UTRECHT UNIVERSITY

CQM

MASTER THESIS

Dynamic Reoptimization of Transport for Elderly and Disabled

Author:
Nick KUIJPERS

University supervisor:
Dr. Johan van ROOIJ

Second reader:
Dr. Mihaela MITICI

CQM supervisors:
Dr. Bart POST
Dr. Frans de RUITER

Computing Science

August 7, 2023

Utrecht University

UTRECHT UNIVERSITY

Abstract

Faculty of Science
Computing Science

Master of Science

Dynamic Reoptimization of Transport for Elderly and Disabled

by Nick KUIJPERS

In this master thesis, we address the challenge of dynamically replanning transportation services for individuals with reduced mobility in the context of Valys: a paratransit agency in the Netherlands. High financial costs and the need for reducing greenhouse gas emissions have put pressure on paratransit agencies to improve their operational efficiency of transport services. Dynamic replanning can prevent changes from rendering the schedule created a day in advance inefficient, thereby averting delays, missed deliveries, and additional expenses. Furthermore, it can enhance flexibility for paratransit users, allowing same-day bookings instead of a day or more in advance.

We propose a Greedy Randomized Adaptive Search Procedure (GRASP) with evolutionary Path Relinking as a solution to the challenges of dynamic replanning. To the best of our knowledge, we are the first to apply GRASP in this field. Therefore, we provide numerical evidence for our GRASP in comparison with a widely used simulated annealing (SA) method. Comparative analysis with an SA approach demonstrates that our GRASP outperforms SA significantly in the setting of limited computation time, typically encountered in real-time planning scenarios. Our GRASP approach relies on high-quality start solutions for better overall solutions, and we provide qualitative insight into the influence of the start solution on the performance. At last, we demonstrate that our GRASP approach is better at frequent real-time replanning than an SA approach. Overall, our findings highlight the effectiveness of our GRASP in addressing the challenges of dynamic replanning.

Furthermore, we assess the opportunity gap of dynamic rescheduling with respect to not rescheduling, quantifying the potential improvement based on the cumulative number of changes. Our results show predictable gains ranging from 3.0% to 16.8% depending on the number of changes. These gains correspond to a decrease in service time from 150 to 600 hours per day. Our GRASP algorithm can effectively close the opportunity gap, improve service delivery, and enhance customer satisfaction.

Acknowledgements

First of all, I would like to express my deepest appreciation to my supervisor Dr. Johan van Rooij for his dedicated guidance and support during the completion of this thesis. Your sharp insights and thought-provoking perspectives consistently challenged my assumptions and broadened my intellectual horizons.

I am extremely grateful for the exceptional supervision provided by Dr. Bart Post at CQM. He consistently offered his support whenever I needed it, engaging in fruitful discussions, providing valuable feedback on my thesis through the numerous pages covered with blue ink, and continuously sparring ideas to enhance the results. Besides, I would like to thank "Groep Hulsen" for the warm welcome into the group and the great time I had at CQM.

I would also like to thank Dr. Mihaela Mitici for taking the time to assess my work. I enjoyed your monthly AI & Mobility Lab sessions and learned interesting things about the research of many PhD candidates.

Moreover, I would like to thank all of my friends for making everything much more enjoyable. Last but not least, I would like to express my gratitude to my family, for their unconditional support, love and understanding.

Contents

List of Symbols

N	Set of trips
A	Set of transporters
M	Set of routes
F	Set of vehicle types
δ^{nt}	Set of new trips
δ^{c}	Set of cancelled trips
$\mathcal{L}$	Set of locations
t_{l_1,l_2}	Shortest travel time from location l_1 to location l_2
L_i^p	Pickup location for trip i
L_i^d	Dropoff location for trip i
g_i	1 if trip i is a normal trip, 0 if trip i is an arrival guarantee trip
P_i^L	Number of passengers for trip i
P_i^W	Number of wheelchairs for trip i
st_i	Service time for trip i
σ_i	Desired pickup/dropoff time for trip i
λ_i	Maximum travel duration for trip i in minutes
B_i^p	Planned pickup time for trip i
B_i^d	Planned dropoff time for trip i
A_i	1 if trip i is not allowed to share a ride, 0 otherwise
F_i	1 if trip i has a passenger that needs to sit in the front seat, 0 otherwise
L_k^A	Set of locations, corresponding to the area of transporter of route k
R_k	Sequence of locations of route k
d_k	Duration of route k in minutes
x_{ik}	1 if trip i is driven by route k, 0 otherwise
q_f^P	Capacity of passengers of vehicle type f
q_f^W	Capacity of wheelchairs of vehicle type f
c_f	Cost per minute of vehicle type f
y_k	Planned vehicle type of route k, $y_k \in F$
tw_p	Minutes before and after σ_i in which trip i can be picked up
tw_a	Minutes before σ_i in which arrival guarantee trip i can be dropped off
λ	Fraction of the shortest driving distance is used for λ_i
τ_c	Minimum number of minutes a cancellation is known in advance
τ_{nt}	Minimum number of minutes a new trip is known in advance
τ_d	Maximum number of minutes a driver may drive without break
τ_b	Length of a break in minutes
τ_f	Minimum number of minutes between two planned breaks
τ_m	Maximum duration of a route in minutes
τ	Time needed for computation and communication of the reoptimization algorithms

Chapter 1

Introduction

In the last few decades, the organization of transport services for people with reduced mobility has become a priority in several countries, as the public transit system can pose significant obstacles to these people. For example, in Canada and the Netherlands, the law requires adequate transport services for individuals with reduced mobility.

High financial costs and the need for reducing greenhouse gas emissions have put pressure on paratransit agencies to improve their operational efficiency of transport services. Effectively, these agencies face the challenge of scheduling a fleet of vehicles to fulfil a set of transportation requests at minimum cost. This challenge, also known as a Dial-A-Ride Problem (DARP), is a generalization of a pickup and delivery problem. However, in addition to the classic pickup and delivery problem, a DARP has service-oriented criteria. Usually, these exist of narrow time windows on the pickup or delivery time, multiple types of service vehicles, and a maximum trip duration for the passengers.

To date, most research is focussed on finding such schedules a day in advance, thus scheduling with all trips known to the scheduler [1]. Creating the schedule in advance has some benefits for the transporters of paratransit agencies: the involved transporters are better informed of the number of vehicles needed, and there is more computation time to create efficient routes.

However, in practice vehicle routing problems are often not static at all due to uncertain driving times, service times, or information being gradually revealed throughout the day, such as new trips or cancellations. In a dynamic routing problem, some input data is revealed or updated during the time in which the operations take place. These problems are called Dynamic Vehicle Routing Problems (DVRP). The occurrence of these changes may render the schedule created a day in advance inefficient, causing delays, missed deliveries, and additional costs. A near-optimal schedule that is created a day in advance may turn out to be inefficient due to cancellations. Additionally, it limits the flexibility of the users, who are restricted to booking a day or more in advance. Allowing same-day bookings increases flexibility and service quality.

Dynamic replanning allows for real-time adjustments to the schedule based on the current situation. This means that changes must be made quickly and efficiently to ensure that the vehicles are always operating near-optimally. In this thesis, we investigate a practical case that arises in the Netherlands involving Valys, a paratransit agency that manages all transportation services for individuals with reduced mobility in the Netherlands. This case is generalizable to the Dynamic variant of the Dial-A-Ride problem (DDARP), as will be shown in Section 2. Our proposed solution can handle the complexity and scale of realistic instances that came from Valys.

We will focus on two main topics: 1) a potential method to address this dynamic problem, Greedy Randomized Adaptive Search Procedure (GRASP), and how that method compares to a simulated annealing algorithm and 2) the opportunity gap that arises with dynamic rescheduling. We will perform five different experiments to address these two topics. To the best of our knowledge, we are the first to use GRASP in the domain of (D)DARP, therefore comparing it to the well-known simulated annealing method should provide insight into how this method performs in this dynamic problem. Addressing the opportunity gap reveals the extent to which improvements can be made upon the current practice of not optimizing changes.

The remainder of this chapter is organized as follows. Section 1.1 gives an extended review of related problems in the literature. Section 1.2 explains our research questions. Finally, in Section 1.3 we give an outline of the rest of this thesis.

1.1 Literature Review

Besides DARP, numerous variations of the Vehicle Routing Problem (VRP) have been studied in the literature. The most common are the Capacitated VRP (CVRP), a VRP with vehicle capacities; the VRP with Pickup and Delivery (PDP or VRPPD), where items should be transported between an origin and a destination; the VRP with Time Windows (VRPTW), where the customers should be served within given time windows; and the Heterogeneous fleet VRP (HVRP), a generalization of CVRP, where a fleet of vehicles with different capacities and cost is available. DARP is a generalization of the Pickup and Delivery Problem with Time Windows, combined with the HVRP.

Many of the above-stated problems and combinations of them relate to real-world problems that are solved every day. For example, grocery stores that deliver groceries to your door, parcel deliverers such as FedEx and DHL, and in the industry almost all distributions and logistics. Solving these vehicle routing problems is key to efficient transportation and decreasing supply-chain costs. Vendors of VRP routing tools often claim that they can offer cost savings of 5%–30% compared to not using the optimization tools [2]. Considering the enormous volumes handled by logistic transporters on a daily basis, even an improvement of a few per cent can make a significant difference in terms of cost savings and reducing greenhouse gas emissions.

1.1.1 Dial-A-Ride Problem

The DARP consists of moving people between different locations, often with tight time windows for pickups, and special service constraints such as maximum trip durations. The focus is on determining a minimum-cost set of vehicle routes, considering operational and service constraints. A traditional application is the door-to-door transportation service for people with reduced mobility, often elderly and disabled.

The human perspective, of moving people instead of objects between different locations, makes a DARP different from other VRPs. When transporting passengers, reducing user inconvenience must be balanced against minimizing operational costs [3]. The standard formulation for a DARP is often contributed to Cordeau and Laporte [4]. Although this formulation has been studied extensively, practical applications of Dial-A-Ride systems often exhibit numerous additional characteristics, like user-dependent maximum trip durations, multiple vehicle types, and multiple

depots. Ignoring these extensions may result in infeasible or unrealistic solutions. For more practical settings, the considered constraints are usually tailored to the specific problem situation, such as special service vehicles for wheelchair passengers, or a maximum wait time until pickup. In a practical setting the objective can range from maximizing the number of people served, to minimizing user waiting time [5], routing costs [6]–[8] or a combination of both [9], [10].

Finding an optimal solution for the DARP is an NP-hard problem [11]. Nevertheless, when the problem is moderately constrained, and the problem size is small, exact solution approaches can be used. For example, Desrosiers et al. [12] proposed a dynamic programming method for solving a single-vehicle DARP that could handle at most 25 trips. Jean-François Cordeau [7] proposed a branch-and-cut algorithm that can find optimal solutions for at most 48 trips, and 4 vehicles, using several families of valid inequalities to strengthen the LP-relaxation. However, for more realistic cases these methods do not suffice.

Since the DARP is an NP-hard problem, most previous work has concentrated on the development of heuristics to solve larger instances of the problem. Cordeau and Laporte [4] proposed a tabu search heuristic that allows the violation of constraints during the search. The method was tested on a randomly generated set of instances, based on realistic cases provided by Montreal Transit Commission. The instances contained between 24 and 144 requests and are used by other authors as benchmarks. Parragh et al. [13] proposed a competitive variable neighbourhood search, where the neighbourhoods make use of simple swap operations, ideas from ejection chains, and the exploitation of route parts with no passengers in a vehicle. They used the benchmarks of Cordeau and Laporte and found in 16 of the 20 instances an improved solution. Dumas et al. [14] presented a two-phase clustering and column generation method that can handle instances with several hundred customers. All methods above, although solving instances with a reasonable amount of customers, still have large running times. For more realistic cases, e.g. a typical weekday of Valys may contain over 3000 trips, these would not scale well. Toth and Vigo [8], [15] proposed a fast and effective parallel insertion heuristic which can determine good solutions in a few seconds, for relatively small real-life instances arising in Bologna.

Although having large similarities, these methods are not created for solving dynamic DARP. More solutions and versions of the DARP and VRP that have been studied over the past 40 years can be found in the reviews of Cordeau and Laporte [3], Laporte [16], and Molenbruch et al. [17].

1.1.2 Dynamic Dial-A-Ride Problem

The vast majority of the vehicle routing literature is dedicated to deterministic and static models, where all problem parameters are assumed to be known, and all decisions are made before the day of operations. In a dynamic routing problem, some input data is revealed or updated during the time in which operations take place. We are going to look into the Dynamic Dial-A-Ride Problem (DDARP).

Psaraftis [5] first introduced the concept of the dynamic single-vehicle DARP, where new requests arise dynamically over time without any prior knowledge of future requests. Psaraftis solved the problem using a dynamic programming approach that was developed for the static DARP. To our knowledge, little research was conducted on dynamic routing between the work of Psaraftis in 1980 and the late 1990s. However, the last two decades have seen a renewed interest in this class of problems, thanks to better hardware, faster algorithms and efficient real-time communication

technologies. As a result, these algorithms can now be realistically implemented in a practical setting.

Savelsbergh and Sol [18] solved a real-world dynamic VRPPD using a rolling-horizon framework. The problem is divided into a series of static problems, with a subset of the known requests. These are then solved using a branch-and-price heuristic. Computational results of the algorithm are given for several test problems with up to 500 trips, and 100 vehicles with a capacity of up to 24 passengers. The large vehicle capacity makes it easier to find routes for trips, as more trips can be scheduled at the same time on the same vehicle. Besides, the absence of a maximum trip duration makes the proposed method not directly applicable to our situation.

Desrosiers et al. [19] used a cluster first, route second method. First, a mini-clustering phase defines clusters of users that should be served by the same vehicle, then a column generation technique optimally combines the clusters to form vehicle routes. The technique can be used to create good schedules for several hundred trips. Madsen et al. [20] presented an algorithm for a similar problem, for a real-life scenario in Copenhagen, where new requests are inserted in the current routes using an efficient insertion heuristic based on that of Jaw et al. [21]. The limited number of requests these algorithms can handle makes them difficult to use in our scenario.

Attanasio et al. [22] solved the same problem using a parallel tabu search heuristic. Interestingly, they found that randomly generated instances are much harder to solve than real-world instances. Because there are some patterns of taxi mobility in realistic instances: for example, moving from home places to workplace areas and between workplace areas. Lazhar Khelifi et al. [23] designed a hybrid approach based on multi-objective simulated annealing in combination with tabu search to solve the DDARP. The hybrid approach is characterized by a better exploration of the search space. Simulations show a noteworthy negative impact on the quality of solutions when the number of trip requests per hour increases from 20 to 35 over a 4-hour time horizon with 10 vehicles.

Lois et al. [24] proposed an online regret-based algorithm for the DDARP. The structure is based on two separate sub-algorithms, one for handling incoming requests and the other for continuously improving the solution. The algorithm was tested on a set of 1619 trip requests and 50 vehicles with promising results and running time. Nonetheless, due to the distinct characteristics of our research context, including a smaller capacity, the presence of driver constraints, and the use of multiple vehicle types, the findings of this study are not directly applicable to our research.

Horn [25] created a scheduling and dispatching system called L2sched. Their idea was to find a succession of optimal schedules for the deployment of the fleet. The difference between any two successive optimal schedules is induced by a small change in scheduling conditions. So the optimal schedules do not change radically between steps, but they evolve. When a new request for a trip arrives the trip is inserted in the schedule such that the marginal cost is minimized. Then an attempt is made to find local improvements in a broader neighbourhood. Afterwards, at fixed decision epochs a local search procedure improves the schedule, to overcome inefficiencies that may accumulate as a result of the incrementally optimal choices made when inserting new trips. While the paper researches a very similar problem, it does not directly address the specific research question of our study. The paper misses constraints on the maximum trip duration and driver requirements.

Other used methods to solve the DDARP are: insertion heuristics by Varone et al. [26], Jaw et al. [21], and Luo et al. [27]; cluster-then-route, by Bertsimas et al. [28], and Santi et al. [29]; simulated annealing by Yu et al. [30].

1.1.3 Related Problems

The ridesharing problem is very similar to the DDARP. The main difference is that DDARP focuses on optimizing the allocation of vehicles to passengers in real-time, while ridesharing focuses on optimizing the sharing of rides among multiple passengers to reduce travel costs for all passengers. The sharing of rides is a fundamental aspect of ridesharing, where multiple passengers share the same vehicle to reach their destinations. In DDARP, the sharing of rides among passengers is not necessarily a primary concern, and vehicles may transport only a single passenger on some parts of a route. DDARP typically involves the use of specialized vehicles, such as minibuses or wheelchair vehicles, that are designed for special passenger transport. Ridesharing, on the other hand, can involve the use of any type of vehicle, including private cars, taxis, and ride-sharing services like Uber and Lyft, as long as the vehicle can accommodate multiple passengers, but is not taking special needs into account.

Santos et al. [31] looked into the ridesharing problem and proposed a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic, that can solve large scale solutions, in a reasonable time. They divided the day into equal-length periods in which incoming requests are buffered and a static version of the problem is constructed and solved using their GRASP method. Their GRASP implementation exists of three phases: the first phase generated a greedy randomized initial solution, and the second phase performed a local search to improve the solution found. After the local search, a third phase used a path-relinking technique to explore new solutions on the path between two known solutions. The procedure ends when some stopping criterium is met. The authors are able to solve instances where for 12 hours every minute 54 new trips arrive and realize up to 30% savings for passengers that shared a ride. Their work is an extension of their previous work [32]. They show that the path-relinking technique works well for the ridesharing problem. Ma et al. [33] tackled the ridesharing problem by designing a system called T-share to increase the number of passengers served while decreasing the driving distance. Their method is based on a fast taxi-searching algorithm with a lazy shortest-path calculation strategy. These works are of particular interest in relation to the present work. Although they address a different problem, the solution methods presented show promising results for large-scale dynamic problem instances.

Häme [34] proposed an adaptive insertion heuristic for the single-vehicle DARP with tight time windows. The proposed method generates optimal routes when the problem size is reasonable (up to ten customers per vehicle). As the problem size increases, the search space is narrowed to achieve locally optimal solutions. This exact method could be used as a subroutine to perform better than the often-used insertion heuristics. However, Hyytiä et al. [35] demonstrated that the performance degradation of using an insertion heuristic instead of an exact insertion is almost negligible as the number of possible routes becomes large enough. They conclude that classical insertion heuristics are indeed well motivated for problems in which the number of customers assigned to a single vehicle is sufficiently small and the amount of vehicles is large at any moment.

1.1.4 Differences to Previous Research

Our problem differs from previous research in several crucial aspects. Firstly, only a few papers in the literature have specifically addressed the challenges associated

with service cancellations, as most papers focus on handling new transportation requests. Secondly, practical constraints such as driver requirements are often ignored. Driver requirements consist of a maximum route duration and legally mandated breaks.

Thirdly, we included several constraints that are specific to our practical case, but not to many others. These constraints provide passengers with the option to choose whether they prefer to sit alone and avoid being combined with other trips. Additionally, we provide an additional service for individuals with guide dogs or car sickness, who may sit in the front seat.

Although these last two constraints are specific to our practical case, our model remains generally applicable to the DDARP. Both constraints generalize the other DDARP models, as they can be disabled by setting them to false for each trip.

Finally, the scale of our research is much larger than what most studies in the literature have considered. While most research in the literature focuses on small-scale problems, we focus on the complexity and scale of realistic instances which can contain over 3000 trips. Our method effectively handles these additional regulations while maintaining good performance on problems of the given size.

1.2 Approach

This thesis addresses the DDARP that arises in the practical setting of Valys by answering two key questions. First, how does GRASP compare to simulated annealing in a dynamic setting? Second, what is the opportunity gap of dynamic rescheduling as compared to not rescheduling?

First Research Question Based on the literature review, we became intrigued by GRASP and its extensions due to its flexibility in computation time and its ability to reuse parts of the old schedule, which aligns perfectly with our problem. Since GRASP is not commonly used in this area of (D)DARP, we aimed to compare our proposed GRASP algorithm with a well-established method. We selected simulated annealing, known for its ability to produce near-optimal results.

Specifically, we evaluate the computation time versus the reached objective when rescheduling once a day. This is an important metric for dynamic problems characterized by their limited allowed computation time. Furthermore, we examine the influence of the start solution on the final best solution obtained between the two different methods for both rescheduling once and for rescheduling more often. This analysis provides insights into the extent to which GRASP relies on the quality of the start solution compared to simulated annealing. At last, we compare the results of GRASP and simulated annealing when frequently replanning during the day. To show which of the two methods is a better option for dynamic problems which deal with repeated replanning.

Second Research Question We examine the opportunity gap, which refers to the missed opportunities for efficient passenger transportation due to limited information beforehand. Specifically, we investigate the benefits of dynamic rescheduling versus not, by comparing our GRASP algorithm against different benchmarks we established. One of the benchmarks simulates the existing practice of not optimizing changes. We compare the performance of our proposed algorithm against this

benchmark to quantify the impact of our approach and assess the potential for optimization. Another benchmark gives a lower bound on the solution quality, which can be used to see how well our GRASP algorithm performs.

1.3 Overview

The rest of this thesis is outlined as follows. The next two chapters are preliminary. In Chapter 2 we present our model and describe two operators on the model. In Chapter 3, we propose a GRASP method to solve the DDARP. The research questions are covered in the two chapters following the preliminary chapters. Chapter 4 compares our algorithm against a simulated annealing in three experiments. Chapter 5 presents the benchmarks, determines a strategy for replanning, and analyses the results to reveal the opportunity gap. Finally, Chapter 6 presents the conclusions of this thesis and gives some ideas for future research.

Chapter 2

Problem Description & Model

In this chapter, we introduce our DARP and DDARP model, all relevant operational constraints, service constraints, and two operators on the model. For the remainder of this thesis, we assume that all distances are in meters and all times in minutes. The "static algorithm" refers to an algorithm that creates the routes a day in advance, and the "dynamic algorithm" refers to an algorithm that can handle changes on the day itself.

The remainder of this chapter is organized as follows. In Section 2.1, we present our model for the static DARP, and in Section 2.2 we present our model for the dynamic DARP. In Section 2.3 we explain two important operators that are frequently used.

2.1 DARP Variant of Valys

The interval $[0, T]$ gives the planning horizon. A static algorithm creates all routes for all (known) trips that must be executed in the planning horizon. We are given a set N of n trips. A trip $i = 1 \dots n$ has the following characteristics: Let P_i^L denote the number of non-wheelchair passengers to be transported, let P_i^W denote the number of wheelchair passengers to be transported, L_i^p is the pickup location where the passengers must be picked up, and L_i^d is the dropoff location where the passengers must be dropped off.

The passengers of a trip can indicate if they want to sit alone or with other people in the same vehicle. A trip that has passengers that indicated that they want to sit alone should never be combined such that they sit with others in a vehicle at the same time. However, these trips can be combined in a route, such that a vehicle picks up other people before or after serving the passengers. Another option the passengers of a trip can specify is if one of the passengers wants to sit in the front. As each vehicle has only one front seat, we assume that at most one passenger of a single trip wants to sit in the front. This applies e.g. to passengers with guide dogs. A guide dog should sit in between the legs of a passenger, and this is only possible in the front seat.

For each trip i the pickup or the dropoff of the passengers must occur at a given time instant, σ_i, depending on the trip type. There are two different trip types:

- Pickup trip, where σ_i denotes the desired pickup time.
- Arrival guarantee trip, where σ_i denotes the desired dropoff time.

All time instants are modelled as the time in minutes since the start of the planning horizon.

For all trips, a time window is set that specifies the earliest and latest time service can begin. For a pickup trip, the passenger should be picked up in between tw_p

before the desired pickup time σ_i, and tw_p minutes after σ_i. An arrival guarantee trip should be dropped off at most tw_a minutes before the desired dropoff time σ_i, and never later than σ_i. If a vehicle arrives too early, it must wait until the opening of the time window.

