odd and $j_n \geq 3$): then we set the first two visited stations to overfull stations each with an excess of one, the third visited station to an underfull station with a deficit of two, and afterwards we alternatively set the visited stations in Γ_j^{tsp} to overfull and underfull stations with an excess of one resp. deficit of one, i.e., we set

$$\boldsymbol{z}^0_{v_{jl}} = \begin{cases} 1, & \text{if } l \text{ is even or } l \in \{1, 2\}, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\boldsymbol{z}^T_{v_{jl}} = \begin{cases} 0, & \text{if } l \text{ is odd and } l > 3, \\ 2, & \text{if } l = 3, \\ 1, & \text{otherwise.} \end{cases}$$

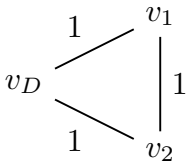
Case 3 ($j_n = 1$): in this case, we add another station to V with a distance of 0 to v^{j1}. Then we can handle this case as done in Case 1.

Case 4 ($j_n = 0$): this means, that the driver does not leave the depot. In this case, nothing has to be done.

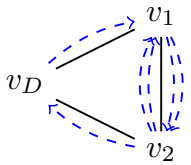
In order to show, that there exists an optimal transportation schedule $\mathcal{S}$ for the Static Min-Cost Relocation Problem with $\ell(\mathcal{S}) = \ell(\mathcal{S}^{tsp})$, it is sufficient to show that for each tour $\Gamma^{tsp} \in \mathcal{S}^{tsp}$ there exists a tour Γ with $\ell(\Gamma) = \ell(\Gamma^{tsp})$. However, by construction of the overfull and underfull stations, in order to solve the Static Min-Cost Relocation Problem, the stations can be visited in the same order as they are visited by the salesperson. Hereby, the cars are picked up in the overfull stations and directly dropped in the underfull stations. Since $\mathcal{S}^{tsp}$ is an optimal solution for the vehicle routing problem, there does not exist a tour which visits all stations with less costs than $\ell(\mathcal{S}^{tsp})$ and $\ell(\Gamma) = \ell(\Gamma^{tsp})$ as well as the optimal statement follow. Since the vehicle routing problem (G^{tsp}, v_D, k) was arbitrarily chosen, the statement is proved. □

From above stated and previous lemma, we can conclude that solving a vehicle routing problem on the set of overfull and underfull stations, a lower bound is given. Due to Lemma 7.22, there are cases, when this lower bound cannot be improved. However, the gap between this lower bound and the optimal solution for a Static Min-Cost Relocation Problem can be arbitrarily high as the next example shows.

Example 7.23. Let us consider a the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, 1, 1)$ with the complete weighted graph $G = (V \cup \{v_D\}, E, w)$ with $V = \{v_1, v_2\}$ where $w(v_D, v_1) = w(v_1, v_2) = w(v_2, v_D) = 1$, and $\boldsymbol{z}^0_{v_1} = \gamma$ and $\boldsymbol{z}^T_{v_2} = \gamma$ (see Figure 7.11a for an illustration). Thus, the tasks are $(v_1, 0, T, \gamma)$ and $(v_2, 0, T, -\gamma)$, and since the capacity of the driver is 1, it follows that the driver has to move between v_1 and v_2 at least $2\gamma - 1$ times. Since the driver has to drive from v_D to v_1 at the beginning and return to v_D from v_2 at the end of the tour, it follows that the minimal costs for an optimal transportation schedule for solving the Static Min-Cost Relocation Problem are $2\gamma + 1$ (see Figure 7.11b for an illustration).



(a) This figure illustrates the complete weighted graph G.



(b) This figure illustrates the optimal transportation schedule solving the Static Min-Cost Relocation Problem. The tour of the driver is indicated by the dashed arcs.

Figure 7.11: This figure illustrates that the ratio between the lower bound computed by solving a vehicle routing problem and the total tour length of the optimal transportation schedule can be arbitrarily large.

An optimal solution for the vehicle routing problem starting at the depot v_D is given by the tour $\Gamma^{tsp} = (v_D, v_1, v_2)$ and its length is 3. Since γ can be selected arbitrarily hight, the ratio

between an optimal transportation schedule for the Static Min-Cost Relocation Problem and an optimal transportation schedule for the vehicle routing problem can be arbitrarily high. ◊

7.7.2 Minimal Perfect p-Matching and Car Flows

The second lower bound we consider is computed from a minimal perfect p-matching (see Section 7.4.1). For that, we first consider a special case of Static Min-Cost Relocation Problems where there is only one driver and only one depot. Afterwards, we show some relations between transportation schedules using a different amount of drivers and multiple depots. However, every driver must return to its starting depot. From that we can conclude a lower bound for the Static Min-Cost Relocation Problem with backhaul.

The next theorem has been stated and proved in [40] by Chalasani, Motwani and Rao.

Theorem 7.24. *Let $\mathcal{S}^\lambda$ be an optimal transportation schedule for a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, 1, \lambda)$ with backhaul. Let x be a minimal perfect p-matching and let $\ell(x)$ be the total length of x. Then it holds*

- $\ell(x) \leq \ell(\mathcal{S}^1)/2$,
- $\ell(\mathcal{S}^1) \leq 2L \cdot \ell(\mathcal{S}^L)$, *for all* $1 \leq L$.

The second statement of the previous theorem has been improved by a factor of 2 by Charikar, Khuller and Raghavachari in [42].

Theorem 7.25. *Let $\mathcal{S}^\lambda$ be an optimal transportation schedule for a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, 1, \lambda)$ with backhaul. Then it holds*

$$\ell(\mathcal{S}^1) \leq L \cdot \ell(\mathcal{S}^L).$$

From Theorem 7.24 and Theorem 7.25 Charikar, Khuller and Raghavachari then concluded the next corollary (see [42]).

Corollary 7.26. *Let $\mathcal{S}^L$ be an optimal transportation schedule for a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, 1, L)$ with backhaul. Let x be a minimal perfect p-matching and let $\ell(x)$ be the total length of x. Then it holds*

$$\ell(x) \leq \frac{L \cdot \ell(\mathcal{S}^L)}{2}.$$

In order to apply this statement to the case that there are k drivers in the system, we have to give some relations between the total tour lengths. For the case that there is only one depot, we already showed in Section 7.4.4 (see the proof of Theorem 7.15) that the total tour length of an optimal transportation schedule with k drivers is equal to the total tour length of an optimal transportation schedule with only one driver. Therefore, we consider the case when there are multiple depots.

Let $\mathcal{S}_k^L$ be an optimal transportation schedule for a Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ with multiple depots V^0 and with backhaul, and let $\mathcal{S}_{k+1}^L$ be an optimal transportation schedule for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k+1, L)$. If the initial number of drivers is greater or equal in every depot in $\mathcal{S}_{k+1}^L$ as in $\mathcal{S}_k^L$ then it holds

$$\ell(\mathcal{S}_k^L) \geq \ell(\mathcal{S}_{k+1}^L).$$

This can be seen as follows. Let us assume that $\ell(\mathcal{S}_k^L) < \ell(\mathcal{S}_{k+1}^L)$. Since the number of drivers in a depot for $\mathcal{S}_k^L$ is less than or equal to the number of drivers for $\mathcal{S}_{k+1}^L$ it follows that there exists a tour Γ^j in $\mathcal{S}_{k+1}^L$ which is not in $\mathcal{S}_k^L$. By fixing $\Gamma^j = \emptyset$ to the empty tour, the transportation schedule $\mathcal{S}_k^L \cup \Gamma^j$ becomes a feasible solution for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k+1, L)$ with a smaller total tour length, which contradicts the condition that $\mathcal{S}_{k+1}^L$ is an optimal transportation schedule for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k+1, L)$.

Since every transportation schedule induces a perfect p-matching (see Section 7.4.4) whose length is greater than or equal to the length of a minimal perfect p-matching x, it follows

$$\ell(x) \leq \frac{L \cdot \ell(\mathcal{S}_k^L)}{2}.$$

From that we can directly develop a lower bound for any Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$ (with a set of depots V^0) with backhaul

$$\frac{2\ell(x)}{L} \leq \ell(\mathcal{S}_k^L). \tag{7.14}$$

Also for this lower bound there exist examples when it cannot be improved.

Example 7.27. Let us consider the complete weighted graph $G = (V, E, w)$ with $V = \{v_D, v_{o1}, v_{o2}, v_{u1}, v_{u2}\}$, where v_D is the depot, v_{o1}, v_{o2} are overfull stations each with one excess car, and v_{u1}, v_{u2} are underfull stations each with one deficit car (see Figure 7.12). The weights between two stations are set to 1, the distance between v_D and v_{o1} is 0, the distance between v_D and all other stations is 1. There is only one driver with a convoy capacity of 1.

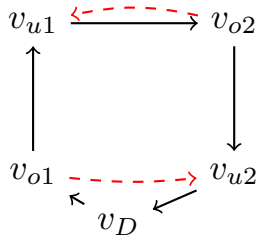


Figure 7.12: This figure illustrates a graph with five nodes, corresponding to one depot, two overfull and two underfull stations. The distance between two stations is set to 1, the distance between v_D and v_{o1} is 0. In this image, the solid arcs correspond to an optimal tour, the dashed arcs correspond to a minimal perfect p-matching.

A minimal perfect p-matching x has weight 2, and an optimal tour has length 4. Thus we have for any transportation schedule $\mathcal{S}$

$$\frac{2\ell(x)}{L} = 4 = \ell(\mathcal{S}),$$

showing that the lower bound can be equal to the total tour length of a transportation schedule. ◊

That the lower bound does not necessarily hold for the Static Min-Cost Relocation Problem without backhaul can be easily seen by slightly modifying previous example.

Example 7.28. Let us consider the complete weighted graph $G = (V, E, w)$ with $V = \{v_{d1}, v_{d2}, v_{o1}, v_{o2}, v_{u1}, v_{u2}\}$, where v_{d1}, v_{d2} are depots, v_{o1}, v_{o2} are overfull stations with one excess car, and v_{u1}, v_{u2} are underfull stations with one deficit car (see Figure 7.13). The weights between two stations are 1, the distance between v_{d1} and v_{o1} as well as the distance between v_{d2} and v_{u2} is 0, the other distances are 1. There is only one driver with a convoy capacity of 1.

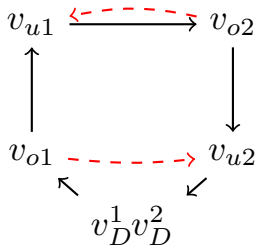


Figure 7.13: This figure illustrates a complete graph with six nodes, corresponding to two depots, two overfull and two underfull stations. The distance between two stations is set to 1, the distance between v_{d1} and v_{o1} is 0, the distance between v_{d2} and v_{u2} is 0. In this image, the solid arcs correspond to an optimal tour, the dashed arcs correspond to a minimal perfect p-matching.

A minimal perfect p-matching x has weight 2, and an optimal tour has length 3. Thus we have for any transportation schedule $\mathcal{S}$

$$\frac{2\ell(x)}{L} = 4 > 3 = \ell(\mathcal{S}),$$

showing that the lower bound can be greater than the total tour length of a transportation schedule when a tour can start and end in different depots. ◊

The previous example motivates a deeper look into the computation of this lower bound. Hereby, we consider the Static Min-Cost Relocation Problem with multiple depots without backhaul. Then a minimal perfect p-matching x gives a lower bound for the part of the transportation schedule where the cars are transfered between the stations. Twice the weight of a minimal perfect p-matching corresponds to a (or more) cycle(s) in the set of over- and underfull stations. However, when there is no backhaul, the tours may not traverse the arcs from the underfull to the overfull stations. Furthermore, neither the distances from the depots to the stations nor the distances from the stations to the depots are considered. In order to take these distances into account, we add the minimal distance from a depot to an overfull station and the minimal distance from an underfull station to a depot. From this description we gain the following formula for a lower bound for the Static Min-Cost Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ with multiple depots without backhaul

$$\frac{\ell(x)}{L} + \min_{a \in V^D \times V^O} \ell(a) + \min_{a \in V^U \times V^D} \ell(a), \tag{7.15}$$

where V_D is the set of depots, V_O the set of overfull, and V_U the set of underfull stations.

Example 7.29. We consider the graph from Example 7.28. Then the weight of a minimal perfect p-matching is 2, the arc with the maximal length has a length of 1, and the minimal distance between the depots and the overfull and underfull stations, respectively, is 0. Thus, we have

$$\frac{\ell(x)}{L} + \min_{a \in V^D \times V^O} \ell(a) + \min_{a \in V^U \times V^D} \ell(a) = \frac{2}{1} + 0 + 0 = 2,$$

which is less than the total tour length of an optimal transportation schedule. ◊

Until now, the convoy capacity has a relatively high impact on the value of the lower bound based on the minimal perfect p-matching. Therefore, we consider a lower bound where we try to reduce this impact. For that we compute a solution from a modified version of the integer linear program (7.1) where we consider only the car flow with the adjusted objective function $\sum_{a \in A_L} f(a)d(a)$. This gives us a solution with minimal costs for transferring the cars between the stations. Then we get the total distance for the drivers by computing

$$\sum_{a \in A_L} \left\lceil \frac{f(a)}{L} \right\rceil d(a),$$

where $d(a)$ is the distance between the two corresponding stations, i.e., for $a = ((v, t^v), (w, t^w))$ we have $d(a) = d(v, w)$. Hereby, only the minimal number of drivers needed to transfer the cars between the stations are taken into account. As before, we add the distances between the stations and the depots, which leads us to the formula

$$\sum_{a \in A_L} \left\lceil \frac{f(a)}{L} \right\rceil d(a) + \min_{a \in V^D \times V^O} \ell(a) + \min_{a \in V^U \times V^D} \ell(a). \tag{7.16}$$

Since the constraint matrix of the modified integer linear program is totally unimodular, this lower bound can be computed efficiently.

Finally, note that previous examples can be used to show that this lower bound cannot be improved in certain situations, or that the differences can be arbitrarily large between the computed lower bound and the total tour length of an optimal transportation schedule.

