Improving Dynamic Route Optimisation by making use of Historical Data

Jelmer Alexander van Lochem

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Improving Dynamic Route Optimisation by making use of Historical Data

by

Jelmer Alexander van Lochem

in partial fulfillment of the requirements for the degree of

Master of Science
in Systems & Control

at the Delft University of Technology,
to be defended publicly on Monday April 1st, 2019 at 12:30 PM.

Supervisor:	Dr. J. Alonso-Mora	TU Delft
External supervisor:	Dr. Ir. P. van 't Hof	ORTEC
Thesis committee:	Prof. dr. ir. J. Hellendoorn	TU Delft
	Dr. B. Atasoy	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

ABSTRACT

In the dynamic world we live in, the transportation of people and goods in a reliable, efficient and timely manner has grown to be more important than ever. Roads and cities are becoming more congested and the impact of greenhouse gasses can already be observed. The need for controlling transportation systems, and specifically fleets of vehicles, more efficiently is therefore now higher than ever. Few methods exist in the literature which utilise historical data to increase the efficiency of dynamic fleets of vehicles. This work therefore proposes a novel anticipatory insertion method which incorporates a set of predicted requests to beneficially adjust the routes of a fleet of vehicles, in real-time. This set of predicted requests is derived, in advance, from historical data by clustering comparable requests and predicting similar requests when assumed patterns in their occurrence are present. This method is combined with a developed dynamic vehicle routing solver which makes use of a range of heuristics and adaptive large neighbourhood search. The proposed method is evaluated using numerical simulations on a range of real-world problem instances with up to 1.655 requests per day. These instances represent dynamic multi-depot capacitated pickup and deliver vehicle routing problems with time windows. The method is compared with several other approaches and in order to quantify the added value of making use of historical data, the method is benchmarked against a comparable reactive approach which also makes use of adaptive large neighbourhood search. It is shown that, by making use of the proposed method, on average, 4,58% less distance is required to be travelled by a fleet vehicles while additionally 3,35% fewer vehicles are required to fulfil the same set of requests.

Preface

I would like to thank my daily supervisor at ORTEC, Pim van 't Hof, for guiding me during my thesis. You have provided me with loads of feedback and have helped me tremendously during the past year. I enjoyed our sometimes lengthy meetings and talks, not only about vehicle routing, but also about research in general and a variety of other topics.

Furthermore I would like to thank Javier Alonso-Mora, my supervisor at the TU Delft for his guidance and the introduction to the subject of vehicle routing. I believe our meetings and your feedback really took my work to the next level.

Finally I would like to thank my parents for giving me the opportunity to pursue this study. And last but not least, my friends and family for making the remaining spare-time very enjoyable. Thank you all for your support during the past year.

Jelmer van Lochem
Rotterdam, March 2019

CONTENTS

LIST OF FIGURES

List of Tables

1 INTRODUCTION

In an ever more dynamic world the transportation of people and goods in a reliable, efficient and timely manner has grown to be more important than ever. Logistics has become the backbone of manufacturing, food, e-commerce and public transportation industries. As a result expenditures on transportation activities have risen with economic growth, passing over 1426 billion dollars in 2012, in the United States alone [1].

Due to the scale of these operations, suppliers and distributors have recognised the importance of efficient distribution strategies in order to reduce operational cost and increase the level of service to customers.

Simultaneously, numerous technological advances in the field of communication and information processing have been made which have the potential to benefit the transportation industry. Especially, the ability to quickly process vast quantities of data, real-time wireless communication and the tracking of assets using the global positioning system (GPS) have enabled coordinated real-time transportation. Mainly due to these technological advances the transportation industry and specifically urban transportation are going through rapid and significant changes.

First of all an explosive growth of on-demand transportation of passengers, performed by for example Uber, Lyft and Grab, can be observed across the globe. These services try to use the mentioned technologies to their full potential and are creating a connected and more centrally controlled transportation network. By using such networks they are able to control their fleets of vehicles more efficiently while offering a higher level of service to customers as compared to their (more traditional) competitors.

Second, e-commerce sales have doubled [2] over the past five years which demands far more from parcel delivery and courier express service providers within cities. This increased demand does not only consists of additional volume but a trend towards tighter and same-day delivery time windows can also be observed. In addition to this a variety of meal and grocery delivery services have been introduced during recent years, both increasing the number of fleets of vehicles inside and around cities even further.

Finally, even though the increase in transportation has offered many comforts, a significant price is also being payed. For example, the cost of traffic congestion in the United States alone is estimated at 300 billion dollar per year [3]. Furthermore, one third of all CO_2 gasses is released in traffic and transport, whereof 80% by vehicles [4]. Without a doubt it can therefore be concluded that controlling fleets of vehicles more efficiently is a topic which has many relatively new applications, is under active development but above all appears to be a necessity in and around ever more congested cities worldwide.

1.1. RESEARCH OBJECTIVE

The objective of this work is therefore to develop an algorithm which is able to more efficiently control a fleet of vehicles in a situation where additional customers may request service while vehicles are already driving. The focus is on improving the efficiency of the fleet of vehicles by making use of historical data. The primary performance indicator while doing this is the total distance travelled by all vehicles while fulfilling all requests. Furthermore, for wide applicability, the focus is on being able to handle relatively large arbitrary problem instances. Specifically, the evaluated instances represent dynamic multi-depot capacitated pickup and deliver vehicle routing problems with time windows of multiple hours. Two clear goals are identified:

- Determine how a state of the art reactive approach performs on large dynamic real-world problem instances.

- Determine if historical data can be used to improve upon a purely reactive approach when solving large dynamic real-world problem instances.

1.2. CONTRIBUTIONS

The contributions presented in this work are related to a new developed method for anticipatory routing. Four contributions can be identified, namely:

- A method, based on anticipatory insertion, is proposed which does not rely on assumed arrival probabilities of future requests but instead derives a set of predicted requests purely from historical data.
- The proposed method is evaluated on real-world problem instances with up to 1.655 requests per day. These instances are multiple times larger than the instances which are generally solved in the literature related to anticipatory routing.
- The added value of including historical data using the proposed method is quantified and put into perspective by comparing it with a dynamic state of the art reactive method and comparable static full information approach.
- It is shown that, by making use of historical data and the proposed method, the comparable state of the art reactive approach can be outperformed on large real-world problem instances without any detrimental side effects.

1.3. CHAPTER OVERVIEW

This work is composed as follows. First, in Chapter 2 an overview of the existing related literature is presented. Chapter 3 formally describes the problem in mathematical terms and gives an overview of the proposed method. In Chapter 4, a method for predicting requests based on a set of historical requests is presented. In Chapter 5 it is described how the set of predicted requests can be incorporated into a dynamic vehicle routing problem. In Chapter 6 the design of a developed vehicle routing solver, which makes use of heuristics, is described. In Chapter 7 the proposed method is evaluated and compared to several other optimisation approaches. Finally, in Chapter 8 conclusions and recommendations for future work are presented.