For each of the trips in N, we have two decision variables: the planned pickup time B_i^p, and the planned dropoff time B_i^d. We also know for each trip the service time st_i at the pickup and dropoff, e.g., representing the time needed to board or disembark a vehicle. The service time of each trip depends on the type and amount of passengers.

The duration of a trip is the time between the pickup and dropoff of the trip excluding the service time of the trip $(B_i^d - (B_i^p + st_i))$. The duration must not exceed a prescribed maximum trip duration λ_i, which is proportional to the fastest time needed to travel from pickup to dropoff directly. Note the service time of the trip itself is not included in the trip duration and thus does not count up to the maximum trip duration. However, the service time of other trips does count.

The vehicle type that can accommodate a trip depends on the type of passengers it has. Wheelchair passengers can only be transported by vehicles that are equipped to handle wheelchairs, whereas other passengers can be served by any vehicle.

Valys assumes an unlimited mixed vehicle fleet for their current static algorithm, and we follow the same assumption for our heuristic algorithm. There are enough vehicles provided by different transporters, which are all taxi companies, to accommodate all trip requests. Let A be the set of transporters. Each transporter serves a region (which is a set of locations), L_a^A where $a \in A$.

A route k denotes a schedule for a single vehicle. A route $R_k = \{u_1, u_2, \ldots, u_z\}$ is an ordered sequence of locations that a vehicle will visit, sorted by visiting time. The first location u_1, is the first pickup location of the route and determines the assigned transporter. A transporter a is assigned to route k if pickup location u_1 lies in the region L_a^A, $u_1 \in L_a^A$. The route starts at the first pickup location, and we do not account for driving time prior to the first pickup. Transporters differ in their allocation strategy for allocating vehicles to a route, therefore it is impossible to know the exact location of any vehicle, and thus we also are unable to determine the needed driving time prior to the first pickup. A route needs to end in the region of the corresponding transporter a. When the location of the last dropoff lies in the region, the route directly ends, then u_z is the dropoff location of the last trip and $u_z \in L_a^A$. If not, the vehicle needs to drive from the last dropoff location (u_{z-1}) to the region of the transporter. Let u_z be the closest location within the region of the transporter to the last dropoff location u_{z-1}. The route duration d_k of a route k is the time difference between u_1 and u_z (including disembarking time if u_z is the dropoff location of a trip).

Different types of vehicles may be used to drive a particular route. Let F denote the set of available vehicle types, and let the decision variable $y_k \in F$ represent the vehicle type assigned to route k. Each vehicle type $f \in F$ has a capacity of q_f^p non-wheelchair passengers and q_f^W wheelchair passengers, where $q_f^W = 0$ if the vehicle does not accommodate wheelchairs. Notably, all vehicles have a single front seat. Additionally, each vehicle type f is associated with a cost per minute of c_f.

We want to find a set of routes M, where each trip is assigned to a route. A fourth decision variable x_{ik} denotes the assignment of a trip i to a route k, one if i is assigned to k and zero otherwise. Note that a trip can only be assigned to a single route, $\sum_{k \in M} x_{ik} = 1$, but a route can contain multiple trips.

Definition 2.1 (Locations) *$\mathcal{L} = \{L_i^p, L_i^d \mid \forall i \in N\} \cup \{u_r \mid u_r \in L_a^A, \forall a \in A\}$ is the set of all locations. Let t_{l_1,l_2} $(l_1, l_2 \in \mathcal{L})$ be the shortest travel time from location l_1 to location l_2.*

Definition 2.2 (Individual trip) *An individual trip is a trip that is combined with no other trips, i.e., a route with only one trip.*

Remark 2.1 In general, t_{l_1,l_2} may not be equal to t_{l_2,l_1}, and the triangle inequality is not always satisfied. Because in the real world, it is sometimes faster to go from A to C via B, instead of directly from A to C, if only access traffic is allowed in the area of B.

We now define the constraints which must be satisfied by the vehicle routes.

Constraint 2.1.1 (Capacity) *For any given route k, at time $t = 0, \ldots, T$, the number of non-wheelchair passengers p_t and the number of wheelchair passengers wp_t in a vehicle must not exceed the corresponding vehicle capacity. We can express this constraint as:*

$$p_t + wp_t \leq q_{y_k}^P + q_{y_k}^W \qquad t = 0, \ldots, T \tag{2.1}$$

$$wp_t \leq q_{y_k}^W \qquad t = 0, \ldots, T \tag{2.2}$$

Constraint 2.1.2 (Pickup trip time window) *The planned pickup and the service time of a trip i at the pickup location should be within the pickup time window of the corresponding trip.*

$$\sigma_i - tw_p \leq B_i^p \leq B_i^p + st_i \leq \sigma_i + tw_p$$

Constraint 2.1.3 (Arrival guarantee trip time window) *The planned dropoff and the service time of a trip i at the dropoff location should be within the time window of the trip.*

$$\sigma_i - tw_a \leq B_i^d \leq B_i^d + st_i \leq \sigma_i$$

Constraint 2.1.4 (Driver break) *A driver is legally obliged to have a break when driving for at least τ_d minutes. The break should be held within these τ_d minutes and should be at least τ_b minutes long. A route is not allowed to have a break within the first τ_f minutes after a break or after the start of the route, and a route should not end with a break. During the break, the vehicle should be empty and is not allowed to drive. When $d_k \geq \tau_d$, a break should be included in the route.*

Constraint 2.1.5 (Maximum route duration) *A vehicle is not allowed to drive longer than τ_m minutes, $d_k \leq \tau_m$.*

Constraint 2.1.6 (Front seat) *When a trip i has a passenger that wants to sit in the front seat, then trip i should not simultaneously share the same vehicle as another trip that also wants to sit in the front seat.*

Constraint 2.1.7 (Sit alone) *A trip i that indicated that the passengers in the trip want to sit alone, should not simultaneously share the same vehicle as another trip.*

Constraint 2.1.8 (Maximum trip duration) *A trip i is not allowed to take longer than its maximum trip duration λ_i, $B_i^d - (B_i^p + st_i) \leq \lambda_i$. We define λ_i in terms of a factor (λ) of the shortest driving time from pickup to dropoff. There are two exceptions to the rule, the minimum excess ride time is 15 minutes and the maximum excess ride time is 60 minutes, i.e.:*

$$\lambda_i = t_{L_i^p, L_i^d} + \min(60,\ \max(15,\ \lambda t_{L_i^p, L_i^d})) \tag{2.3}$$

To summarize, the Dial-a-Ride problem is to create a set of routes for vehicles that contain all trips, and minimizes the weighted sum of route durations ($\sum_{k \in M} c_{y_k} d_k$), subject to the constraints 2.1.1 - 2.1.8.

2.2 DDARP Variant of Valys

In the dynamic version of the DARP, we have a series of time points $e_1, \ldots, e_q$, at each of which a "periodic" reoptimization is started. This reoptimization incorporates all changes into the current schedule that became known in between the time points while maintaining feasibility and taking routes that are in progress into account.

At time point e_i, we receive a set N of trips, a set M of vehicle routes, and a time in minutes τ. The vehicle routes in M are created by a static algorithm a day in advance or by a previous optimization. Let τ denote the allowed computation time of the algorithm and the time transporters need to communicate the changes in the routes to their drivers.

During the day of operations, we receive two types of changes: a new trip δ_{nt} or a cancelled trip δ_c.

An incoming cancellation message δ_c becomes known at time t_c^δ. It contains a trip $i \in N$ that has been cancelled. A characteristic of a cancellation is that it is known at least τ_c minutes in advance. So for a cancelled trip $B_i^p \geq t_c^\delta + \tau_c$.

A new trip message δ_{nt} becomes known at time t_{nt}^δ. It contains a new trip i that is not yet known, i.e. $i \notin N$, and where L_i^p, L_i^d, g_i, P_i^L, P_i^W, σ_i, A_i and F_i are defined but B_i^p, B_i^d and x_{ik} $\forall k \in M$ are undefined. A new trip becomes known at least τ_{nt} minutes in advance, thus $\sigma_i \geq t_{nt}^\delta + \tau_{nt}$.

Then for time point e_i all changes with $e_{i-1} \leq t_c^\delta, t_{nt}^\delta < e_i$ are taken into account, where $e_0 = 0$. All changes with $t_c^\delta, t_{nt}^\delta < e_{i-1}$ have already been taken into account by a previous reoptimization. Let $\Delta = \Delta^c \cup \Delta^{nt}$ be the set of changes of the current periodic interval, $[e_{i-1}, e_i)$, where the superscripts denote the change types, cancellation and new trip respectively. Figure 2.1, shows the process of a periodic approach graphically.

A change might take place before the execution of the next reoptimization. A change that becomes known within the periodic interval and the corresponding trip starts before the beginning of the next periodic reoptimization is called locked. A cancellation or new trip is locked if at the next periodic reoptimization e_i one of the conditions in Definition 2.3 applies. Locked changes can not be handled any more. These changes should be handled by the default rules. The default rules give rise to a baseline benchmark that we compare to, see Section 5.1.1

During the periodic reoptimization, we want to find a new schedule where all trips in Δ^{nt} are incorporated in a new route set M and all trips in Δ^c are removed from the routes. Hereby we can create new routes (M), change the trips in a route

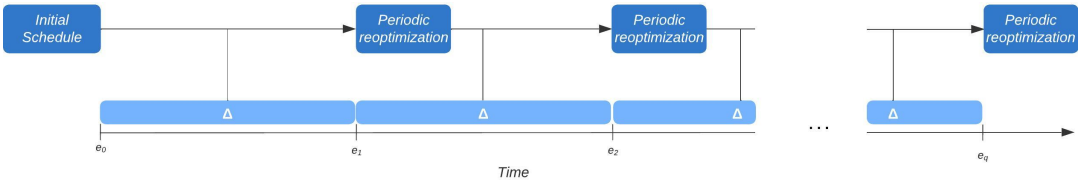


FIGURE 2.1: Periodic approach graphically displayed.

(x_{ik}), the order of pickup and dropoffs (R_k), the vehicle type of a route (y_k), and the pickup and dropoff time of a trip (B_i^p and B_i^d). However, we are not allowed to change all parts of a route. At the time of rescheduling, some trips might already have been executed, and some others might currently be in progress. These trips can not be changed any more and are called locked. Thus, for locked trips, we can't change any of the above-named changes in the routes.

Definition 2.3 (Locked trips) *A trip i is locked if one of the following conditions hold:*

1. *The trip has finished before the rescheduling interval starts:*
$$B_i^d \leq e_i + \tau \tag{2.4}$$
2. *The trip is still in progress, thus already picked up but not yet dropped off:*
$$B_i^p \leq e_i + \tau < B_i^d \tag{2.5}$$
3. *We are driving towards the trip, thus the next location of a vehicle that is currently executing a route is the pickup location of a trip i, let j be the location visited just before pickup of i:*
$$B_j^{p/d} \leq e_i + \tau \leq B_i^p \tag{2.6}$$
4. *A trip is locked if it simultaneously shares the same vehicle as another trip that is locked.*

Condition 4 extends the locked part until the vehicle load of the route is empty. We do this because inserting something in between the pickup and the dropoff of a locked trip becomes unnecessarily complicated, often it is also not desired by the transporters.

Definition 2.4 (In progress) *A route is in progress at the start of the rescheduling interval if:*
$$B_i^p \leq e_i + \tau < \tau_z \tag{2.7}$$
where trip $i \in N$ is the first trip that is picked up by the route, $R_k = \{L_i^p, \ldots, u_z\}$, and τ_z is the finish time at location u_z, so either the time of the dropoff of a trip or the return time at the working area.

Definition 2.5 (Finished routes) *A finished route $k \in M$ is a route where the end time of the route is before the start of the planning interval. All finished routes can be removed from M, before reoptimizing, together with all trips they served, trip $i \in N$ can be removed from the set N if: $x_{ik} = 1$.*

At the start of e_i, a route $k \in M$ is assigned a vehicle type that can carry a certain amount of wheelchair passengers and other passengers. If a route is currently in progress, see Definition 2.3, we can't change its vehicle type (y_k). When a route is not in progress and not finished, we can change its vehicle type, e.g., to be able to transport more or fewer passengers.

The challenge of the DDARP is to find a set of routes that minimizes the weighted sum of the route durations ($\sum_{k \in M} c_{y_k} d_k$). The set of routes should at least contain all routes from the input that are in progress. Of the routes in progress the order of the locked trips should not be changed, and the vehicle capacity of the routes that are in progress should also not be changed. Besides, all cancelled trips should be removed from the routes and all new trips should be incorporated.

2.3 Route Model and Route Operators

Previously in Section 2.1 a route was defined as a sequence of locations. In our implementation, we model a route as a list of nodes. A node represents the pickup of a trip, the dropoff of a trip, or a driver break. The nodes store the following information:

Earliest arrival time (EAT): The earliest possible moment to reach the node.
Earliest departure time (EDT): The earliest time we can leave the node
Wait time: The duration before the trip's time window opens (zero for breaks).
Vehicle load: The number and type of passengers.
Last break time: The time of the last break (initially set to the EAT of the first node).
Slack time: The maximum allowable shifting of preceding nodes forward in time while maintaining a feasible route.

A solution consists of a list of routes, where every trip is present in exactly one route. Permutations of the solutions are created by two operators: 1) inserting a trip into a route and 2) removing a trip from a route. These operators always respect the constraints 2.1.1 - 2.1.8.

With these operators, we can generate various neighbourhoods for the solution, such as inserting a trip into a route or swapping two trips between routes. As the two operators take care of all constraints, we do not need to check them anywhere else and it will be enough to know if an operation was successful or if it was infeasible. Given that these two operators will be frequently applied, their computation complexity becomes crucial.

2.3.1 Trip Insertion

To insert a trip into a route, we use a stack of nodes to represent a route. In Listing 1 we show a simplified version of our trip insertion implementation. The process starts with an empty stack unless the route contains locked trips. In the case of locked trips, we push all nodes that are in the locked part of the route onto the stack.

Subsequently, we continuously pop and push nodes onto the stack while recalculating the aforementioned parameters. This process allows us to explore all possible insertions of a trip, thereby calculating the best possible insertion while preserving the order of the nodes in the route. Whenever we mention pushing a node to the stack it implies that all parameters are recalculated accordingly.

We maintain the earliest arrival principle, meaning that we arrive at every node at the earliest possible moment. Figure 2.2 shows how node parameters are calculated. To adhere to the principle of earliest arrival, we calculate the EAT by adding the driving time between the previous node and the current node to the EDT of the previous node. Whenever we have to wait, we consult the slack time of the previous node to determine if all preceding nodes can be shifted forward in time, thereby reducing the wait time and thus the total service duration of the route. Note that this will never result in infeasible pushes to the stack, as we simply reduce the wait time. In Figure 2.2, the wait time can partly be reduced by the slack time remaining for trip b, indicating that the EAT of trip a, b and c is shifted forward in time with the amount of slack available.

In Line 4 of Listing 1 we consider all locations for a pickup. In Line 5, if there are already nodes on the stack, we first use the *PopNodes* function to remove all nodes

```
1  simplified procedure Insertion(route_stack, trip)
2      stack = new Stack()
3      PushLockedNodes(route_stack, stack)
4      for pickup_location in  range(len(stack), len(route_stack)):
5          if pickup_location > 0:
6              PopNodes(stack, pickup_location)
7              stack.push(route_stack[pickup_location - 1])
8          Validate(stack.push(trip.pickup))
9          for dropoff_location in range(pickup_location, len(route_stack)):
10             if pickup_location ≠ dropoff_location:
11                 PopNodes(stack, dropoff_location)
12                 Validate(stack.push(route_stack[dropoff_location - 1])
13             Validate(stack.push(trip.dropoff))
14             for k in range(dropoff_location, len(route_stack) + 1):
15                 if k == len(route_stack):
16                     SaveFeasibleInsert(stack)
17                     break
18                 Validate(stack.push(route_stack[k])
19     return BestSavedFeasibleInsert()
```

LISTING 1: Simplified version of the insertion procedure. Adding breaks, checking feasibility and the acceleration steps are hidden in the function *Validate*.

until the length of the stack is i again, as these nodes were pushed when checking another insert combination. Line 7 pushes the first not yet pushed route node onto the stack. Then, in Line 8, we push the pickup node of the trip we want to insert onto the stack and validate the push using the *Validate* function. This *Validate* function handles adding breaks to the route (explained later), checks feasibility and performs some acceleration steps. When *Validate* encounters an infeasibility, it continues at the next pickup position $(i + 1)$, as the route can never become feasible any more. The for loop declared in Line 9 ensures that we check all insert combinations of the pickup on the current position and the dropoff on later positions.

Similar to Line 5, Line 10 removes nodes that were pushed when checking another insert combination. Then, in Line 12, we push the next node of the route onto the stack, increasing the dropoff insert position by one. Line 13 inserts the dropoff

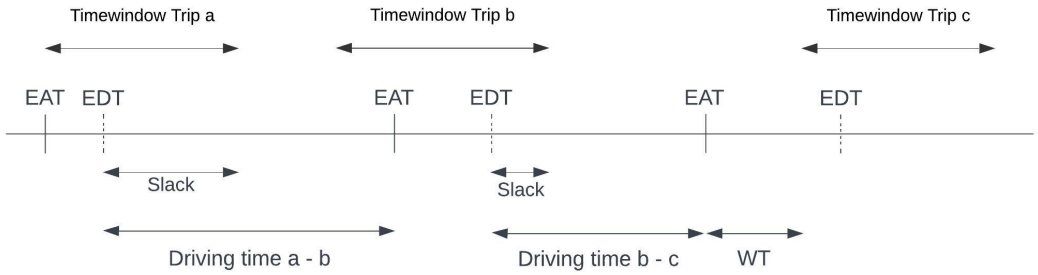


FIGURE 2.2: Values of the parameters after recalculation. EAT stands for earliest arrival time, EDT for earliest departure time and WT for the wait time.

and validates its feasibility again, when infeasible the next dropoff position is considered ($j + 1$). Now we have pushed both pickup and dropoff, with possibly some route nodes before the pickup or in between the pickup and dropoff. Line 14 to 18 inserts all remaining nodes in the route after the dropoff. When we reach Line 16, we found a feasible insertion and save the stack. Finally, in Line 19 the best-saved stack is returned. If there is no feasible insert, no stack was saved and an empty stack is returned to indicate infeasibility.

Breaks A break is required if any of the pushed nodes became infeasible because the sum of the last break time and τ_d is smaller than the EDT of the current node. To insert a break, we begin at the end of the stack and traverse backwards through all the nodes, checking if a break can feasibly be inserted in between any two nodes. A break is inserted at the first feasible position encountered. It is important to note that the vehicle load must be empty in order to insert a break. If no feasible break position is found, we stop the procedure and the node is deemed infeasible for pushing onto the stack. This is handled in Listing 1 in the *Validate* function. When a break was successfully added, the corresponding break is removed when the node that triggered the insertion of a break is popped from the stack, handled in the *PopNodes* function.

Speed Up Steps As this procedure of inserting a trip is essentially a "smart" brute-force method for inserting a trip into a route, we have devised several branching rules to expedite the procedure.

1. Prior to pushing the pickup of the insert trip onto the stack, we verify if the EDT of the current top node is no later than the end of the pickup's time window. In such a case, we can no longer achieve a feasible solution with the pickup node on this position or later positions, as we can never get in time to the pickup location. Thus we terminate the insertion procedure. This is a strong branching rule as it allows us to skip numerous combinations.

2. If any node is infeasible and the dropoff of the inserted trip is not yet pushed onto the stack, all subsequent combinations involving the pickup at the current position will also be infeasible. Consequently, we backtrack to continue the search with the pickup on the next position.

3. If the EDT of the current top node is no later than the close time of the dropoff's time window, the dropoff node cannot be feasibly pushed any more. Hence, we continue to the next position for the pickup.

4. If the EDT of the last node minus the EAT of the first node exceeds the maximum driving time (τ_m), we can cease further search and backtrack until we removed the pickup.

The four branching rules listed above significantly speed up the insertion procedure. Officially, this procedure has a cubic time complexity. However, empirical tests show that a linear number of insertion positions are often considered. This is due to the infrequent activation of the break procedure. Additionally, the stack ensures that only the combinations where pickups precede dropoffs are considered. Finally, the branching rules further reduce the number of positions by quickly discarding combinations that will never result in a feasible insertion.

Remark 2.2 The trip insertion procedure does not capture all feasible insertions. Sometimes it might be feasible to insert a trip but our implementation will conclude that it is not feasible. This is due to the maximum trip duration. When an insertion is infeasible because the maximum trip duration of trip i is exceeded it might be possible that waiting at the pickup node of trip i decreases the duration of i such that the maximum trip duration isn't exceeded. We didn't include this case as it would significantly increase the running time of the procedure. Additionally, we saw that when this occurs there often would be another order that was feasible, thereby still returning a feasible insertion, even though it might not be the best insertion any more.

2.3.2 Trip Removal

Removing a trip from a route that consists of two trips is straightforward, as both trips become individual. For longer routes or routes containing locked trips, we again make use of a stack. We iterate through the current route, pushing nodes onto the stack from front to back. When we encounter the pickup or dropoff node of the removed trip, we skip pushing this node to the stack and continue at the next node in the route. When the last node of the route is considered we are finished. There are three special cases that require attention:

1. The first two nodes correspond to the pickup and dropoff nodes of the removed trip and the third node is a break. The break should be ignored, as starting with a break is useless.

2. The last two nodes represent the pickup and dropoff nodes of the removed trip. In this case, the new last node shouldn't be a break, if it is, it should be popped off the stack, as ending with a break is unnecessary.

3. Skipping the removed trip nodes results in two breaks too soon after each other (i.e., the time between the two breaks is smaller than τ_f). In this case, we do not push the second break to the stack and proceed with the next node.

All parameters mentioned in Section 2.3.1 should still be recomputed. Note that removing a trip may increase driving times (see Remark 2.1), and hence make the route infeasible. When that happens a trip can not feasibly be removed. If no infeasibility was found the trip can successfully be removed.

Chapter 3

Greedy Randomized Adaptive Search Procedure

Many types of metaheuristics have been used to solve dynamic routing problems. Most researchers used well-known techniques such as simulated annealing, tabu search or genetic algorithms. In this thesis, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) with evolutionary Path Relinking to solve the DDARP. To the best of our knowledge, we are the first to implement GRASP for the DDARP.

We chose to implement GRASP instead of other well-known metaheuristics for a few reasons. Foremost, GRASP can build upon the existing planning using already well-researched (insertion) heuristics. Since dynamic replanning is often constrained by limited computation time, having a schedule that is likely to be near-optimal provides a good starting point, such as the one created a day before by the static algorithm. By using this schedule as a foundation, GRASP can quickly build upon it to find even better solutions. This ensures that the limited computation time is used effectively and efficiently.

Secondly, the total computation time of GRASP is predictable. Since the computation time does not vary significantly from iteration to iteration, the total computation time increases linearly with the number of iterations. Therefore, the longer the computation time, the more solutions we can explore and the better the final solution becomes. Even with only a few minutes available, GRASP is able to find and connect dozens of local optima using path relinking. This flexibility makes the method highly adaptable, as we can run GRASP with limited computation time and still obtain a good solution, or allocate more computation time to achieve an even better solution.

The remainder of this chapter is organized as follows. In Section 3.1, we will explain the working of our GRASP algorithm. Then in Section 3.2, we will introduce path relinking and how it can be combined with GRASP.

3.1 GRASP Heuristic

GRASP was proposed by Feo and Resende in 1995 [36]. GRASP is a multi-start metaheuristic, which involves running a metaheuristic multiple times with different starting solutions to increase the likelihood of finding a globally optimal solution. The pseudocode in Listing 2 illustrates the main block of a GRASP algorithm. Each iteration consists of a construction phase, where a feasible solution is constructed, and a local search phase, which starts at the constructed solution and applies iterative improvement until a local optimum is found. This process is repeated until

some stopping criteria have been met. The best local optimal solution found in all GRASP iterations is returned as the final solution.