7.7.3 Lower Bounds Based on LiftFlow

In Section 7.5.4 we showed that the algorithm LiftFlow computes under certain conditions an optimal solution. In this section, we show that when "any" of these conditions does not hold, an optimal solution of the integer linear program (7.10) can still serve as a lower bound. Hereby, we must still assume that there exists an optimal solution, so that no balanced station is used as preemption station. In other words, we show in the next theorem that if any of the conditions (i)–(iv) of Theorem 7.19 does not hold (but conditions (v) holds), then an optimal solution of the integer linear program 7.10 gives a lower bound for the Static Relocation Problem.

Theorem 7.30. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^{T^A}, \gamma, k, L)$ be a Static Relocation Problem. Let f^A (car flow) and F^A (driver flow) be an optimal solution in an aggregated network G^A, so that one of the following conditions holds:*

(i) there is a cycle in the precedence graph,

(ii) there is an isolated cycle in the flows,

(iii) the capacities of the stations are not sufficiently large, and

(iv) the time horizon is not sufficiently large.

Furthermore, let there exist an optimal solution so that no balanced station is used as a preemption station. Then every feasible flows f^T (car flow) and F^T (driver flow) in the time-expanded network G^T induce

$$\sum_{a \in A_L^T} d(a)F^T(a) \geq \sum_{a \in A^A} w(a)F^A(a). \tag{7.17}$$

Proof. Under the above assumptions, it follows from the optimality of F^T and f^T, that (7.17) holds for any solution of $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$.

"(iv)": let T^A be the makespan of the transportation schedule induced from the solution f^A and F^A. Then every solution with a makespan of T is also a solution for the Static Relocation Problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^{T^A}, \gamma, k, L)$. Thus, the optimal solution for this problem cannot be greater than any solution for the original problem $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$, and it follows that

$$\sum_{a \in A_L} d(a) F^T(a) \geq \sum_{a \in A_A} w(a) F^A(a),$$

holds, proving the statement if condition (iv) holds.

"(iii)": this condition only affects preemption stations. Otherwise, either $\boldsymbol{z}^0$ or $\boldsymbol{z}^T$ would be infeasible, contradicting the definition of a Static Relocation Problem.

If there exists an optimal solution so that condition (iii) does not hold, then two objective function values are equal. Otherwise, since f^A and F^A are optimal for the integer linear program (7.10), and due to Theorem 7.18 it follows that there exists an optimal solution f^T and F^T for the integer linear program (7.1), so that

$$\sum_{a \in A_L} d(a) F^T(a) \geq \sum_{a \in A_A} w(a) F^A(a),$$

holds.

"(i)": in order to prove this case, let us assume that neither (iii) nor (iv) holds. Otherwise, the statement is already shown above.

If there exists an optimal solution for the integer linear program (7.10) so that none of the stated conditions holds, then it follows from Theorem 7.19 that this solution is optimal for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$, proving the statement.

Therefore, let us assume that there does not exist an optimal solution for the integer linear program (7.10) so that none of the stated conditions holds. Since f^T and F^T are an optimal solution for the Static Relocation Problem, it follows from Theorem 7.18 that there exist flows $\tilde{f}^A$ and $\tilde{F}^A$ with

$$\sum_{a \in A_L} d(a) F^T(a) = \sum_{a \in A_A} w(a) \tilde{F}^A(a).$$

Since f^A and F^A are a minimal solution for the integer linear program (7.10) it follows that

$$\sum_{a \in A_A} w(a) F^A(a) \leq \sum_{a \in A_A} w(a) \tilde{F}^A(a),$$

holds, and thus,

$$\sum_{a \in A_A} w(a) F^A(a) \leq \sum_{a \in A_L} d(a) F^T(a),$$

proving the statement if condition (i) holds.

"(ii)": This case can be proved analogously to (i). □

As we already stated above, whether condition (v) holds can (until now) only be ensured by optimally solving the Static Relocation Problem or by considering a further restricted problem, where only solutions are accepted which allow drivers to pickup or drop cars at overfull and underfull stations. However, in the case when condition (v) does not hold, we can still give a lower bound with the help of the maximal gaps between preemptive and non-preemptive transportation schedules (see Section 9.1). Since we consider the Static Relocation Problem with one depot, and we do not allow inner preemption in balanced stations, the next statement follows together with previous theorem and Theorem 9.3 (ii).

Corollary 7.31. *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$ be a Static Relocation Problem. Let f^A and F^A be an optimal solution in an aggregated network G^A. Then*

$$\frac{1}{8} \sum_{a \in A^A} C(a) F^A(a)$$

is a lower bound for an optimal solution for $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$.

Chapter 8

Static Max-Profit Relocation Problem

In general, there does not always exist a feasible solution for the Static Min-Cost Relocation Problem for the given time horizon T. In this case, it is generally of interest having a transportation schedule which transforms the start system state into a system state as similar as possible to the target system state. Therefore, we consider in this chapter, a max-profit version of the Static Relocation Problem, where the objective is to maximize the number of cars transferred to their destination, while minimizing the costs of transferring cars. That means, for a given destination system state $\boldsymbol{z}^T$, the objective is to find a transportation schedule $\mathcal{S}$ that minimizes $\left|\boldsymbol{z} - \boldsymbol{z}^T\right|_1 + cost(\mathcal{S})$, where $\boldsymbol{z}$ is the state reached at the end of the time horizon, and $cost(\mathcal{S})$ are the costs induced by $\mathcal{S}$. Since the target system state is not a hard constraint, as it is in the Static Min-Cost Relocation Problem, there is always a feasible solution for the Static Max-Profit Relocation Problem.

In order to solve the Static Max-Profit Relocation Problem, we present an exact solution based on flows in a time-expanded network. Firstly, we model the max-profit problem in a natural way. Obviously, the part of the objective function

$$\left|\boldsymbol{z} - \boldsymbol{z}^T\right|_1 = \sum_{v \in V} \left|z_v - z_v^T\right|$$

is non-linear and, therefore, an integer linear program based on this objective function is non-linear as well (Section 8.1). Therefore, we give an alternative model for the Static Max-Profit Relocation Problem with an alternative linear objective function whose optimization problem is linear (Section 8.2).

In this chapter we consider the preemptive situation when there is only one depot in the system. In order to obtain a non-preemptive transportation schedule, and/or a transportation schedule with (resp. without) backhaul, one can adjust the flows in the network and apply the constructions and ideas from Sections 7.1.

The name for this problem is motivated by the consideration of profits attached to system states which are similar to the destination state. A further motivation is also given in the second section of this chapter.

8.1 Integer Non-Linear Program

For solving a max-profit version of the Static Relocation Problem, we construct a time-expanded version G_T of the original graph G as done before in Section 7.1. However, there are some small differences to that time-expanded network, which we briefly discuss in this section. The node set

V_T contains beside the nodes $(v,t) \in V_T$, representing station v at time t, also a sink node D. Additionally, the arc set $A_T = A_H \cup A_L \cup A_S$ of G_T is extended by the set A_S of sink arcs connecting (v,T) to D.

As before, we define two flows on the time-expanded network, a driver and a car flow. The flows are initialized with the nodes $(v,0) \in V_T$ as sources, and on all internal nodes we use the standard flow conservation constraints. Unlike before, we ensure that the destination state can be reached and each driver returns to the depot, by bounding the flows on the sink arcs. On the sink arc $a = [(v_D,T),D] \in A_S$, connecting the node representing the depot to the sink, i.e., $(v_D,T) \in V_T$ to D, we set $F(a) = k$ (see (8.1c)), for all other sink arcs $a' \in A_S \setminus \{a\}$ we set $F(a') = 0$ (see (8.1c)). For the car flow we set the sum of all sink arcs to the total number of cars, i.e., $\sum_{a \in A_S} f(a) = \sum_{v \in V} z_v^0$ (see (8.1c)). With the help of the sink arcs, it is easier to count the number of cars reaching a certain station.

The objective function (8.1a) measures and minimizes the costs of transporting cars in convoys between the stations, while minimizing the difference between the final system state at time T and the target system state. Since car flow on the sink arcs corresponds to the number of cars at time t at the corresponding station, we have as objective function

$$\sum_{a=((v,T),D) \in A_S} \Delta \left| f(a) - z_v^T \right| + \sum_{a \in A_L} c(a) f(a) + \sum_{a \in A_L} C(a) F(a).$$

In order to force the solver to stress more on reaching a system state more similar to the destination state than on reducing the costs for transferring cars within the system, we add a constant Δ to the objective function. Obviously, for this the constant must be selected sufficiently large.

In order to solve Static Max-Profit Relocation Problem exactly, we present an integer nonlinear programming formulation for a min-cost flow problem in the time-expanded network $G_T = (V_T, A_T)$ as follows:

$$\min \sum_{a=((v,T),D) \in A_S} \Delta \left| f(a) - z_v^T \right| + \sum_{a \in A_L} c(a) f(a) + \sum_{a \in A_L} C(a) F(a) \tag{8.1a}$$

$$\sum_{a \in \delta^-(v,0)} f(a) = z_v^0, \quad \sum_{a \in \delta^-(v_D,0)} F(a) = k \qquad \forall (v,0) \in V_T \tag{8.1b}$$

$$\sum_{a \in A_S} f(a) = \sum_{v \in V} z_v^T, \quad \sum_{a \in \delta^+(v_D,T)} F(a) = k \qquad \forall (v,T) \in V_T \tag{8.1c}$$

$$\sum_{a \in \delta^-(v,t)} f(a) = \sum_{a \in \delta^+(v,t)} f(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{8.1d}$$

$$\sum_{a \in \delta^-(v,t)} F(a) = \sum_{a \in \delta^+(v,t)} F(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{8.1e}$$

$$0 \le f(a) \le \mathrm{cap}(v) \qquad \forall a = [(v,t),(v,t+1)] \in A_H \tag{8.1f}$$

$$f(a) \le L \cdot F(a) \qquad \forall a \in A_L \tag{8.1g}$$

$$f, F \text{ integer.} \tag{8.1h}$$

As before, the flows in the time-expanded network have to be interpreted as a transportation schedule. Due to the construction of the time-expanded network, all dependencies over time are properly respected by the constraints, implying the next theorem.

Theorem 8.1. *Flows constrained by the system* (8.1) *correspond to a preemptive transportation schedule for the Static Max-Profit Relocation Problem* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \gamma, k, L)$.

8.2 Linearizing the Integer Non-Linear Program

The objective function can be modified to be quadratic

$$\min \sum_{a=((v,T),D)\in A_S} \Delta \left(f(a) - z_v^T\right)^2 + \sum_{a\in A_L} c(a)f(a) + \sum_{a\in A_L} C(a)F(a)$$

which may speed up the computation times of solving the problem.

However, usually the best performance improvement can be achieved by finding a natural formulation resulting in an integer linear program. In the literature, there exist several standard methods to linearize integer quadratic programs, e.g., [24, 89, 116]. In this section, we modify the construction of the time-expanded network of the previous section so that the resulting program is linear. This resulting integer linear program can then be used by standard solver to solve the Static Max-Profit Relocation Problem.

Underfull stations can be considered as stations from which customers will take cars early in the morning. Therefore, we artificially add some customer requests which represent the deficits of underfull stations. Thus, serving a customer request corresponds to bringing a car to an underfull station. There is a profit attached to each customer request, and serving a customer request increases the total profit. This forces the solver to find solutions where as many customer requests are served as possible. From that one can follow, that if every customer request is served, the target system state $\boldsymbol{z}^T$ is reached at the end of the time horizon.

Node Set. In addition to the time-expanded network of previous section, we add a further sink node D^{cr} to the node set A_T. This sink node corresponds to the "target station/node" of the artificially added customer requests.

Customer Requests. Next, we describe customer requests that we artificially add to the system. Every added customer request can be rejected. However, to ensure that as many customer requests are served as possible, there is a profit on each added customer request.

The set of customer requests $R = \emptyset$ is initialized with the empty set. For every underfull station $v \in V$, i.e., every station $v \in V$ with $z_v^0 - z_v^T < 0$, we add $\left|z_v^0 - z_v^T\right|$ customer requests of the form (v, T, D^{cr}, T) to the set R.

Arc Set To the arc set $A_T = A_H \cup A_L \cup A_R \cup A_S$ of G_T we append the set customer request arcs (denoted by A_R). Hereby, we add an arc to A_R, connecting (v, T) to D^{cr}, for each customer request $r = (v, T, D^{cr}, T) \in R$.

Flow Model. On the time-expanded network G_T, we define two different flows, the car flow f and the driver flow F.

Customer request arcs can only be used by cars and not by drivers and, therefore, have an upper bound of 1 for the car flow f (see (8.2h)) and an upper bound of 0 for the driver flow F (see (8.2i)). In order to encourage the solver to serve as many customer requests as possible, there is a profit p for each served customer request. Note that in the case when there are only depots separate from the stations, there cannot be a driver flow on a request arc, and the upper bound of 0 for the driver flow becomes obsolete.

To ensure that the destination state is reached and each driver returns to the depot, we use as sink D, for both flows, and the sink D^{cr} only for the car flow. For the sink arc $a = [(v_D, T), D]$, connecting the depot v_D to the sink D, we set $F(a) = k$, and for every other sink arc $F(a') = 0$ (see (8.2c)). With respect to the car flow f the sink arcs are bounded to the capacities of the

corresponding station. Thus, we must take the cars moved to the station as well as the cars located from the beginning at the station into account (see (8.2c))

$$\sum_{a \in \delta^+(v,T)} f(a) + b_v \leq \text{cap}(v),$$

with $a = ((v,T), D)$ and

$$b_v = \begin{cases} z_v^0 - \max\{z_v^0 - z_v^T, 0\} = z_v^T, & \text{if } v \text{ is overfull,} \\ z_v^0, & \text{otherwise.} \end{cases}$$

Hereby, remember that the flows are initialized only by the number of cars that need to be transferred from the overfull stations.

The objective function (8.2a) measures and minimizes the costs for transferring cars by the drivers through the system and maximizes the profit for serving customer requests.