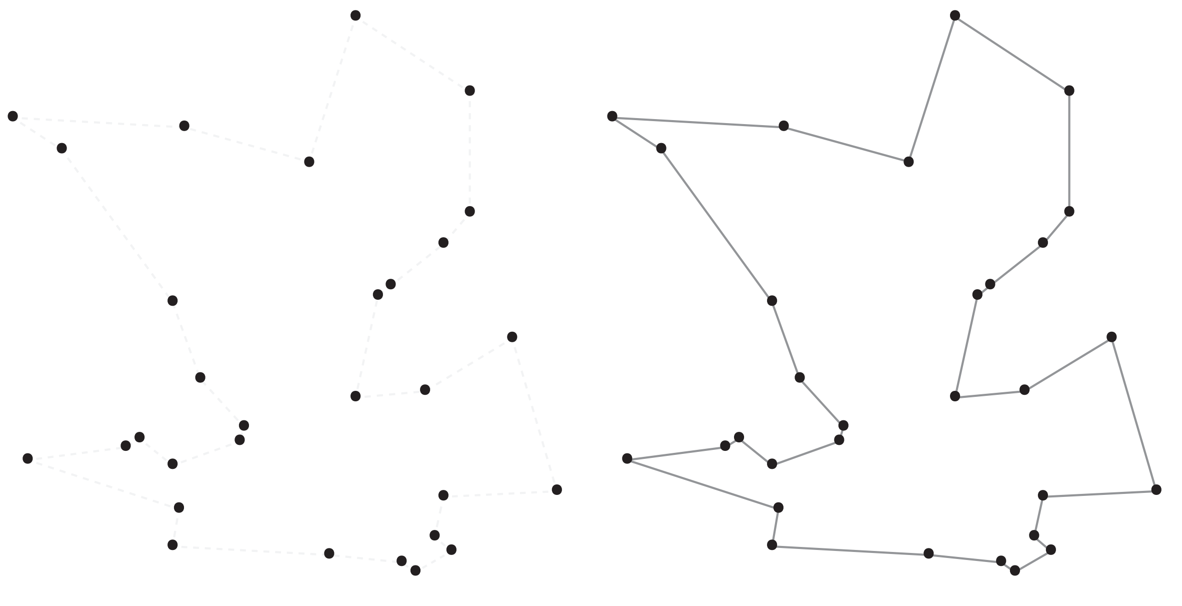
2
Related Work

In this chapter related work is presented in three different sections. First litature related to the Vehicle Routing Problem (VRP) is introduced in Section 2.1. Then, several concepts related to the Dynamic Vehicle Routing Problem are covered in Section 2.2. Finally, literature related to anticipatory routing is covered in Section 2.3.

2.1. Vehicle Routing Problem

The goal of the Vehicle Routing Problem (VRP) is to determine the set of routes for a fleet of vehicles to optimally serve a set of customers. This problem was first introduced by Dantzig and Ramser in 1959 to improve the delivery of petrol to gas-stations [5]. Now, over 50 years later, the problem is one of the most researched combinatorial optimisation problems with countless variations

The vehicle routing problem is a generalisation of the Travelling Salesman Problem (TSP) which was formulated by the Irish mathematician W.R. Hamilton in the 1800s. The goal of the TSP is to find the shortest possible route along a given set of cities with known distances between them. Each city can only be passed once and the route should end at the same city from which it started. An instance of a TSP can be seen in Figure 2.1.



(a) Input of the problem (b) The solution

Figure 2.1: An instance of a Travelling Salesman Problem

2.1.1. PROBLEM VARIATIONS
The VRP has numerous variations, amongst others, the Capacitated Vehicle Routing Problem, the Vehicle Routing Problem with Time Windows and the Pickup and Delivery Vehicle Routing Problem. These variations are briefly described below.

CAPACITATED VEHICLES
The Capacitated Vehicle Routing Problem (CVRP) is the most researched variation of the vehicle routing problem [6]. This variation poses a capacity constraint on the vehicles that have to serve customers. This constraint is either modelled as a finite amount of customers that a single vehicle can visit or as a maximum amount of load that it can carry at any point during its route execution. Furthermore, it assumes the presence of a depot from which vehicles start and return to. Also, it is assumed that the fleet of vehicles is *homogeneous*, or in others words, that vehicles all have the same capacity. Two ways to model the problem have recently been presented by Borcinova [7]. In Section 3.1 the problem definition used in this work, which contains capacitated vehicles, is presented.

TIME WINDOWS
The Vehicle Routing Problem with Time Windows (VRPTW) includes time windows which stem from the fact that customers impose deadlines on earliest and latest service [6]. These constraints add a temporal scheduling aspect to the, generally only spatial, vehicle routing problem. Because of this extra dimension the problem complexity also increases. Numerous variations of time constrained vehicle routing problems exist in literature. Amongst these variations, an important distinction can be made in the "hardness" of the time windows. Hard time windows can, under no circumstance, be violated whereas soft time windows can often be violated at the cost of a certain penalty. Regardless of hardness, time constraints at any location i are generally defined by an earliest start of service e_i and latest start of service l_i [8]. Furthermore, generally, a restriction is also posed on the total route duration of a vehicle k by a certain maximum allowable duration T_k. For more on time window constrained routing and scheduling problems the reader is referred to an extensive survey by Solomon and Desrosiers [8]. The problem definition, presented in Section 3.1, states that both customers and vehicles have limited availability caused by time windows.

PICKUP AND DELIVERY
The Pickup and Delivery Vehicle Routing Problem (PDVRP) expands on the "regular" CVRP by allowing pickups to occur at other locations than at a single depot [6]. This generally means capacity constraints can be met more easily as not all load is required to be present in the vehicle at the start of a route. However, it should be noted that, a pickup should always occur before its corresponding delivery as the load is required to be in the vehicle. The problem definition, presented in Section 3.1, describes requests wich consist of both a pickup and deliver task.

2.1.2. PROBLEM COMPLEXITY
The TSP has been categorised as an NP-hard problem [9]. This means it is highly unlikely that instances of this problem can be solved within polynomial time (polynomial in the size of the problem instance). Furthermore, the TSP can be seen as a variation of the more general CVRP. This because when the vehicle capacity is set sufficiently large the CVRP is in essence equal to the TSP [5]. As a result the CVRP belongs to the same category of problem complexity. In addition to this, Savelsbergh [10] showed that when time windows are introduced the complexity increases even further. It was proven that even finding a feasible solution for a TSP with time windows is an NP-complete problem in itself. Because of this exact algorithms are rarely able to solve problem instances with more than a few dozen or hundreds of customers and vehicles (depending on the problem variation) [11]. Therefore, heuristics are often used to solve larger real-world (or rich) problem instances. These and other solution methods are described in the following Section 2.1.3.

2.1.3. SOLUTION METHODS
Several categories of solution methods exist for most variations of the vehicle routing problem. A clear separation between four kinds of methods can be identified. This is the separation between exact methods, classical heuristics, meta-heuristics and neural networks. These categories of methods are be covered below.

EXACT METHODS
Exact solution methods distinct themselves by the ability to find the provable optimal solution to an optimisation problem. The downside of these kinds of methods is that they generally do not scale well. This means

even for medium sized problems (containing several hundreds of customers) the required computation time increases drastically and is generally not acceptable in real-world applications. To illustrate this, Pecin et. al [12] recently have shown that even with a start of the art branch and bound based method, often several hours are required to solve problems containing only 360 customers.

However, it is worth mentioning that, when faced with the appropriate problem, results on large problem instances may still be obtained within a small amount of computation time when using exact methods. For example in ride-sharing for passenger transport often very tight time constraints need to be met. Because of these constraints the number of possible solutions is severely reduced. By making use of this fact Alonso et. al [13] have shown to be able to solve instances with hundreds of requests within seconds using exact methods. For more on exact methods for solving vehicle routing problems the reader is referred to several extensive surveys written by Laporte et al. [14, 15].

CLASSICAL HEURISTICS

A *heuristic* can be defined as an approach to solve a certain problem by using a practical method which is not guaranteed to yield an optimal solution but is sufficient for reaching an immediate goal. In vehicle routing research this immediate goal is often finding the best possible solution, according to a certain objective, within a given amount of computation time. Heuristics distinct themselves by the ability to do this for relatively large instances of rich problem variations. The downside of these methods however is that an optimal solution cannot be guaranteed.

According to Laporte et al. [16] and a more recent review paper by Braekers et al. [17] classical heuristics have the distinctive property that they do not allow the intermediate deterioration of the solution during the process of finding a better solution. Classical heuristics can roughly be separated into construction heuristics and local search heuristics. In this section these heuristics are presented in way specifically for the problem variation described in Section 3.1. However they are, in general, also applicable to solving other variations of the vehicle routing problem.

Construction heuristics are used to create an initial solution to a vehicle routing problem. The goal of every construction heuristic is therefore to insert unscheduled requests into an existing (empty) solution. The process and cost of adding a request into an existing route is visualised in Figure 2.2.

The most famous construction heuristic is the Clarke and Wright Savings heuristic [18]. The Clarke and Wright Savings heuristics works by inserting each request into its own route and then sequentially combining routes which result in the largest saving in the primary objective (often the total distance travelled). Also the sequential and parallel cheapest insertion heuristics [19] are often used. The sequential cheapest insertion heuristic works by computing all insertion possibilities for a request into a single solution and then realising the possibility which results in the smallest increase in the primary objective. The parallel cheapest insertion heuristic expands on this by computing all possibilities for all unscheduled requests and thus, besides choosing the 'cheapest' possibility, also chooses the 'cheapest' request to insert first. The regret insertion heuristic again expands on the parallel cheapest insertion heuristic by computing, for each request, the possible increase in insertion cost (the regret of not inserting it right away) and bases the order by which requests are inserted on this metric [20].