The goal of a metaheuristic is to find a sufficiently good solution to a problem, it samples a subset of solutions which is otherwise too large to be completely enumerated or otherwise explored. A metaheuristic has two goals that have to be balanced against each other, exploration, and exploitation. Exploration refers to searching in the unexplored area of the solution space, while exploitation refers to the search of the neighbourhood of a promising region. The construction phase includes both exploration and exploitation. Depending on some parameter α, the construction phase focuses more on exploration or exploitation. The local search phase of GRASP focuses solely on the exploitation, it tries to quickly move to a local optimum.

```
1 procedure GRASP(start_solution, changes)
2     bestSolutionFound = None
3     while GRASP stopping criterion not satisfied:
4         solution = ConstructionPhase(start_solution, changes)
5         solution = LocalSearchPhase(solution)
6         UpdateBestSolution(solution, bestSolutionFound)
7     return bestSolutionFound
```

LISTING 2: GRASP procedure

3.1.1 Construction Phase

During the construction phase, the algorithm uses a randomized greedy heuristic to iteratively construct an initial solution to the problem. One element at a time is considered to build up the solution. In this approach, the randomized greedy heuristic ranks all elements that can be added to the solution by a greedy function according to the quality of the solution they can achieve. To introduce variability, a part of these elements are placed in a restricted candidate list (RCL) and selected randomly from this list when building the initial solution.

The original GRASP procedure starts from an empty solution. However, as we are dealing with a dynamic problem and therefore receive the current schedule, we decided to take the current schedule as a start solution. This makes it possible to reuse parts of solutions from previous optimizations.

To construct an initial solution, we start with the current schedule and adapt it to exclude all cancelled trips and include all new trips using a basic set of rules, also defined in the baseline benchmark described in Section 5.1.1. This benchmark handles the changes as in current practice. New trips are driven individually (see Definition 2.2). Cancellations are removed from their route, as explained in Section 2.3.2.

After constructing this initial solution, we implemented a concept used in large neighbourhood search (LNS) [37]. LNS alternates between a destroy phase and a repair phase. A destroy phase destructs part of the current solution to escape local optima, while a repair phase rebuilds the destroyed solution. We implemented the idea of a destroy phase, to introduce an extra level of diversification. The initial solution is for every GRASP iteration the same, so the destroy phase ensures that the initial solutions are sufficiently different from each other. This way, the GRASP iterations search in different areas of the search space. Another reason for the destroy phase is to destroy parts of the solution that otherwise would never be explored.

E.g., routes that would never be split in the local search phase because the removal (or swap) of a trip never leads to an improvement, and first a decrease in solution quality is needed to later improve the route.

The degree of destruction is an important choice when implementing the destroy method. If only a small part of the solution is destroyed then the heuristic may repeatedly enter the same local optima and is not able to search the entire search space. However, when a large part of the solution is destroyed LNS may act as a random restart method, where no parts of the previous solutions are reused.

Our destroy method takes a random percentage, in between 0% and 30%, trips and removes them from their route. Note that individual trips are kept individual and if removal results in an infeasible route, the route is left unchanged.

The rest of the construction phase can be seen as the repair phase of LNS, where we rebuild the solution element by element. We repeatedly insert or combine individual trips into routes as long as the solution stays feasible. At the start, there is a higher probability of selecting insertion as the method of modifying the solution, while combination is less likely to be chosen randomly. When insertion was selected, we randomly select a route and calculate all possible insertions of all individual trips into this route. All feasible insertions into the route are called candidate insertions and are kept in a candidate list. The candidate list is prioritized according to the actual insertion cost of the candidate insertions, which is calculated by subtracting the cost of driving the route individually from the cost of inserting the trip into the route.

Let the insertion cost of a trip i into a route k be μ_i^k, and let CL^k be the candidate list that contains all individual trips with a feasible insertion into route k. Only the best insertions in CL^k are considered and stored in a restricted list of candidates called RCL^k. Let $\mu_{min}^k = \min_{i \in CL^k} \mu_i^k$ and $\mu_{max}^k = \max_{i \in CL^k} \mu_i^k$, respectively, the minimum and the maximum cost attained by all candidate insertions in CL^k. The RCL^k can be defined as:

$$\mathrm{RCL}^k = \{i \in CL^k \mid \mu_i^k \leq \mu_{min}^k + \alpha(\mu_{max}^k - \mu_{min}^k)\} \tag{3.1}$$

for a given value of $\alpha \in \{\alpha_1, \ldots, \alpha_L\}$. The choice of the trip to be inserted in the partially constructed feasible solution is chosen uniformly at random among all requests included in the RCL^k. Once a trip i has been chosen from the RCL^k, the trip is inserted into route k, and removed from the set of individual trips. When CL^k is empty, no feasible insertion is possible and the route is not considered in the next iterations.

When combine is selected, we build a CL based on all possible combinations of two individual trips combined into a new route. Then the same procedure as above is used to determine which two trips will be combined. We decided to implement this combine method because it is an extra diversification step. Ensuring that some trips that otherwise will never be combined still have a possibility to end up together. At most 70 new routes are created in the construction phase, to prevent combining too much.

This process is repeated until no trip can feasibly be inserted into one of the routes or all trips have been inserted into a route. A formal description of the construction phase is reported in Listing 3.

```
procedure ConstructionPhase(start_solution, changes)
    solution = BenchmarkSolution(start_solution, changes)
    solution = RemoveTripsFromRoute(solution, rand(0, 31))
    routes = all routes in solution, new_routes = rand(50, 71)
    CE = all individual trips in solution
    α = pick random from {α_1,...,α_L} according to probabilities p(α_l), l = 1,...,L
    while routes not empty and CE not empty:
        x = rand(0, len(routes) + (new_routes > 0)), CL = []
        if x == len(routes):
            CL = FeasibleCombinations(CE)
            RCL = BuildRCL(CL, α)
            (trip_a, trip_b) = RCL[rand(0, len(RCL))]
            new_route = solution.Combine(trip_a, trip_b)
            Update(CE, routes, trip_a, trip_b, new_route)
            new_routes -= 1
            if len(CL) == 1: new_routes = 0
        else:
            CL = FeasibleInsertions(routes[x], CE)
            RCL = BuildRCL(CL, α)
            trip = RCL[rand(0, len(RCL))]
            solution.Insert(routes[x], trip)
            Update(CE, routes, trip, routes[x])
    return solution
```

LISTING 3: GRASP construction phase

Effect of the Greediness Factor

In the construction phase, the amount of greediness is determined by the parameter α. For $\alpha = 1$ the algorithm acts completely random and chooses from all insertion candidates ($RCL^K = CL^k$). For $\alpha = 0$ the algorithm is fully greedy and only chooses from the candidates that have insertion cost equal to μ^k_{min}.

Prais and Ribeiro in [38] have shown that using a fixed value for α very often hinders finding a high-quality solution, which eventually could be found if another value was used. That's why we decided to select the value of α from a discrete set of 11 values: $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. The value is chosen according to pre-assigned probabilities for each α value.

We based the pre-assigned probabilities on the following reasoning, mentioned by Resende et al. [39] (the quote has been edited to fit the thesis): "each application of the GRASP construction procedure produces a sample solution from an unknown distribution, whose mean and variance strongly depends on the value of α. If the RCL has one element, then the same solution will be produced in all iterations. The variance of the distribution will be zero and the mean will be equal to the value of the greedy solution. Instead, when the RCL contains more elements, many solutions will be produced, implying a larger variance. Since we are acting less greedy, the mean solution value should be worse than that of the greedy solution. However, the value of the best solution found outperforms the average value and can be near-optimal or even optimal. It is unlikely that GRASP will find an optimal solution if the average solution value is high, even if there is a large variance in the overall solution values. On the other hand, if there is little variance in the overall solution construction, it is also unlikely that GRASP will find an optimal solution, even if the mean solution value is low. What often leads to good solutions are the values for

α that produce relatively low average solution values in the presence of a relatively large variance, such as is the case for $\alpha = 0.2$."

Besides influencing the solution quality, the greediness factor also has an impact on the computational time needed in the local search phase. Low values of α produce more greedy solutions. Those solutions are closer to a local optimum and hence less time is needed in the local search phase. When α is large, more random solutions are created. This increases the time in the local search phase since a random solution is on average further away from a local optimum. Consequently, a higher probability of picking a lower α value has a positive impact on the computation time in the local search phase and thus on the overall computation time.

3.1.2 Local Search Phase

During the local search phase, a local search is performed on the solution constructed in the construction phase until a local minimum is found. In each iteration of the local search, we keep a current solution, which will explore its neighbourhood to obtain a better solution. Anytime a better solution is found, we replace the current solution with the better one and a new iteration begins. A local optimum is found when no solution in the neighbourhood is better than the current solution.

We chose variable neighbourhood descent (VND) as the local search procedure. This is a variant of variable neighbourhood search (VNS) [40]. VNS systematically changes neighbourhoods during the search. The underlying principle is that a local optimum under one neighbourhood structure is not necessarily the optimum of another, but a global optimum is a local optimum regarding all possible neighbourhoods. VND only accepts solutions strictly better than the current solution. When a local optimum is reached with the current neighbourhood structure, the search resumes with a different neighbourhood structure to escape from the current optimum. We repeat the process until the current solution is optimal in all neighbourhood structures.

We have six neighbourhood structures, in the search order: move, move-remove, swap, combine, remove and move sequence. These neighbourhood structures will be explained in depth in Sections 3.1.2.1 – 3.1.2.6.

There exist two typical strategies to explore a neighbourhood, 1) best improvement, also known as steepest descent, and 2) first improvement. In best improvement, the associated neighbourhood is completely explored by a deterministic procedure performing each iteration the best possible operation. The order in which all neighbours are evaluated doesn't matter, since the entire neighbourhood is explored. The first improvement strategy tries to avoid the complexity of exploring the entire neighbourhood by moving to the first neighbour encountered which improves the current solution. The order in which the neighbours are explored can have a significant influence on the final result. Instead of using a deterministic order as in the best improvement strategy, a random order is usually performed, since the same local optima are encountered when using a deterministic order and starting from the same solution. For the efficiency reason mentioned above, we chose to implement the first improvement strategy for all neighbourhoods.

Most VND variants traverse the neighbourhood structures sequentially. Basic VND, pipe VND, cyclic VND and union VND are the most used search procedures. These variants differ in how they implement the neighbourhood change procedure. Specifically, if an improvement has been detected in some neighbourhood, it defines how the search is continued. We implemented the pipe VND that after an improving operation continues the search in the same neighbourhood until no improving

```
procedure LocalSearch(Solution solution)
    neighbourhoods = [move, move_remove, swap, combine, remove, move_sequence]
    current_neighbour = 0
    found_improvement = false
    while current_neighbour < 6:
        if current_neighbour == 0:
            found_improvement = false
        while not neighbourhoods[current_neighbour].is_exhausted():
            sol = neighbourhoods[current_neighbour].single_step()
            if sol is better than solution:
                solution = sol
                found_improvement = true
        current_neighbour++
        if current_neighbour >= 5 and found_improvement:
            current_neighbour = 0
    return solution
```

LISTING 4: GRASP local search phase

neighbours are found, then the search is continued in the next neighbourhood structure. When no new improvements can be found in the remove, second to last, neighbourhood, and we made at least one improvement in the current or previous four neighbourhoods, we continue again at the first neighbour. If this is not the case, we continue the search at the move sequence neighbourhood. Exploration of this neighbourhood is explained in Section 3.1.2.6. When all six neighbourhoods didn't find an improving operation, we stop the local search and the current solution is locally optimal with respect to all neighbourhoods. A formal description of the local search phase is covered in Listing 4.

3.1.2.1 Move

The move neighbourhood consists of inserting an individual trip into a route. The trip is inserted as described in Section 2.3.1. We explore the entire neighbourhood by looking efficiently at all individual trip, and route combinations.

3.1.2.2 Move-Remove

The move-remove neighbourhood looks if an individual trip can be inserted in a route by removing a trip of that route. The removed trip becomes individual and can again be reinserted in another route during the same iteration of this neighbourhood.

This neighbourhood comes after the move neighbourhood, thus we know that at the start no individual trip can be placed in any of the routes and thereby improve the solution. This neighbourhood looks one step further and checks if we can create a better route by removing a trip from the route and then insert the individual trip into the route.

3.1.2.3 Swap

The swap neighbourhood consists of the swap operator. It takes two trips that are currently in a route (not individual trips) and checks if we can remove the trips from their routes and then insert them in the other route.

We extend the swap operator to look at two more cases for the same two trips. The other cases remove the two trips from their routes and insert only one of the trips in the route of the other trip. The other trip that is not inserted is made individual. The operator with the largest cost decrease is chosen. Note that when evaluating the original swap operator, we already have all information for the extra operators. This operator is also applied for two trips that are in the same route, by removing them both and then reinserting them again one by one.

3.1.2.4 Combine

The combine neighbourhood takes two individual trips and inserts them in a new route if the cost is less than driving them both individually. We do not want too many new routes at a time, therefore we do not fully explore this neighbourhood. This is because if we fully explore this neighbourhood we get a lot of new routes and the number of individual trips decreases. This leads to worse and fewer operations in the first two neighbourhoods.

Therefore, each neighbourhood iteration creates at most x new routes at a time, where x is randomly chosen in the domain: $x \in [4, 15]$. When x new routes are created, or no trips can be combined we continue the search in the next neighbourhood.

3.1.2.5 Remove

The remove neighbourhood takes a random non-individual trip and checks if it can be removed from its route, and made individual.

3.1.2.6 Move Sequence

The operator of this neighbourhood selects at most 350 "similar" trips based on the Shaw operator [37]. The trips can be both individual trips and non-individual trips. However, no two trips that are on the same route are selected. All trips are removed from their route (if this is not feasible another trip is chosen to replace this trip). Then we calculate for each trip the insertion into each route of the other trips. If the route originally contained an individual trip, it is now empty and an insert will create an individual trip. If inserting the trip in a route is not feasible then the route is left as is and the trip is made individual. The cost of inserting the trips into the routes is placed in a 350×350 matrix, where the rows correspond to routes and the columns to trips. Therefore, a cell represents the cost of inserting the trip corresponding to the column into the route corresponding with the row.

An assignment algorithm, that uses the min-cost max-flow algorithm, is used to calculate the cheapest way of inserting all trips into the routes. Note that the cost difference of the best assignment is at most zero, as we can reinsert all trips back into their original routes and get the same solution as before. Therefore, we never generate worse solutions.

When reaching this neighbourhood we know that the solution is locally optimal with respect to all other neighbourhoods. This neighbourhood is used to escape from that local optimum by looking at a very large part of the solution space. A successful application with a non-zero cost delta implies that we have reached the region of another local optimum that is better than the current. Therefore, this neighbourhood is perfect to escape local optima. But the time complexity of a single operation in this neighbourhood is very large, as 350 trips have to be removed and reinserted. Hence, we only execute this neighbourhood when all other neighbourhoods are fully

explored. Besides, every time we reach this neighbourhood we execute this neighbourhood at most x times, where x is randomly chosen in the domain: $x \in [3, 5]$. In a single application of the local search phase, the procedure can be executed at most fifteen times to further reduce the time complexity. Furthermore, this operator is only executed when the current solution is "promising". A solution is promising if the cost is at most 0.3% higher than the best-found solution in all GRASP iterations.

3.2 Path Relinking

Path relinking is a search intensification strategy, originally proposed by Glover [41]. It is used as an enhancement to heuristic search methods for solving combinatorial optimization problems. Its use in metaheuristic frameworks has led to significant improvements in both solution quality and running times [42].

During path relinking, we explore trajectories connecting high-quality solutions with each other. Path relinking starts with two solutions, the initial solution and the guiding solution. The idea is to create a path from the initial to the guiding solution using a set of operations. When the operations are applied one by one over the initial solution, we obtain in each application a new solution that is more similar to the guiding solution. After the application of all operations, the initial solution becomes equal to the guiding solution. At the end of this procedure, the best-found solution on the constructed path undergoes a local search procedure as described in Section 3.1.2 and the result is returned.

Path relinking can be viewed as a search strategy that seeks to incorporate elements of the guiding solution into the initial solution. At the start of the path relinking procedure, we map all routes of the guiding solutions onto routes of the initial solution, such that each route in the guiding solution is paired with a route in the initial solution. Two routes are paired together if they are the same route, so both come from the start solution and have the same ID or when not in the start solution we pair it with the most similar route in the guiding solution that is unpaired. The similarity is computed as the number of equal trips both routes serve. Ties are broken arbitrarily. If the initial solution uses fewer routes we create empty routes with no trips and pair the new routes with an unpaired guiding route.

The idea is to make at least one route of the initial solution more similar to its paired route in the guiding solution at every step. At each iteration of path relinking, we evaluate all possible combinations of three types of operations. First, we look at trips that are non-individual in the guiding solution but individual in the initial solution, second we look at non-individual trips in the initial solution but who are individual in the guiding solution, and last we look at trips that are served in both solutions but are in the wrong route (their routes are not mapped to each other).

An operation of the first type exists of inserting an individual trip of the initial solution into the route that is mapped to the route that contains the trip in the guiding solution. An operation of the second type removes a trip from the route in the initial solution that is individual in the guiding solution. The third operation moves a trip that is in the wrong route in the initial solution to the correct route, which is the route in the initial solution that is mapped to the route in the guiding solution that contains this trip. We evaluate all possible operations and select the operation that results in the lowest cost delta of the objective function, ensuring a greedy approach to transition from the initial solution to the guiding solution.

If no feasible operation is possible any more, and there are still trips being served by the incorrect vehicle, we have encountered a deadlock. In such cases, the path relinking procedure would be terminated prematurely.

3.2.1 GRASP + Path Relinking

In the previous sections, we described the basic GRASP procedure, which explores the solution space through a series of independent searches. Each search begins with a unique greedy randomized solution, and no information is shared between the searches, making them all independent. As a result, GRASP is considered a memoryless metaheuristic. In contrast, other successful metaheuristics, such as tabu search, genetic algorithms, and ant colony optimization, make extensive use of information gathered during the search process to guide their search in the solution space.

Path relinking adds a long-term memory to GRASP, by including an elite set of "diverse", high-quality solutions that are found during the search. At each iteration, the path relinking procedure is applied between the solution found at the end of the local search phase and a randomly selected solution in the elite set. The result is a candidate solution to enter the elite set.

The elite set is of fixed length. In the beginning, all solutions can enter the elite set until the limit is reached. When the limit is reached a solution can only enter the elite set if it is better than at least one solution currently in the elite set. When it is better than more than one solution the symmetric difference between these solutions and the current solution is calculated. The solution with the highest similarity is removed from the elite set and the current solution enters the elite set. The symmetric difference between two solutions is defined as the number of trips that are served by the same route. This is calculated by first mapping all routes of one solution onto the other, as explained in the previous section, and then calculating the trips that are served in both routes of the mapping. When a trip is individual in both solutions this counts as served by the same route.

Different types of strategies exist for choosing the initial and guiding solution. Forward path relinking uses the solution from the elite set as guiding solution and the result from the local search as initial solution; backward path relinking is the other way around. We decided to implement backward path relinking because the possibility of a deadlock does not guarantee that we reach the guiding solution. Now we always start with improving an already good solution, hoping to enter a nearby region with a different (lower) local optimum.

3.2.2 GRASP + Evolutionary Path Relinking

At the end, when the GRASP with path relinking procedure terminates, we have an elite set of high-quality solutions. Evolutionary path relinking [43] implements a post-optimization step by applying path relinking on all pairs of elite set solutions, to search for new high-quality solutions and to improve the quality of the final result. We found that for almost all instances the best solution was found during the post-optimization step of evolutionary path relinking.

We decided to apply the evolutionary path relinking step not only as a post-optimization step but also after every ten GRASP iterations. Thus, after ten GRASP iterations, we perform path relinking among all solutions in the elite set and add the best found to the elite set. This further improved the final solution across all instances. Empirical tests revealed that an elite set of length three yielded the highest

quality solutions. By doing so, each application of evolutionary path relinking considers six combinations of solutions. This improved the quality while maintaining manageable additional computation time.

Chapter 4

GRASP vs Simulated Annealing

In this chapter, we compare our GRASP method against a proven simulated annealing (SA) method. Since GRASP is not commonly used in the field of dynamic rescheduling, we analyze its performance to identify the advantages and disadvantages of applying GRASP in this domain. We selected SA as the comparison method because it is a well-known and widely used method which often produces very good solutions for varying optimization problems.

The remainder of this chapter is organized as follows. In Section 4.1, we describe the test instances and the parameters used. In Section 4.2, we describe the SA procedure used for the purpose of comparing it against GRASP. Then we perform three experiments to compare both methods to each other.

In Section 4.3, we demonstrate that GRASP quickly finds high-quality solutions, while SA seems to have a minimum iteration threshold. When the number of iterations falls below this threshold, SA fails to reach high-quality solutions.

In Section 4.4 we establish that our implementation of GRASP requires a start solution of reasonable quality to attain high-quality solutions, while SA appears to be independent of the start solution. Frequently replanning on the same day reduces GRASP's reliance on the start solution.

Furthermore, in Section 4.5 we present results that reveal that GRASP is better at real-time frequent replanning than SA. Finally, Section 4.6 concludes this chapter.

4.1 Instances

We use test instances that are based on real-world scenarios of Valys. A case generator created by CQM provides us with the trips, a set of changes consisting of cancellations and new trips, and a day-ahead planning generated by the static algorithm that is currently in use by Valys. This static algorithm was created by CQM and uses SA.

The region of the sub-transporters is divided into postal codes such that each region corresponds to a specific area in the Netherlands, excluding the Frisian islands. All regions are disjoint.

All driving times between any two locations are real driving times based on maps from OpenStreetMap. The calculation between thousands of locations takes only seconds using CQMaps.

Furthermore, we have two types of vehicles: a) normal cars with a capacity of 4 passengers and b) specialized vehicles that can carry up to 6 passengers, including a maximum of 2 wheelchair passengers. The service time for a trip is 7 minutes per wheelchair passenger and 4 minutes for non-wheelchair passengers, independent of the number of passengers. Arrival guarantee trips have a time window of 30 minutes before the desired arrival time until the desired arrival time. The other trips

have a time window of 10 minutes before to 10 minutes after the desired pickup time. The maximum driving time is calculated using $\lambda = 0.5$.

A driver can't work for more than 5.5 hours consecutively without a break. Breaks last for 30 minutes, and no break is allowed in the first 2 hours after a break or after the start of the route. The maximum duration of a route is 9 hours, including breaks. Changes can only be submitted after 6 a.m, furthermore, we assume that all cancellations and new trips are known at least one hour in advance. For the exact parameter settings, we also refer to Appendix A.

All instances have a name as follows: (x, y, z). This indicates that the instance starts with x trips, that it has y cancellations, and z new trips.

All experiments were made in an Intel(R) Xeon(R) CPU E5-2690 v3, 2.60GHz, 12 cores, with 128 GB of memory, using Windows Server 2012 R2 Standard. The source code was implemented in C++. We ran 8 experiments in parallel on this hardware, to reduce calculation time from months to days.

4.2 Simulated Annealing

To compare our proposed GRASP method against the proven SA method [44], we begin SA with the baseline benchmark (see Section 5.1.1) version of the initial solution and changes. Then, we apply SA to this solution.

We have developed an SA with identical neighbourhoods as GRASP (see Sections 3.1.2.1 - 3.1.2.5). "Move Sequence" was not used as it served as a diversification strategy for GRASP to escape local optima. SA already has its own diversification strategy and rarely gets trapped in local optima. Hence, the Move Sequence neighbourhood is unnecessary and would consume additional time with minimal benefits for SA.