To solve the Static Max-Profit Relocation Problem we give an integer linear programming formulation for a max-profit flow problem in the time-expanded network $G_T = (V_T, A_T)$ as follows:

$$\max \sum_{a \in A_R} pf(a) - \sum_{a \in A_L} c(a)f(a) - \sum_{a \in A_L} C(a)F(a) \tag{8.2a}$$

$$\sum_{a \in \delta^-(v,0)} f(a) = \max\{z_v^0 - z_v^T, 0\}, \sum_{a \in \delta^-(v_D,0)} F(a) = k \qquad \forall (v,0) \in V_T \tag{8.2b}$$

$$\sum_{a \in \delta^+(v,T)} f(a) + b_v \leq \text{cap}(v), \sum_{a \in \delta^+(v_D,T)} F(a) = k \qquad \forall (v,T) \in V_T \tag{8.2c}$$

$$\sum_{a \in \delta^-(v,t)} f(a) = \sum_{a \in \delta^+(v,t)} f(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{8.2d}$$

$$\sum_{a \in \delta^-(v,t)} F(a) = \sum_{a \in \delta^+(v,t)} F(a) \qquad \forall (v,t) \in V_T, 0 < t < T \tag{8.2e}$$

$$0 \leq f(a) + b_v \leq \text{cap}(v) \qquad \forall a = [(v,t),(v,t+1)] \in A_H \tag{8.2f}$$

$$f(a) \leq L \cdot F(a) \qquad \forall a \in A_L \tag{8.2g}$$

$$f(a) \leq 1 \qquad \forall a \in A_R \tag{8.2h}$$

$$F(a) = 0 \qquad \forall a \in A_R \tag{8.2i}$$

$$f, F \text{ integer.} \tag{8.2j}$$

Finally, note that if all customer requests are served then the derived transportation schedule is also a feasible solution for the Static Min-Cost Relocation Problem. Thus, one can use this model to try to compute a solution for the min-cost version of the Static Relocation Problem, and has a fallback solution if there does not exist a transportation schedule with a makespan less than or equal to T.

Chapter 9

Computational Results

In this chapter, we present some computational results for the Static Min-Cost Relocation Problem. Since some algorithms compute a preemptive and some a non-preemptive transportation schedule, we first give some theoretical results on the ratio between optimal preemptive and non-preemptive transportation schedules (Section 9.1). Hereby, we show that, under certain assumptions, the ratio is constant. Afterwards, in Section 9.2, we present and discuss the computational results.

9.1 Preemption vs Non-Preemption

In this section, we compare the optimal (probably) preemptive transportation schedule with the optimal non-preemptive transportation schedule. That means, we compare a transportation schedule $\mathcal{S}^{pre}$ with a shortest total tour length $\ell(\mathcal{S}^{pre})$ which can be preemptive (or not) with a transportation schedule $\mathcal{S}^{non}$ with a shortest total tour length $\ell(\mathcal{S}^{non})$ which is non-preemptive. Since $\mathcal{S}^{pre}$ is allowed to be non-preemptive as well it is obvious that

$$\ell(\mathcal{S}^{pre}) \leq \ell(\mathcal{S}^{non})$$

holds, i.e., that the total tour length of an optimal preemptive transportation schedule is always smaller than or equal to the total tour length of a non-preemptive version. So the question we try to answer in this chapter is whether there exists a constant $c \in \mathbb{R}^+$ so that the total tour length of the non-preemptive transportation schedule $\mathcal{S}^{non}$ can be bounded by $c \cdot \ell(\mathcal{S}^{pre})$, i.e., so that

$$\ell(\mathcal{S}^{non}) \leq c \cdot \ell(\mathcal{S}^{pre}) \tag{9.1}$$

holds.

Throughout this section we assume that the time horizon and capacity of the stations are infinite (resp. sufficiently large).

In the literature, there are some results known based on experimental comparison between generated preemptive and non-preemptive tours, see, e.g., for the dial-a-ride problem [62]. To the best of our knowledge, there is no theoretical result for the Relocation Problem with k drivers with a constant worst-case ratio. A theoretical result for the case with only one driver is given by Charikar, Khuller and Raghavachari in [42].

Theorem 9.1 (Charikar, Khuller, and Raghavachari [42])**.** *Let $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, 1, L)$ be a Static Relocation Problem with only one driver. Let $\mathcal{S}^{pre}$ be an optimal transportation schedule with*

inner preemption and $\mathcal{S}^{non}$ *be an optimal transportation schedule without inner preemption. Then it holds*

$$\ell(\mathcal{S}^{non}) \leq 4 \cdot \ell(\mathcal{S}^{pre}).$$

We concentrate on the case that there are k drivers with $k \geq 2$, and that there are preemptions between the tours.

It is relatively easy to prove an upper bound on the gap which depends on the number of drivers in the system and their convoy capacities, respectively.

Theorem 9.2. *Let* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, \mathcal{Z}, \gamma, k, L)$ *be a Static Relocation Problem. Let* $\mathcal{S}$ *be an optimal transportation schedule with preemption, and let* $\mathcal{S}^*$ *be an optimal transportation schedule without preemption. Then*

$$\ell(\mathcal{S}^*) \leq \min\{(8k-1), (8L-1)\} \cdot \ell(\mathcal{S})$$

holds. If $\mathcal{S}^*$ *is allowed to contain tours with inner preemption, then*

$$\ell(\mathcal{S}^*) \leq \min\{(2k-1), (2L-1)\} \cdot \ell(\mathcal{S})$$

holds.

Next, we state and prove the main statement of this section, the existence of a constant so that (9.1) holds when there are several drivers.

Theorem 9.3. *Let* $(G, \boldsymbol{z}^0, \boldsymbol{z}^T, k, L)$ *be a Static Relocation Problem for* k *drivers and so that there exists an optimal preemptive transportation schedule* $\mathcal{S}^{pre}$ *and an optimal non-preemptive transportation schedule* $\mathcal{S}^{non}$*.*

If there is only one depot (or each driver can end its tour in any depot), then we have

(i) $\ell(\mathcal{S}^{non}) < 2 \cdot \ell(\mathcal{S}^{pre})$ *if inner preemption is allowed, and*

(ii) $\ell(\mathcal{S}^{non}) < 8 \cdot \ell(\mathcal{S}^{pre})$ *if inner preemption is not allowed.*

If there are several depots and each driver has to end its tour in a given depot, then we have

(iii) $\ell(\mathcal{S}^{non}) \leq 3 \cdot \ell(\mathcal{S}^{pre})$ *if inner preemption is allowed, and*

(iv) $\ell(\mathcal{S}^{non}) \leq 12 \cdot \ell(\mathcal{S}^{pre})$ *if inner preemption is not allowed.*

Proof. Obviously, it is sufficient to show that (i) (resp. (iii)) holds. Then statement (ii) (resp. (iv)) directly follow from Theorem 9.1. Furthermore, we show (i) and then conclude that (iii) holds as well. Hereby, the very basic idea is to construct a new tour with only one driver serving all transportation tasks and where the length of the constructed tour is at most twice the total tour length of the original preemptive transportation schedule. For that, the driver transfers "all" cars to the next precedence station before continuing the tours. Since there is only one tour in the end, there are obviously no precedences between the tours. However, in the constructed tour there are inner preemptions. Finally, note that the total tour length decreases when more drivers are used (if these transportation schedules are optimal).

For the first part of the proof, we assume, for the sake of simplicity, that there is only one depot. However, the proof can be easily generalized.

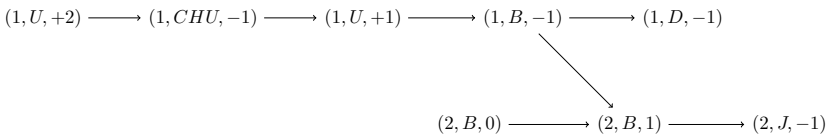
Let $\mathcal{S}$ be a transportation schedule with at least two drivers and let there be preemption between the tours. Then there exists a precedence graph G^P. We start by considering the case when there are no undirected cycles in G^P. Afterwards, we show that the statement also holds when there are undirected cycles in the precedence graph. Furthermore, let G^P be connected, otherwise we consider the components separately.

Next, we construct a directed graph G^T from G^P. Afterwards, we construct a path in G^T and then a tour from this path.

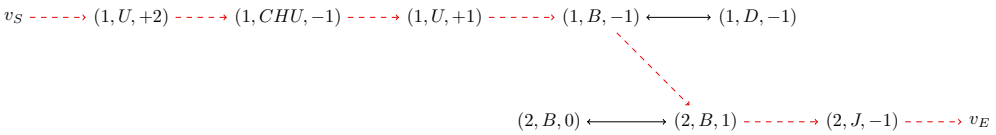
The tour construction graph $G^T = (V^T, A^T)$ is defined as follows. The node set V^T contains the nodes from the precedence graph G^P, as well as two further nodes, a start node v_S and an end node v_E. These two additional nodes correspond to the start and end node of the newly constructed tour. Due to technical reasons, the nodes v_S and v_E are empty actions in the depot.

The arc set A^T contains all arcs from G^P. Furthermore, we add an arc from v_S to a node in G^P which corresponds to the first action of a tour, and an arc from a node in G^P which corresponds to the last action of a tour. Finally, we add some more arcs to G^T as follows: Let σ be a path in G^T from v_S to v_E, the so-called main path. For every arc $(a_1, a_2) \in A^T$ with $(a_1, a_2) \notin \sigma$ we add a backward arc (a_2, a_1) to A^T. Arcs in the set $A^T \setminus \sigma$ are called forward arcs. Note that forward and backward arcs can be canonically transformed to moves.

See Figure 9.1 for an illustration of the construction of G^T.



(a) This figure illustrates the precedence graph of the transportation schedule from Example 2.4.



(b) In this figure, the tour construction graph is shown. The dashed arcs correspond to the main path, the solid arcs to the forward and backward arcs.

Figure 9.1: This figure illustrates the construction of the tour construction graph from the precedence graph of the transportation schedule from Example 2.4.

A forward arc which corresponds to a canonical precedence in a tour is called tour arc, and forward arcs corresponding to the precedences between tours are called precedence arcs.

If one contracts the two nodes v_S and v_E, for each node the number of incoming arcs and the number of outgoing arcs are equal (see Figure 9.2 for an illustration). Thus, it follows from Euler's theorem that there exists an Eulerian cycle in G^T, motivating that there exists a path in G^T from that one can construct a tour with a tour length at most twice the total tour length of the original transportation schedule.

Next, we construct a path from v_S to v_E in G^T traversing each arc exactly once and show how to construct a tour from this path.

Claim 9.3.1. Let the current node in Algorithm 15 be $v \in \sigma$. Then Algorithm 15 traverses all forward and back arcs that are reachable from v and returns to v without traversing an arc in σ.

Proof. Note, that Algorithm 15 is in fact a depth-first search algorithm with some special rules given to lead the direction of the search. One can easily prove the statement by transforming the subgraph G of G^T, containing all forward and back arcs that are reachable from v without traversing an arc in σ and the respecting nodes, into a maze. Then Algorithm 15 corresponds to the Wall-Follower Algorithm (e.g., [23, 133, 142, 148]). ◊

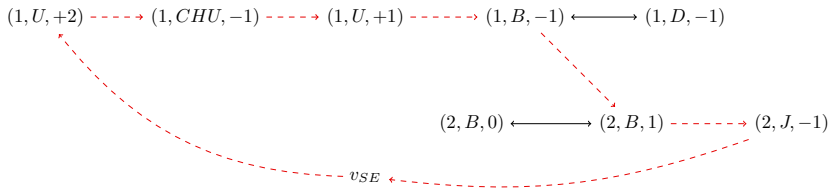


Figure 9.2: This image illustrates how to prove that there always exists a path from v_S to v_E traversing every arc exactly once. For that, the two nodes v_S and v_E are merged to the node v_{SE}. Then there exists an Eulerian path in this graph. Since this walk has to traverse the incoming and outgoing arcs of v_{SE}, there exists a path in G^T from v_S to v_E.

Algorithm 15 Construct path in G^T

Input: a connected tour construction graph G^T
Output: a path traversing each arc G^T exactly once

```
initialize the path P ← ∅
initialize current node v ← v_S
let σ the be main path in G^T
follow the successor arc of v_S and add it to P
while v ≠ v_E do
    if there is an unmarked outgoing backward arc a_b from v then
        follow and mark a_b                          ▷ updates v as well
    else if there is an unmarked outgoing forward arc a_f from v then
        follow and mark a_f
    else
        follow and mark the arc from σ
return the constructed path P
```

Next, we show the construction of a tour from the path $P = ((v_S, a_1), (a_1, a_2), \ldots, (a_q, v_E))$ returned by Algorithm 15. For that we iterate through the arcs of P, and, depending on the current arc (a_i, a_{i+1}), we add the following moves and actions to a sequence Φ:

(ct.i) (a_{i+1}, a_i) is a backward arc: add action that unloads all cars in the station $loc(a_{i+1})$, then add a move from $loc(a_{i+1})$ to $loc(a_i)$;

(ct.ii) (a_i, a_{i+1}) is a tour arc: add a_i to the tour, a move from $loc(a_i)$ to $loc(a_{i+1})$;

(ct.iii) (a_i, a_{i+1}) is a precedence arc: add action that unloads all cars in the station $loc(a_i) = loc(a_{i+1})$. Add a pickup action which picks up that many cars as the original tour contains before performing a_{i+1}.

Afterwards, all empty actions and moves are removed from Φ, and two successive actions (resp. moves) are contracted to one action (resp. move) to transform Φ into a tour.

Example. Let us consider the tour construction graph G^T from Figure 9.1. Then we gain the

following sequence Φ from G^T

{
$(1, v_D, 0)$, % add empty action at v_D ($\equiv v_S$) (Case (ct.ii)), and
$(1, v_D, 0, U, 2)$, % move to U
$(1, U, +2)$,
$(1, U, 2, CHU, 3)$,
$(1, CHU, -1)$,
$(1, CHU, 3, U, 4)$,
$(1, U, +1)$,
$(1, U, 4, B, 5)$,
$(1, B, -1)$,
$(1, B, 5, D, 6)$,
$(1, D, -1)$,
$(1, D, 0)$, % unload all cars (Case (ct.i)), and
$(1, D, 6, B, 7)$, % move to B
(1, B, 0), % unload all cars in B (Case (ct.iii)), and
(1, B, 0), % in the original tour there are 0 cars before arriving at B
(1, B, 0), % unload all cars (Case (ct.i)), and
(1, B, 7, B, 7), % move to B
(1, B, 0), % add action $(2, B, 0)$ with adjusted driver (Case (ct.i)), and
(1, B, 7, B, 7), % move to B
(1, B, 1), % add action $(2, B, 1)$ with adjusted driver (Case (ct.ii)), and
$(1, B, 7, J, 9)$, % move to B
$(1, J, -1)$,
$(1, J, 9, v_D, 13)$ % after arriving at v_D it is not necessary to add the artificially added action v_E
}

Finally, the highlighted moves and actions at B are either removed or contracted. ◊

Next, we show that this construction always constructs a feasible tour. Since all tours end in the depot, the last action of a tour unloads the remaining cars from a convoy. If the last action a of a tour is on the main path, then it follows from (ct.ii) that all cars are unloaded, otherwise, it follows from either (ct.i) (if a is a leaf in G^P) or (ct.iii) (if a is the start or end node of a precedence arc).