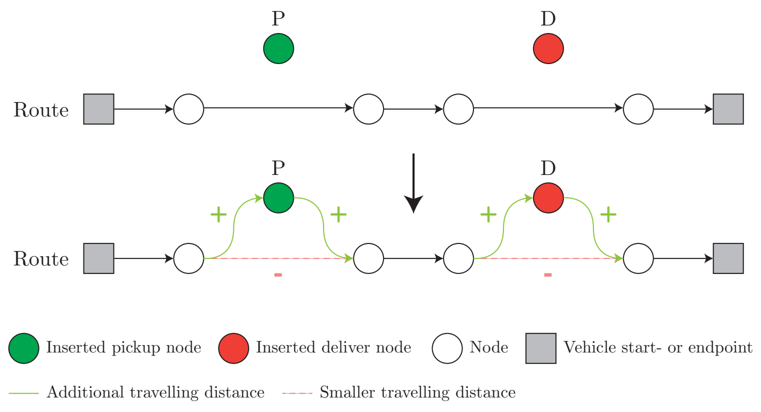

P
D
Route
P
D
+
+
+
+
-
-
Route
Inserted pickup node
Inserted deliver node
Node
Vehicle start- or endpoint
Additional travelling distance
Smaller travelling distance

Figure 2.2: Process and cost of inserting a request into an existing route

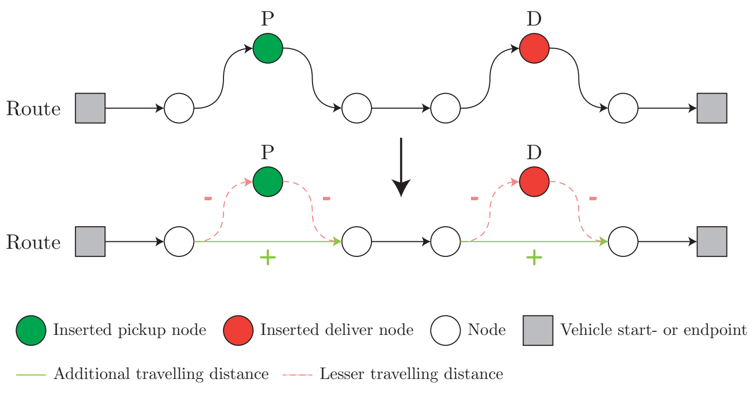

P
D
Route
Route
-
+
Inserted pickup node
Inserted deliver node
Node
Vehicle start- or endpoint
Additional travelling distance
Lesser travelling distance

Figure 2.3: Process and cost of removing a request from an existing route

Local Search (LS) operators perform relatively simple actions in an attempt to improve the quality of a solution. Some of the most widely used LS methods are the shift, rearrange, 2-opt and exchange operators [21]. How these operators work has been visualised in Figures 2.4 - 2.7. For more on local search heuristics and their implementation we refer the reader to a survey by Groër et al. [22].

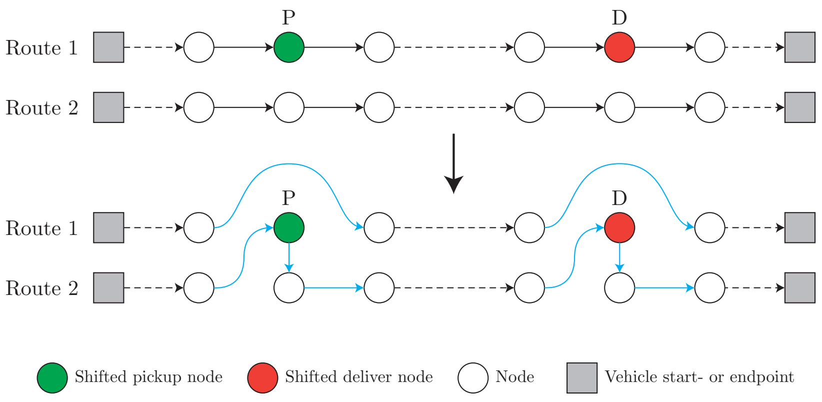


Figure 2.4: Example of the effect of the shift local search operator on existing routes

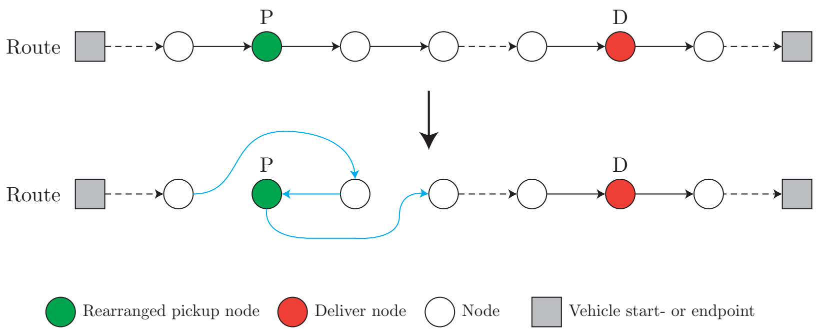


Figure 2.5: Example of the effect of the rearrange local search operator on an existing route

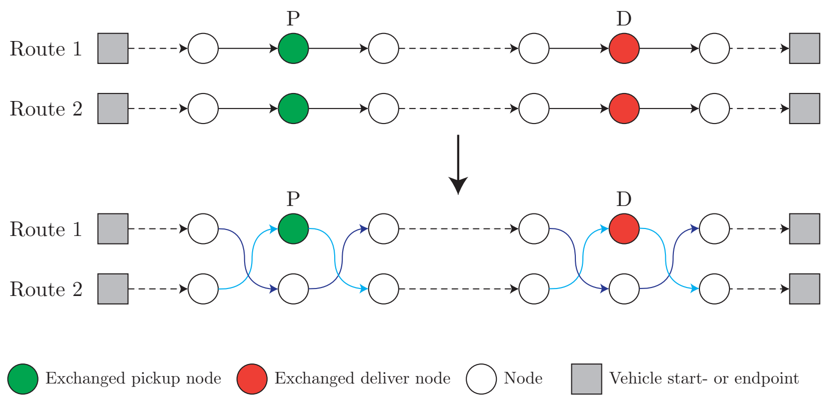


Figure 2.6: Example of the effect of the exchange local search operator on two existing routes

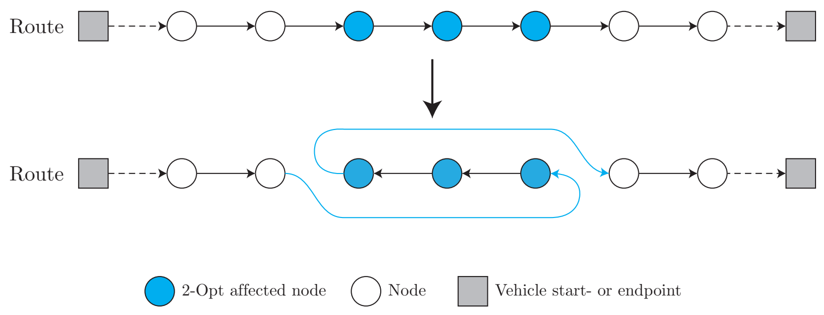


Figure 2.7: Example of the effect of the 2-opt local search operator on an existing route

META-HEURISTICS

Meta-heuristics form a different category of solution methods which, generally, make use of classical heuristics. They are in essence higher-level methods which try to select certain classical heuristics that are able to provide a good solution to the problem at hand. In an attempt to improve on a single heuristic and to achieve the best results both in terms of solution quality and computation time, generally, multiple heuristics are used. Several ways to do this have been developed over the past decades.