As discussed in Section 3.1.2, different variants of variable neighbourhood descent (VND) exist. One commonly used variant for SA is the union VND, where neighbourhoods are selected without a specific order, and the next neighbourhood is chosen randomly. The probabilities for selecting the five neighbourhoods are adaptively tuned during the process. The Move neighbourhood always maintains the same probability, while the probabilities for Swap and Move-Remove slowly increase, and the probabilities for Remove and Combine slowly decrease. The initial probabilities were set to ensure an equal number of successful operations in all neighbourhoods. For Example, Remove has a higher chance of being feasible than Move, and thus the initial probability of Remove is lower than that of Move.

Lowering the probability for Combine ensures that not all individual trips are combined into routes of two, as the other neighbourhoods require individual trips for their operations. Furthermore, routes containing more trips often yield greater benefits. The probability of Remove is decreased because initially, we want to explore quickly, but eventually, it becomes more beneficial to have as many trips as possible in routes. The probability for Move remains the same as it is one of the core neighbourhoods responsible for placing trips into routes. The probability of Swap is increased since it becomes more advantageous later in the process when more trips are already in routes, and we aim to retain as many trips in routes as possible. At last, Move-Remove also receives an increased probability over time since it combines aspects of the Move while also removing a trip, allowing the removal of trips in incorrect locations while maintaining an equal number of trips in routes.

Unlike GRASP where we explore the entire neighbourhood, SA randomly selects a single operation from the neighbourhood and checks feasibility. If the operation

is infeasible we repeat the process of selecting a neighbourhood, followed by an operation. If the operation is feasible and results in an improvement, we execute the operation and this process is repeated. For feasible operations with an increase in cost, we execute the operation with a probability determined by the temperature.

To determine the temperature, we use the cooling schedule and start/end temperature of Valys's static algorithm. the cooling schedule linearly decreases the temperature dependent on the ratio between the number of iterations executed and the target number of iterations.

The proposed SA performs on average 1.64 million iterations per second. When running our SA and Valys's static algorithm an equal (large) number of iterations, the solution quality between the two differs by approximately 0.3%, indicating that Valys's static algorithm is a bit better. The difference can likely be attributed to the different insertion and removal procedures Valys's static algorithm uses. This procedure calculates the best possible order of pickup and dropoffs in a route.

4.3 Comparative Results

In this section, we examine the objective reached when given a limited amount of computation time, such that we can determine how effective GRASP and SA are within the given (limited) computation time.

Setup Twelve o'clock in the afternoon is taken as the rescheduling moment, and all changes known before twelve o'clock will be taken into account. We set $\tau = 15$, meaning all trips taking place until 12:15 p.m. are locked.

Subsequently, we ran both the SA algorithm and our GRASP algorithm multiple times with these parameters and different iterations and computation times for SA and GRASP, respectively. The maximum computation time is approximately ten minutes, which is already considered quite long for dynamic problems.

For SA, we test 29 settings with a different amount of iterations ranging from 100,000 to 1,200,000,000, which took 0.3 to 800 seconds to complete respectively. Each of these settings is repeated ten times with varying random seeds, to get a more accurate estimate of the results reached with each setting. Later, we map the iterations of SA to seconds by taking the average time it took to perform the number of iterations for each setting.

For GRASP, we adopt a similar approach. We conduct tests on 22 settings, with varying calculation times, ranging from 1 second to 660 seconds. Each setting is repeated ten times, with different random seeds. It is worth noting that we test SA on more settings, to ensure a more accurate mapping from iterations to seconds.

Four instances were used in this experiment: (3300, 660, 660), (3300, 660, 1320), (4015, 400, 1200), and (4015, 1200, 400).

Results Figure 4.1 presents the results for instance (3300, 660, 660), where the thick blue (GRASP) and light blue (SA) lines show the average best objective reached when the algorithm is given a specific computation time. The filled areas surrounding the mean lines show the spread of the best objective the algorithm reached in the corresponding time. The figure can roughly be divided into three regions. In region A, from 0 to 180 seconds, our GRASP outperforms SA in terms of solution quality. In region B, from 180 to 350 seconds, the solution quality achieved between the two methods seems to be comparable. At last in region C, from 350 to 650 seconds, SA performs slightly better than our GRASP.

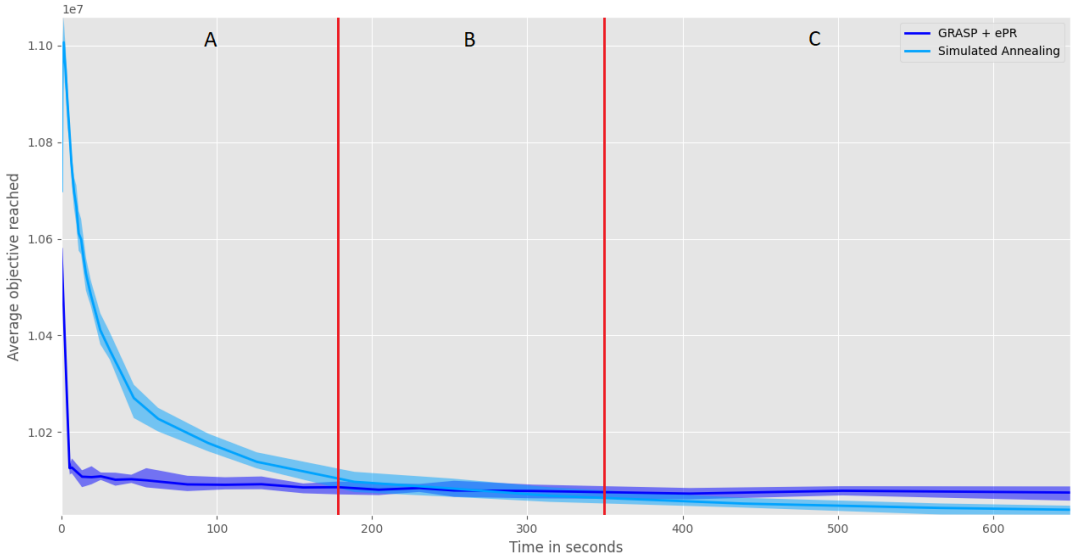


FIGURE 4.1: Allowed computation time against the average objective reached for instance $(3300, 660, 660)$.

The results of the other three instances are presented in Appendix B, demonstrating similar results, i.e. the same three regions can be observed in all figures. In region A, the differences between the methods in the instances at 100 seconds range from 0.77% to 1.83%. In region C the differences at 650 seconds are in between 0.12% and 0.56%. The interval of the regions differs per instance. For larger instances such as (4015, 400, 1200) and (4015, 2100, 400), region A is larger than for smaller instances, when typically region C is larger.

Discussion We can conclude that SA can achieve better solutions with longer computation times. However, as we are dealing with a dynamic problem we are given limited computation time. Our GRASP performs much better in the first minutes and performs similarly in the minutes after. This demonstrates the power of using GRASP on this type of problem, we have more flexibility in computation time and it is able to rapidly deliver high-quality solutions. With more computation time, our GRASP only performs slightly worse than SA.

The reason that SA is unable to reach high-quality solutions in region A could be that it needs a minimum number of iterations before being able to produce good solutions. We call this the Minimum Iteration Threshold (MIT). We expect that SA needs at least this MIT amount of iterations for thoroughly exploring the solution search space. With more iterations than the MIT, the exploration phase is long enough and SA also has enough time for exploitation, reaching high-quality solutions. Our GRASP however does not show a need for a MIT, within five seconds a high-quality solution is found. After that, it slowly improves the solution and does not realize any significant improvement from 200 to 650 seconds.

Besides, we can also notice that SA has a higher MIT for larger instances. A higher MIT for larger instances is needed to explore the larger search space better. On the other hand, our GRASP doesn't seem affected by the change in problem size. It still quickly finds a high-quality solution. This shows that GRASP is more robust against instances with varying problem sizes. This is also one of the benefits

of GRASP over SA, as the problem instances at Valys can vary from 2,000 to 5,000 trips depending on the day of the week and then the new trips aren't even included.

4.4 Impact of Start Solution

GRASP uses the start solution as a building block to build further upon, while SA is probably largely unaffected by the start solution since the high temperature in the early stages of the algorithm completely destroys this solution.

To gain insight into the impact of the quality of the start solution, we look at two scenarios: 1) replanning only once on the day, indicating how well GRASP can handle varying quality start solutions, and 2) replanning frequently during the day. The first scenario indicates how well GRASP can handle varying quality start solutions. The second scenario tells us if GRASP can smooth out the effect of the quality of the start solution when replanning multiple times or if errors accumulate due to the dependency on the start solution.

4.4.1 Replanning Once

In this experiment, we look at varying quality start solutions with a single run reschedule of GRASP and SA. The results demonstrate a dependence of GRASP on the start solution, while SA seems to be independent.

Setup The basis of this experiment is the same as the experiment explained in Section 4.3. However, we create 4 new instances per instance that were used in the previous section. These new instances vary the quality of the start solution by using Valys's static algorithm with different numbers of iterations: 1 million, 5 million, 20 million, and 100 million. Normally, the solution is created by running the static algorithm for a few billion iterations. As these iterations are lower the quality of the start solutions is worse. The worst start solution, 1 million iterations, is on average 14.5% worse than the best start solution. This leaves us with 20 instances, where we have 4 distinct instances with different trips, new trips and cancellations, and per distinct instance we have 5 instances that vary only in their start solution.

Since we still use noon as the replanning moment, the start solution quality will partly differ due to the cost in the locked part of the schedule (from 6 a.m. to 12:15 p.m.). This cost exists of the cost of routes that are already finished before twelve and of the cost in the locked parts of routes that are in progress. A worse-quality start solution has poorer-quality routes, so the cost of these parts will be higher. It is not possible to optimize this cost and the results would differ due to this cost. To prevent this we fix the schedule until 12:15 p.m. for all instances in the same distinct case and only vary the quality of the start solutions after 12:15 p.m. Resulting in an equal cost in the locked part so that comparison of the different solutions is fair.

Results Figure 4.2 presents the results for instance (3300, 660, 660). The figure illustrates the average objective reached against the allowed computation time for the varying quality start solutions. The legend provides the objective of the start solution. To enhance readability, the minimum and maximum values of the objective reached for each setting are excluded from the figure.

The figure shows that across all the different start solutions SA reaches similar results within the same computation time. This indicates that the start solution seems to have no influence on the final solution quality for SA, as may be expected.

Our GRASP algorithm on the other hand clearly shows different performances with changing quality of the start solution. Although all follow the same pattern of quickly reaching a good quality solution and then slowly improving the solution until converging (as seen in the previous section) we see differences in the solution quality where the results of our GRASP on the instances converge. Our GRASP on the start solution with objective 10.08 M never comes very close to our GRASP on the best start solution, and scores roughly 0.7% worse at any point. Interesting is that the worst objective is not reached on the instance with the worst start solution but on the second worst.

For the other start solutions, we see that our GRASP algorithm also performs similarly within the given computation time. The results are better than on the 10.08 M instance, however, the quality is still very far away from our GRASP on the best quality start solution.

In Figure 4.3, 4.4 and 4.5 we present results for the remaining three instances. Once again, the results obtained from SA are close together on the different quality start solutions for all three instances. None show any significant difference in the solution quality reached.

The results for GRASP demonstrate a different phenomenon. In Figure 4.3 and Figure 4.4 we see that there is a ranking in quality. GRASP reaches the best solutions on the instance with the best quality start solution, followed by the second best, followed by the worst and after that the middle two are performing similarly. The last instance, shown in Figure 4.5, reveals that GRASP reaches similar results on the four start solutions that were newly created and GRASP on the best start solution reaches the highest objective by far.

Discussion We can directly conclude from the four figures that SA is independent of the start solution. For all distinct instances it doesn't matter what the start solution is, SA reaches similar results at every moment for all quality start solutions. This is

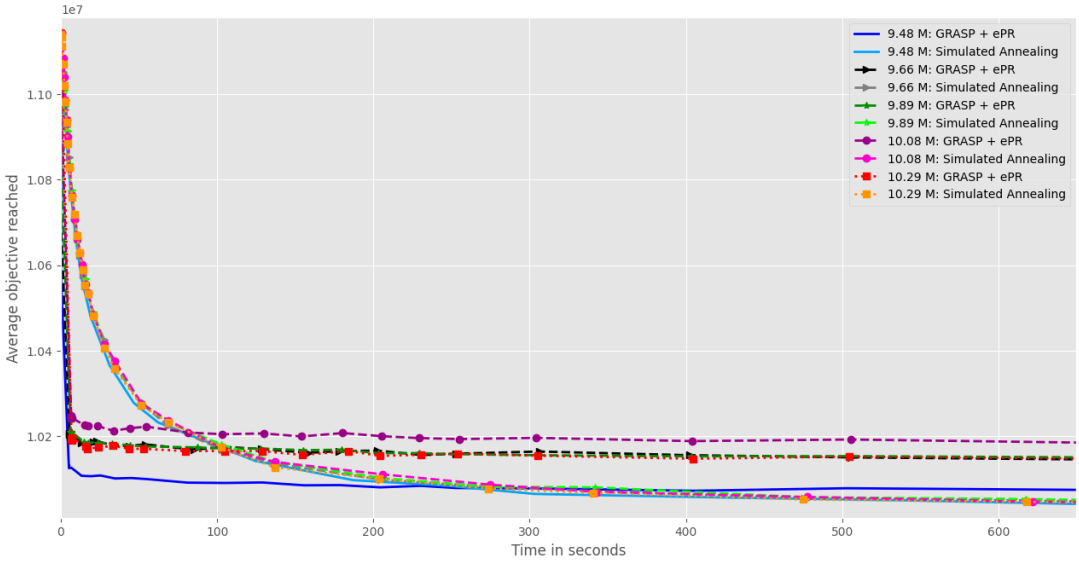


FIGURE 4.2: Computation time against average objective reached, for instance (3300, 660, 660), for different quality start solutions. The legend shows the objective of the start solution.

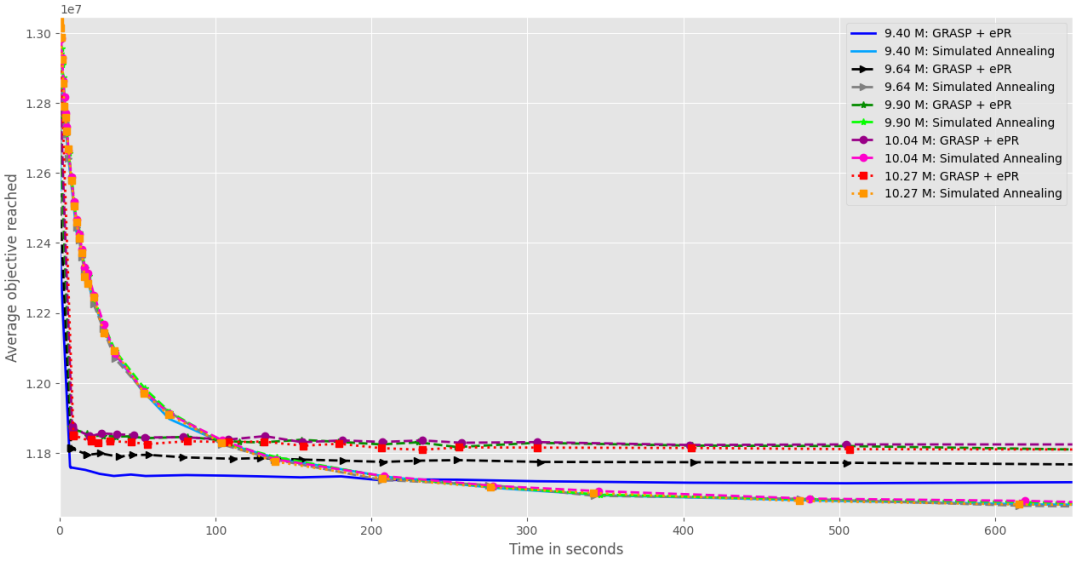


FIGURE 4.3: Computation time against average objective reached, for instance (3300, 660, 1320), for different quality start solutions. The legend shows the objective of the start solution.

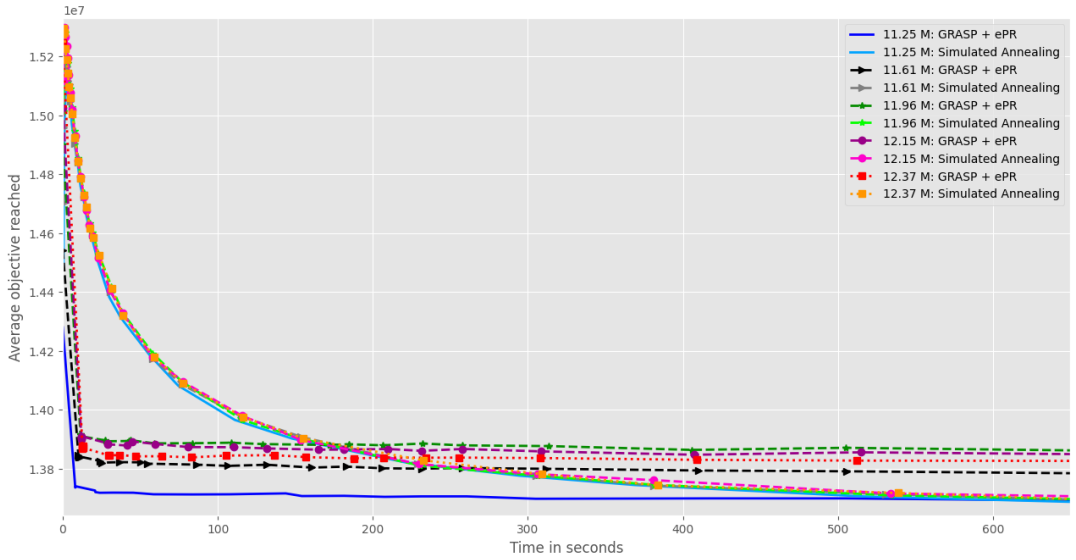


FIGURE 4.4: Computation time against average objective reached, for instance (4015, 400, 1200), for different quality start solutions. The legend shows the objective of the start solution.

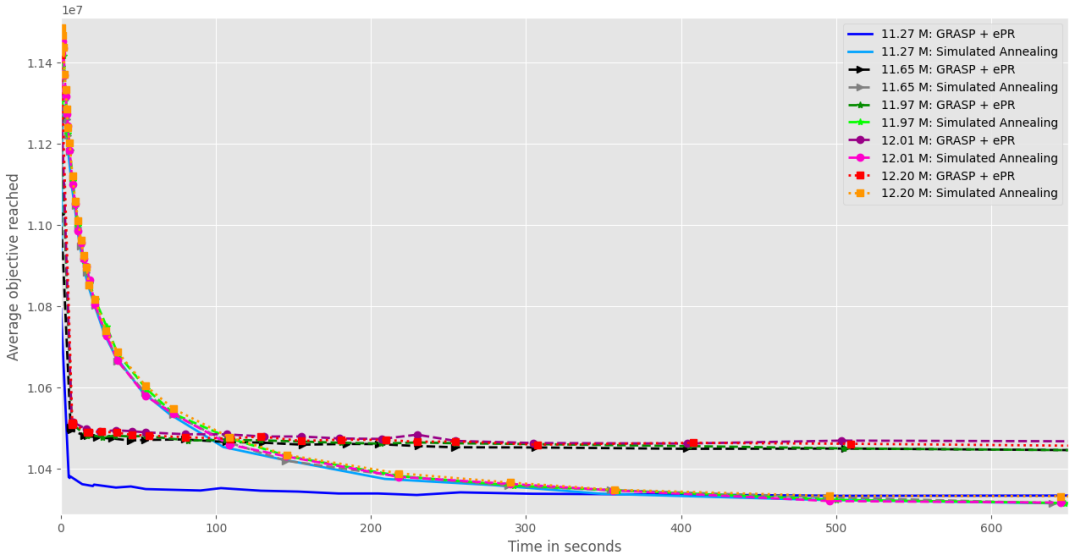


FIGURE 4.5: Computation time against average objective reached, for instance (4015, 1200, 400), for different quality start solutions. The legend shows the objective of the start solution.

due to the high-temperature SA starts with. The high temperature in the early stages of the algorithm completely destroys the start solution. In almost all cases there was some point that the solution didn't have any combined routes, only individual routes, indicating that nothing of the start solution was reused which can be seen in the figures.

GRASP on the best quality start solution reaches significantly better results than GRASP on the other start solutions. However, the dependence doesn't seem only related to the quality of the solution. As the figures do not show a clear ranking that worse start solutions always lead to worse solutions. This can be seen in figures 4.3 and 4.4, where GRASP on the middle two solutions scores poorly and GRASP on the second best and worst solution scores better. A possible reason for this could be the heavy focus on exploitation in the local search phase, where we only accept improvements. In this case, GRASP on semi-good start solutions performs poorly. Some routes in the start solutions are semi-good and some are terrible. Therefore, the local search procedure is focussing on the terrible routes, but not on the semi-good routes. These semi-good routes are suboptimal but to improve them it would take a few operations that increase the cost before decreasing it again, as that is not possible in our local search implementation the routes are left unchanged. Unlike poor start solutions, where most routes are bad and thus local search improves all routes.

The reason that in Figure 4.5 the results of GRASP are similar on the four instances with varying start solution quality is that even though the start solutions differ in quality the result of the baseline benchmark on these instances is similar. So when we remove all cancellations from their routes and when we plan new trips individually, we see that the difference in the quality of the resulting solution is similar for the four instances. As explained in Section 3.1.1 the construction phase uses this baseline benchmark as a start solution to build further upon. Hence, the results are similar as they start from similar quality solutions. This indicates that with many cancellations a really high-quality start solution is needed to reach good results, such

that the quality of the baseline benchmark remains reasonable.

A possible explanation for the necessity of a high-quality start solution for our GRASP implementation lies in the degree of destruction in the destroy phase of the construction phase. Initially, we assumed a high-quality start solution. Empirical tests revealed that the degree of destruction was best kept low when using a high-quality start solution. As GRASP could reuse routes of high-quality from the start solution. However, with a low-quality start solution, this assumption breaks down and a higher degree of destruction might be required to reconstruct larger portions of the start solution. As fewer routes in the start solution are reusable.

4.4.2 Replanning Multiple Times

In this section, we show that the dependency of GRASP on the start solution decreases when real-time frequently replanning compared to replanning once, while simulated annealing still shows no dependence. This is done by simulating an entire day for instances with varying quality start solutions.

Setup For both SA and GRASP, the replanning process starts at 6 a.m. We then start replanning every 15 minutes for five minutes until midnight. In Section 5.2 we demonstrate that this strategy yields the best results for GRASP. For our implementation of SA, five minutes of computation time roughly translates to 500 million iterations.

We examine four (new) instances with the same number of trips, new trips and cancellations as the instances in the previous sections. New instances were created to spread the changes more throughout the day to better represent a realistic scenario. We assess the impact of the start solution's quality by creating an additional four instances for each of the new "main" instances. These instances differ only in the quality of the start solution. We used Valys's static algorithm to vary the quality of the start solutions. The start solution for these instances was created using four different amounts of iterations, namely: 1 million, 5 million, 20 million, and 100 million.

We anticipate obtaining poor-quality solutions with SA due to the limited computation time of five minutes. We have seen in Section 4.3 that SA performs equal to or worse than our GRASP algorithm when given 5 minutes of computation time. Nonetheless, we can still analyse the dependence of SA on the start solution. If there is a dependency SA would show differences in solution quality when given varying quality start solutions.

Results Table 4.1 shows the objective at the end of the day for replanning with our GRASP algorithm and Table 4.2 shows the results for replanning with SA for the different quality start solutions. The column headings show the number of iterations used for creating the start solution.

We clearly see the (expected) large differences in solution quality between our GRASP and SA, where the difference in objective value can get up to 2.6%. Also notable is that our GRASP algorithm still realized better results on the worst start solution than SA on the best start solution.

Furthermore, for our GRASP algorithm, we see that the better the start solution, the better the final solution, indicating again that our GRASP algorithm is dependent on the start solution. Our algorithm's performance difference between using the best and worst start solution can go up to almost 0.92%, where on average the difference is 0.64%. Similar to the previous section we see that in three of the four instances,

our GRASP algorithm realized worse solutions using start solutions created with 5 million iterations than on the instances with start solutions created with 1 million iterations.