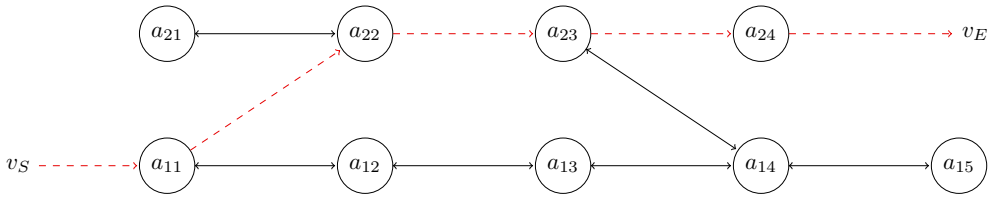
Since the precedence graph is induced by a transportation schedule, it follows that one can always compute the number of cars in any point of time within a tour. Thus, the number needed for (ct.iii) can easily be computed. However, it is still necessary to show that there are enough cars available when the successive action is performed. However, this follows from the assumption that there are no cycles within the precedence graph, and that the outgoing backward arcs of a node v are traversed before the outgoing forward arcs of v. Therefore, all precedences are satisfied and the number of cars at the corresponding station is sufficiently large.

Finally, note that all actions in G^T are performed at most once, namely only when Case (ct.ii) is applied. Since every arc is traversed exactly once, it follows that each action is performed at most once.

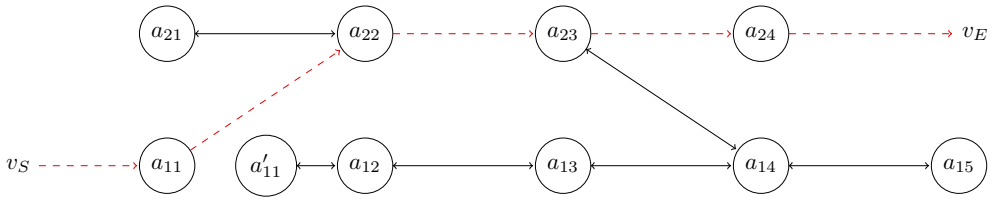
Since every arc in G^T is traversed exactly once, and the arcs correspond to moves in a tour within the original transportation schedule $\mathcal{S}$ (or to a waiting move), it follows that the length of the newly constructed tour is at most twice the total tour length of $\mathcal{S}$, and the statement (i) follows for the case when there are no cycles in G^P.

Next, we prove the statement when there are (undirected) cycles in the precedence graph G^P. As before, we construct from G^P the corresponding tour construction graph G^T. Then, we apply an algorithm on G^T, which "breaks" cycles in G^T in such a way that Algorithm 15 can be applied to the tour construction graph and so that the resulting tours are feasible.

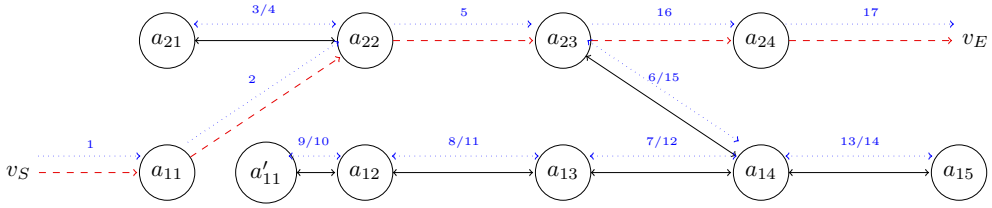
In order to compute a path in G^T, we extend Algorithm 15. Hereby, we dynamically "break" the cycles in a way such that it is ensured that the resulting path can be used to compute a feasible tour. In Figure 9.3 we give an illustration of Algorithm 15.



(a) This image shows a tour construction graph G^T.



(b) In this image, the current node v is a_{11}. Since the arc (a_{11}, a_{22}) is in the main path, it follows that the arc (a_{11}, a_{12}) is considered first. However, this arc is within a cycle and, thus, it is removed (line 10). Furthermore, since (a_{11}, a_{12}) is a tour arc, a new node/action and arc are added to G^T (lines 11–13). Afterwards, G^T is cycle-free, and the remaining steps on this graph are like the steps of Algorithm 15.



(c) This image show the order in that the arcs are traversed by Algorithm 16. The number j above the dotted arcs corresponds to the jth traversed arc. Hereby, the second arc is traversed after the cycle is "broken" as illustrated in 9.3b.

Figure 9.3: This image illustrates the breaking of a cycle as in Algorithm 15. For that we consider two tours with two preemptions so that the precedence graph contains an undirected cycle. From the precedence graph of these tours, a tour construction graph G^T is computed. The dashed arcs correspond to the main path, the solid arcs to the forward and backward arcs.

Algorithm 16 Construct path in G^T (induced from G^P with cycles)

Input: a connected precedence graph G^P, the tour construction graph G^T of G^P
Output: a path traversing each arc G^T exactly once

1: initialize the path $P \leftarrow \emptyset$
2: initialize current node $v \leftarrow v_S$
3: let σ the be main path in G^T
4: follow the successor arc of v_S and add it to P
5: **while** $v \neq v_E$ **do**
6: | **if** there is an unmarked outgoing backward arc a_b from v **then**
7: | | follow and mark a_b ▷ updates v as well
8: | **else if** there is an unmarked outgoing forward arc a_f from v **and** the corresponding backward arc a_b to a_f is also unmarked **then**
9: | | **if** a_f is in a cycle in G^P **then**
10: | | | remove a_f and a_b (also from G^P)
11: | | | **if** $a_f = (a_1, a_2)$ is a tour arc **then**
12: | | | | add action $a = (1, v, x)$ with x is the number of cars in the convoy after a_1 is performed in the original tour
13: | | | | add tour arc (a, a_2) and (a_2, a) to G^T
14: | | **else**
15: | | | follow and mark a_f
16: | **else if** there is an unmarked outgoing forward arc a_f from v **then** ▷ the corresponding backward arc a_b to a_f is marked
17: | | follow and mark a_f ▷ a_f cannot be in a cycle in G^P
18: | **else**
19: | | follow and mark the arc from σ
20: **return** the constructed path P

In the following, we prove the correctness of Algorithm 15. Hereby, we also show that resulting path can be used to construct a feasible tour.

Since an outgoing forward of a node v is traversed after all incoming forward arcs of v are traversed, it is ensured that there are enough cars for the artificially added actions.

Next, we show that if an outgoing forward arc a_f of v is unmarked and the corresponding backward arc is marked, that a_f cannot be in a cycle in G^P. Hereby, it is possible that a_f was originally in a cycle but it has been removed in Line 10. Note, when there are no (undirected) cycles in G^P, the construction of a tour is independent of the order in which the forward arcs are traversed.

Since there are no directed cycles, it is not possible that v can be reached twice by only traversing backward arcs. Cycles are removed when there are no outgoing backward arcs to a node. Therefore, if v is the current node for the second time, all cycles "behind" v must have already been removed and v is reached for the second time by traversing a forward arc. Thus, it follows, whenever an outgoing forward arc a_f is not marked, but the corresponding incoming backward arc is marked, that a_f is not within a cycle.

Finally, since a tour can be constructed from the path returned by Algorithm 15, and the path traverses each arc in G^T exactly once, the statement (i) follows in the case that there are cycles in G^P.

In order to prove statement (iii) note that all other driver follow their original tour without performing any action. Therefore, all tour arcs are traversed at most three times, all other arcs at

most twice, and the statement directly follows from above. This finally proves this theorem. □

From the proof in previous theorem, one can ensure that the longest path in G^P is traversed only once. Thus, one can easily follow that in the special case when there is only one precedence between two tours, then the maximal ratio between the preemptive and non-preemptive transportation schedule is at most 3/2 if inner preemption is allowed (resp. 6 if inner preemption is not allowed) and there is only one depot.

For the case when the capacities of the stations are limited, the ratio is unknown to the best of our knowledge. Until now, we have not found an example with a ratio larger than 4. However, a proof or counter example that the ratio is maximal 4 (or larger) is still an open task for the future.

9.2 Test Instances

The tests have been run on a server with Intel Xeon E7-8870 processors clocked at 2.40 GHz. In total there are 160 kernels, and in total 1 TB RAM available. The operating system is CentOS 6.6 with the Linux kernel 2.6.32.

The combinatorial approaches are implemented either in Java 6 or C++ (gcc 4.4.7). Optimization problems (i.e., the integer linear programs) are solved with Gurobi 6.0 using the Python interface. In order to run several instances at the same time we use the gnu tool "parallel" [151].

Besides the optimal approach with flows in time-expanded networks, we tested the following algorithms: Greedy, ReOpt and LiftFlow. Since the solutions computed by Greedy and LiftFlow often exceed the time-horizon, they have been tested in combination with the approach to handle solutions exceeding the time-horizon from Section 7.6 together with the randomized approach from Section 7.3. Furthermore, we tested the lower bounds based on computing a minimal perfect p-matching or the car flows (see Section 7.7.2) and the lower bound based on the approach LiftFlow from Section 7.7.3.

Since the computation of a lower bound based on solving a vehicle routing problem (see Section 7.7) is itself an $\mathcal{NP}$-hard problem, we did not test this lower bound as it does not give us any additional value.

All approaches have been tested on a total of 480 randomly generated "small" test instances. Hereby, the instances have 15 stations; the number of overfull and underfull is to 4 or 7, the number of drivers 2 and 5, convoy capacities of 2 and 5, time horizons are set to 90 time units, and the excess and deficit of the overfull and underfull stations is between 1 and 12. Furthermore, there are 30 instances per permutation, this means that there are, e.g., 30 instances with 2 drivers with a convoy capacity of 3, and 4 overfull and 7 underfull stations. Note that the sizes of these instances corresponds to small car- or bikesharing systems or to clusters of larger systems.

Furthermore, we tested the approaches on 30 medium sized test instances with 30 stations. The number of overfull and underfull stations are each set to 12, and the excess and deficit of the overfull and underfull stations is between 1 and 5. There are three drivers with a convoy size of 5 in the system. The time horizons are set to 300 time units.

Finally, we generated 30 "big" test instances. Each test instance contains 50 stations, 23 overfull and 23 underfull stations with an excess and deficit between 1 and 5. There are 5 drivers with a convoy size of 5. As for the medium sized test instances, the time horizons are set to 300 time units.

In all test instances, the stations are uniformly distributed over a plane with a width and height of 10 units (resp. 20 for the medium sized test instances), and added distances between two stations correspond to the rounded Euclidean distances. Hereby, not all roads are added to

the system, but it is ensured during the generation of the carsharing system that the resulting graphs are connected. Figure 9.4 illustrates an example for a graph of such a test instance.

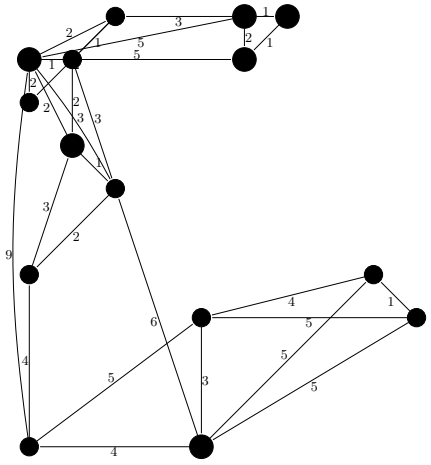


Figure 9.4: The image shows the graph of a test instance. The weights of the edges correspond to the distances between the connecting stations.

For computing an exact solution, we set a time limit of 2 hours, for the greedy heuristic itself there is no time limit but for the randomized approach we set the time limit to 1 hour (60 iterations, each limited to 1 minute), LiftFlow has a time limit of 20 minutes which was never reached. The combinatorial approach ReOpt does not have a time limit. The exact approach found an optimal solution for five test instances within the given time limit. Finally, the approach LiftFlow always found an optimal solution in the aggregated network, thus, giving at least a lower bound on the other instances.

The results of the computational experiments are summarized in Table 9.1. Hereby, we give the averages of the percentages of the gaps between the different approaches and the best known lower bounds. That the gap becomes bigger with an increasing convoy capacity, can be explained due to the fact that the lower bounds depend on the convoy capacity (see Section 7.7).

In Table 9.2 we highlight the results for which an optimal solution is known. Note that the combinatorial algorithm ReOpt computes non-preemptive transportation schedules which may lead to larger total tour lengths even if the optimal (non-preemptive) solution is found. That in some cases no solution could be found may result from a too tight time horizon where either no or only few non-preemptive transportation schedules exists.

As expected the exact approach takes generally too much time to solve even small instances (see Table 9.3). The relative fast average computational times for the smaller instances can be explained since there are several infeasible test instances, which could be proven within seconds. Furthermore, it does not always find the best solution of all the tested approaches.

The relative fast computation times for the instances with 12 overfull and 12 underfull stations for the combinatorial approach ReOpt but the relative slow computation times for the instances with 7 overfull and 7 underfull stations can be explained by the (likely) tight time horizon of the "smaller" instances, while the "medium" sized instances have a generous time horizon. Interestingly, the computation times differ slightly but noticeable when there are more overfull than underfull stations and the inverse, respectively. Currently, it is unknown whether this is an effect of the considered test instances or a general phenomenon. However, since there are different numbers of infeasible test instances in both sets, it is not unlikely that this observation

Table 9.1: This table summarizes the results of all approaches on the test instances and gives the percentage of the gap between the approach and the best known lower bound. In columns one to five, show the different variables of the test instances. The other columns show the total tour lengths by the corresponding algorithm. We consider the following algorithms: the exact approach (exact) with a time limit of 2 hours, GREEDY with the randomized approach (greedy), LIFTFLOWwith the randomized approach (liftflow), and REOPT (reopt). Only those values are taken into account where a solution has been found with the corresponding approach.