Simulated Annealing (SA) is an optimisation method inspired by the annealing of metals [23]. In material science this is a technique which basically encompasses the heating of a metal after which it is slowly cooled back down. During the heating process molecules are able to reorder or re-orientate themselves which increases the size of the crystalline structures in the material. This is generally seen as improvement of the quality of the material. While cooling it back down, the molecules slowly become fixed and the new (better) material structure persists. In optimisation the solution is analogue to the molecules of the material. Classical heuristics or a set of mathematical operations represents the heat which is used to alter the solution (or the molecules). Furthermore the value of the temperature is analogue to the degree by which solutions with deteriorating performance are accepted. Therefore, the higher the temperature the more likely it is that these sets of operations which result in solutions with a worse objective value are accepted. This, initially high, decaying probability of the acceptance of worse solutions yields a minima-escaping behaviour which will eventually converge to a (local) optimum as the temperature decreases.

Tabu Search (TS) is a meta-heuristic which was developed by the mathematician Fred. W. Glover [24]. It also builds upon classical heuristics as it uses them to modify a provided initial solution. However, it differentiates itself from LS methods in two ways. First of all, it allows for worsening solutions to be accepted when no improvement can be found. Secondly, from where it lends it name, it remembers previously visited solutions and marks these as a *tabu*. This basically means it prevents itself from visiting these solutions again. With these two properties TS is able to remember and escape previously visited local minima. For more on TS and several examples of implementations for dynamic vehicle routing the reader is referred to an article by Branchini et al. [25].

Genetic Algorithms (GA) are meta-heuristics inspired by the process of natural selection. They originate from the work done by the American psychology professor John Holland [26]. These methods basically consists of three parts, namely: initialisation, selection and reproduction. During the first step a set of solutions is (randomly) generated. These solutions form the initial population. Then each solution is rated using a certain fitness function. In vehicle routing this might be the total distance travelled by all vehicles. Now, as happens in natural selection, only the best performing solutions survive (or are chosen) during the selection step. Using the obtained subset, new solutions are generated by combining parts of the different remaining solutions and adding some random permutations (as also happens with genomes in nature). The selection and reproduction steps are then repeated until a certain predefined stopping criterion is reached. For more on GA and their application to vehicle routing the reader is referred to an article by Hanshar and Ombuki-Berman [27].

Large Neighbourhood Search (LNS) is a meta-heuristic which was introduced by Shaw in 1997 [28]. As the name suggests this method tries to increase the search neighbourhood as compared to LS methods. It does this by relaxing a certain part of the decision variables in the solution when no further improvements can be found. This basically means that a certain part of the solution remains the same and the other part may completely be changed. This new problem is then re-optimised. When a better solution is found the process restarts. When the solution did not improve the method reverts back to the best known solution and selects a new neighbourhood to modify.

Some of the best results for vehicle routing problems, in terms of quality of the obtained solutions, have been found using a kind of LNS called Ruin & Recreate (R&R). To illustrate, in the year 2000 Schrimpf et. al [29] broke several records by combining a R&R approach with SA. A year later Li and Lim [30] introduced sets of PDPTW instances which they solved using a method which combined LS, SA and TS but did not make use of R&R. Since then records on these instances have been broken multiple times by methods which do make use of R&R. First notably in 2004 by Ropke and Pisinger [19] by using a R&R method which makes use of multiple removal and construction heuristics and a variable search neighbourhood. Then, again in 2006 by Bent and Van Hentenryck, [31] by combining LNS and SA. Finally, in 2016 and 2019, Christiaens and Berghe [32, 33] have shown to be able break almost every record by implementing a range of heuristics and introducing a new kind of removal method called string-removal.

Neural Networks

During recent years, neural networks, belonging to the field of research called machine-learning, have received a lot of attention. Primarily because they have outperformed classic approaches in speech recognition, machine translation and image captioning [34]. A unique property of this category of methods is that it imposes and requires little to no structure of the problem at hand. Instead this method tries to learn it by itself. This has both advantages and disadvantages. Advantages are that the learning process itself requires little human interaction and that it is almost instantaneous in run-time. Disadvantages are that it can be hard to "mold" the data in to a format which can be used to train a *neural network* and that the learning process requires a lot of computation power. Furthermore, for vehicle routing, in terms of solution quality it does not seem to rival with conventional solution methods. However, a recent preprint by Kool et al. does show promising results in terms of solving a TSP with up to a 100 nodes [35].

2.2. Dynamic Vehicle Routing

The Dynamic Vehicle Routing Problem (DVRP) is a variation of the vehicle routing problem which has received considerably more attention over the last three decades [36]. In contrary to static variations the dynamic vehicle routing problem tries to adapt to changing conditions (which in reality often happen every day). Since the first paper by Wilson and Colvin [37] numerous problem variations and solutions methods have been developed to handle all kinds of problems. This work focuses on additional customers requesting service vehicles are driving. Several concepts related to the DVRP are covered below.

2.2.1. Degree of Dynamism

Dynamic vehicle routing problems can be classified according to a widely accepted measure, namely the Degree of Dynamism (DOD). This measure is the number of initially unknown or immediate requests n_{imm} as a fraction of the total number of requests n_{tot} [38, 39]. The Effective Degree of Dynamism (eDOD) expands on this definition by taking into account how far ahead information becomes known before it should be acted upon [39, 40]. In this definition the time each immediate request becomes known is denoted as t_i and the entire planning horizon stretches from $[0, T]$. The *eDOD* is then defined as denoted by Equation 2.1.

$$eDOD = \frac{\sum_{i=1}^{n_{imm}} \frac{t_i}{T}}{n_{tot}} \tag{2.1}$$

For problems with time windows the *eDOD* can be expanded to the $eDOD_{tw}$ by defining the available response time r_i as a fraction of of the planning horizon. The response time is defined as $r_i = l_i - t_i$. Here l_i is the end of the time window. The $eDOD_{tw}$ is then defined as denoted by Equation 2.2.

$$eDOD_{tw} = \frac{\sum_{i=1}^{n_{imm}} 1 - \frac{r_i}{T}}{n_{tot}} \tag{2.2}$$

2.2.2. PERFORMANCE EVALUATION

The performance of solution methods for DVRP's can be evaluated using competitive analysis. This method was first introduced by Sleator and Tarjan [41]. In this method, online algorithm performance is evaluated by comparing it to the performance of an offline algorithm which directly has all information available. More formally, let P be a minimisation problem and S a set of instances of P. Let $c^*(I_{\text{off}})$ be the optimal cost for the *offline* instance I_{off} which holds for $I \in S$. In creating a solution for I_{off} all dynamic and stochastic information is available. This in contrary to an algorithm A used for DVRP's to which this information is revealed in real-time. Furthermore, let $x_A(I)$ be the final solution to instance I, generated by this algorithm A. Then, similar to the optimal cost, the cost to this solution can be defined as $c_A(I) = c(x_A(I))$. Now, the *competitive ratio* c_r of an algorithm A is defined by Equation 2.3 [39].

$$c_r = \frac{c_A(I)}{c^*(I_{\text{off}})} \qquad \forall \quad I \in S \tag{2.3}$$

This ratio basically quantifies the loss of efficiency stemming from the fact there is not full information. It is worth mentioning that this ratio requires the optimal solution $c^*(I_{\text{off}})$ which is therefore required to be solved using an exact method. This renders comparison impossible for many real-life instances as these are generally unsolvable using these methods.

To overcome this problem, often, a slightly modified ratio is used namely the ratio of *value of information* $V_A(I)$. This ratio is defined as stated in Equation 2.4.

$$V_A(I) = \frac{c_A(I) - c_A(I_{\text{off}})}{c_A(I_{\text{off}})} * 100 \qquad \forall \quad I \in S \tag{2.4}$$

This ratio, expressed as a percentage, describes the relative loss in solution quality as compared to the instance which had all information available from the start. In this work competitive analysis is used in Chapter 7 for the evaluation of the proposed method.

2.2.3. SOLUTION METHODS

A wide range of solution methods, ranging from linear programming techniques to meta-heuristics, as covered in Section 2.1.3, are used in the literature to solve DVRP's. [42–45]. However, depending on the degree of dynamism, maximum allowable computation time and scale of the problem instance certain methods might be more suitable than others. Unfortunately, no definitive conclusions have previously be drawn regarding an optimal solution method for certain problems. It can however, be stated that when dealing with large, loosely constrained problem instances which have to be solved within a limited amount of computation time, heuristics might be more suitable as compared to exact methods.