For SA, we observe no dependence on the start solutions. Table 4.2 shows that sometimes SA realized the best solutions on the instances created with 5 million iterations, sometimes on the one with 100 million iterations, and sometimes on the instance with the original number of iterations. Furthermore, the results of the same main instance are close together, where the difference between best and worse is in three of the four instances between 0.14% and 0.22% and on the fourth larger with 0.81%.

Discussion The large differences between SA and GRASP in the final objective shows the power of GRASP with limited computation time. SA is unable to reach a high-quality solution within the given computation time of five minutes. This can be attributed, at least in part, to SA's Minimum Iteration Threshold (MIT) phenomena. The 500 million iterations are insufficient for thorough exploration and exploitation, as discussed in Section 4.3. The remaining part can probably be explained by inefficiencies in the created solution that are stacking up in the locked parts. This will be further explained in the next section. This suggests that our GRASP algorithm is better at replanning than SA when given five minutes of computation time.

In the instance (4015, 1200, 400) when replanning once at noon for five minutes there is a difference of 1.21% between our GRASP on the worst and best start solution. However, when replanning more frequently this difference is reduced to 0.64%, indicating that our GRASP is able to reduce the effect of the start solution on the final solution when replanning frequently. Our GRASP algorithm reduces the difference from 1.01% to 0.92% on the instance (4015, 400, 1200). Our GRASP is able to reduce the difference from 0.82% to 0.20% on the instance (3300, 660, 1320). At last, our GRASP algorithm on the instance (3300, 660, 660) is the only case where the difference is equal between the worst and best start solution when replanning frequently and only once. However, even though there still is a dependency on the start solution this dependency decreases when frequently replanning.

Our GRASP algorithm is probably less dependent on the start solution when frequently replanning as the start solution matters less as the day progresses. In the first few hours, the solutions after replanning still look similar to the start solution, however after every replan it deviates further from the start solution. Hence, it has some influence at the start of the day, but as time progresses, it becomes less sensitive to the quality of the start solution.

Instance	1M	5M	20M	100M	Original
(3300, 660, 660)	9,644,967	9,647,837	9,606,751	9,592,402	9,572,712
(3300, 660, 1320)	11,461,030	11,463,600	11,458,747	11,447,412	11,437,932
(4015, 400, 1200)	13,727,478	13,754,305	13,738,286	13,609,963	13,602,158
(4015, 1200, 400)	9,456,721	9,453,262	9,445,452	9,429,332	9,396,329

TABLE 4.1: The final objective of replanning every 15 minutes on a day with GRASP. 1M, 5M, 20M and 100M stand for the number of simulated annealing iterations that were used to create the start solution. "Original" is the normal amount of iterations.

Instance	1M	5M	20M	100M	Original
(3300, 660, 660)	9,808,000	9,799,620	9,818,709	9,818,361	9,804,641
(3300, 660, 1320)	11,676,542	11,658,220	11,674,595	11,651,167	11,661,606
(4015, 400, 1200)	14,052,590	14,023,762	14,009,411	14,012,785	13,940,230
(4015, 1200, 400)	9,623,538	9,604,212	9,615,619	9,613,477	9,609,289

TABLE 4.2: The final objective of replanning every 15 minutes on a day with simulated annealing. 1M, 5M, 20M and 100M stand for the number of simulated annealing iterations that were used to create the start solution. "Original" is the normal amount of iterations.

Given that our GRASP algorithm still exhibits a dependency on the start solution, we recommend employing a good static algorithm together with GRASP to maximize its effectiveness. However, we note that the differences in quality start solutions can get up to 14.24% which is really high. A basic static algorithm should already improve upon that and would come closer to the quality of the start solutions that are created with 100 million iterations. But to really utilize the potential of GRASP a good static algorithm is recommended such as Valys's simulated annealing.

4.5 Frequent Replanning

In this section, we look into frequent replanning on the same day with GRASP and SA. We demonstrate that GRASP is better in frequent replanning than SA and give insight into the reason.

We wanted to explain the gap between a lower bound created by the perfect foresight benchmark (explained in Section 5.1.2) and our proposed GRASP algorithm (Chapter 5) by obtaining a tighter lower bound. We created an experiment where we started replanning every 2 minutes for 20 minutes with SA. The communication time and official computation time were set to 0 minutes ($\tau = 0$), such that SA could create the best possible schedule with the new changes while keeping the locked part into account, something that the perfect foresight benchmark couldn't do.

The results of this experiment were however unexpected. SA showed worse results than GRASP which was only allowed to replan every 15 minutes with a computation time of 5 minutes. Furthermore, the results showed no direct explanation for why this happened. Therefore, we wanted to research what caused this difference. Because if it had something to do with repeated replanning, SA might not be the best candidate for dynamic problems.

Setup We run both SA and GRASP on 5 instances independent of each other. Both replan every 15 minutes, starting at 6 a.m. until midnight. GRASP is allowed a computation time of 5 minutes, therefore $\tau = 10$. SA is allowed to run 2 billion iterations, which translates to roughly 20 minutes. The official computation time of SA is set at 0 minutes, so $\tau = 5$. We want to make our SA partly independent of the implementation and give a lower bound. By setting it at 0, we act as if we can do 2 billion iterations instantly.

Besides running these two methods on the same instances we also run the other method every 15 minutes in parallel on the same input as the main method. Figure 4.6 shows the experiment graphically, where GRASP is used as the main method and

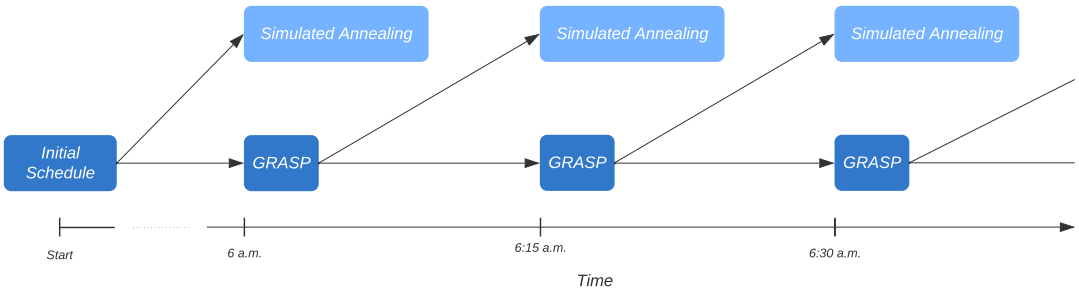


FIGURE 4.6: The experiment when we take GRASP as the main method and reschedule every 15 minutes.

SA is run on the side. Here GRASP produces a solution every 15 minutes that will be used as a start solution for the next run. We also run SA on this solution with the same input as GRASP but do not use the generated solution as input for the next. This is done for both GRASP as the main method with SA on the side and for SA as the main method with GRASP on the side.

By doing this we can compare GRASP with SA on the separate instances created every 15 minutes. This allows us to assess how SA and GRASP perform on instances created by GRASP and on instances generated by SA. These instances are similar to those used in the experiment described in Section 4.3, although having fewer changes. Additionally, we can compare the results between the two methods as the main method, determining which method achieves better results in different replanning periods and produces better final results.

Note that the experiment was only executed once per instance due to the long computation times.

Results Figure 4.7 shows the results for two of the five instances. Figures 4.7a and 4.7b display the outcomes of frequent real-time replanning. The objective value at each replanning moment is the objective value over the entire day, therefore including all finished routes, locked routes and all changes known thus far. The first figure clearly shows the differences between the two methods. However, in the second figure, the differences are not as apparent due to the larger scale. The objective increases significantly due to the type of changes, more new trips are added to the solution than trips are cancelled. For this reason, we have created Figure 4.7c, which depicts the deviation from the average solution quality at every replanning moment, providing a clearer view of the results. The results of the other three instances and their supplementary figures can be found in Appendix C.2. Exact results are available in Appendix C.1.

What we see is that for all five instances, GRASP as the main method has a better solution at the end of the day compared to SA as the main method, with a difference in objective ranging from 0.39% to 1.16%. When we look at the results produced every 15 minutes by GRASP and SA both as the main method we see that GRASP produces better results in the first few reschedules, then SA produces better results for a few hours, and after that starting from approximately 1 p.m. our GRASP algorithm consistently produces better solutions.

Moreover, in all instances, we see that our GRASP algorithm produces better results in the first replanning period at 6 a.m. (especially visible in Figure C.1). Because changes become known after 6 a.m. this is just an optimization of the start solution.

Furthermore, when we look at the results on the separate replanning moments between the main method and the side method (between the two solid lines and between the two dotted lines in Figure 4.7), we see that in all instances, independent of the main method, our GRASP is better in the first few replanning moments and the methods produce similar results in the last few replanning moments. For the middle part, it depends if we look at our GRASP as the main method or SA as the main method. When taking our GRASP as the main method we see that SA produces better results than our GRASP on the solutions reached every 15 minutes. However, when we take SA as the main method, the best method alternates between our GRASP and SA in the middle part. But no significant performance differences can be found between the two methods.

Discussion At start time 6 a.m., our GRASP consistently generates better solutions than the start solution as SA fails to produce locally optimal solutions. By applying GRASP to the non-locally optimal solution created by SA, we can hill climb to the nearest local optimum, thereby yielding better results.

That is why our GRASP consistently produces better results in the first few reschedules, due to its extensive use of the high-quality start solution. On the other hand, SA recreates the entire schedule at each replanning moment, mostly disregarding the use of the high-quality start solution that was created with over three billion iterations. In the first few replanning moments, we often see that SA's best solution found is the baseline benchmark version of the start solution, therefore it is not completely disregarding the start solution, but it is not improving upon it.

In the final rescheduling moments, our GRASP and SA generate similar solutions since there are only a few trips that do not fall in the locked part. Because of this limited amount of trips, the search space is relatively small and both methods are able to produce good results.

With our GRASP as the main method and SA as the side method, we observed that SA generally performed better than our GRASP in the middle of the day. This is because the instances are still large enough, the influence of the start solution becomes less significant and as we have seen in Section 4.3 SA realizes better results than our GRASP with longer computation times. Consequently, SA produced superior results.

Conversely, when SA serves as the main method and GRASP as the secondary method, the two methods do not show significant differences. This can be attributed to the fact that SA fails to produce locally optimal solutions, as observed earlier. Consequently, every 15 minutes GRASP seizes the opportunity for dual optimization. Firstly, it enhances the previous solution generated by SA, making it locally optimal. Something we have seen GRASP do really well in the first replanning moment, at 6 a.m. Secondly, GRASP optimizes the changes themselves. This allows GRASP to achieve substantial improvements in solution quality, while SA is again constructing an entirely new solution to reach a high-quality, non-locally optimal solution.

At last, the superiority of GRASP as the main method over SA as the main method in replanning can be attributed to the local vs. global search characteristics of GRASP and SA, respectively. A near-globally optimal solution tends to leave minimal room for changes, as everything is planned so tightly that almost nothing can fill the gap when a trip is cancelled. In contrast, searching locally provides more

flexibility for changes, allowing remaining trips to fill the gaps left by cancellations and creating routes that can easily be extended with the arrival of new trips. GRASP produces more locally optimal solutions and therefore is better at handling changes compared to the globally-focused SA.

Another reason for the superiority of our GRASP as the main method over SA as the main method lies in the non-locally optimal solutions produced by SA. It is very likely that inefficiencies are present when a solution is not locally optimal. Since we replan every 15 minutes the locked part also shifts forwards every 15 minutes. Inefficiencies occurring precisely within these 15-minute intervals remain locked until the end. With a total of 72 replannings on a day, some inefficiencies probably occur directly after the locked part and as a result, SA accumulates inefficiencies in its locked part, resulting in a lower-quality solution.

4.6 Conclusion

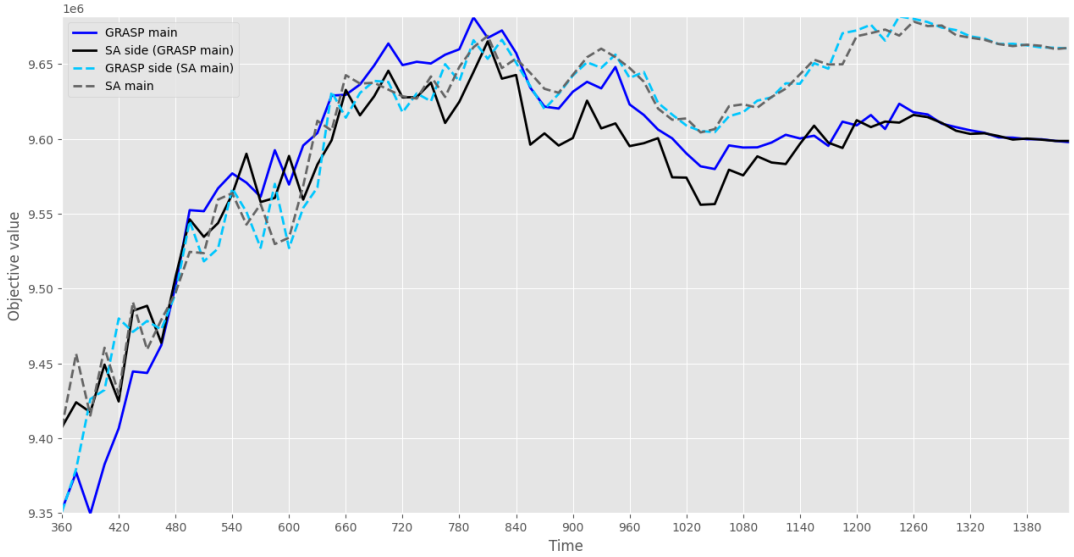
We compared the performance of GRASP with SA to assess how well GRASP performs on this dynamic problem compared to a proven method. We evaluated both methods in three experiments that examined different aspects of the algorithms: computation time versus the objective reached, the influence of the start solution and the ability to replan frequently during the day.

GRASP finds high-quality solutions quickly, while SA appears to have a minimum iteration threshold. When the number of iterations falls below this threshold, SA fails to reach high-quality solutions. On the other hand, with more iterations than the threshold, SA performs better. However, considering the variation in this threshold across different problem sizes and the relatively small difference in quality between GRASP and SA when given additional computation time, GRASP proves to be a favourable choice. This is especially true when considering the limited computation time typically available for dynamic algorithms.

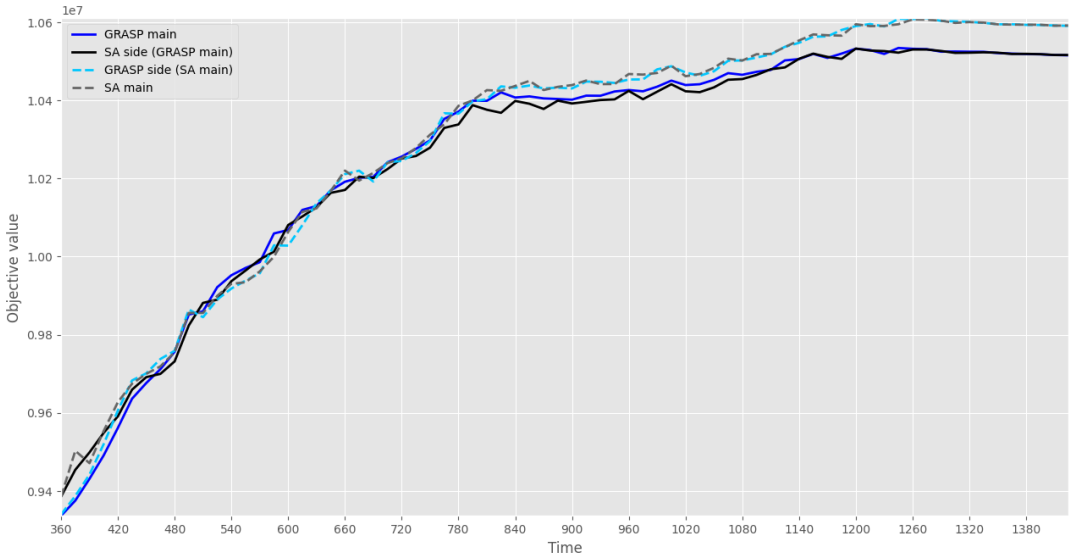
Additionally, our GRASP implementation requires a reasonable start solution to achieve high-quality solutions, while SA shows independence from the start solution. Increasing the degree of destruction in the destroy phase of the construction phase could improve the quality on instances with poor start solutions. When replanning every 15 minutes on a day, the dependence on the start solution decreases. In one instance, it reduced the difference between GRASP on the worst start solution and on the best start solution from 0.82% to 0.20%. To fully leverage the potential of GRASP, we recommend using a good static algorithm to generate the start solutions.

Furthermore, our results demonstrated that GRASP outperforms SA in real-time frequent replanning. Despite the significantly longer computation time given to SA, GRASP consistently produces better results, with differences up to 1.16%. This can be attributed to GRASP and SA's local vs global search characteristics, respectively. A near-globally optimal solution leaves little room for changes, while a locally optimal solution allows for adjustments. Additionally, SA fails to produce locally optimal results, indicating potential inefficiencies. These inefficiencies may accumulate in the locked part during replanning, resulting in lower-quality solutions.

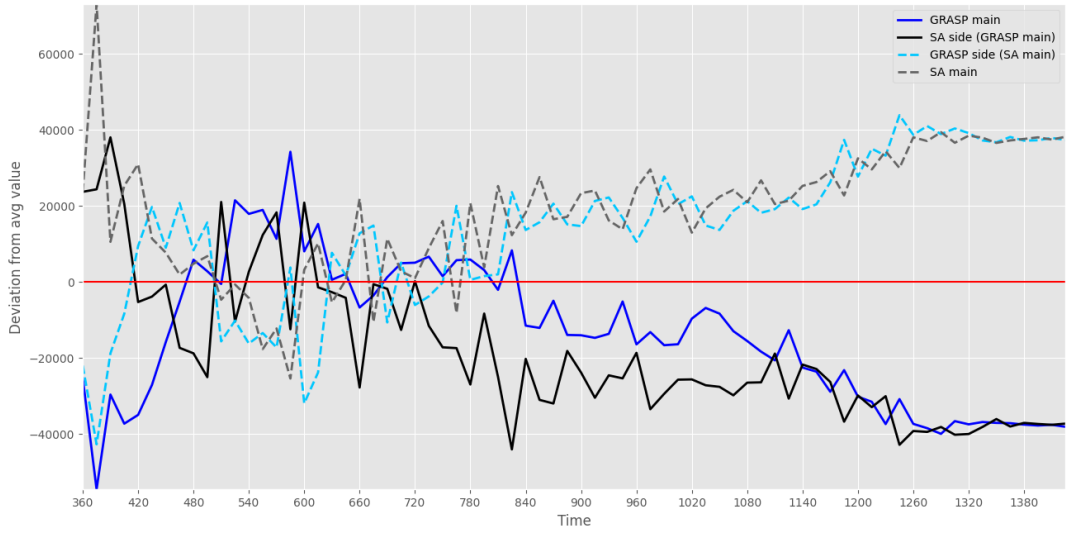
In conclusion, GRASP proves to be a strong choice for this problem resembling a DDARP. It effectively and efficiently reuses parts of previous solutions to quickly generate high-quality results. Moreover, GRASP excels in frequent replanning on the same day, which is often desirable in dynamic problems. To maximize the benefits of GRASP, starting with a high-quality solution is advisable.



(A) Instance (3300, 660, 660).



(B) Instance (3300, 660, 990).



(C) Relative improvement of instance (3300, 660, 990). The figure shows the deviation from the average solution quality at every replanning moment.

FIGURE 4.7: Frequent real-time replanning for GRASP as main and SA as side and the other way around.

Chapter 5

Comparative Results of the Opportunity Gap

In this chapter, we present an analysis of the opportunity gap, which refers to the potential benefit of rescheduling as compared to not rescheduling. Failing to take advantage of dynamic rescheduling can lead to inefficiencies in the transportation system, resulting in longer waiting times for passengers, higher operational costs for the fleet, and increased environmental impact.

We will focus on the opportunity gap of the DDARP in the context of Valys and discuss how it can be effectively managed through our proposed GRASP algorithm. We will compare our algorithm to different benchmarks, including a different approach that continuously optimizes the schedule, and highlight the benefits and challenges associated with our method. Overall, this chapter aims to provide a comprehensive understanding of the opportunity gap in the DDARP and to present a practical solution for its effective exploitation.

We found that when real-time rescheduling with GRASP every 15 minutes for a duration of 5 minutes yields the best results across all instances. Furthermore, we will demonstrate that the opportunity gap is predictable when knowing the total number of changes, where the improvement can go up to 17% for instances with 2640 changes.

The remainder of this chapter is organized as follows. In Section 5.1 we present three benchmarks that we use to compare with. In Section 5.2, we analyse the different rescheduling strategies we can use with GRASP. Finally, in Section 5.3 we present and discuss the results of the opportunity gap for different approaches.

5.1 Benchmarks

To evaluate the performance of our proposed algorithm we compare it against several benchmarks which we will explain in this section. To evaluate the quality of our GRASP method, we need a simple benchmark that creates an upper bound on the solution quality. This benchmark should incorporate the changes in the simplest way possible, doing as little as reasonably possible. This baseline benchmark is defined in Section 5.1.1. Section 5.1.2 defines a benchmark that will act as a lower bound by using Valys's static algorithm. At last, in Section 5.1.3 we propose a method to solve the DDARP continuously, such that we can compare our GRASP that uses a periodic approach against an approach that continuously optimizes the schedule.

5.1.1 Baseline Benchmark

The baseline benchmark sticks to the original plan as much as reasonably possible. It represents the strategy of not optimizing changes. We can compare our algorithm against this benchmark to get an idea of how much our algorithm improves upon not optimizing the changes. In the end, this benchmark can be interpreted as an upper bound on the solution quality. The opportunity gap is defined by this benchmark as it represents the performance when no action is taken in response to the changes. It reveals to which extent improvements can be made to the schedule when taking the changes into account.

To modify a schedule to the baseline schedule, the following operations should be applied:

- All new trips should be scheduled as individual trips.
- A cancelled trip that was planned as an individual trip should be removed from the solution.
- In the case of a cancelled trip where the pickup location is the first location of a route, the trip should be removed. The driver should then proceed directly to the (new) first pickup location, even if it is outside its designated region.
- If both the pickup and dropoff locations of the cancelled trip are not the first and not the last location, the trip should be removed from the route, as defined in Section 2.3.2
- When the dropoff location of the cancelled trip is the last location of a route, the trip should be removed from the route. The driver should then drive directly from the (new) last dropoff location to the closest point within the transporters' region if they are not already in their region. If the pickup was not directly before the dropoff, the pickup should be handled the same as explained above.

It is possible that a route becomes infeasible after removing a trip, see Section 2.3.2. In this case, the entire route is split and all trips in it will be driven individually. However, trips that are locked will remain in the route, as these are already finished or in progress.

5.1.2 Perfect Knowledge Benchmark

The perfect knowledge benchmark serves as a lower bound for the dynamic algorithm. This benchmark has complete information about the day's cancellations and new trips at the start of the day, enabling it to create an optimal schedule without the constraints of vehicles already en route. To calculate the final schedule, we use Valys's static algorithm, which has been shown through numerous empirical tests to perform near-optimally.

However, since the perfect knowledge benchmark has access to information that a dynamic algorithm does not, we expect that there will be a gap between the dynamic algorithm's performance and the benchmark's near-optimal schedule. As this benchmark is not limited by the information given at the time, whereas, dynamic algorithms are unable to predict changes in the future. Consequently, we expect that part of the gap between the dynamic algorithm and this benchmark will never be closed. Nevertheless, the benchmark provides a useful reference point for evaluating the dynamic algorithm's performance.

5.1.3 Continuous Approach Benchmark

In the continuous approach, changes to the routes are updated in real-time and sent back in a matter of seconds. The communication time in this approach is almost always negligible and is often assumed to be zero. Because of the quick response time, often-used methods include quick insertion heuristics and sometimes post-insertion optimization. It has as benefit that all changes can be incorporated into the schedule.