$\|V\|$	over / under	T	k	L	exact	greedy	aggnet	reopt
15	4/4	90	2	2	5.29	5.80	5.27	9.44
15	4/4	90	2	5	11.70	13.01	8.67	29.64
15	4/7	90	2	2	4.61	6.85	5.13	7.04
15	4/7	90	2	5	11.58	12.25	7.81	32.85
15	7/4	90	2	2	4.91	6.78	4.70	9.22
15	7/4	90	2	5	9.70	10.04	6.99	27.27
15	7/7	90	2	2	5.52	8.78	5.32	9.34
15	7/7	90	2	5	10.01	10.42	7.68	32.63
15	4/4	90	5	2	5.74	5.94	6.19	7.59
15	4/4	90	5	5	16.89	16.97	7.6	21.30
15	4/7	90	5	2	6.18	6.43	5.03	8.36
15	4/7	90	5	5	16.94	17.00	7.33	23.05
15	7/4	90	5	2	6.26	6.43	3.97	9.20
15	7/4	90	5	5	15.38	15.52	6.08	21.34
15	7/7	90	5	2	6.10	6.16	5.95	9.89
15	7/7	90	5	5	16.03	16.10	9.09	23.20
30	4/4	300	2	2	44.07	14.45	8.76	31.15
50	4/4	300	2	2	40.77	24.10	15.74	42.05

Table 9.2: This table highlights the results of all approaches on the test instances where an optimal solution is known. In columns one to four, show the different variables of the test instances. The other columns show the total tour lengths by the corresponding algorithm. We consider the following algorithms: the exact approach (exact) with a time limit of 2 hours, GREEDY with the randomized approach (greedy), REOPT (reopt), and LIFTFLOWwith the randomized approach (liftflow). A hyphen '-' indicates that no solution has been found for the test instance.

drivers	L	T	over/under	exact	greedy	reopt	liftflow
2	2	90	4/4	123	123	-	123
2	2	90	4/4	151	152	-	151
5	2	90	4/4	122	122	123	122
5	2	90	4/4	124	124	130	124
5	2	90	7/7	155	155	164	155

is based on the considered test instances. The "other" runtimes correspond to the expectations, i.e., if the sum of overfull and underfull stations increases, then the runtimes for solving these instances increase as well.

As already mentioned, the exact approach could find an optimal solution for 8 test instances within the given time limit. Hereby, all the instances where in the set of the "small" instances. In 12 small instances, the exact approach could not find any solution within the time limit. For the medium sized test instances, the exact approach could find a solution only for four instances. Within the big sized test instances, only 2 instances lead to feasible solution for the exact approach.

In most cases, the approach LiftFlow combined with the randomized approach generally lead to the best results in acceptable time. That LiftFlow often shows better results than the exact approach can be explained by the fact that the exact approach nearly never found the optimal solution. In the small test instances, LiftFlow found the optimal solution of the first step in all cases. Furthermore, due to the tight time horizon, the tours constructed in the second step of LiftFlow almost always extend the time horizon. Only in six medium sized instances, the first step of LiftFlow did not find the optimal solution within the time limit. The maximal duality gap for these instances is 8.63% and the average is 4.9%. For the test instances with 50 stations, the approach LiftFlow found the optimal solution for the first step only in six cases. Hereby, we have a maximal duality gap of 10.18% and an average of 4.87%.

The results state that the Greedy approach combined with the randomized approach also lead to acceptable results in a reasonable time. That the results of LiftFlow are almost always better than the results of Greedy also shows that the start solution given to RandTEN plays an important role for the quality of the final solution. Furthermore, it follows that RandTEN can take a tremendous amount of time to find the optimal (or near optimal) solution. However, an advantage of Greedy over LiftFlow is that the runtime only increase slightly when the total number of overfull and underfull stations increase. Furthermore, Greedy was the algorithm which found the most feasible solutions.

Since the algorithm ReOpt computes only non-preemptive transportation schedules with backhaul, it is hard to compare this approach with the other three tested approaches. This most likely the reason, why this approach generally returns the transportation schedules with the largest total tour lengths. However, one can see that the ratio between the best known value and the solution computed by ReOpt is always much smaller than the theoretical upper bound (see Sections 7.4 and 9.1). Especially, ReOpt profits from adding drivers to the system. Hereby, not only the results get much closer to the other approaches but also the likeliness that a feasible solution is found. However, the algorithm ReOpt shows by far the best results w.r.t. the runtime.

Generally, adding more drivers seem to lead to better results and a higher probability for finding a feasible solution than increasing the capacity when the time horizon is (supposedly) tight. Hereby, the total tour lengths seem to increase only slightly. However, since the costs for moving a vehicle are not the only costs, but also paying more drivers and the need of purchasing more equipment, the overall costs for adding more drivers to the system may increase too much in practice.

Although, the randomized approach largely improves the given solution, it has the big disadvantage that infeasible test instances cannot be detected and the computation does not stop before the limits are reached. Since nowadays integer linear program solver can quickly detect infeasible models, we propose to check the instance for feasibility on the integer linear program of the exact approach before launching any other heuristic to calculate a solution.

While the exact approach benefits rather from a shorter than a large time-horizon, especially the algorithm ReOpt more likely finds a solution when the time-horizon is large.

Table 9.3: This table summarizes the average time in seconds needed to solve the instances. In columns one to four, show the different variables of the test instances. The other columns show the average time needed to solve a set of instances by the corresponding algorithm. We consider the following algorithms: the exact approach (exact) with a time limit of 2 hours, GREEDY with the randomized approach (greedy), REOPT (reopt), and the total amount of time used by LIFTFLOWwith the randomized approach (liftflow). Hereby, we had a closer look and also give the amount for solving the flows in the aggregated network (lf-agg), to compute the transportation schedule (lf-tour), and the time spent to optimize with the randomized approach (lf-rnd).

over/under	exact	greedy	reopt	liftflow	lf-agg	lf-tour	lf-rnd
4 / 4	6852.82	1742.50	9.82	1739.09	0.19	0.31	1738.60
4 / 7	6783.98	1768.22	32.32	1788.65	0.98	1.41	1786.26
7 / 4	6600.62	1787.01	26.04	1794.87	1.01	6.09	1787.77
7 / 7	6420.83	1752.64	64.91	1797.66	6.60	10.62	1780.44
12 / 12	7205.84	1892.06	340.52	2170.16	340.52	19.66	2161.07
23 / 23	7203.21	1836.99	565.63	2982.85	1062.14	142.32	1839.71

Finally, we observed that the "best" lower bounds are computed during the computation of the exact approach and by LIFTFLOW. However, the lower bounds from Section 7.7.2 can be computed quickly and, thus, can provide some initial information. On the other hand, the gap between the lower bound and the optimal solution is quite large in general.

Part IV

Conclusions

In this thesis, we studied a carsharing system where customers can rent cars and take them from any station and return them at any time later at any station. Especially, when many customers use the carsharing system, the stations become imbalanced over time, i.e., some stations have a deficit of cars while others have a deficit of parking places. Therefore, an operator monitors the loads of the stations and relocates cars if necessary. Hereby, the operator has to decide when and how the cars have to be relocated, i.e., the operator has to solve a Relocation Problem.

The Relocation Problem is an $\mathcal{NP}$-hard problem which generalizes other known $\mathcal{NP}$-hard problems like the traveling salesperson problem. We considered the Relocation Problem from a theoretical point of view and performed computational experiments on the presented algorithms. Hereby, we focused on the dynamic and static aspects of this problem.

We modeled and analyzed the Dynamic Relocation Problem within the framework of online optimization. Hereby, we used two analysis tools to evaluate the performance of different online algorithms: the competitive analysis and the max/max ratio. While the max/max ratio never lead to a meaningful result, we could give some competitive online algorithms. Due to the pessimism of competitive analysis, we had to restrict the adversaries as well as the carsharing system to find a competitive online algorithm. Furthermore, we could show that the proper selection of the objective function is important in the question of the existence of a competitive online algorithm. Hereby, also problems which seem to be equal at first glance turn out to be extremely different w.r.t. this question, e.g., the Online Min-Reject Relocation Problem and the Online Max-Accept Relocation Problem, where the objective is to minimize the number of rejected customer requests and to maximize the number of accepted customer requests, respectively.

For the Online Min-Reject Relocation Problem, we could prove the non-existence of competitive online algorithms unless we restricted the considered adversary, the metric space induced by the carsharing system and the number of cars and drivers within the system. For the Online Max-Accept Relocation Problem, there are fewer restrictions necessary to find a competitive online algorithm. Considering these online problems, naturally lead to consider the offline version of these two Dynamic Relocation Problems. The online algorithm RePlan requires that instances of these offline problems have to be solved repeatedly. Our studies resulted in a fast flow-based heuristic which is able to solve even bigger instances (w.r.t. the number of stations in the system).

For the optimization problems where all customer requests have to be served while aiming at minimizing the waiting times of the customers, we could show that even in very restricted scenarios there does not exist a competitive online algorithm. To the best of our knowledge, it is still an open question how these online problems can be theoretically evaluated. The negative results and the positive results of the decision problems motivated to consider a mixed decision/optimization problem. However, aiming at minimizing the total tour length while getting negative profits for each non-served customer request, leads to competitive online algorithms MarkRePlan which recomputes the transportation schedule by taking only unmarked cars into account. Hereby, cars are marked when they serve a customer request.

For some problems we could state competitive online algorithms under certain conditions while for other problems, we could prove that none exist. We made experiments for the Online Max-Accept Relocation Problem on several test instances. To get a better overview of the performance of the online algorithms in practice, the parameters of these instances are set so that there does not exist a competitive online algorithm. The results, w.r.t. the number of accepted customer requests, point to two online algorithms: RePlan and EST. In some cases RePlan outperforms EST and in some cases EST outperforms RePlan. However, it is still an open questions when the performance of RePlan is better and when EST shows better results. Furthermore, the theoretical background to explain the performances in more detail is still work

for the future.

However, the runtimes of both online algorithms differ enormously. While EST needs in average 5 seconds to solve a big test instance with 250 stations, RePlan (and even its flow-based heuristic version) needs more than 50 seconds in average (resp. 7 seconds) even to solve a small test instance. Therefore, EST clearly outperforms RePlan when the runtimes are into account and, thus, should be used in practice.

Although, EST provides a good online algorithm, there is still room for improvements in the future.

Since in the static version of the Relocation Problem there are no customer requests which can be rejected, considering decision problems is not meaningful in this situation. Thus, for the static aspect, we focus only on optimization problems.

In order to solve the Static Min-Cost Relocation Problem, we provided an integer linear program with which one can compute an optimal solution. Since it can be $\mathcal{NP}$-hard to find even a feasible solution, it is quite evident that this approach rarely finds the (proven) optimal solution within a reasonable time even for small carsharing systems (w.r.t. the number of stations), we presented some heuristic approaches. Although, some of these approaches alone do not lead to good results, a combination of these lead to astonishingly good results, e.g., the combination of using the solution from Greedy or LiftFlow as an initial solution for RandTEN. Under certain conditions, the heuristic LiftFlow computes even an optimal solution. However, when these conditions are not satisfied, the approach LiftFlow provides at least a lower bound.

A possible practical disadvantage of LiftFlow may be that it (generally) computes preemptive transportation schedules. Non-preemptive transportation schedules are computed by the combinatorial approach ReOpt, which has a proven worst-case approximation ratio. However, in our experiments, this approach performed better and the worst-case ratio has never been reached.

The experiments could show that the given time horizon is responsible whether an instance is feasible or not. Thus, it would be interesting to have a good lower bound for a "feasible" time horizon. Hereby, the lower bound should not be too tight since otherwise finding a feasible solution becomes very hard. On the other hand, if the lower bound is chosen too large, finding an optimal or good solution becomes hard with some approaches. Furthermore, a large time horizon implies that the relocation process may take more time than necessary, which might not be useful in practice.

Since the number of drivers is highly responsible for the costs of a running system, it is of interest to have a lower and an upper bound for the number of drivers needed in order to solve an instance. This becomes even more interesting, when there is some other work for the drivers within the enterprise whenever they are idle. To the best of our knowledge, both bounds as well as a good bound for the time horizon are still open for future work.

Finally, we studied the worst-case ratio between preemptive and non-preemptive transportation schedules. Hereby, we could provide a constant ratio which, however, only holds when there are enough parking places at each station. When the number of parking places is not sufficiently large at the stations, it is still an open questions whether there exists a constant worst-case ratio between preemptive and non-preemptive transportation schedules. The examples we considered in order to find a way to prove a worst-case ratio, point to the existence of a constant worst-case ratio.

Bibliography

[1] Luca Allulli, Giorgio Ausiello, Vincenzo Bonifaci, and Luigi Laura. On the power of lookahead in on-line server routing problems. Theoretical Computer Science, 408(2-3):116 – 128, 2008. Excursions in Algorithmics: A Collection of Papers in Honor of Franco P. Preparata.

[2] Luca Allulli, Giorgio Ausiello, and Luigi Laura. On the power of lookahead in on-line vehicle routing problems. In Lusheng Wang, editor, Computing and Combinatorics, volume 3595 of Lecture Notes in Computer Science, pages 728–736. Springer Berlin Heidelberg, 2005.

[3] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, pages 229–237, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[4] Spyros Angelopoulos and Pascal Schweitzer. Paging and list update under bijective analysis. J. ACM, 60(2):7:1–7:18, May 2013.

[5] M. Antosiewicz, G. Koloch, and B. Kamiński. Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed. Journal of Theoretical and Applied Computer Science, 7:46–55, 2013.

[6] Robert Arnold, Vance C. Smith, John Q. Doan, Rodney N. Barry, Jayme L. Blakesley, Patrick T. DeCorla-Souza, Mark F. Muriello, Gummada N. Murthy, Patty K. Rubstello, and Nick A. Thompson. Reducing congestion and funding transportation using road pricing in europe and singapore. Technical report, American Trade Initiatives, December 2010. Report no. FHWA-PL-10-030.