2.3. ANTICIPATORY ROUTING

Anticipatory routing describes a field of research related to the VRP where an attempt is made to anticipate additional requests and routes are adjusted according to the expectation. Three categories of methods are identified in the literature, namely anticipatory insertion, the multiple scenario approach and waiting- and relocation strategies. The three categories are described in the following subsections.

2.3.1. ANTICIPATORY INSERTION

The idea of anticipatory insertion is to change routes in advance by incorporating a representation of expected requests that will become known in the future.

An example is the method proposed by Hemert and La Poutre [46] which defines regions where requests are likely to occur as *fruitful regions*. By selectively routing vehicles trough these regions they modify existing routes and end up with different solution as compared to not doing this. They show that by using this method they are able to improve upon a purely reactive approach. It worth mentioning that their problem instances are constructed in such a way that they indeed contain *fruitful regions*, which their method is specifically designed to work on. Furthermore, additional requests are also assumed to arrive according to the same distribution which is used by their method to assess which regions should be routed through. Finally, for optimisation, they use a GA which solely makes use of an exchange LS operator for the mutation of solutions.

A slightly different method is proposed by Ghiani et al. [43]. They propose sampling requests from an assumed future arrival distribution and than adding these sampled requests to the current problem definition. This problem is then solved. Where additional requests should then be inserted is based on the solution which includes the sampled requests instead of on the one which does not. For optimisation they use sequential cheapest insertion and the shift and rearrange LS operators. They report "dramatic" improvements upon a purely reactive approach on synthesised problem instances with up to 300 customers. Finally it is worth mentioning that they "have assumed that the requests arrive according to a known stochastic process" and that "Future work should be aimed at verifying how robust our approach is whereas demand sampling is based on an approximation of the arrival process".

2.3.2. MULTIPLE SCENARIO APPROACH

The Multiple Scenario Approach (MSA) is a method proposed by Bent and Van Hentenryck [47] which expands upon anticipatory insertion methods. They propose a method which creates different *scenarios* by sampling requests from an assumed future arrival distribution multiple times. In each *scenario* a different set of requests is assumed to arrive in the (near) future and is added to the problem definition. These scenarios are then seperatly optimised in parallel. When an additional request arrives, the best insertion location within all scenarios is determined. The insertion location which is best most frequently, amongst 50 scenarios (according to a consensus function), is then chosen. For optimisation they make use of an unspecified local search method. On synthesised problem instances with up to 100 customers they report "dramatic" improvements upon purely reactive approaches. It is worth mentioning that, in obtaining these results, again the actual arrival distributions of future requests are used by the proposed method.

More recently Ghiani et al. [48] compared the performance of anticipatory insertion to the multiple scenario approach in solving the dynamic and stochastic TSP. They concluded that anticipatory insertion offers comparable performance to MSA while requiring less computational resources.

Finally, Saint-Guillain et al. [49] expand on the MSA with their Global Stochastic Assessment (GSA) approach which, instead of computing a set of solutions, creates a single solution which best suits all scenarios. They conclude that this method produces competitive results with respect to MSA. However they also report that their stochastic evaluation over 150 scenarios is approximately 10^3 more expensive (in terms of required computational resources) as their reactive approach which makes use of LS and SA.

2.3.3. WAITING- AND RELOCATION STRATEGIES

The idea of waiting strategies is to let vehicle wait at certain locations when it is estimated that additional requests will be made known near the current location of the vehicle. In such a situation it may be more appropriate to idle for a certain amount of time than to move away from the specific location and having to return to it at a later time. Thomas [50] shows that in a single vehicle case, on instances with up to 50 customers, waiting strategies which make use of stochastic information can improve upon strategies which do not. Branke et al. [51] show, that for the one and two vehicle case and instances with up to 200 customers, making use of the appropriate waiting strategy can significantly increase the probability of being able to service additional customers while, at the same time, reducing the average required detour for servicing customers. Ichoua et al. [44] show comparable results on relatively larger problem instances with up to 6 vehicles and 200 customers. To determine the waiting locations they segment space and time into cubes which each receive their own Poisson arrival distribution. When the probability of arrival within a cube is high enough, and vehicle currently resides within this cube, a "dummy" customer is generated which forces the vehicle to wait at that specific location. For optimisation they make use of TS which makes use of the cross exchange LS operator for the modification of existing solutions. They show a reactive approach can be improved upon, especially in the case of "critical" situations involving a small fleet size and high requests arrival rates. Finally, in obtaining these results, again the actual arrival distributions of future requests are used by the proposed method.

The idea of relocation strategies is to move vehicle to locations where requests are expected and no vehicles are available. These strategies are used in urgent problem contexts (where time windows are narrow), often to be able to service more customers as compared to not using them. Given that relocation causes additional distance to be travelled, employing this strategy often works contradictory to reducing operational cost. These strategies find their origin in ambulance dispatching. A variety of ambulance dispatching and relocation methods are covered by Andersson and Värbrand [52]. More recently, these methods have also found applications in other problem contexts such as the rebalancing of empty taxi's to be able to better service future demand [53]. It is worth mentioning that, for relocation, the method requires vehicles to be idle. Depending on the length of time windows of requests this may or may not ever happen.

3
Preliminaries

In this preliminary chapter the solved problem is formulated in mathematical terms in Section 3.1. Furthermore, an overview of the proposed method is presented in Section 3.2.

3.1. Problem Definition

In this section the definition of the problem that is solved is covered. This is done in two parts. First, the static problem definition is described in Section 3.1.1. Second, the actual problem, the dynamic problem, is described in Section 3.1.2 by making use of the static problem definition.

3.1.1. Static Problem

Formally, the static problem can be defined as follows. Let $G = (N, A)$ be a directed graph where A defines a set of arcs and $N = P \cup D \cup S \cup E$ defines a set of nodes. Let P define a set of pickup nodes, D define a set of the corresponding deliver nodes, S defines a set of vehicle starting nodes and E define a set of vehicle end nodes. Furthermore, R defines a set of requests of length n where each request r_i is defined by the pickup node N_i and deliver node N_{n+i}. Each pickup node N_i has an associated positive load q_i which should be picked up at that node. This load is required to be delivered to its corresponding deliver node N_{n+i} which means its associated load q_{n+i} has the same value, only negative. All pickup and deliver nodes also have a non-negative service duration d_i which defines for how long a vehicle should stay at node N_i before it is considered to be serviced. The service of each node N_i should start within a uniquely configurable time window starting at e_i and ending at l_i. Let V define a set of vehicles of length m where each vehicle V_k should start its route at starting node N_{2n+k} (or S_k) and end its route at end node N_{2n+m+k} (or E_k). Furthermore each vehicle V_K has a limited capacity equal to Q_k. Indirectly, each vehicle V_k also has a time window during which it is able to service pickup and deliver nodes. This time window is defined by the time windows of the start and end node of the route. More specifically, vehicles may only depart from their starting node N_{2n+k} (or S_k) after a time e_{2n+k} and should be at their end node before a time l_{2n+m+k} (or E_k). Lastly, all arcs in A from node N_i to node N_j are defined by an associated travel cost $c_{i,j}$ and travel time $t_{i,j}$.

The objective of the problem is to find a circuit, of minimal travelling cost, servicing all pickup and deliver nodes while meeting all constraints. The problem, using a three-index vehicle flow formulation, can be defined as denoted by Equation 3.1. In this formulation all symbols are defined as listed in Table 3.1.

The objective function denoted by Equation 3.1a calculates the total travelling cost. The total travelling cost is a summation over the routing cost of all possible arcs multiplied by a corresponding binary variable $x_{i,j}^k$ which indicates whether the arc is traversed or not. Specifically, the set of binary variables $x_{i,j}^k (i \in N, j \in N, k \in K)$ indicate which arcs from node N_i to node N_j are performed by each vehicle V_k. This objective function is subject to multiple constraints to ensure the problem definition is obeyed.