The current schedule is constantly kept in memory to react as fast as possible to the changes. The key idea is, that we always want to have a schedule that is locally optimal. At the start of the day, we have the probably near-optimal schedule created a day before by the static algorithm. After making small changes to incorporate a new trip or cancellation, we will probably enter a local optimum close to this near-optimal solution. These local optima are often also of high quality. However, after many small changes, there will be some accumulation of inefficiencies made by taking these myopic steps, and the dynamic schedule might get further and further away from a near-optimal schedule. However, because we start with a high-quality solution we expect not to deviate too far from this solution and thus still reach a good solution.

Every time a change arrives, we redetermine the locked trips for all routes, such that at any time we have an up-to-date schedule that is currently in operation. Then the action taken depends on the type of change: a cancellation or a new trip, described in Procedure 5.1 and Procedure 5.2, respectively.

Procedure 5.1 (Cancellation) We remove the cancelled trip from its route. If the cancelled trip is an individual trip we simply remove the trip from the solution. When the cancelled trip is not an individual trip, we start a small optimization step on the route that will be introduced in Procedure 5.4.

When removing the trip leads to an infeasible route we make all trips in the route individual. For each of the trips, we apply the optimization explained in Procedure 5.3. If the route contained locked trips the route is added again with only the locked trips in it and an optimization step is started on the route, as explained in Procedure 5.4.

Procedure 5.2 (New trip) New trips are added to the solution as individual trips. Afterwards, we are going to try to insert the trip into a route by applying the optimization as explained in Procedure 5.3.

Reoptimization The reoptimization step after each new change ensures that at all times we will stay in a local minimum. Because only a single change is considered at a time, it often takes only a few operations to reach a new local minimum. The optimization procedures only accept improvements to the current schedule.

Procedure 5.3 (Trip reoptimization) Trip reoptimization is applicable only to individual trips. It involves three neighbourhoods for consideration: moving a trip into a route, performing a move-remove operation with a route and exploring the possibility of combining the trip with another individual trip. All are explained in Section 3.1.2, named Move, Move-Remove and Combine, respectively.

In the order of the mentioned neighbourhoods, we systematically explore all potential operations within each neighbourhood. The operation that yields the greatest cost reduction is selected and executed. E.g. firstly, we examine all possible moves into a route. If a feasible move with a negative cost delta is identified, we execute

the move with the highest cost reduction. If no feasible move with a negative cost delta is found, we proceed to the move-remove neighbourhood, and so on.

If none of the operations are possible, the procedure ends. However, if an operation is executed, we initiate the route reoptimization procedure explained in Procedure 5.4 on the route where the new trip is inserted in.

Procedure 5.4 (Route reoptimization) Route reoptimization consists of three neighbourhoods, two of which are similar to those in trip reoptimization but from the other perspective: moving an individual trip into the route, performing a move-remove operation with an individual trip and swapping a trip in the route with a trip in another route. All are explained in Section 3.1.2, named Move, Move-Remove and Swap, respectively.

The neighbourhoods are explored the same as explained in Procedure 5.3. If a successful operation is performed, the route reoptimization function is recursively called on the route. In the case of applying the swap operator, this route reoptimization is also performed on the other route involved.

5.2 GRASP Rescheduling Strategy

Our GRASP algorithm uses the periodic approach, where the changes are buffered until a predefined time point when a reoptimization is initiated on all buffered changes and the current schedule. A periodic approach requires a rescheduling strategy to determine how often, how long, and when to reschedule. We will look at the realized average quality of the solutions at the end of the day to determine the best rescheduling strategy

Setup We looked into four different rescheduling strategies for GRASP: every 2 hours for 10 minutes, every hour for 10 minutes, every 30 minutes for 10 minutes and every 15 minutes for 5 minutes. We evaluated all four strategies on four different instances. We start rescheduling at 6 a.m. and continue until midnight. The communication time was set at 5 minutes for all strategies, meaning that $\tau = 15$ for the first three strategies and $\tau = 10$ for the last strategy.

For the experiment, we evaluated every strategy on the four instances, as well as the baseline benchmark (Section 5.1.1) and the continuous approach benchmark (Section 5.1.3). We repeated every strategy ten times and averaged the results.

Results In Table 5.1 we present the percentage improvement with the baseline benchmark for the continuous approach and the four replanning strategies. This difference with the baseline benchmark is the opportunity gap of replanning. The table reveals that replanning every 15 minutes outperforms the other strategies and the continuous approach, achieving the best results in all instances. Furthermore, it shows that in every instance the more frequently you replan the greater the improvement upon the baseline benchmark. The continuous approach is better in most instances than the 2h strategy but never better than the 1h strategy.

Discussion These results demonstrate that more frequent replanning leads to an improvement in the final objective. The best replanning strategy is to replan every 15 minutes for a duration of 5 minutes. This could be because the shorter replanning time reduces the length of the locked part, leaving more trips unlocked, which in turn increases the number of trips that can be operated on. Additionally, there is a

Instance	Continuous (%)	Replanning Interval (%)			
		2 h	1 h	30 min	15 min
(3300, 660, 660)	9.10	8.88	9.87	10.33	10.46
(3300, 330, 990)	8.84	8.90	9.97	10.22	10.45
(3300, 990, 330)	8.69	8.40	9.69	10.03	10.42
(3300, 1320, 1320)	15.21	14.00	15.69	16.29	16.76

TABLE 5.1: The results of GRASP with the different types of strategies, taken as the percentage improvement with the baseline benchmark.

higher likelihood that cancelled trips and new trips do not fall within the locked part. A cancelled trip or new trip can still be locked even though the changes are known an hour in advance, as we enlarge the locked part until the car is empty (explained in Definition 2.3). These locked changes cannot be taken into account and are handled as the baseline benchmark (explained in Section 5.1.1), leaving gaps in routes or making new trips individual, which is often more expensive than combining them with other trips.

Our experiment demonstrates that our periodic approach method is better than a continuous approach when replanning sufficiently often. This can be explained by two factors. Firstly, our method has more computation time, resulting in better solutions compared to the quick optimization steps taken by the continuous approach. Secondly, with more frequent replanning, we are able to take all changes into account, avoiding changes falling into the locked part, thus creating a better schedule. The big advantage of the continuous approach is that all changes can be included but as more frequent rescheduling also has the same effect this benefit of continuous rescheduling falls away. However, in the case of replanning every 2 hours, the continuous approach outperforms our GRASP method. This can be attributed to the number of changes that fall in the locked part when using a 2-hour rescheduling strategy.

5.3 Results & Discussion Opportunity Gap

In this section, we compare our GRASP algorithm introduced in Chapter 3, to the benchmarks described in Section 5.1. Furthermore, we reveal the opportunity of rescheduling compared to not rescheduling.

Setup We used the strategy of rescheduling every 15 minutes for a duration of 5 minutes, as was determined to be the best approach in our previous analysis in Section 5.2. Again, we rescheduled from 6 a.m. to midnight, with a communication time of 5 minutes, so $\tau = 10$. We tested our GRASP on 20 different instances with varying numbers of new trips and cancellations. We determined the relative improvement with respect to the baseline benchmark to reveal the extent to which improvements can be made.

Results Table 5.2 lists the final results in relative improvement with the baseline benchmark for our method, the continuous benchmark, the perfect foresight benchmark and the schedule created a day before by Valys's static algorithm. We can see

Instance	Day B. (%)	Perf F. (%)	Cont. (%)	GRASP (%)
(0, 330)	11.87	3.47	2.38	3.23
(0, 660)	21.33	7.25	4.87	6.30
(0, 990)	28.96	9.51	6.61	8.13
(0, 1320)	35.04	11.25	8.50	9.89
(330, 0)	-6.27	3.08	2.00	3.01
(330, 330)	6.69	6.69	4.39	5.70
(330, 660)	17.25	9.78	6.94	8.48
(330, 990)	25.66	12.31	8.84	10.45
(660, 0)	-13.96	7.33	4.37	6.14
(660, 330)	0.71	9.82	6.51	8.18
(660, 660)	12.13	12.48	9.10	10.46
(660, 990)	21.82	14.63	10.52	12.24
(660, 1320)	29.11	16.30	12.39	13.85
(990, 0)	-25.68	9.26	6.59	7.73
(990, 330)	-7.73	12.32	8.69	10.42
(990, 660)	5.83	14.63	10.78	12.50
(990, 990)	16.82	16.70	12.48	14.36
(1320, 0)	-42.29	11.62	7.76	9.45
(1320, 660)	-2.88	16.38	12.18	13.87
(1320, 1320)	19.42	19.87	15.21	16.76

TABLE 5.2: The results of the different types of approaches, taken as the relative improvement with the baseline benchmark. Day before is shortened to Day B, Continuous to Cont. and perfect foresight benchmark to Perf F. Each instance contains 3300 trips, so the instance name is shortened to only show the number of cancellations and new trips, e.g. (660, 990) stands for (3300, 660, 990).

that our GRASP procedure is better than the continuous benchmark on all instances. Even for smaller instances, the difference is already around 1%, and for larger instances this can go up to 1.88%.

The differences between GRASP and the perfect foresight benchmark range from 0.07% to 3.11%, from the smallest instance to the largest instance respectively. On average, the deviation between our method and this upper bound is 1.68%. This difference tends to be slightly higher for instances with a larger number of changes and slightly lower for instances with fewer changes.

Furthermore, the table reveals that the size of the opportunity gap is similar for instances with a similar amount of changes, e.g. we can see that with instances in which the total number of changes (both cancellations and new trips) sums up to 1320, the realized relative improvement is between 9.4% and 10.5%. Similar behaviour can be seen for the other cases, shown in Table 5.3.

Discussion The small ranges for instances with the same number of changes indicate a predictable opportunity gap between these instances. We not only see this in the relative improvement of our GRASP algorithm but also in the results of the perfect foresight benchmark. This is noteworthy as it seems that it doesn't matter

Changes	330 (%)	660 (%)	990 (%)	1320 (%)	1650 (%)	1980 (%)
Opportunity gap	$3.0 - 3.3$	$5.7 - 6.3$	$7.7 - 8.5$	$9.4 - 10.5$	$12.2 - 12.5$	$13.8 - 14.4$

TABLE 5.3: Range of the opportunity gap, as relative improvement with the baseline benchmark, for the number of changes accumulated.

what the changes themselves are, suggesting that the potential improvement is not dependent on the type of change.

Thus, when we roughly know the number of changes a paratransit agency has, we can predict the relative improvement we can make. Note that the difference between the schedule created a day before and the baseline benchmark does not follow the same pattern, meaning, we can not predict how much worse a schedule becomes when not rescheduling with a certain amount of changes. The differences between the day-before results are mostly due to the arrival of new trips or the cancellations of existing trips. The total cost increases with more trips and decreases with fewer trips.

Furthermore, our results show that there is an opportunity for dynamic rescheduling, even when the number of changes is relatively small, e.g. for the instances with only 660 changes the improvement can be 6% compared to not optimizing. This translates in the specific instance of $(3300, 330, 330)$ to a saving of 150 work hours. For instances with more changes, this can go up to 17%, which roughly translates in the specific instance of $(3300, 1320, 1320)$ to a decrease of 600 hours in which a vehicle and driver are needed.

The gap between our GRASP algorithm and the perfect foresight benchmark can be explained by the flexibility of the perfect foresight benchmark. This upper bound is not bound by real-time operational restrictions and thus can create routes that otherwise would likely be impossible to create. Therefore, some gap between our GRASP method and the perfect foresight benchmark is to be expected. Increasing the number of changes widens the gap even more, as the difference in the amount of information received between the perfect knowledge benchmark and our method is larger. Our GRASP can only optimize based on the information received in the 15 minutes prior to the optimization period, while the perfect knowledge benchmark has all information. Therefore, our GRASP makes myopic decisions when optimizing the current trips, making errors early on as information later reveals that other choices would have been better.

Nevertheless, we think that our method is effective in closing the gap between the baseline benchmark and a near-optimal schedule, as the average deviation of 1.68% is not very large compared to how much more information the benchmark receives.

Chapter 6

Conclusion

In this thesis, we presented an algorithm for the Dynamic Dial-A-Ride Problem that arises in the practical setting of Valys. We demonstrated the ability of our GRASP method to quickly generate high-quality solutions, specifically in settings relevant to real-time planning. In comparison, simulated annealing (SA) requires a minimum number of iterations before producing high-quality solutions, something we call the minimum iteration threshold (MIT). SA fails to generate a high-quality solution when using fewer iterations than the MIT, as it doesn't have enough time to explore the search space. Consequently, our GRASP method is more flexible in its computation time, generating good solutions with limited computation time and increasing the quality with more computation time.

Our GRASP method outperformed SA significantly in the initial minutes and performs similarly in the minutes after. With a longer computation time, our GRASP only performs slightly worse than SA. Furthermore, our tests revealed that the MIT of SA is larger for bigger instances, indicating that more computation time is needed to reach good solutions for larger instances. In contrast, our GRASP doesn't seem to be affected by the larger instances, suggesting that it is more robust against instances with varying problem sizes.

We analysed the influence of varying quality start solutions on the final solution quality for both our GRASP and SA when replanning once and frequently. Our findings showed that our GRASP on high-quality start solutions led to better solutions, indicating that the start solution's quality influences the final objective. On the other hand, SA seems to be independent of the start solution. Experiments show that when replanning more frequently the dependency of our GRASP method on the start solution diminishes, but never vanishes. Therefore, when using GRASP it is advised to use a good static algorithm to create the start solution.

We showed that SA performed 0.39% to 1.16% worse than GRASP in the setting of frequent replanning during the day, even though SA had four times more computation time. An explanation could be that our GRASP is focussing on local search, providing more flexibility for changes later on. SA focuses on global search, leaving minimal room for improvement when confronted with changes. Furthermore, the solutions of SA are not locally optimal and therefore inefficiencies may accumulate in the locked part, leading to lower-quality solutions.

Finally, we showed that the opportunity gap is predictable with respect to not replanning. The improvement compared to taking no action ranges from 3.0% for a few changes to 16.8% for many changes. This can translate to a 150 to 600 hours decrease in daily service time. We showed that our GRASP method is effective in closing the gap between not optimizing changes and the perfect knowledge benchmark, where all changes are assumed to be known in advance. Our GRASP method performed on average 1.68% worse than the perfect knowledge benchmark. This

gap can mostly be explained by the difference in the amount of information received between the perfect knowledge benchmark and our method. Our GRASP algorithm can only optimize based on the information received in the 15 minutes prior to the optimization period, while the perfect knowledge benchmark has all information. Therefore, our GRASP algorithm makes myopic decisions when optimizing the current trips, making errors early on as information later reveals that other choices would have been better. However, note that we do not exactly know how much of the gap is explained by the difference in information.

All in all, by implementing the proposed GRASP method, Valys can leverage dynamic rescheduling techniques to enhance its transport service. When faced with sudden changes in the schedule, our GRASP algorithm can quickly generate high-quality solutions, resulting in improved service delivery and enhanced customer satisfaction. In addition, we showed the predictability of the opportunity gap with respect to not optimizing. Consequently, Valys can estimate their potential gain based on cumulative changes. This insight allows Valys to make informed decisions and calculate the expected decrease in service time, which can range from 150 to 600 hours per day, depending on the cumulative changes.

6.1 Suggestions for Future Research

In this section, we list some promising directions for future research. The parameters τ_c (time a cancellation is at least known in advance) and τ_{nt} (time a new trip is at least known in advance) may have a significant effect on the final solution quality. Looking into this parameter and its influence on the final solution can give a trade-off between more flexibility for the transportation service users and the solution's quality. We showed that a 1-hour advance notice can close the opportunity gap quite well. This might even be reduced to only 30 minutes, giving more flexibility to the users.

Furthermore, the degree of destruction in the construction phase of GRASP and varying qualities start solutions may offer potential improvement in the quality. Better solutions could be reached when the degree of destruction is increased for low-quality start solutions.

Another promising research direction is to apply a hill climbing algorithm as a post-optimization step to simulated annealing. We showed that simulated annealing is not really suitable for dynamic problems partly because of the non-local optimal solutions simulated annealing produces. This step may provide an improvement to the solution.

Schedule Stability Motivated by the operational challenges of transporters, an innovative idea in the setting of real-time replanning is to take schedule stability into account. We have made a promising start, but in view of the time we weren't able to finish testing everything and running experiments.

Schedule stability is an important factor in practice which could improve the solution for the transporters. Conversations with Valys and CQM led to the following definition of schedule stability: "At any time and for each transporter the number of vehicles needed should be similar to the schedule created in advance." So at any moment, we should try to minimize the deviations in the number of planned vehicles per transporter.

This is important for Valys and probably other paratransit organizations as it is really hard to call-in extra drivers for the new routes that were created. Needing

fewer drivers is also bad, as these drivers still should be paid because they are already on duty. Valys mentioned that a transporter that needs two more vehicles is twice as bad as needing one more vehicle, needing three more vehicles is twice as bad as needing two more vehicles etc. They favour that the deviations for the transporters are more fairly spread out. Such that not one transporter needs 10 extra vehicles while another needs 5 fewer vehicles.

To represent the schedule stability, we came up with a binning approach, where we penalize deviations from the number of a priori scheduled vehicles for a transporter, looking at each transporter individually. We split the time horizon ($[0, T]$) up into small bins of size τ_{bin}, thereby creating T/τ_{bin} bins per transporter. Let bin b_i^a be the bin ranging from $[(i-1) \times \tau_{bin}, i \times \tau_{bin})$ for transporter $a \in A$, for $i = 0, \ldots, T/\tau_{bin}$. Each bin keeps track of the deviation in the number of vehicles of the current solution compared to the start solution. Our new problem then is to create a set of routes for vehicles subject to the constraints 2.1.1 - 2.1.8, while minimizing:

$$\sum_{k \in M} c_{y_k} d_k + \sum_{a \in A} \sum_{i=1}^{T/\tau_{bin}} (b_i^a)^2 \times \begin{cases} c^+ \text{ if } b_i^a > 0 \\ c^- \text{ if } b_i^a < 0 \\ 0 \text{ otherwise} \end{cases} \tag{6.1}$$

Where c^+ is the penalty for having more vehicles, and c^- is the penalty for having fewer vehicles.

Now the method which is used to solve the problem becomes a multi-objective problem where a trade-off has to be made between creating good routes and spreading all vehicles more fairly over the transporters. Such that if a transporter has many cancelled routes it becomes beneficial to assign new routes to that transporter.

Preliminary results showed that this helped spread all routes equally over the transporters, such that no transporter had more than 3 extra or fewer vehicles at any time, compared to the baseline benchmark where a transporter sometimes needed 18 extra vehicles. Future research could continue this research.

Appendix A

Parameters used in testing

The parameter settings used in our problem.

Parameter	Value
q_a^L	4
q_a^W	0
c_a	45
q_b^L	4
q_b^W	2
c_b	54
tw_p	10
tw_a	30
λ	0.5
τ_c	60
τ_{nt}	60
τ_d	330
τ_b	30
τ_f	120
τ_m	540
τ	10
st_i	$7 \times P_i^W + 4 \times y^*$

TABLE A.1: Parameters used for our problem. With two types of vehicles, a and b.
* where y is 1 if $P_i^L > 0$ and 0 otherwise

Appendix B

Simulated Annealing vs GRASP

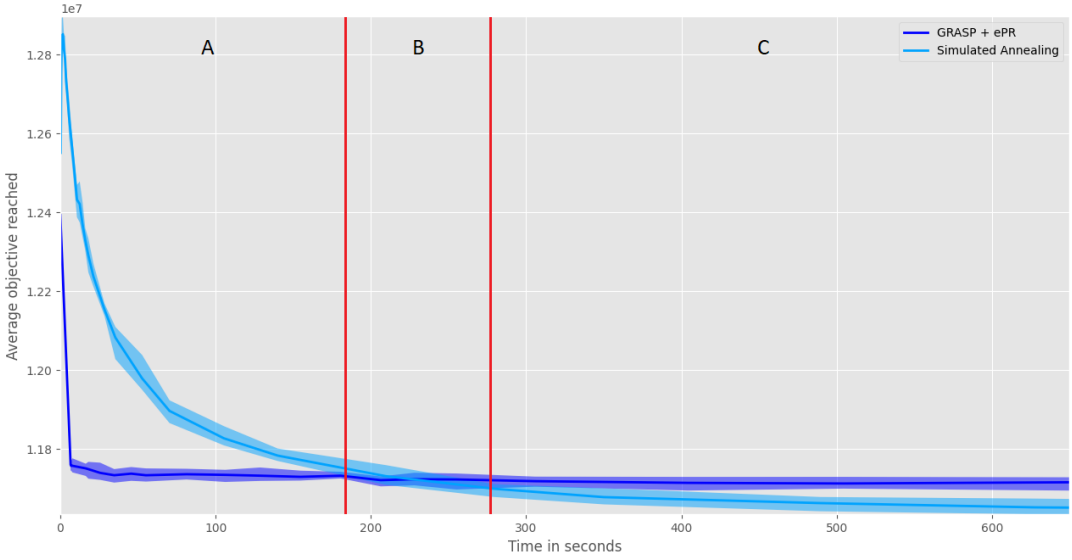


FIGURE B.1: Allowed computation time against the average objective reached for instance $(3300, 660, 1320)$.

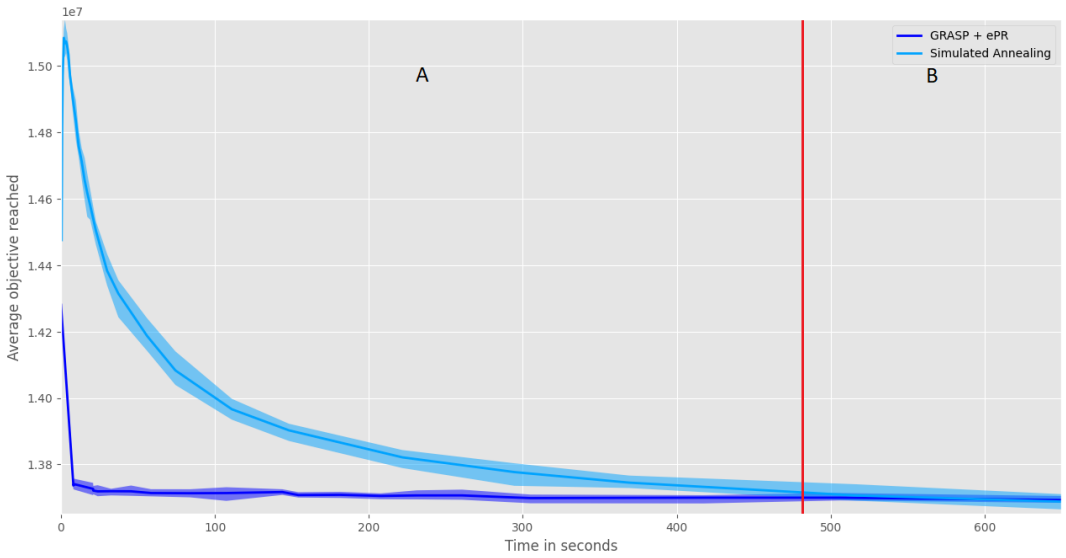


FIGURE B.2: Allowed computation time against the average objective reached for instance $(4015, 400, 1200)$.