[7] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM, 45(5):753–782, September 1998.

[8] N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, volume 1770 of Lecture Notes in Computer Science, pages 639–650. Springer, 2000.

[9] J. Bachand-Marleau, B. Lee, and A. El-Geneidy. Towards a better understanding of the factors influencing the likelihood of using shared bicycle systems and frequency of use. Transportation Research Record, 2314:66–71, 2012.

[10] Jørgen Bang-Jensen, Gregory Gutin, and Anders Yeo. When the greedy algorithm fails. Discrete Optimization, 1(2):121 – 127, 2004.

[11] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In Rafail Ostrovsky, editor, FOCS, pages 267–276. IEEE, 2011.

[12] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Metrical task systems and the k-server problem on HSTs. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, Automata, Languages and Programming, volume 6198 of Lecture Notes in Computer Science, pages 287–298. Springer Berlin Heidelberg, 2010.

[13] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, STOC '97, pages 711–719, New York, NY, USA, 1997. ACM.

[14] Sumanta Basu. Tabu search implementation on traveling salesman problem and its variations: A literature survey. American Journal of Operations Research, 2:163–173, 2012.

[15] Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In Leah Epstein and Paolo Ferragina, editors, ESA, volume 7501 of Lecture Notes in Computer Science, pages 169–180. Springer, 2012.

[16] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, G. Schäfer, and T. Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on, pages 462–471, Oct 2003.

[17] R. Bellman. On a routing problem. Quarterly of Applied Mathematics, 16:87–90, 1958.

[18] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. Algorithmica, 11(1):73–91, 1994.

[19] Mike Benchimol, Pascal Benchimol, Benoît Chappert, Arnaud de la Taille, Fabien Laroche, Frédéric Meunier, and Ludovic Robinet. Balancing the stations of a self service "bike hire" system. RAIRO - Operations Research, 45:37–61, 0 2011.

[20] Terence Bendixson and MartinG. Richards. Witkar: Amsterdam's self-drive hire city car. Transportation, 5(1):63–72, 1976.

[21] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. European Journal of Operational Research, 202(1):8 – 15, 2010.

[22] Kristin Berg Bergvinsdottir, Jesper Larsen, and Rene Munk Jørgensen. Solving the Dial-a-Ride Problem using Genetic algorithms. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.

[23] N. Biggs, E. K. Lloyd, and R. J. Wilson. Graph Theory, 1736-1936. Clarendon Press, New York, NY, USA, 1986.

[24] Alain Billionnet, Sourour Elloumi, and Amélie Lambert. Linear reformulations of integer quadratic programs. In MCO'08, 2nd international conference on Modelling, Computation and Optimization in Information Systems and Management Sciences, LNCS, pages 43–51, Metz, France, September 2008.

[25] Markus Bläser, Konstatinos Panagiotou, and B.V.Raghavendra Rao. A probabilistic analysis of christofides' algorithm. In Fedor V. Fomin and Petteri Kaski, editors, Algorithm Theory - SWAT 2012, volume 7357 of Lecture Notes in Computer Science, pages 225–236. Springer Berlin Heidelberg, 2012.

[26] Michiel Blom, Sven O. Krumke, Willem E. De Paepe, and Leen Stougie. The online TSP against fair adversaries. INFORMS Journal on Computing, 13(2):138–148, 2001.

[27] Avrim Blum and John Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms, pages 905–914, 2002.

[28] V. Bonifaci, M. Lipmann, and L. Stougie. Online multi-server dial-a-ride problems. other TUE - SPOR-2006-04, 2006.

[29] Pierre Borgnat, Patrice Abry, Patrick Flandrin, Céline Robardet, Jean-Baptiste Rouquier, and Eric Fleury. Shared Bicycles in a City: A Signal processing and Data Analysis Perspective. Advances in Complex Systems, 14(3):1–24, June 2011.

[30] Pierre Borgnat, Céline Robardet, Patrice Abry, Patrick Flandrin, Jean-Baptiste Rouquier, and Nicolas Tremblay. Dynamics On and Of Complex Networks, Volume 2: Applications to Time-Varying Dynamical Systems, chapter A Dynamical Network View of Lyon's Vélo'v Shared Bicycle System, pages 267–284. Modeling and Simulation in Science, Engineering and Technology. Springer, June 2013.

[31] A. Borodin and R. El-Yaniv. On randomization in on-line computation. Information and Computation, 150(2):244–267, 1999.

[32] Allan Borodin and Ran El-Yaniv. Online computation and competitive analysis. Cambridge University Press, New York, NY, USA, 1998.

[33] Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for online algorithms. ACM Trans. Algorithms, 3(2), May 2007.

[34] Joan Boyar, Sandy Irani, and Kim S. Larsen. A comparison of performance measures for online algorithms. In Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, Algorithms and Data Structures, volume 5664 of Lecture Notes in Computer Science, pages 119–130. Springer Berlin Heidelberg, 2009.

[35] Sahar Bsaybes, Sven O. Krumke, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener. Reopt: an algorithm with a quality guaranty for solving the static relocation problem. 2015. `arXiv:1511.02751 [cs.DS]`.

[36] Sahar Bsaybes, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener. Two flow-based approaches for the static relocation problem in carsharing systems. 2015. `arXiv:1511.02650 [cs.DS]`.

[37] Rainer E. Burkard, Vladimir G. Deineko, and Gerhard J. Woeginger. The travelling salesman and the PQ-tree. In William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, editors, Integer Programming and Combinatorial Optimization, volume 1084 of Lecture Notes in Computer Science, pages 490–504. Springer Berlin Heidelberg, 1996.

[38] Roberto Wolfler Calvo and Nora Touati Moungla. A matheuristic for the dial-a-ride problem. In Julia Pahl, Torsten Reiners, and Stefan Voß, editors, INOC, volume 6701 of Lecture Notes in Computer Science, pages 450–463. Springer, 2011.

[39] Sean V. Casley, Adam S. Jardim, and Alex M. Quartulli. A study of public acceptance of autonomous cars. Interactive qualifying project, Worcester Polytechnic Institute, April 2013.

[40] Prasad Chalasani, Rajeev Motwani, and Anil Rao. Algorithms for robot grasp and delivery. 2nd International Workshop on Algorithmic Foundations of Robotics, 1996.

[41] Yaw Chang and Lin Chen. Solve the vehicle routing problem with time windows via a genetic algorithm. Descrete and continous dynamical systems supplement, pages 240–249, September 2007.

[42] Moses Charikar, Samir Khuller, and Balaji Raghavachari. Algorithms for capacitated vehicle routing. SIAM J. Comput., 31(3):665–682, 2001.

[43] Daniel Chemla, Frédéric Meunier, and Roberto Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. pages 120–146, 2013.

[44] Daniel Chemla, Frédéric Meunier, Thomas Pradeau, Roberto Wolfler Calvo, and Houssame Yahiaoui. Self-service bike sharing systems: simulation, repositioning, pricing. March 2013.

[45] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[46] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Mathematical Programming, 20(1):255–282, 1981.

[47] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In SIAM Journal on Discrete Mathematics, pages 291–300, 1990.

[48] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k servers on trees. SIAM Journal on Computing, 20:144–148, 1996.

[49] Leandro C. Coelho and Gilbert Laporte. The exact solution of several classes of inventory-routing problems. Computers & Operations Research, 40(2):558 – 565, 2013.

[50] C. Contardo, C. Morency, and L-M. Rousseau. Balancing a dynamic public bike-sharing system. Technical Report 9, CIRRELT, 2012. `https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2012-09.pdf`.

[51] Don Coppersmith, Peter Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs, and applications to on-line algorithms (extended. In Journal of the ACM, pages 369–378, 1990.

[52] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: Variants, modeling issues and algorithms. Draft, GERAD-HEC Montreal, 2002.

[53] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Draft, GERAD-HEC Montreal, 2002.

[54] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. Annals of Operations Research, 153(1):29–46, 2007.

[55] Aaron Coté, Adam Meyerson, and Laura Poplawski. Randomized k-server on hierarchical binary trees. In Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC '08, pages 227–234, New York, NY, USA, 2008. ACM.

[56] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America, 2:393–410, 1954.

[57] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. Management Science, 6(1):80–91, 1959.

[58] G.B. Dantzig. Linear Programming and Extensions. Princeton landmarks in mathematics and physics. Princeton University Press, 1963.

[59] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A^*. J. ACM, 32(3):505–536, July 1985.

[60] Vladimir Deineko and Alexander Tiskin. Fast minimum-weight double-tree shortcutting for metric TSP: Is the best one good enough? J. Exp. Algorithmics, 14:6:4.6–6:4.16, January 2010.

[61] Samuel Deleplanque and Alain Quilliot. Insertion techniques and constraint propagation for the DARP. Federated conference on computer science and information systems, pages 393–400, September 2012.

[62] Samuel Deleplanque and Alain Quilliot. In Proceedings of MISTA 2013, Multidisciplinary International Scheduling Conference: Theroy & Applications, volume 6, August 2013.

[63] Samuel Deleplanque and Alain Quilliot. Constraint propagation for the dial-a-ride problem with split loads. In Stefka Fidanova, editor, Recent Advances in Computational Optimization, volume 470 of Studies in Computational Intelligence, pages 31–50. Springer International Publishing, 2013.

[64] Xiaotie Deng and Sanjeev Mahajan. Server problems and resistive spaces. Information Processing Letters, 37(4):193 – 196, 1991.

[65] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. Operations Research, 40(2):342–354, 1992.

[66] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

[67] Reza Dorrigiv. Alternative Measures for the Analysis of Online Algorithms. PhD thesis, University of Waterloo, Ontario, Canada, 2010.

[68] Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. SIGACT News, 36(3):67–81, 2005.

[69] Earth Policy Institute. Data Center - Population, Health, and Society. http://www.earth-policy.org/data_center/C21, 2014. Accessed: 2014-07-11.

[70] Ecoplan. World Carshare Cities Inventories. http://ecoplan.org/carshare/general/cities.htm, October 2008. Accessed: 2014-10-15.

[71] M. EL-Zaher, B. Dafflon, F. Gechter, and J.-M. Contet. Vehicle platoon control with multi-configuration ability. Proc. Computer Science, 9(0):1503–1512, 2012.

[72] Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Comparing online algorithms for bin packing problems. Journal of Scheduling, 15(1):13–21, 2012.

[73] Côme Etienne and Oukhellou Latifa. Model-based count series clustering for bike sharing system usage mining: A case study with the vélib' system of paris. ACM Trans. Intell. Syst. Technol., 5(3):39:1–39:21, July 2014.

[74] Anke Fabri and Peter Recht. Online dial-a-ride problem with time windows: An exact algorithm using status vectors. In Karl-Heinz Waldmann and UlrikeM. Stocker, editors, Operations Research Proceedings 2006, volume 2006 of Operations Research Proceedings, pages 445–450. Springer Berlin Heidelberg, 2007.

[75] Joy Fang. Legal liability issues 'preventing mass adoption of self-driving cars'. http://www.todayonline.com/singapore/legal-liability-issues-preventing-mass-adoption-self-driving-cars, May 2015. Accessed: 2015-09-14.

[76] Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. Theor. Comput. Sci., 268(1):91–105, October 2001.

[77] Marshall L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. Operations Research, 42(4):626–642, 1994.

[78] Food and Agriculture Organization of the United Nations. FAOSTAT. http://faostat3.fao.org/, 2014. Accessed: 2014-06-25.

[79] L. R. Ford, Jr. Network flow theory. Technical Report Paper P-923, The RAND Corporation, Santa Monica, California, 1956.

[80] Santo Fortunato. Community detection in graphs. CoRR, abs/0906.0612, 2009.

[81] Jon Froehlich, Joachim Neumann, and Nuria Oliver. Measuring the pulse of the city through shared bicycle programs. In Proceedings of the International Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense08), November 2008.

[82] Xiaobing Gan, Yan Wang, Shuhai Li, and Ben Niu. Vehicle routing problem with time windows and simultaneous delivery and pick-up service based on mcpso. Mathematical Problems in Engineering, vol. 2012, 2012.

[83] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. Management Science, 40(10):1276–1290, October 1994.

[84] Ian P. Gent and Toby Walsh. The TSP phase transition. Artificial Intelligence, 88:105–109, 1996.

[85] Fatma Pinar Goksal, Ismail Karaoglan, and Fulya Altiparmak. A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. Computers and Industrial Engineering, 65(1):39–53, May 2013.

[86] Inge Li Gørtz, Viswanath Nagarajan, and R. Ravi. Minimum makespan multi-vehicle dial-a-ride. CoRR, abs/1102.5450, 2011.

[87] Joachim Gudmundsson and Christos Levcopoulos. A fast approximation algorithm for TSP with neighborhoods and red-blue separation. In Takano Asano, Hideki Imai, D.T. Lee, Shin-ichi Nakano, and Takeshi Tokuyama, editors, Computing and Combinatorics, volume 1627 of Lecture Notes in Computer Science, pages 473–482. Springer Berlin Heidelberg, 1999.

[88] Joachim Gudmundsson, Marc van Kreveld, and Giri Narasimhan. Region-restricted clustering for geographic data mining. Computational Geometry, 42(3):231 – 240, 2009.

[89] Akshay Gupte, Shabbir Ahmed, Myun Seok Cheon, and Santanu Dey. Solving mixed integer bilinear problems using MILP formulations. SIAM Journal on Optimization, 23(2):721–744, 2013.

[90] Gregory Gutin and Anders Yeo. Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. Discrete Applied Mathematics, 119(1–2):107 – 116, 2002. Special Issue devoted to Foundation of Heuristics in Combinatoria l Optimization.

[91] Gregory Gutin, Anders Yeo, and Alexey Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. Discrete Applied Mathematics, 117(1–3):81 – 86, 2002.

[92] Zulqarnain Haider, Alexander Nikolaev, Jee Eun Kang, and Changhyun Kwon. Inventory rebalancing through pricing in public bike sharing systems. 2014. Article under Review. Available at `http://www.chkwon.net/papers/haider2014.pdf`.

[93] Pascal Halffmann, Sven O. Krumke, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener. On the online max-accept relocation problem. Procedings of 6th International Workshop on Freight Transportation and Logistics (ODYSSEUS 2015), 2015. (to appear).