The first set of constraints 3.1b ensures that the set of variables $x_{i,j}^k$ can only be binary. Constraints 3.1c ensure that all arcs have different starting and end nodes. Constraints 3.1d ensure that each pickup is serviced a single time by a single vehicle. Constraints 3.1e ensure that each individual pickup and deliver node have as many arcs towards it as there are arcs originating from it. Constraints 3.1f ensure that the number of arcs originating from a deliver node by a vehicle is equal to the number of arcs originating from its corresponding pickup node by that same vehicle. Together constraints 3.1d, 3.1e and 3.1f ensure that each pickup is assigned to a single vehicle, that the corresponding deliver is assigned to the same vehicle and that continuous routes are formed.

Constraints 3.1g ensure that the starting node for each vehicle can only be serviced by its appropriate vehicle. Similarly, constraints 3.1h ensure that the end node for each vehicle can only be serviced by its appropriate vehicle. Constraints 3.1i, 3.1j ensure that a single arc originates from each start node and no arcs go towards them, forcing them to be the starting node for each route. Similarly, constraints 3.1k, 3.1l ensure that a single arc goes towards each end node and no arcs originate from them, forcing them to be the end node for each route.

Constraints 3.1m ensures that the time between the start of service of two nodes, when an arc occurs, is at least equal to the travelling time between the two nodes and the service duration of the first node. These constraints also ensure subtours are eliminated. Constraints 3.1n ensures that each pickup node is serviced before its corresponding deliver node. Constraints 3.1o ensure that the start of service for each node is within its defined time window. Constraints 3.1p ensure that the load on board of each vehicle after servicing a node is at least equal to the load it had on board at the previous node plus the load of the node it just serviced. Together with constraints 3.1q this ensures that the upper and lower bounds on the capacity of each vehicle are not exceeded.

This three-index vehicle flow formulation is basically the same as described by Laporte and Cordeau [54], however additionally, this formulation allows vehicles to have unique start and end locations, different from a single depot.

Finally, it is worth mentioning that even though this problem formulation can be used in conjunction with one of many available mixed-integer programming solvers, a solution method based on heuristics will be used during the evaluation in Section 7. This is because exact methods do not scale as has been described in Section 2.1.3. The latter means that this formulation is merely used as a way to convey the specifics of the static problem.

$$\min_{x_{i,j}} \quad \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} c^k_{i,j}x^k_{i,j} \tag{3.1a}$$

$$\text{subject to} \quad x^k_{i,j} \in 0,1 \qquad (i \in N, j \in N, k \in K) \tag{3.1b}$$

$$x^k_{i,i} = 0 \qquad (i \in N, k \in K) \tag{3.1c}$$

$$\sum_{k\in K}\sum_{j\in N} x^k_{i,j} = 1 \qquad (i \in P) \tag{3.1d}$$

$$\sum_{j\in N} x^k_{j,i} - \sum_{j\in N} x^k_{i,j} = 0 \qquad (i \in P \cup D, k \in K) \tag{3.1e}$$

$$\sum_{j\in N} x^k_{i,j} - \sum_{j\in N} x^k_{n+i,j} = 0 \qquad (i \in P, k \in K) \tag{3.1f}$$

$$x^k_{2n+l,j} = 0 \qquad (k \in K, l \in K \neq k, j \in N) \tag{3.1g}$$

$$x^k_{i,2n+m+l} = 0 \qquad (k \in K, l \in K \neq k, i \in N) \tag{3.1h}$$

$$\sum_{j\in N} x^k_{2n+k,j} = 1 \qquad (k \in K) \tag{3.1i}$$

$$\sum_{j\in N} x^k_{j,2n+k} = 0 \qquad (k \in K) \tag{3.1j}$$

$$\sum_{i\in N} x^k_{i,2n+m+k} = 1 \qquad (k \in K) \tag{3.1k}$$

$$\sum_{j\in N} x^k_{2n+m+k,j} = 0 \qquad (k \in K) \tag{3.1l}$$

$$B^k_j \geq (B^k_i + d_i + t_{i,j})x^k_{i,j} \qquad (i \in N, j \in N, k \in K) \tag{3.1m}$$

$$B^k_i + d_i + t_{i,n+i} \leq B^k_{n+i} \qquad (i \in P, k \in K) \tag{3.1n}$$

$$e_i \leq B^k_i \leq l_i \qquad (i \in N, k \in K) \tag{3.1o}$$

$$Q^k_j \geq (Q^k_i + q_j)x^k_{i,j} \qquad (i \in N, j \in N, k \in K) \tag{3.1p}$$

$$\max(0, q_i) \leq Q^k_i \leq \min(Q_k, Q_k + q_i) \qquad (i \in N, k \in K) \tag{3.1q}$$

Table 3.1: Symbols used in the vehicle flow formulation

Symbol	Description
V	Set of vehicles
V_k	Vehicle in set V
m	Number of vehicles in set V
Q_k	Capacity of vehicle V_k
G	Undirected graph consisting of nodes N and arcs A
N	Set of nodes in graph G
R	Set of requests
R_i	Request in set R
n	Number of requests in set R
P	Subset of N containing all nodes where a load needs to be picked up
D	Subset of N containing all nodes where a load needs to be delivered to
S	Subset of N containing a single starting node for each vehicle in V.
E	Subset of N containing a single end node for each vehicle in V.
N_i	Node in set N
$x^k_{i,j}$	Boolean which indicates the presence of an arc in the solution from node N_i to node N_j performed by vehicle V_k
A	Set of arcs from nodes N_i to nodes N_j in graph G which are represented by their associated routing cost $c_{i,j}$ and travel time $t_{i,j}$
q_i	Load being picked up or delivered at node N_i
d_i	Non-negative service duration at node N_i
$c_{i,j}$	Travel cost from node N_i to node N_j
$t_{i,j}$	Travel time from node N_i to node N_j
e_i	Earliest starting time at which service may start at node N_i
l_i	Latest starting time at which service may start at node N_i
$[e_i, l_i]$	Time window for start of service at node N_i
B^k_i	Time at which vehicle k started servicing node N_i
Q^k_i	Load of vehicle V_k after servicing node N_i
k_i	Time at which request i becomes known
S	Solution
R_s	Set of scheduled requests in solution S
$n_s = \|R_s\|$	Number of scheduled requests
$r_{i,s}$	Request i from set of scheduled requests
R_{us}	Set of unscheduled requests
$n_{us} = \|R_{us}\|$	Number of unscheduled requests
$r_{i,us}$	Request i from set of unscheduled requests
W	Set of routes
w_k	Route belonging to set W fulfilled by vehicle k
Z_k	Sequence of tasks (or nodes) fulfilled by route w_k
$z_k = \|Z_k\|$	Number of tasks (or nodes) fulfilled by route w_k
$z_{k,l}$	Task at position l in sequence Z_k fulfilled by route w_k
$es_{k,l}$	Earliest possible start of task at position l in sequence Z_k fulfilled by route w_k
$ls_{k,l}$	Latest possible start of task at position l in sequence Z_k fulfilled by route w_k
T	Time
R_a	Dynamic stream of additional requests that are made known at a certain time
R_f	Dynamic stream of requests that are (being) fulfilled at a certain time

3.1.2. DYNAMIC PROBLEM

The static problem described in Section 3.1.1 becomes dynamic with the introduction of, for each request, a time k_i at which the request becomes known and a time dimension along which these events occur. While progressing along this time dimension, because of the time at which requests become known, the set of requests R will become larger. This means the previously described static problem definition will change at discrete time events. Because of this, many updated static problem variations are required to be solved consecutively.

Furthermore, while the time T progresses, the time window during which a vehicle is available may start.

It may also happen that, as a result of solving a previous static problem variation, a solution is found where this vehicle is required to service several nodes. This means that a vehicle may already be driving around and may already have started servicing nodes. It is assumed that once service has started, it can not be undone (or it is not efficient to do so). This means that while time progresses and nodes are being serviced, additional constraints are being put on the solutions of future static problem definitions based on the solutions which were previously obtained and are being executed.

More specifically, the presence of an arc ($x^k_{i,j} = 1$) within future solutions is required when a vehicle has already started servicing the node the arc is going towards (thus if $T \geq B^k_j$). Furthermore it is also assumed that once a vehicle has started driving towards a node, the arc towards that node is also required to be present (thus if $T \geq B^k_j - t_{i,j}$). Given that requests consist of both a pickup and a deliver task which are required to be serviced by the same vehicle, an arc towards the deliver node is also required to occur, when a vehicle is constrained to service the corresponding pickup task. However, because of the structure of the problem formulation listed in Equation 3.1, this constraint will be met when an arc towards the pickup node is required to be present.