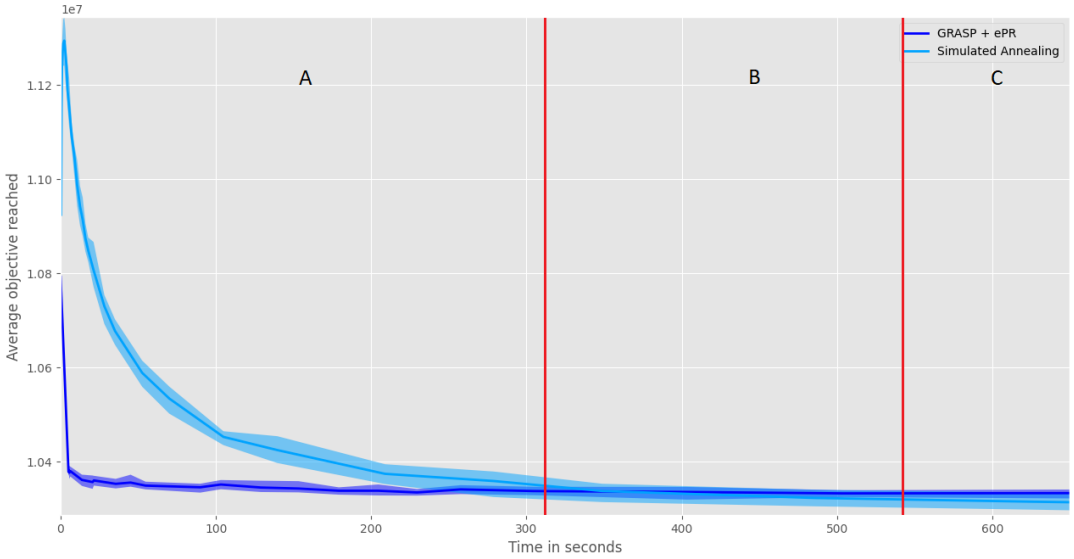


FIGURE B.3: Allowed computation time against the average objective reached for instance $(4015, 1200, 400)$.

Appendix C

Results Frequent Replanning

C.1 Results Frequent Replanning

	(3300, 660, 660)				(3300, 660, 990)				(3300, 1320, 660)			
	GRASP main		SA Main		GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
360	9,352,227	9,407,516	9,350,869	9,407,516	9,335,916	9,384,898	9,339,193	9,384,898	9,334,623	9,377,854	9,333,642	9,377,854
375	9,377,258	9,424,091	9,379,491	9,456,489	9,374,673	9,453,631	9,386,512	9,502,353	9,263,936	9,362,082	9,268,829	9,378,619
390	9,349,627	9,417,477	9,426,214	9,415,255	9,430,480	9,498,173	9,441,337	9,470,684	9,198,490	9,266,544	9,294,151	9,274,338
405	9,382,615	9,449,257	9,432,363	9,460,528	9,490,564	9,548,389	9,519,483	9,553,175	9,077,844	9,133,301	9,137,529	9,103,115
420	9,406,693	9,424,649	9,480,108	9,429,581	9,560,983	9,590,680	9,605,423	9,626,979	9,048,882	9,067,435	9,082,609	9,075,353
435	9,444,679	9,485,220	9,471,135	9,491,050	9,635,812	9,659,067	9,682,669	9,674,340	9,022,964	9,037,806	9,045,327	9,022,351
450	9,443,689	9,488,540	9,478,446	9,459,383	9,675,984	9,691,081	9,700,805	9,699,508	8,907,497	8,917,356	8,921,238	8,904,706
465	9,462,185	9,463,859	9,472,704	9,479,220	9,711,676	9,699,524	9,737,654	9,718,804	8,827,515	8,824,788	8,841,100	8,830,901
480	9,501,656	9,507,610	9,496,550	9,497,180	9,756,085	9,731,442	9,758,557	9,755,021	8,781,396	8,796,607	8,788,685	8,782,916
495	9,552,468	9,546,338	9,545,208	9,524,592	9,851,575	9,823,714	9,864,414	9,855,553	8,774,358	8,750,208	8,779,481	8,738,686
510	9,551,764	9,534,649	9,518,272	9,523,755	9,859,864	9,881,456	9,844,757	9,855,724	8,678,021	8,682,487	8,648,417	8,669,519
525	9,567,055	9,543,843	9,526,854	9,559,565	9,921,573	9,889,636	9,889,900	9,899,408	8,644,058	8,610,911	8,616,026	8,620,149
540	9,577,022	9,563,153	9,566,919	9,563,818	9,952,291	9,937,081	9,918,171	9,930,166	8,602,028	8,570,867	8,560,379	8,564,003
555	9,570,843	9,590,139	9,551,332	9,542,837	9,970,969	9,964,357	9,938,572	9,934,333	8,522,339	8,473,544	8,486,884	8,502,120
570	9,561,413	9,557,883	9,527,314	9,556,481	9,985,762	9,992,717	9,957,176	9,962,177	8,497,399	8,481,476	8,478,213	8,484,116
585	9,592,535	9,560,630	9,570,032	9,529,830	10,059,376	10,012,666	10,028,931	9,999,704	8,512,525	8,481,178	8,505,408	8,485,370

	(3300, 660, 660)				(3300, 660, 990)				(3300, 1320, 660)			
	GRASP main		SA Main		GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
600	9,569,563	9,588,663	9,527,174	9,534,060	10,068,139	10,080,960	10,028,153	10,063,281	8,448,604	8,412,880	8,439,554	8,470,618
615	9,595,680	9,559,482	9,553,982	9,568,651	10,119,659	10,102,985	10,080,575	10,114,564	8,395,543	8,361,346	8,421,429	8,383,659
630	9,603,998	9,582,939	9,567,612	9,612,183	10,129,624	10,126,356	10,136,712	10,123,681	8,353,271	8,326,331	8,356,203	8,353,712
645	9,629,029	9,599,120	9,631,671	9,605,588	10,169,683	10,163,385	10,169,309	10,168,046	8,349,038	8,346,525	8,347,006	8,362,774
660	9,629,539	9,632,792	9,614,240	9,642,644	10,191,855	10,170,790	10,211,382	10,220,468	8,333,598	8,322,481	8,335,271	8,304,965
675	9,636,411	9,615,822	9,630,761	9,637,036	10,201,847	10,204,819	10,220,248	10,194,868	8,303,336	8,291,064	8,271,864	8,284,060
690	9,649,129	9,628,529	9,638,820	9,637,570	10,204,337	10,201,264	10,192,419	10,214,465	8,274,956	8,256,592	8,245,732	8,233,226
705	9,663,809	9,645,602	9,638,105	9,632,999	10,241,765	10,224,209	10,241,937	10,239,614	8,259,921	8,238,904	8,225,267	8,241,431
720	9,649,357	9,627,773	9,617,833	9,628,553	10,256,300	10,251,417	10,245,166	10,252,238	8,242,587	8,248,883	8,227,913	8,223,901
735	9,651,580	9,628,012	9,630,438	9,627,027	10,275,977	10,257,775	10,265,518	10,278,223	8,235,219	8,226,383	8,212,502	8,220,630
750	9,650,476	9,637,672	9,625,249	9,641,797	10,297,977	10,279,266	10,296,311	10,312,500	8,239,108	8,238,031	8,218,784	8,195,633
765	9,656,323	9,610,707	9,650,022	9,628,090	10,352,981	10,329,844	10,367,300	10,339,080	8,236,063	8,192,672	8,191,740	8,208,350
780	9,659,963	9,624,854	9,638,578	9,648,156	10,371,601	10,338,720	10,366,253	10,386,433	8,240,348	8,211,200	8,201,970	8,230,969
795	9,681,271	9,644,787	9,666,011	9,661,313	10,399,621	10,388,177	10,398,034	10,400,344	8,221,126	8,198,738	8,211,947	8,189,043
810	9,667,547	9,665,222	9,653,590	9,668,923	10,399,314	10,376,401	10,403,393	10,426,646	8,184,903	8,186,209	8,165,112	8,163,463
825	9,672,516	9,640,307	9,666,298	9,647,415	10,420,891	10,368,513	10,436,225	10,424,891	8,191,997	8,182,275	8,172,967	8,162,722
840	9,657,503	9,642,778	9,650,979	9,653,739	10,407,661	10,398,920	10,432,836	10,437,422	8,168,336	8,163,488	8,157,001	8,158,478
855	9,634,312	9,596,223	9,635,278	9,644,165	10,410,649	10,391,712	10,438,461	10,450,350	8,156,081	8,155,130	8,146,458	8,143,644
870	9,621,672	9,603,757	9,620,027	9,633,512	10,405,435	10,378,387	10,430,990	10,426,884	8,106,170	8,104,675	8,097,323	8,103,455
885	9,620,350	9,595,630	9,630,141	9,630,898	10,403,962	10,399,758	10,433,064	10,435,044	8,084,599	8,086,259	8,078,537	8,081,941
900	9,631,580	9,600,652	9,642,367	9,642,880	10,402,212	10,392,394	10,430,938	10,439,608	8,079,633	8,071,679	8,079,390	8,071,779
915	9,638,202	9,625,611	9,651,272	9,654,632	10,412,481	10,396,716	10,448,510	10,451,254	8,062,643	8,048,820	8,062,545	8,063,993
930	9,633,833	9,607,085	9,647,557	9,660,362	10,412,236	10,401,318	10,448,135	10,442,082	8,046,321	8,040,435	8,044,157	8,052,467
945	9,648,115	9,610,414	9,656,405	9,654,681	10,423,171	10,402,938	10,445,164	10,442,170	8,041,130	8,020,717	8,050,528	8,036,414
960	9,623,105	9,595,248	9,640,810	9,647,470	10,426,899	10,424,667	10,453,906	10,467,997	8,022,177	8,022,244	8,025,486	8,027,958
975	9,616,073	9,597,187	9,644,793	9,638,288	10,423,683	10,403,389	10,454,130	10,466,498	8,011,497	8,004,493	8,016,621	8,019,348
990	9,606,325	9,600,564	9,624,165	9,620,330	10,435,464	10,422,661	10,479,859	10,470,638	7,988,486	7,977,059	7,994,117	7,998,409
1005	9,600,436	9,574,408	9,616,313	9,612,865	10,451,017	10,441,699	10,487,880	10,489,247	7,979,668	7,965,832	7,986,944	7,990,923
1020	9,590,376	9,574,203	9,609,028	9,613,750	10,439,697	10,423,708	10,471,869	10,462,322	7,970,743	7,959,561	7,981,404	7,973,903
1035	9,581,770	9,556,088	9,605,270	9,604,405	10,441,773	10,421,422	10,463,474	10,467,973	7,959,468	7,940,550	7,964,359	7,965,226
1050	9,579,930	9,556,531	9,604,589	9,606,616	10,452,945	10,433,676	10,474,967	10,483,710	7,951,627	7,940,426	7,956,679	7,968,206
1065	9,595,709	9,579,477	9,615,197	9,621,851	10,470,151	10,453,266	10,501,819	10,507,334	7,946,173	7,926,139	7,965,891	7,953,661
1080	9,594,301	9,575,705	9,617,916	9,623,196	10,465,974	10,455,011	10,502,778	10,502,488	7,916,001	7,914,672	7,925,319	7,933,792
1095	9,594,427	9,588,419	9,625,612	9,620,910	10,473,601	10,465,518	10,510,138	10,518,682	7,895,721	7,894,622	7,914,868	7,911,363
1110	9,597,636	9,584,304	9,627,808	9,628,057	10,478,373	10,480,154	10,518,168	10,519,499	7,876,235	7,870,458	7,895,848	7,902,707

	(3300, 660, 660)				(3300, 660, 990)				(3300, 1320, 660)			
	GRASP main		SA Main		GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
1125	9,602,821	9,583,271	9,637,215	9,633,617	10,502,840	10,484,839	10,537,724	10,536,927	7,870,262	7,862,131	7,908,960	7,888,432
1140	9,600,368	9,596,695	9,636,878	9,643,206	10,506,228	10,507,009	10,547,929	10,554,039	7,852,708	7,853,455	7,875,069	7,884,680
1155	9,602,206	9,608,897	9,650,656	9,652,986	10,519,498	10,520,195	10,563,527	10,569,395	7,853,959	7,855,677	7,878,937	7,874,327
1170	9,595,434	9,597,577	9,646,987	9,649,768	10,509,271	10,511,839	10,564,401	10,567,308	7,838,779	7,838,537	7,864,418	7,871,108
1185	9,611,585	9,594,014	9,670,638	9,649,918	10,520,340	10,506,773	10,580,938	10,566,310	7,842,725	7,828,323	7,869,367	7,857,536
1200	9,609,103	9,612,556	9,672,542	9,668,682	10,532,882	10,533,190	10,590,786	10,595,602	7,851,046	7,849,646	7,884,554	7,886,770
1215	9,616,013	9,607,951	9,676,367	9,670,729	10,529,499	10,528,092	10,596,109	10,590,655	7,846,728	7,842,809	7,877,555	7,877,923
1230	9,606,700	9,611,634	9,665,756	9,673,132	10,519,387	10,526,724	10,590,026	10,591,198	7,840,234	7,845,440	7,872,317	7,875,711
1245	9,623,556	9,610,948	9,681,908	9,669,103	10,534,739	10,522,718	10,609,460	10,595,566	7,851,496	7,842,708	7,883,699	7,875,698
1260	9,617,874	9,616,091	9,680,288	9,678,464	10,532,627	10,530,758	10,608,691	10,608,040	7,849,936	7,847,840	7,883,588	7,884,094
1275	9,616,400	9,614,605	9,678,140	9,675,458	10,531,703	10,530,730	10,611,224	10,607,281	7,850,488	7,847,707	7,885,680	7,882,409
1290	9,610,240	9,610,720	9,674,397	9,675,808	10,525,214	10,527,019	10,604,093	10,604,610	7,842,937	7,844,870	7,879,436	7,881,201
1305	9,607,929	9,605,523	9,672,767	9,669,380	10,525,662	10,522,076	10,602,687	10,598,923	7,844,289	7,841,034	7,879,985	7,876,216
1320	9,605,814	9,603,414	9,668,606	9,667,699	10,525,114	10,522,571	10,601,766	10,601,087	7,840,213	7,838,323	7,874,955	7,874,634
1335	9,604,156	9,603,836	9,667,092	9,666,287	10,524,929	10,523,632	10,599,027	10,599,757	7,839,953	7,839,809	7,874,417	7,873,698
1350	9,601,073	9,601,997	9,663,450	9,663,463	10,521,592	10,522,602	10,595,438	10,595,277	7,836,843	7,837,472	7,870,547	7,870,391
1365	9,600,909	9,599,619	9,663,610	9,662,074	10,520,236	10,519,354	10,595,560	10,594,647	7,836,130	7,835,028	7,870,737	7,870,421
1380	9,599,857	9,600,307	9,662,558	9,663,008	10,519,184	10,519,634	10,593,916	10,594,366	7,834,219	7,834,979	7,868,980	7,869,586
1395	9,599,842	9,599,575	9,661,176	9,662,276	10,518,427	10,518,830	10,593,498	10,594,266	7,834,514	7,834,569	7,869,886	7,869,510
1410	9,598,739	9,598,739	9,660,961	9,660,073	10,516,726	10,516,726	10,592,217	10,591,797	7,833,411	7,833,411	7,869,203	7,868,783
1425	9,597,879	9,598,739	9,660,440	9,660,961	10,515,889	10,516,726	10,591,380	10,592,217	7,832,890	7,833,411	7,868,682	7,869,203

TABLE C.1: Results at the end of the day when frequently replanning for GRASP and simulated annealing (SA). Where the time is the number of minutes since midnight.

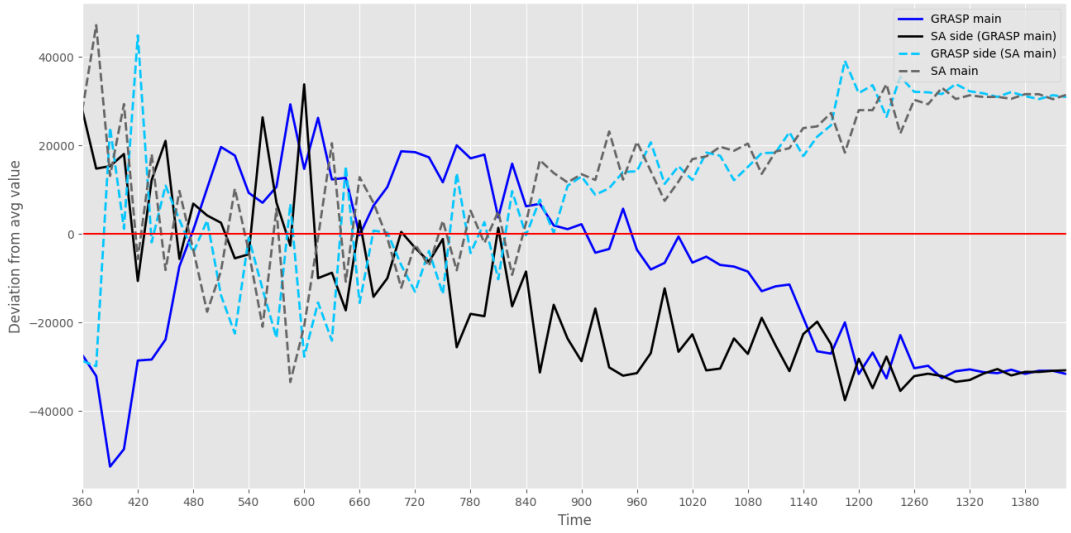
	(3300, 990, 330)				(3300, 330, 990)			
	GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
360	9,354,963	9,397,538	9,354,416	9,397,538	9,344,660	9,398,752	9,346,287	9,398,752
375	9,280,884	9,350,579	9,282,090	9,368,032	9,447,228	9,500,669	9,449,767	9,536,555
390	9,196,144	9,241,895	9,255,130	9,275,748	9,515,169	9,588,832	9,591,400	9,606,723
405	9,113,551	9,148,432	9,177,508	9,137,077	9,598,867	9,664,867	9,694,341	9,670,095
420	9,058,305	9,080,880	9,076,926	9,080,733	9,710,554	9,738,409	9,763,578	9,746,790
435	8,999,320	8,983,157	9,006,201	9,007,280	9,825,278	9,881,497	9,843,349	9,877,468
450	8,917,104	8,897,982	8,916,394	8,912,933	9,906,267	9,936,136	9,957,338	9,949,933
465	8,840,406	8,820,291	8,823,855	8,840,461	9,979,220	9,990,368	10,015,370	9,987,414
480	8,791,676	8,787,805	8,781,201	8,801,439	10,064,314	10,063,961	10,062,701	10,087,857
495	8,733,391	8,692,159	8,738,687	8,705,107	10,186,901	10,202,711	10,208,747	10,170,950
510	8,680,516	8,667,140	8,642,674	8,666,328	10,235,433	10,243,249	10,218,888	10,230,632
525	8,622,354	8,626,889	8,602,421	8,593,621	10,316,212	10,303,811	10,293,236	10,290,321
540	8,583,026	8,589,107	8,542,128	8,565,284	10,365,541	10,398,552	10,336,248	10,347,140
555	8,536,031	8,508,670	8,502,695	8,497,114	10,411,208	10,394,063	10,385,013	10,422,508
570	8,499,839	8,482,927	8,444,627	8,466,971	10,431,322	10,472,254	10,451,235	10,433,608
585	8,506,076	8,485,182	8,468,554	8,434,778	10,517,368	10,542,920	10,516,804	10,503,533
600	8,422,262	8,418,280	8,371,944	8,375,229	10,554,911	10,574,591	10,554,146	10,604,906
615	8,417,700	8,388,595	8,365,113	8,381,343	10,629,240	10,644,986	10,670,039	10,655,861
630	8,403,357	8,423,620	8,379,483	8,397,420	10,657,999	10,681,051	10,689,540	10,716,894
645	8,395,683	8,399,723	8,394,181	8,371,191	10,723,940	10,735,129	10,777,756	10,764,481
660	8,355,529	8,343,679	8,340,642	8,353,377	10,770,441	10,781,273	10,809,036	10,821,632
675	8,315,305	8,328,910	8,310,048	8,291,635	10,794,280	10,813,334	10,833,545	10,842,782
690	8,291,188	8,284,432	8,267,701	8,284,899	10,809,496	10,834,490	10,854,606	10,844,329
705	8,281,909	8,284,745	8,263,204	8,274,015	10,856,100	10,867,959	10,881,406	10,859,827
720	8,261,295	8,245,233	8,246,091	8,262,222	10,869,241	10,879,120	10,868,130	10,884,043
735	8,238,102	8,226,762	8,240,195	8,225,053	10,900,350	10,910,471	10,910,588	10,928,509
750	8,217,572	8,215,787	8,204,549	8,214,808	10,932,333	10,924,344	10,943,535	10,971,341
765	8,191,914	8,190,712	8,183,692	8,181,483	10,983,441	10,984,879	11,019,975	11,001,903

	(3300, 990, 330)				(3300, 330, 990)			
	GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
780	8,174,021	8,173,853	8,171,555	8,177,636	11,007,307	11,011,846	11,037,164	11,041,657
795	8,175,294	8,167,476	8,175,703	8,182,159	11,040,907	11,039,308	11,073,546	11,081,689
810	8,149,982	8,162,859	8,153,945	8,165,069	11,056,022	11,059,562	11,095,692	11,101,936
825	8,141,221	8,124,366	8,150,843	8,136,799	11,087,023	11,059,581	11,125,232	11,113,135
840	8,125,778	8,129,880	8,137,698	8,139,733	11,088,997	11,086,997	11,131,395	11,143,096
855	8,105,659	8,115,080	8,122,417	8,109,282	11,104,048	11,090,542	11,146,753	11,164,125
870	8,088,656	8,094,579	8,088,492	8,098,662	11,110,624	11,111,200	11,167,294	11,165,831
885	8,083,962	8,051,682	8,085,088	8,089,792	11,125,751	11,117,491	11,178,296	11,189,606
900	8,071,259	8,063,969	8,076,091	8,083,509	11,146,994	11,118,608	11,199,582	11,223,247
915	8,071,884	8,064,947	8,081,583	8,076,779	11,163,828	11,171,970	11,238,680	11,247,736
930	8,056,777	8,042,286	8,062,683	8,063,493	11,158,425	11,152,975	11,234,711	11,235,899
945	8,039,903	8,015,180	8,044,475	8,045,112	11,181,182	11,167,676	11,260,354	11,249,073
960	8,010,189	7,993,978	8,020,828	8,015,770	11,193,368	11,182,378	11,268,319	11,256,409
975	7,999,975	7,996,296	7,994,671	8,007,907	11,196,208	11,192,416	11,261,730	11,265,437
990	7,985,557	7,970,159	7,986,327	7,983,628	11,203,549	11,187,543	11,277,812	11,281,590
1005	7,982,977	7,963,844	7,981,717	7,983,599	11,211,298	11,206,561	11,287,214	11,292,716
1020	7,969,022	7,938,970	7,967,672	7,970,761	11,211,338	11,193,984	11,290,446	11,296,208
1035	7,953,793	7,929,703	7,952,649	7,953,481	11,214,884	11,195,006	11,302,059	11,295,387
1050	7,937,819	7,922,934	7,944,093	7,939,678	11,215,411	11,217,731	11,301,433	11,311,996
1065	7,925,019	7,916,037	7,925,861	7,926,937	11,239,500	11,234,946	11,336,211	11,333,906
1080	7,910,157	7,902,828	7,909,850	7,920,560	11,251,268	11,252,197	11,345,940	11,353,079
1095	7,908,519	7,910,037	7,915,829	7,918,716	11,267,641	11,270,799	11,367,747	11,363,630
1110	7,896,214	7,892,637	7,913,184	7,912,580	11,270,854	11,276,754	11,364,464	11,372,449
1125	7,896,408	7,884,396	7,912,120	7,899,013	11,294,667	11,284,001	11,394,574	11,388,896
1140	7,865,319	7,869,636	7,878,739	7,881,992	11,292,389	11,297,012	11,400,308	11,407,276
1155	7,862,668	7,862,201	7,872,083	7,880,562	11,305,270	11,308,484	11,416,035	11,419,214
1170	7,848,321	7,852,211	7,864,519	7,869,272	11,304,154	11,304,269	11,420,573	11,422,494
1185	7,854,271	7,839,978	7,879,063	7,862,053	11,318,537	11,302,152	11,439,348	11,422,293
1200	7,857,328	7,856,799	7,882,809	7,882,201	11,325,320	11,328,060	11,441,762	11,451,181