[94] Pascal Halffmann, Sven O. Krumke, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener. On the online min-wait relocation problem. Procedings of LAGOS2015, 2015. (to appear).

[95] Keld Helsgaun. An effective implementation of k-opt moves for the Lin-Kernighan TSP. In Roskilde University, 2007. Case, page 109, 2006.

[96] Benjamin Hiller and Tjark Vredeveld. Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In Dan Halperin and Kurt Mehlhorn, editors, Algorithms - ESA 2008, volume 5193 of Lecture Notes in Computer Science, pages 528–539. Springer Berlin Heidelberg, 2008.

[97] Brady Hunsaker and Martin Savelsbergh. Efficient feasibility testing for dial-a-ride problems. Operations Research Letters, 30(3):169 – 173, 2002.

[98] P. Jaillet and M. Wagner. Online routing problems: value of advanced information as improved competitive ratios. Transportation Science, 40(2), 2006.

[99] Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In Jimmy Lee, editor, Principles and Practice of Constraint Programming – CP 2011, volume 6876 of Lecture Notes in Computer Science, pages 400–413. Springer Berlin Heidelberg, 2011.

[100] David S Johnson, Gregory Gutin, Lyle A McGeoch, Anders Yeo, Weixiong Zhang, and Alexei Zverovitch. Experimental analysis of heuristics for the atsp. In The traveling salesman problem and its variations, pages 445–487. Springer US, 2007.

[101] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization, 1:215–310, 1997.

[102] Nicolas Jozefowiez, Fred Glover, and Manuel Laguna. Multi-objective meta-heuristics for the traveling salesman problem with profits. Journal of Mathematical Modelling and Algorithms, 7(2):177–195, 2008.

[103] Noah Kazis. Theft and vandalism just not a problem for American bike-sharing. Blog at `http://www.streetsblog.org/2010/11/29/theft-and-vandalism-just-not-a-problem-for-american-bike-sharing/`, November 2010. Accessed: 2014-07-25.

[104] Claire Kenyon. Best-fit bin-packing with random order. In Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '96, pages 359–364, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[105] Byron Kidd. Free Bicycle Sharing in Japan Runs Into Problems. `http://www.tokyobybike.com/2013/06/free-bicycle-sharing-in-japan-runs-into.html`, June 2013. Accessed: 2014-09-22.

[106] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, May 1983.

[107] Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy. Incremental heuristic search in ai. AI Mag., 25(2):99–112, June 2004.

[108] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. Journal of the ACM, 42:507–511, 1995.

[109] S. O. Krumke. Online Optimization: Competitive Analysis and Beyond. Habilitationsschrift, Technische Universität Berlin, 2002.

[110] S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, volume 2462 of Lecture Notes in Computer Science, pages 200–214. Springer, 2002.

[111] S. O. Krumke, A. Quilliot, A. K. Wagler, and J.-T. Wegener. Relocation in carsharing systems using flows in time-expanded networks. In Joachim Gudmundsson and Jyrki Katajainen, editors, Experimental Algorithms, volume 8504 of LNCS, pages 87–98. Springer, 2014.

[112] Sven O. Krumke, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener. Models and algorithms for carsharing systems and related problems. Electronic Notes in Discrete Mathematics, 44(0):201 – 206, 2013.

[113] Karim Labadi, Taha Benarbia, Samir Hamaci, and A-Moumen Darcherif. Petri nets models for analysis and control of public bicycle-sharing systems. Petri Nets - Manufacturing and Computer Science, 2012.

[114] Andrea Lancichinetti, Filippo Radicchi, Jose J. Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. CoRR, abs/1012.2363, 2010.

[115] Hoong Chuin Lau, Melvyn Sim, and Kwong Meng Teo. Vehicle routing problem with time windows and a limited number of vehicles. European Journal of Operational Research, 148(3):559 – 569, 2003.

[116] Leo Liberti. Compact linearization for binary quadratic problems. 4OR, 5(3):231–245, 2007.

[117] Jenn-Rong Lin, Ta-Hui Yang, and Yu-Chung Chang. A hub location inventory model for bicycle sharing system design: Formulation and solution. Computers and Industrial Engineering, 65(1):77–86, May 2013.

[118] László Lovász. Random walks on graphs: A survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, Combinatorics, Paul Erdős is Eighty, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.

[119] D.R. Mallampati, P.P. Mutalik, and R.L. Wainwright. A parallel multi-phase implementation of simulated annealing for the traveling salesman problem. In Distributed Memory Computing Conference, 1991. Proceedings., The Sixth, pages 488–491, Apr 1991.

[120] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88, pages 322–333, New York, NY, USA, 1988. ACM.

[121] Alfonsas Misevičius, Antanas Lenkevicius, and Dalius Rubliauskas. Iterated tabu search: an improvement to standard tabu search. Information Technology and Control, 35(3):187–197, 2006.

[122] Alfonsas Misevičius, Jonas Smolinskas, and Arūnas Tomkevičius. Iterated tabu search for the traveling salesman problem. Information Technology and Control, 34(4):327–337, 2005.

[123] Edward F. Moore. The shortest path through a maze. In Proc. Internat. Sympos. Switching Theory 1957, Part II, pages 285–292. Harvard Univ. Press, Cambridge, Mass., 1959.

[124] Enda Murphy and Joe Usher. An analysis of the role of bicycle-sharing in a European city : the case of Dublin. Irish Transport Research Network, 2011.

[125] Rahul Nair, Elise Miller-Hooks, Robert C. Hampshire, and Ana Bušić. Large-scale vehicle sharing systems: Analysis of vélib'. International Journal of Sustainable Transportation, 7(1):85–106, 2013.

[126] Eoin O'Mahony and David Shmoys. Data analysis and optimization for (citi)bike sharing. Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015.

[127] Feargus O'Sullivan. In Paris, thefts and vandalism could force bike-share to shrink. http://www.citylab.com/commute/2013/09/paris-thefts-and-vandalism-could-force-bike-share-shrink/7014/, September 2013. Accessed: 2014-07-25.

[128] Christos H. Papadimitriou. The euclidean travelling salesman problem is NP-complete. Theoretical Computer Science, 4(3):237 – 244, 1977.

[129] Stephen D. Parkes, Greg Marsden, Susan A. Shaheen, and Adam P. Cohen. Understanding the diffusion of public bikesharing systems: evidence from Europe and North America. Journal of Transport Geography, 31:94 – 103, 2013.

[130] R. C. Prim. Shortest connection networks and some generalizations. Bell System Technology Journal, 36:1389–1401, 1957.

[131] Marian Rainer-Harbach, Petrina Papazek, Bin Hu, and Günther R. Raidl. Balancing bicycle sharing systems: A variable neighborhood search approach. In Martin Middendorf and Christian Blum, editors, EvoCOP, volume 7832 of Lecture Notes in Computer Science, pages 121–132. Springer, 2013.

[132] Andreas Rau and Daniel Bongardt. Facts and figures on transport. Technical report, GTZ Transport Policy Advisory Service, 2010.

[133] Abu Bakar Sayuti Saman and Issa Abdramane. Solving a reconfigurable maze using hybrid wall follower algorithm. International Journal of Computer Applications, 82(3):22–26, November 2013.

[134] Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. Theoretical Computer Science, 341(1–3):216 – 246, 2005.

[135] Brandon Schoettle and Michael Sivak. A survey of public opinion about autonomous and self-driving vehicles in the U.S., the U.K., and Australia. Technical report, University of Michigan, July 2014.

[136] A. Schrijver. Theory of Linear and Integer Programming. Wiley Series in Discrete Mathematics & Optimization. John Wiley & Sons, 1998.

[137] Jasper Schuijbroek, Robert Hampshire, and Willem-Jan van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. 2013. working paper.

[138] Susan Shaheen, Daniel Sperling, and Conrad Wagner. Carsharing and partnership management: An international perspective. Transportation Research Record, (1666):118–124, 1999.

[139] Susan A. Shaheen and Adam P. Cohen. Worldwide carsharing growth: An international comparison. March 2008. The text is part of a series Institute of Transportation Studies (UCD).

[140] Susan A. Shaheen, Stacey Guzman, and Hua Zhang. Bikesharing in Europe, the Americas, and Asia: Past, present, and future. Transportation Research Record: Journal of the Transportation Research Board, (2143):159–167, 2010.

[141] Susan A. Shaheen, Daniel Sperling, and Conrad Wagner. A short history of carsharing in the 90's. World Transport Policy and Practice, 5(3), September 1999.

[142] Keshav Sharma and Chirag Munshi. A comprehensive and comparative study of maze-solving techniques by implementing graph theory. IOSR Journal of Computer Engineering (IOSR-JCE), 17(1 (Ver. IV)):24–29, Jan.–Feb. 2015.

[143] Radwa El Shawi, Joachim Gudmundsson, and Christos Levcopoulos. Quickest path queries on transportation network. CoRR, abs/1012.0634, 2010.

[144] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. Commun. ACM, 28(2):202–208, February 1985.

[145] Daniel A. Spielman. Lecture notes in spectral graph theory, September 2009. Available at `http://www.cs.yale.edu/homes/spielman/561/lect08-09.pdf`.

[146] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. CoRR, cs.DS/0111050, 2001.

[147] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis. In Frank Dehne, Jörg-Rüdiger Sack, and Michiel Smid, editors, Algorithms and Data Structures, volume 2748 of Lecture Notes in Computer Science, pages 256–270. Springer Berlin Heidelberg, 2003.

[148] Adam Srebro. Fault-tolerant algorithm for a mobile robot solving a maze. Challenges of Modern Technology, 4(1):21–29, 2013.

[149] K.C. Tan, L.H. Lee, and K. Ou. Artificial intelligence heuristics in solving vehicle routing problems with time window constraints. Engineering Applications of Artificial Intelligence, 14(6):825 – 837, 2001.

[150] K.C. Tan, L.H. Lee, Q.L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. Artificial Intelligence in Engineering, 15(3):281 – 295, 2001.

[151] O. Tange. Gnu parallel - the command-line power tool. ;login: The USENIX Magazine, 36(1):42–47, Feb 2011.

[152] The World Bank Group. World bank open data. `http://data.worldbank.org`, 2014.

[153] Paolo Toth and Daniele Vigo. Heuristic algorithms for the handicapped persons transportation problem. Transportation Science, 31(1):60–71, 1997.

[154] Tali Eilam Tzoreff, Daniel Granot, Frieda Granot, and Greys Sošić. The vehicle routing problem with pickups and deliveries on some special graphs. Discrete Applied Mathematics, 116(3):193 – 229, 2002.

[155] United Nations, Department of Economic and Social Affairs, Population Division. The world at six billion. October 1999.

[156] United Nations, Department of Economic and Social Affairs, Population Division. World urbanization prospects, the 2011 revision. March 2012.

[157] United Nations, Department of Economic and Social Affairs, Population Division. World population prospects: The 2012 revision, key findings and advance tables. 2013. Working Paper No. ESA/P/WP.227.

[158] United Nations, Economic and Social Council. World population monitoring, focusing on population distribution, urbanization, internal migration and development, January 2008.

[159] Jian Yang, Patrick Jaillet, and Hani Mahmassani. Real-time multivehicle truckload pickup and delivery problems. Transportation Science, 38:135–148, 2004.

[160] Fanglei Yi and Lei Tian. On the online dial-a-ride problem with time-windows. In Algorithmic Applications in Management: First International Conference, AAIM 2005, volume 3521 of LNCS, pages 85–94. Springer-Verlag Berlin Heidelberg, 2005.

[161] Bin Yu, Hanbing Zhu, Wanjun Cai, Ning Ma, Qiji Kuang, and Baozhen Yao. Two-phase optimization approach to transit hub location – the case of dalian. Journal of Transport Geography, 33(0):62 – 71, 2013.

[162] Jie Yu, Yue Liu, Gang-Len Chang, Wanjing Ma, and Xiaoguang Yang. Cluster-based hierarchical model for urban transit hub location planning. Transportation Research Record: Journal of the Transportation Research Board, 2112:8–16, December 2009.

[163] Alkın Yurtkuran and Erdal Emel. A new hybrid electromagnetism-like algorithm for capacitated vehicle routing problems. Expert Syst. Appl., 37(4):3427–3433, April 2010.

[164] K. Zavitsas, I. Kaparias, and M.G.H. Bell. Transport problems in cities. CONDUITS, 2010.

Appendix **A**

Basic Definitions

A.1 Graph Theory

It is natural to model urban areas as graphs. Therefore, many routing and transportation problems are defined on graphs or metric spaces induced by graphs. In this section, we give a brief introduction in some basic notions of graph theory which are used throughout this thesis.

A graph $G = (V, E)$ is a tuple containing a set V of nodes (also called vertices) and a set $E \subseteq V \times V$ of edges, connecting the nodes. Edges are undirected, i.e., if $(v, v') \in E$ then it follows that $(v', v) \in E$ as well[1]. If E contains all possible edges between two distinct nodes then G is called complete.

In a directed graph $G = (V, A)$, the set $A \subseteq V \times V$ of arcs does not necessarily include both edges[2], i.e., from $(v, v') \in A$ does necessarily follow that $(v', v) \in A$. Let $a = (v, v') \in A$ be an arc. Then a is said to be an outgoing arc of v and an incoming arc of v'. Arcs (or edges) of the form (v, v) are called loops.

A (directed) graph $H = (V', E')$ is a subgraph of a (directed) graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

A weighted graph $G = (V, E, w)$ contains besides the set of nodes and edges, an edge weight function $w : E \to \mathbb{N}$. If in a graph $G = (V, E, \text{cap})$ the nodes are weighted, i.e., if $\text{cap} : V \to \mathbb{N}$, then the function is called node weight function. When we simply write weight function, we mean an edge weight function. For a directed graph, these notations are defined analogously.

When a graph is illustrated, then the nodes usually correspond to cycles (or dots) and the edges (resp. arcs) to lines (resp. arrows) between the nodes (see Figure A.1 for an illustration).

A sequence $P = ((v_1, v_2), (v_2, v_3), \ldots, (v_{\ell-1}, v_\ell))$ is a (directed) path if all edges (resp. arcs) are distinct from each other. In the case that $v_1 = v_\ell$ holds, then P is a (directed) cycle[3]. A node v or an edge/arc e that belongs to P is denoted by $v \in P$ or $e \in P$. Graphs without cycles are called cycle-free or acyclic.