Because of the effects described in the previous two paragraphs, the structure of the dynamic problem, regardless of the used solution method, is visualised in Figure 3.1. In this visualisation it can be seen that time acts upon the world. As time progresses additional requests (R_a) become known. Furthermore, as time progresses, vehicles are driving, which at discrete time events generates requests that are being fulfilled (R_f). These vehicles do this based on a previous solution (S) which is created using a solution method (or solver). Based on the two information streams and the previous solution a new solution is required to be continuously generated.

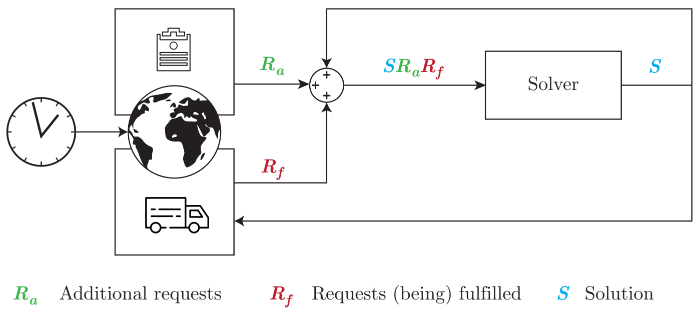


Figure 3.1: Dynamic problem structure

3.2. METHOD OVERVIEW

The related work presented in Chapter 2 shows that:

- Presented methods which incorporate historical data often require a significant amount of computational resources and have therefore only been evaluated on instances with up to 300 customers (and are generally evaluated on problem instances with up to 100 customers) [43, 46–49].

- Instead of deriving an arrival probability for additional requests from historical data, often additional requests are generated based on an assumed arrival probability which is also made available to the used method [43, 46–49]. This is not possible in any real-world application.

- Solving large instances of VRP's, with arbitrary time window lengths, to optimality, is currently not possible. However, using heuristics and a large amount of computational resources, relatively good results can be obtained [19, 29, 31, 32, 55].

- Presented methods which incorporate historical data often use a relatively small set of heuristics as compared to the state of the art in static and dynamic vehicle routing [43, 46–49].

In order to improve upon the current state to the art for solving large dynamic real-world problem instances, while making use of historical data, we propose to:

- Develop an anticipatory insertion method which only makes use of a single representation of reality and solely relies on historical data as input.

- Develop a dynamic vehicle routing solver which makes use of variable large neighbourhood search, a state of the art method for solving vehicle routing problems.

Specifically, the proposed method for improving upon a purely reactive approach consists of two parts. First, based on historical data, a representation for a set of requests which are predicted to occur later is created in advance. Second, using this representation, the solution of the DVRP is adjusted so that requests which actually appear later can be serviced more efficiently when they become known. How these two parts affect the dynamic problem structure shown in Figure 3.1 is visualised in Figure 3.2. The two parts, creating the representation (predicting requests) and adjusting the solution (incorporating predicted requests) are covered in Chapters 4 and 5. The design of the solver, the remaining component, is covered in Chapter 6.

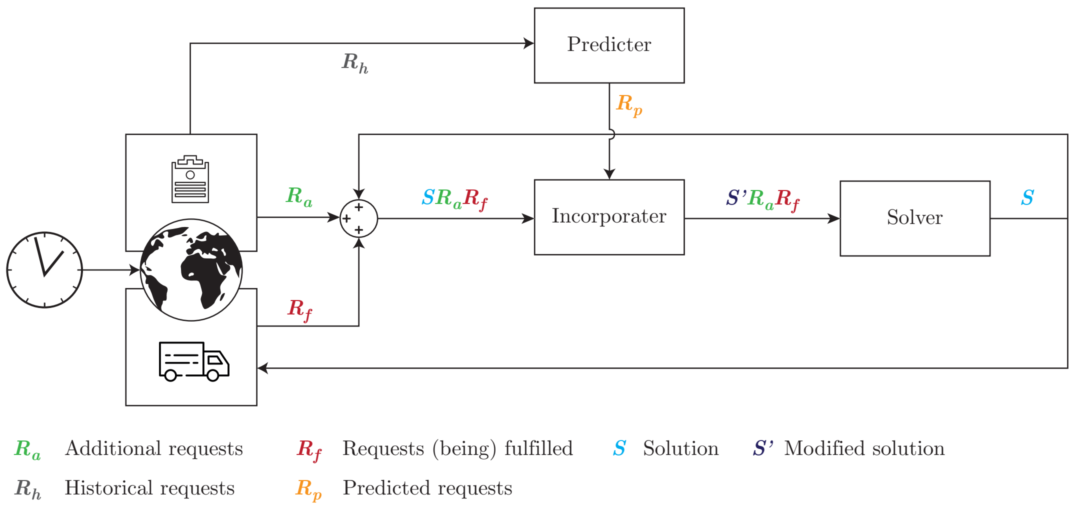


Figure 3.2: Method overview

4
PREDICTING REQUESTS

The goal of predicting requests is to find a representation of a set of requests which are currently unknown, will be made known and are required to be serviced within in a certain future time frame. As this representation will be used to steer routes of vehicles to specific locations at specific times, both the spatial and temporal properties of predicted requests should, to a certain degree, match with the actual later appearing requests.

Existing methods used for clustering and time series forecasting are by themselves only able to solve part of this problem. Therefore a structure, which combines several of these methods, is proposed. It is worth mentioning that this structure can be used for the prediction of arbitrary multi-dimensional objects which occur along a time dimension.

The general idea is to group historical requests which are similar so that it might be possible to reveal patterns in their occurrence across the time dimension. The method consists of three parts. First, requests are clustered on relevant dimensions to create groups named *request types*. Second, the number of occurrences of each request type is predicted using a prediction model. Then, by creating the same number of requests based on the requests belonging to a request type, predicted requests are generated. How request types are exactly generated and how predicted requests are generated using these requests types will be described in Sections 4.1 and 4.2. Finally, guidelines for how often such a prediction should be performed are covered in Section 4.3.

4.1. GENERATING REQUEST TYPES

Request types are generated by making use of clustering. As a wide range of clustering methods exists, an appropriate method should be chosen. Therefore an appropriate method for clustering multi-dimensional objects which occur along a time dimension is first chosen in Section 4.1.1. This chosen clustering method is briefly covered in Section 4.1.2. Specific measures that are used to improve the performance for the purpose of clustering requests are covered in Sections 4.1.3 and 4.1.4. Finally which parameters should be chosen when clustering is described in Section 4.1.5.

4.1.1. CHOOSING A CLUSTERING METHOD

To choose an appropriate method for clustering requests, several considerations should be taken into account. These considerations are as follows:

- When no knowledge about requests types or patterns within the data are available, supervised clustering methods can not be used. An unsupervised clustering method is therefore required.
- Given that the amount of request types that need to be generated is unknown, clustering methods which require the number of clusters as a parameter can not be used. This means the relatively fast and highly popular K-means clustering algorithm[56] or Ward clustering [57] are less appropriate.
- Requests may be called 'similar' when values across most or all dimensions do not differ much. This means that density-based clustering methods such as DBSCAN [58] or HDBSCAN [59], which allow wide non-spherical clusters to be created, may be less appropriate. This is because, for example, requests which have a very dissimilar pickup location may be clustered together. It may even be argued

that, when clustering requests, it is preferred to specify a maximum allowable dissimilarity for all requests belonging to the same cluster (or request type).

- To be able to cluster requests, the (dis)similarity of requests should be determined. Any implementation should either allow for the input of a (dis)similarity function so that values can be calculated while clustering or should allow for the input of a (dis)similarity matrix so that values can be pre-computed. However, when such a matrix is computed for hundreds of thousands of requests the available amount of random-access memory within most computers is quickly exceeded. For example, when dealing with a 100 thousand requests, an upper triangular (and therefore symmetrical) dissimilarity matrix contains 5 billion values. As the storage of a double in `C++` requires 8 bytes, storing such a matrix requires at least 32,25 gigabytes of memory. Requiring this amount of memory makes handling such matrices troublesome. Depending on the problem size, having an implementation available which allows to input a custom (dis)similarity function is therefore preferable or the only viable option.
- When large sets of historical requests are required to be clustered, methods which have a relatively low time complexity are preferred. It is worth mentioning that the number of methods which do scale well is very limited, when only considering implementations available for Python, as can be seen in Appendix B.
- When large sets of historical requests are required to be clustered, methods which already have been implemented very efficiently are preferred.