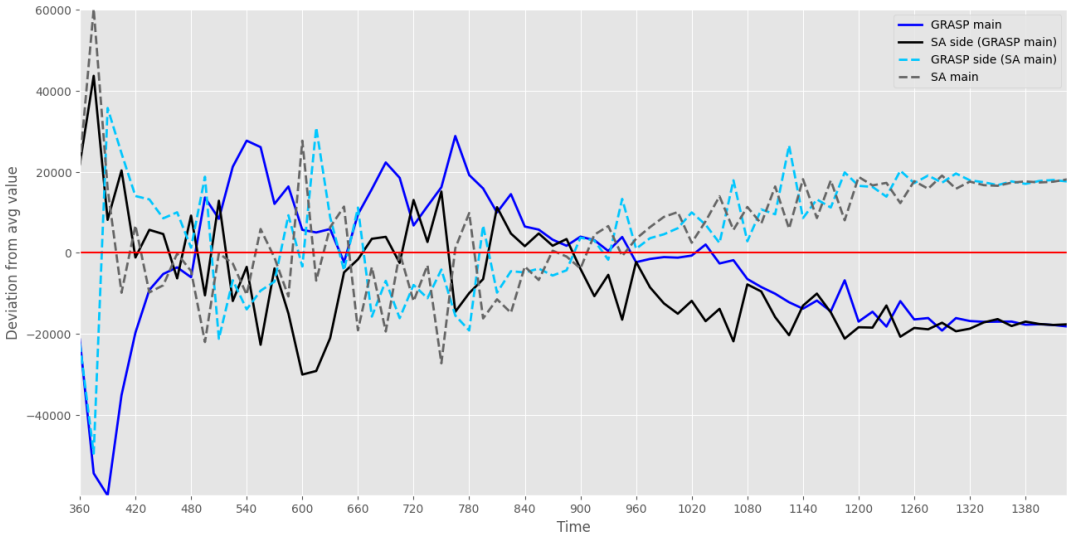
	(3300, 990, 330)				(3300, 330, 990)			
	GRASP main		SA Main		GRASP main		SA Main	
Time	GRASP	SA	GRASP	SA	GRASP	SA	GRASP	SA
1215	7,852,763	7,852,276	7,883,275	7,881,501	11,324,541	11,320,599	11,451,548	11,447,471
1230	7,841,047	7,843,619	7,871,287	7,874,231	11,319,142	11,323,333	11,447,120	11,454,354
1245	7,847,526	7,836,984	7,878,715	7,866,548	11,338,604	11,322,026	11,468,435	11,452,725
1260	7,846,835	7,845,666	7,880,354	7,878,240	11,336,757	11,334,617	11,463,943	11,464,646
1275	7,848,452	7,844,423	7,880,250	7,878,252	11,334,261	11,333,771	11,464,410	11,462,152
1290	7,843,491	7,843,535	7,873,363	7,873,245	11,328,006	11,328,040	11,456,234	11,456,729
1305	7,841,743	7,840,456	7,873,646	7,871,785	11,331,383	11,324,387	11,460,041	11,454,972
1320	7,836,149	7,837,354	7,868,101	7,868,833	11,330,015	11,329,284	11,460,735	11,458,336
1335	7,833,559	7,833,831	7,865,358	7,865,524	11,326,836	11,327,400	11,458,420	11,458,385
1350	7,832,115	7,832,560	7,863,824	7,864,359	11,320,695	11,322,625	11,454,276	11,453,902
1365	7,833,201	7,831,989	7,864,070	7,863,150	11,321,216	11,320,504	11,452,082	11,451,328
1380	7,831,818	7,833,201	7,861,851	7,862,767	11,320,132	11,320,991	11,450,547	11,451,153
1395	7,831,950	7,831,638	7,861,929	7,861,671	11,320,530	11,320,336	11,450,139	11,449,802
1410	7,831,493	7,831,298	7,861,472	7,861,277	11,319,249	11,318,829	11,449,090	11,449,090
1425	7,831,177	7,831,493	7,861,156	7,861,472	11,318,412	11,319,249	11,448,253	11,449,090

TABLE C.2: Results at the end of the day when frequently replanning for GRASP and simulated annealing (SA). Where the time is the number of minutes since midnight.

C.2 Frequent Replanning Extra Figures

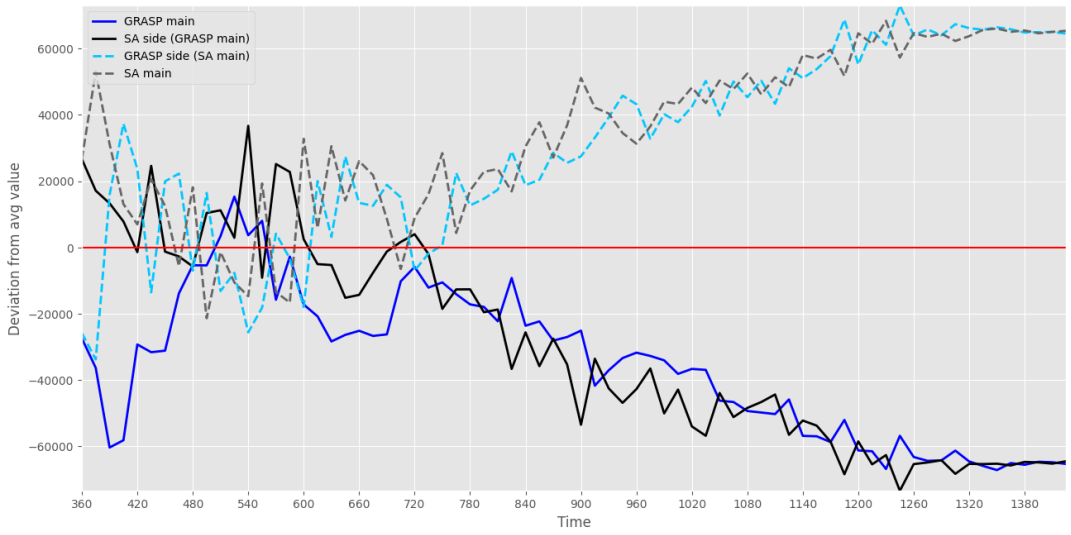


(A) Instance (3300, 660, 660).

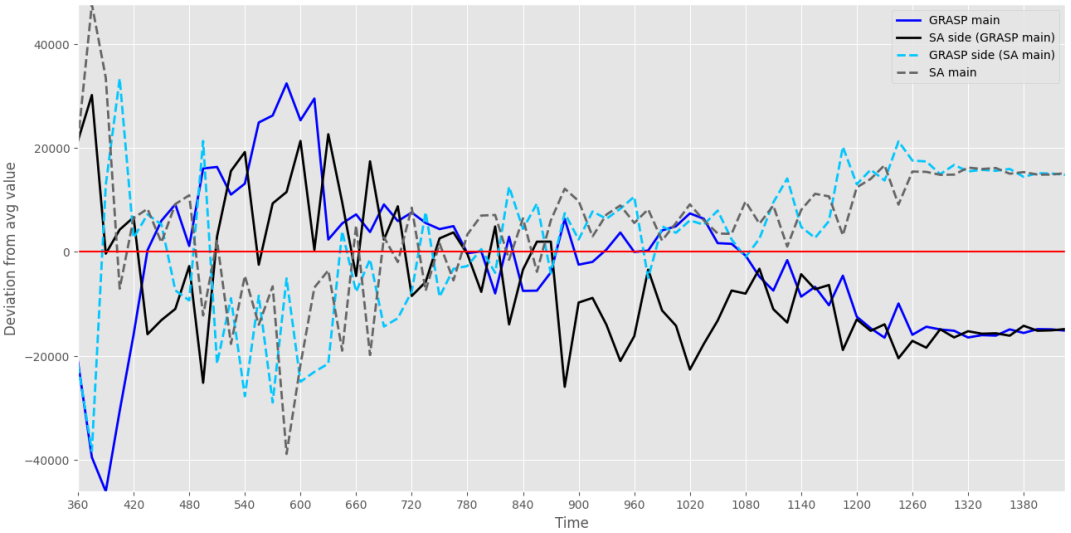


(B) Instance (3300, 1320, 660).

FIGURE C.1: Frequent real-time replanning for GRASP as main and simulated annealing (SA) as side and the other way around. The figure shows the deviation from the average solution quality at every replanning moment.

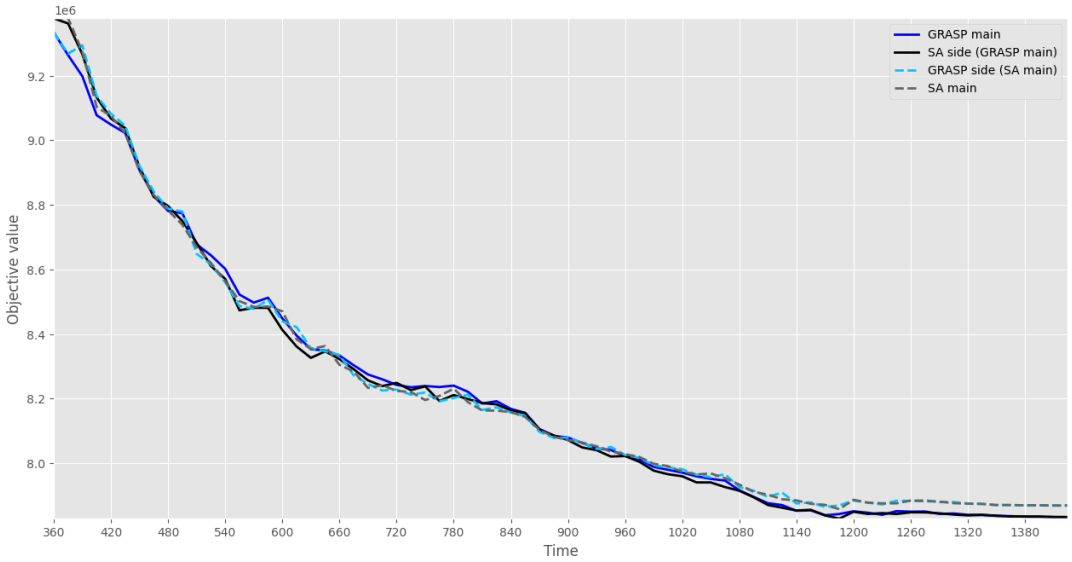


(C) Instance (3300, 330, 990).

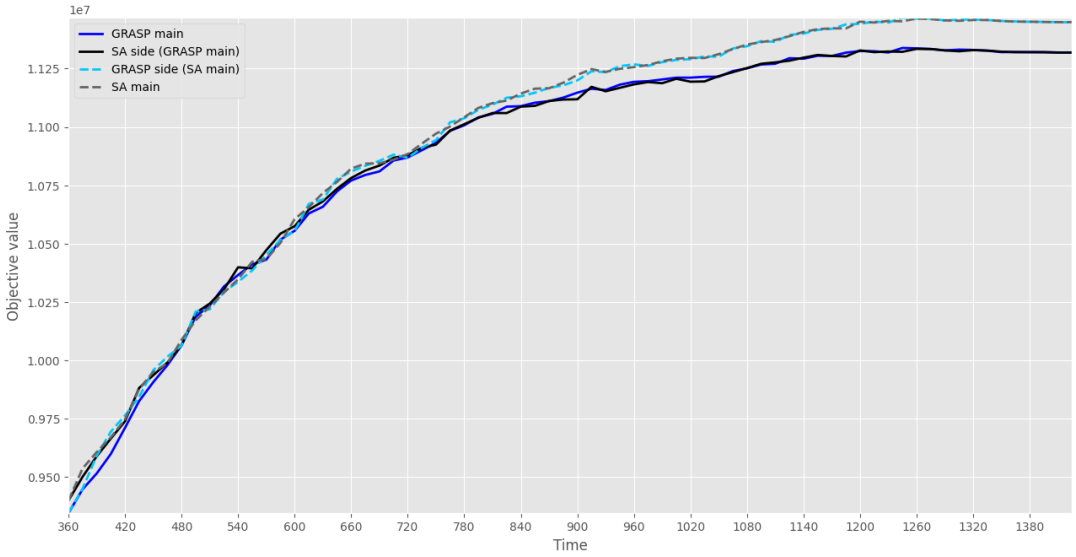


(D) Instance (3300, 990, 330).

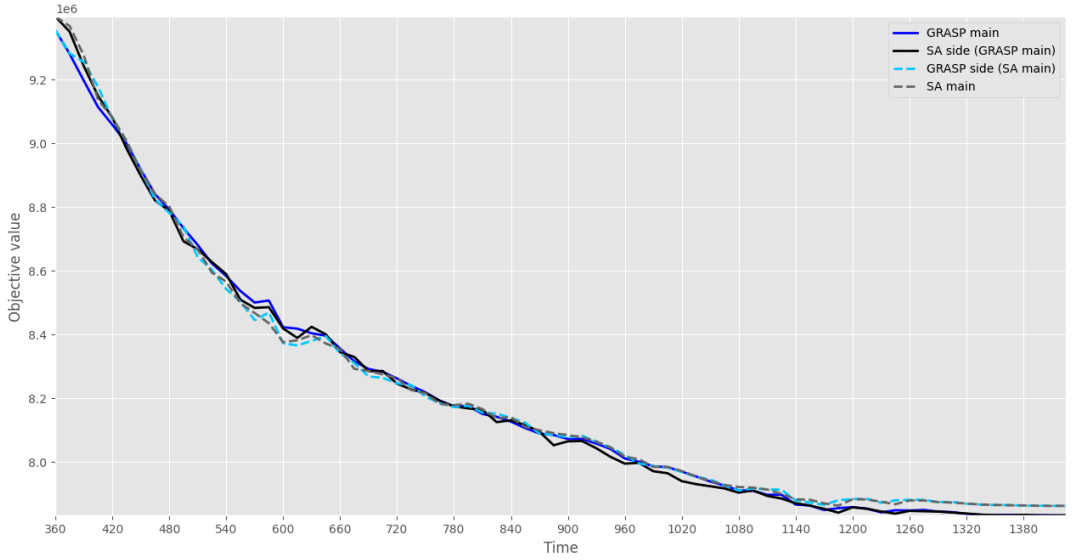
FIGURE C.1: Frequent real-time replanning for GRASP as main and simulated annealing (SA) as side and the other way around. The figure shows the deviation from the average solution quality at every replanning moment.



(A) Instance (3300, 1320, 660).



(B) Instance (3300, 330, 990).



(C) Instance (3300, 990, 330).

FIGURE C.2: Frequent real-time replanning for GRASP as main and simulated annealing (SA) as side and the other way around.

Bibliography

[1] G. Berbeglia, J.-F. Cordeau, and L. Gilbert, "Dynamic pickup and delivery problems," *European Journal of Operational Research*, vol. 202, no. 1, pp. 8–15, Apr. 2010. DOI: 10.1016/j.ejor.2009.04.024.

[2] G. Hasle, K. Lie, and E. Quak, *Geometric Modelling, Numerical Simulation, and Optimization:: Applied Mathematics at SINTEF*. Springer Berlin Heidelberg, 2007, p. 398, ISBN: 9783540687832.

[3] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: Models and algorithms," *Annals of Operations Research*, vol. 153, no. 1, pp. 29–46, Jun. 2007. DOI: 10.1007/s10479-007-0170-8.

[4] J.-F. Cordeau and L. Gilbert, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transportation Research Part B: Methodological*, vol. 37, no. 6, pp. 579–594, Jul. 2003, ISSN: 0191-2615. DOI: 10.1016/S0191-2615(02)00045-0.

[5] H. Psaraftis, "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem," *Transportation Science*, vol. 14, pp. 130–154, May 1980. DOI: 10.1287/trsc.14.2.130.

[6] M. Aldaihani and M. Dessouky, "Hybrid scheduling methods for paratransit operations," *Computers & Industrial Engineering*, vol. 45, no. 1, pp. 75–96, Jun. 2003, ISSN: 0360-8352. DOI: 10.1016/S0360-8352(03)00032-9.

[7] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, Jun. 2006. DOI: 10.1287/opre.1060.0283.

[8] P. Toth and D. Vigo, "Heuristic algorithms for the handicapped persons transportation problem," *Transportation Science*, vol. 31, no. 1, pp. 60–71, Feb. 1997. DOI: 10.1287/trsc.31.1.60.

[9] J. Baugh JR., G. Kakivaya, and J. Stone, "Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing," *Engineering Optimization*, vol. 30, no. 2, pp. 91–123, Feb. 1998. DOI: 10.1080/03052159808941240.

[10] S. Parragh, K. Doerner, R. Hartl, and X. Gandibleux, "A heuristic two-phase solution approach for the multi-objective dial-a-ride problem," *Networks*, vol. 54, no. 4, pp. 227–242, Dec. 2009, ISSN: 0028-3045. DOI: 10.1002/net.20335.

[11] M. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations Research*, vol. 4, no. 1, pp. 285–305, Dec. 1985. DOI: 10.1007/BF02022044.

[12] J. Desrosiers, Y. Dumas, and F. Soumis, "A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows," *American Journal of Mathematical and Management Sciences*, vol. 6, no. 3, pp. 301–325, Feb. 1986. DOI: 10.1080/01966324.1986.10737198.

[13] S. Parragh, K. Doerner, and R. Hartl, "Variable neighborhood search for the dial-a-ride problem," *Computers & Operations Research*, vol. 37, no. 6, pp. 1129–1138, Jun. 2010, ISSN: 0305-0548. DOI: 10.1016/j.cor.2009.10.003.

[14] Y. Dumas, J. Desrosiers, and F. Soumis, *Large Scale Multi-vehicle Dial-a-ride Problems*. Groupe d'études et de recherche en analyse des décisions, Sep. 1989. [Online]. Available: https://www.gerad.ca/en/papers/G-89-30.

[15] P. Toth and D. Vigo, "Fast local search algorithms for the handicapped persons transportation problem," in *Meta-Heuristics: Theory and Applications*. Boston, MA: Springer US, 1996, pp. 677–690, ISBN: 978-1-4613-1361-8. DOI: 10.1007/978-1-4613-1361-8_41.

[16] L. Gilbert, "Fifty years of vehicle routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, Nov. 2009. DOI: 10.1287/trsc.1090.0301.

[17] Y. Molenbruch, K. Braekers, and A. Caris, "Typology and literature review for dial-a-ride problems," *Annals of Operations Research*, vol. 259, no. 1-2, pp. 295–325, May 2017. DOI: 10.1007/s10479-017-2525-0.

[18] M. Savelsbergh and M. Sol, "Drive: Dynamic routing of independent vehicles," *Operations Research*, vol. 46, no. 4, pp. 474–490, Aug. 1998. DOI: 10.1287/opre.46.4.474.

[19] J. Desrosiers, Y. Dumas, F. Soumis, S Taillefer, and D. Villeneuve, "An algorithm for mini-clustering in handicapped transport," *Les Cahiers du GERAD*, pp. 1–15, Jan. 1991. [Online]. Available: https://www.gerad.ca/en/papers/G-91-02.

[20] O. Madsen, H. Ravn, and J. Rygaard, "A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives," *Annals of Operations Research*, vol. 60, no. 1, pp. 193–208, Dec. 1995. DOI: 10.1007/bf02031946.

[21] J.-J. Jaw, A. Odoni, H. Psaraftis, and N. Wilson, "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows," *Transportation Research Part B: Methodological*, vol. 20, no. 3, pp. 243–257, Jun. 1986. DOI: 10.1016/0191-2615(86)90020-2.

[22] A. Attanasio, J.-F. Cordeau, G. Ghiani, and L. Gilbert, "Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem," *Parallel Computing*, vol. 30, no. 3, pp. 377–387, Mar. 2004. DOI: 10.1016/j.parco.2003.12.001.

[23] L. Khelifi, I. Zidi, K. Zidi, and K. Ghedira, "A hybrid approach based on multi-objective simulated annealing and tabu search to solve the dynamic dial a ride problem," May 2013, pp. 227–232, ISBN: 978-1-4799-0314-6. DOI: 10.1109/ICAdLT.2013.6568464.

[24] A. Lois and A. Ziliaskopoulos, "Online algorithm for dynamic dial a ride problem and its metrics," *Transportation Research Procedia*, vol. 24, pp. 377–384, Jan. 2017. DOI: 10.1016/j.trpro.2017.05.097.

[25] M. Horn, "Fleet scheduling and dispatching for demand-responsive passenger services," *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 1, pp. 35–63, Feb. 2002. DOI: 10.1016/S0968-090X(01)00003-1.

[26] S. Varone and V. Janilionis, "Insertion heuristic for a dynamic dial-a-ride problem using geographical maps," in *French National Centre for Scientific Research*, Nov. 2014. [Online]. Available: https://hal.science/hal-01166662.

[27] Y. Luo and P. Schonfeld, "Online rejected-reinsertion heuristics for dynamic multivehicle dial-a-ride problem," *Transportation Research Record*, vol. 2218, no. 1, pp. 59–67, Jan. 2011. DOI: 10.3141/2218-07.

[28] D. Bertsimas, "The edge of optimization in large-scale vehicle routing for paratransit," *Proceedings - Operations Research*, 2020, visited: 23/03/2023. [Online]. Available: https://dbertsim.mit.edu/pdfs/papers/2020-yan-transporation-science-paratransit.pdf.

[29] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13290 –13 294, Sep. 2013. DOI: 10.1073/pnas.1403657111.

[30] V. Yu, S. Purwanti, A. Perwira Redi, C.-C. Lu, S. Suprayogi, and P. Jewpanya, "Simulated annealing heuristic for the general share-a-ride problem," *Engineering Optimization*, vol. 50, no. 7, pp. 1178–1197, Mar. 2018. DOI: 10.1080/0305215X.2018.1437153.

[31] D. Santos and E. Xavier, "Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6728–6737, Nov. 2015. DOI: 10.1016/j.eswa.2015.04.060.

[32] D. Santos and E. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, AAAI Press, 2013, 2885–2891, ISBN: 9781577356332. DOI: 10.5555/2540128.2540544.

[33] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," *Proceedings - International Conference on Data Engineering*, pp. 410–421, Apr. 2013. DOI: 10.1109/ICDE.2013.6544843.

[34] L. Häme, "An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows," *European Journal of Operational Research*, vol. 209, no. 1, pp. 11–22, Feb. 2011. DOI: 10.1016/j.ejor.2010.08.021.

[35] E. Hyytiä, L. Häme, A. Penttinen, and R. Sulonen, "Simulation of a large scale dynamic pickup and delivery problem," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010. DOI: 10.4108/ICST.SIMUTOOLS2010.8701.

[36] M. G. C. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, Mar. 1995. DOI: 10.1007/bf01096763.

[37] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," pp. 417–431, Jan. 1998. DOI: 10.1007/3-540-49481-2\{_}30.

[38] M. Prais and C. C. Ribeiro, "Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 164–176, Aug. 2000. DOI: 10.1287/ijoc.12.3.164.12639.

[39] M. Resende and C. Ribeiro, "Greedy randomized adaptive search procedures: Advances and extensions," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Springer International Publishing, 2019, pp. 169–220, ISBN: 978-3-319-91086-4.

[40] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, Nov. 1997, ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(97)00031-2.

[41] F. Glover, "Tabu search and adaptive memory programming — advances, applications and challenges," in *Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers, 1996, pp. 1–75. DOI: 10.1007/978-1-4615-4102-8_1.

[42] M. Resende and C. Ribeiro, "Path-relinking," in *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer New York, 2016, pp. 167–188, ISBN: 978-1-4939-6530-4. DOI: 10.1007/978-1-4939-6530-4_8.

[43] M. Resende and R. Werneck, "A hybrid heuristic for the p-median problem," *Journal of Heuristics*, vol. 10, no. 1, pp. 59–88, Jan. 2004. DOI: 10.1023/B:HEUR.0000019986.96257.50.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983. DOI: 10.1126/science.220.4598.671.