A graph G is connected if for any two distinct nodes, there is a path one to the other. If there is a directed path between any two distinct nodes, then G is strongly connected. Graphs which are not (strongly) connected, contain maximal connected subgraphs, the so-called (strongly) connected components.

If a directed path (or cycle) traverses every arc in A exactly once, it is said to be Eulerian. The next well-known result gives conditions whether there exists an Eulerian cycle in a graph or

[1] An edge can also be defined as an unordered pair of nodes.
[2] An arc can also be defined as an ordered pair of nodes.
[3] A directed cycle is also called circuit.

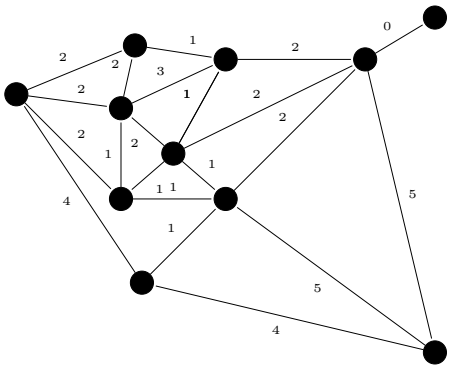


Figure A.1: This image shows a weighted graph with 11 nodes and 21 edges. The numbers next to the edges correspond to the edge weights. The graph corresponds to the bikesharing system from Figure 2.1.

not.

Theorem A.1 (Euler's Theorem)**.** *Let $G = (V, A)$ be a strongly connected directed graph. There exists an Eulerian cycle in G if and only if for every node $v \in V$ the number of incoming arcs and the number of outgoing arcs of v are equal.*

Finally, we define two further properties on graph: a "minimum spanning tree" and a "perfect matching".

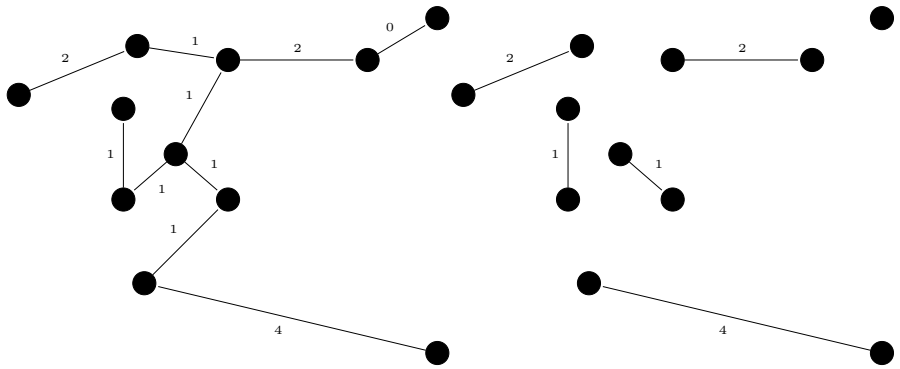
A connected graph without cycles is called tree. Every connected graph contains a subgraph T which is a tree and so that between all distinct nodes of G there is a path in T. Such a graph is called a spanning tree of G.

Let G be a weighted graph G and T a spanning tree of G. Then, the weight of a spanning tree is the sum of the weights of all edges in T, and T is called minimum spanning tree if there does not exists another spanning tree T' of G with a weight strictly less than the weight of T.

A minimum spanning tree can be computed in polynomial time with Prim's algorithm [130]. The algorithm starts the construction of T at any arbitrary node v. Afterwards, the least-weighted non-added edge is added to T as long as T remains a tree. The spanning tree shown in Figure A.2a has been constructed with Prim's algorithm and, thus, is a minimum spanning tree. Note that there can be more than one minimum spanning tree in a graph.

Spanning trees are often used as a basis to solve some transportation problems (see Section 1.3). Especially, the problem of finding shortest tours can be solved with the help of (minimum) spanning trees. In certain transportation problems, e.g., where a pickup location must be visited before a corresponding delivery location, a "matching" can be used to find these pickup and delivery locations.

A matching in a graph is a set of edges without common nodes (see Figure A.2b for an illustration). When all nodes are connected by edges of a matching, then it is called a perfect matching. It is easy to see, that if the number of nodes in a graph G is odd, there cannot exist a perfect matching for G.



(a) In this figure, a spanning tree is illustrated.

(b) This figure shows a matching. Since the number of nodes is odd, there cannot be a perfect matching for the graph.

Figure A.2: This image illustrates a spanning tree and a matching of the weighted graph from Figure A.1. The numbers next to the edges correspond to the edge weights.

A.2 Metric Space

A metric space (X, d) is a pair containing a set X and a distance function $d : X \times X \to \mathbb{R}$ with

(i) $d(x, y) = d(y, x)$ for all $x, y \in X$ (symmetry),

(ii) $d(x, y) + d(y, z) \leq d(x, z)$ for all $x, y, z \in X$ (triangular inequality), and

(iii) $d(x, y) = 0$ if and only if $x = y$ for all $x, y \in X$ (identity of indiscernibles).

Note that $d(x, y) \geq 0$ for all $x, y \in X$.

Probably, the most famous metric space is the Euclidean metric space $(\mathbb{R}^n, d)$ with $d(x, y) = \sqrt{\sum_{i=1}^{n} x_i^2 + y_i^2}$. In this thesis, we consider the special case of the Euclidean metric space where the set is the real line, and we simply denote this metric space by $\mathbb{R}$.

Given a connected weighted graph $G = (V, E, w)$, one can define a metric space (V, d) by setting $d(u, v)$ as the shortest path distance between the nodes u and v.

A uniform metric space is a metric space where the distance between two distinct points is 1. When the set has a finite number n of nodes, the uniform metric space (denoted by $\mathbb{U}(n)$) can be considered to be induced by a complete weighted graph, where the weights of the arcs are 1.

Appendix B

Optimal Offline Solution for the Online Min-Wait Relocation Problem

In this section, we give an integer linear program to compute an optimal offline solution for the Online Min-Wait Relocation Problem, where every request must be served so that the total waiting time for the customers is minimal. Hereby, we assume that customers spontaneously arrive at a station to take a car. If their request cannot be served immediately, they wait until there is a free car at the station and a free parking space at their destination station.

The Offline Min-Wait Relocation Problem $(G, \boldsymbol{z}^0, z^d, \gamma, k, L, R)$ consists of the following data:

- a weighted graph $G = (V, E, \mathrm{cap}, w)$, where the nodes correspond to stations, edges to their links, node weights to the station's capacities, and the edge weights $w : E \to \mathbb{R}_+$ determine the driving times between two points $v, v' \in V$ as length of a shortest path from v to v';
- the total number k of drivers, the maximum number $L \in \mathbb{N}$ of cars which can be simultaneously moved in one convoy, the initial quantities $0 \leq z_v^0 \leq \mathrm{cap}(v)$ and z_v^d of cars and drivers located at v at the start time $t = 0$, with $\gamma = \left|\boldsymbol{z}^0\right|$ and $k = \left|\boldsymbol{z}^d\right|$; and
- a sequence $R = (r_1, \ldots, r_\lambda)$ of floating customer requests.

The output of the Offline Min-Wait Relocation Problem is a transportation schedule for a metric task system, whose tasks are directly induced by the customer requests. Hereby, all customer requests are served. The objective function is to minimize the total waiting time for the customers, i.e.,

$$\sum_{j=1}^{\lambda} \tau_j - t_j,$$

is minimal and where τ_j is the time when a floating customer request $r_j = (t_j, v_j, v'_j, \delta_j)$ is served.

In order to solve the Offline Min-Wait Relocation Problem, we present an exact solution based on flows in a time-expanded network with two coupled flows: a car and a driver flow.

Time-Expanded Networks. We build a time-expanded version $G_T = (V_T, A_T)$ of the original network G.

The node set V_T is constructed as follows. Let T be a given time horizon. For each station $v \in V$ and each time point $t \in [0, T]$, there is a node $(v, t) \in V_T$ which represents station v at time t as a capacitated station where convoys can simply pass or cars can be picked up, delivered and exchanged by drivers.

The arc set $A_T = A_H \cup A_L \cup A_R$ of G_T is composed of several subsets:

- A_H contains, for each station $v \in V$ of the original network and each $t \in \{0, 1, \ldots, T-1\}$, the holdover arc connecting (v, t) to $(v, t+1)$.
- A_L contains, for each edge (v, v') of G and each point in time $t \in T$ such that $t+d(v, v') \leq T$, the relocation arc from (v, t) to $(v', t + d(v, v'))$.
- A_R contains, for each floating customer request $r_i = (t_i^{rel}, v_i, v_i', \delta_i) \in R$, request arcs from (v_i, t) to $(v_i', t + \delta_i)$ for every $t_i^{rel} \leq t \leq T$ with $t + \delta_i \leq T$. Hereby, we denote a request arc corresponding to a floating customer request r_i by a_{ij} where the request arc connects the nodes $(v_i, t_i^{rel} + j)$ to $(v_i', t_i^{rel} + j + \delta_i)$. The subset of A_R containing all request arcs which correspond to r_i is denoted by A_{R^i}.

Note, that the time-expanded network G_T is acyclic by construction.

A Min-Wait Flow Problem. On the time-expanded network G_T, we define two different flows, the car flow f and the driver flow F, to encode the relocation of cars in convoys.

Note that a flow on a relocation arc corresponds to a (sub)move in a tour, i.e., some cars are moved by drivers in a convoy from a station v to another station v'. Thus, a relocation arc from (v, t) to $(v', t + d(v, v'))$ has infinite capacity for the drivers, but to ensure that cars can be moved only in convoys and at most L cars per driver, we require that $f(a) \leq L \cdot F(a)$ holds for all $a \in A_R$ (see (B.1h)). Thus, the capacities for f on the relocation arcs are not given by constants but by a function. Note that due to these flow coupling constraints, the constraint matrix of the network is not totally unimodular (as in the case of uncoupled flows) and therefore integrality constraints for both flows are required (B.1k)). Furthermore, these constraints reflect that solving the Offline Min-Wait Relocation Problem is $\mathcal{NP}$-hard.

Flows on holdover arcs correspond to cars and drivers remaining at the station in the time interval $[t, t+1]$. Thus, the capacity of all holdover arcs for the car flow f is set to $\text{cap}(v)$ (see (B.1g)), whereas there is no capacity constraint for F on such arcs. Moreover, a car flow on a customer request arc corresponds to an accepted request, whereas driver flow is forbidden on such arcs (see (B.1j)). To assure that a customer takes only one car, the car flow on customer request arcs is bounded by 1 (see (B.1i)), and in order to ensure that every floating customer request $r_i = (t_i^{rel}, v_i, v_i', \delta_i)$ is served exactly once, we couple all request arcs $a_{ij} \in A_{R^i}$ (see (B.1f)) by

$$\sum_{a_{ij} \in A_{R^i}} f(a_{it}) = 1.$$

The aim is to receive a transportation schedule so that the total waiting time is minimal. We reflect the waiting time of a customer by setting costs for serving a corresponding floating customer request $r_i = (t_i^{rel}, v_i, v_i', \delta_i)$. For that we set on each customer request arc $a_{ij} = ((v, t_i^{rel} + j), (v', t_i^{rel} + j + \delta_i))$ the costs

$$c(a_{ij}) := j$$

where the arcs a_{ij} correspond to r_i. One can easily see, that these costs correspond to the waiting time of the customer to whom r_i corresponds. All other arcs have zero costs.

To correctly initialize the system, we use the nodes $(v, 0) \in V_T$ as sources for both flows and set their balances accordingly to the initial numbers of cars and drivers at station v and time 0 in z_v^0 and z_v^d (see (B.1b) and (B.1c)). For all internal nodes $(v, t) \in V_T$ with $t > 0$, we use normal flow conservation constraints (which is possible due to the fact that the entire flow of cars is

modeled by taking parked cars, convoy moves and customer actions into account), see (B.1d) and (B.1e). The flow conservation constraints also ensure that every car and driver finally arrives at a node $(v, T) \in V_T$.

The objective function (B.1a) minimizes the total waiting time, i.e., the total costs for serving all floating customer requests.

The corresponding integer linear program is as follows:

$$\min \sum_{a \in A_R} c(a) f(a) \tag{B.1a}$$

$$\sum_{a \in \delta^-(v,0)} f(a) = z_v^0 \qquad \forall (v,0) \in V_T \tag{B.1b}$$

$$\sum_{a \in \delta^-(v,0)} F(a) = z_v^d \qquad \forall (v,0) \in V_T \tag{B.1c}$$

$$\sum_{a \in \delta^-(v,t)} f(a) = \sum_{a \in \delta^+(v,t)} f(a) \qquad \forall (v,t) \in V_T, t > 0 \tag{B.1d}$$

$$\sum_{a \in \delta^-(v,t)} F(a) = \sum_{a \in \delta^+(v,t)} F(a) \qquad \forall (v,t) \in V_T, t > 0 \tag{B.1e}$$

$$\sum_{a_{ij} \in A_{Ri}} f(a_{it}) = 1 \qquad \forall r_i \in R \tag{B.1f}$$

$$0 \leq f(a) \leq \text{cap}(v) \qquad \forall a = [(v,t),(v,t+1)] \in A_H \tag{B.1g}$$

$$f(a) \leq L \cdot F(a) \qquad \forall a \in A_L \tag{B.1h}$$

$$f(a) \leq 1 \qquad \forall a \in A_R \tag{B.1i}$$

$$F(a) = 0 \qquad \forall a \in A_R \tag{B.1j}$$

$$f, F \text{ integer}, \tag{B.1k}$$

where $\delta^-(v,t)$ denotes the set of outgoing arcs of (v,t), and $\delta^+(v,t)$ denotes the set of incoming arcs of (v,t).

In order to avoid unnecessary movements of the drivers one can set some costs for driver flows F on relocation arcs, e.g,

$$C(a) := 1 \text{ for all } a \in A_L.$$

Then one can adjust the objective function by adding these costs to the objective function resulting in

$$\min \sum_{a \in A_R} c(a) f(a) + \sum_{a \in A_L} C(a) F(a).$$

Finally note that the Offline Min-Wait Relocation Problem always has a feasible solution as long as the time horizon is chosen large enough.