When taking these considerations into account, Hierarchical Agglomerative Complete Linkage Clustering (HACLC) is found to be most suitable for the purpose of clustering requests. This is because of the following reasons:

- It is an unsupervised clustering method and thus no prior information is required about the data.
- It does not require the number of clusters (or request types) as an input parameter.
- It is possible to specify a maximum allowable dissimilarity for all requests belonging to the same cluster (or request type).
- Even though the worst case time complexity of the implementation of $O(n^2 log(n))$, where n is the number of requests, is relatively high, several problem specific measures can be used, to improve clustering speed, as is described in the following section.
- A fast C++ based implementation for Python is available within SciPy [60].
- The implementation allows for the input of a custom (dis)similarity function whereby a (dis)similarity matrix, which might exceed random-access memory limits, does not have to be pre-computed.

The only real alternative, BIRCH clustering [61], is not chosen because the available implementations do require the computation of a (dis)similarity matrix. Also, the clustering result depends on the order of the inputted requests which is not favourable in terms of reproducibility.

4.1.2. HIERARCHICAL AGGLOMERATIVE COMPLETE LINKAGE CLUSTERING

In HACLC [62], at the beginning of the process, each object is assigned to its own cluster. Then, the two clusters which have the minimum distance (or maximum similarity) according to a linkage method, are merged into a new single cluster. For example, two requests which have similar pickup locations according to travelling distance between may be clustered together. With this modified set of clusters the process is repeated, when no stopping condition is supplied even until all objects belong to the same cluster.

The result of the clustering process can be visualised using a dendogram as shown Figure 4.1. It can be seen that all objects start as individual clusters at the bottom and have been merged into a single cluster at the top. Now, by defining a minimum similarity (or maximum dissimilarity), the number and composition of clusters can be chosen. In Figure 4.1 this choice has been visualised using the red dashed line. In this example six clusters are created.

Furthermore, complete linkage clustering states that the distance between two clusters is defined by the maximum distance between two objects belonging to either one of the clusters. This means that two clusters will only be merged if the maximum distance between all objects in the resulting cluster is below the chosen threshold.

Finally, with the mentioned time complexity and the fact that historical data-sets may quickly contain more than hundreds of thousands of requests, the computation time required for clustering can quickly grow beyond practically usable bounds. To overcome this problem, two measures to reduce the required computation time, are used. These measures are be covered in the following section.

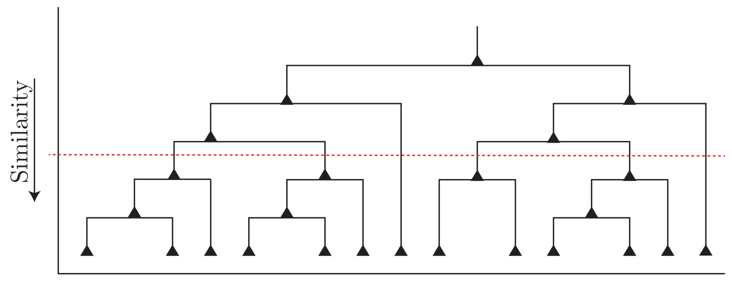


Figure 4.1: Visualisation of hierarchical clustering. The tree structure is called a dendogram. The red dashed line visualises the chosen measure of (dis)similarity at which the dendogram is 'cut off' which determines the number and composition of clusters

4.1.3. IMPROVING CLUSTERING SPEED

To speed up clustering, two measures, which may reduce the number of requests that need to be clustered simultaneously, are used. The first measure exploits the property of HACLC that the most 'similar' clusters are clustered into larger clusters initially. By only considering unique requests (and effectively already merging duplicate requests) the number of requests that need to be clustered is reduced while maintaining the same clustering result as compared to not doing this. It is worth mentioning that duplicate requests are not actually removed from historical data but they simply receive the same cluster label. Second, to reduce the number of requests that need to be clustered even further, clustering on multiple levels is used. This measure is covered in the next section.

4.1.4. CLUSTERING ON MULTIPLE LEVELS

When clustering requests, similarity might not be scaled uniformly across all dimensions. To still be able to cluster on these dimensions a custom function which scales certain dimensions for calculating the similarity of requests can be used [63]. However, it could be that a dissimilarity in a leading dimension means two requests are not similar, regardless of their similarity across other dimensions. For example, when the pickup location of two requests are not close to each other it may already be concluded that two requests are not similar, regardless of possibly similar deliver locations and time windows. To make use of this knowledge, clustering on multiple levels is used. Depending on problem context, it might be beneficial to simultaneously cluster on multiple dimensions, cluster sequentially on single dimensions or sequentially do a combination of both.

When clustering on multiple levels, a set of requests is first clustered on a portion of the relevant dimensions to separate the complete set into multiple subsets. These smaller subsets are then further separated. This process has been visualised in Figure 4.2. In this figure a set of requests is first separated along the "x" dimension where after individual subsets are then further separated along the "y" and "z" dimensions. By using this method it is possible to create subsets of requests which show similarity across all dimensions.

An advantage of clustering on multiple levels is that the number of requests which are being clustered simultaneously is (heavily) reduced at each level. This makes the usage of clustering methods with higher time complexities viable or more efficient depending on the number of requests that need to be clustered.

A disadvantage of clustering on multiple levels is of course that when a separation has been made on a higher level based on one dimension, requests will not be transferred to another subset even though this might be more logical based on other dimensions at lower levels.

It is worth mentioning that when a set of requests is separated again and again, very few requests might remain within certain subsets. When this happens it might be impossible to discover any pattern across the time dimension, regardless of the used prediction model. As the problem allows for a partial prediction (in the sense that only a certain part of the requests is predicted and others remain unexpected) it might be beneficial to ignore these subsets, increasing computational efficiency. In Figure 4.2 ignoring these subsets, possibly on different levels, has been visualised with red crosses.

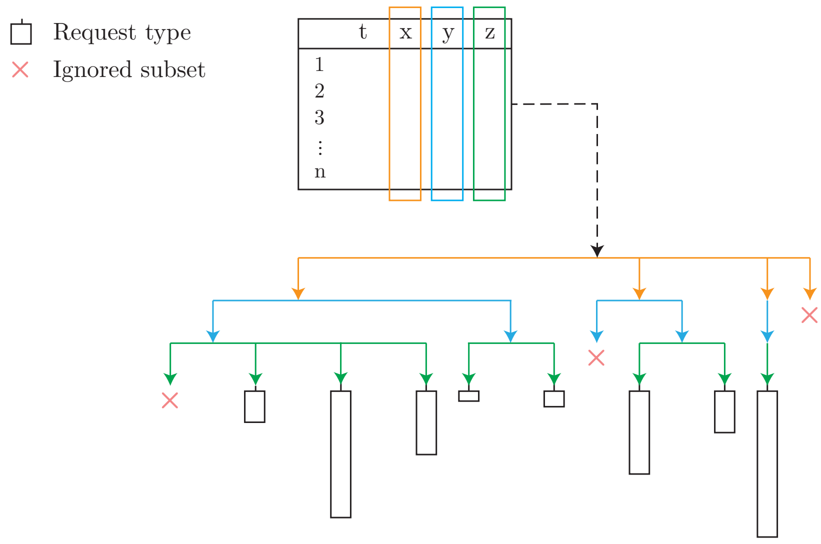


Figure 4.2: Clustering multidimensional data on multiple levels. To small remaining subsets of the data may be ignored at different levels. This is visualised with red crosses.

4.1.5. CLUSTERING PARAMETERS

The methods described in the previous sections have several unspecified parameters for the clustering of requests. The first parameter is if requests should be clustered using multiple levels. Second, for each individual cluster level several parameters should be specified. The required parameters for each cluster level are as listed in Table 4.1.

Table 4.1: Parameters to be defined at each clustering level

Column	**Description**
Dimensions	Which dimensions or parameters of a request are used for clustering at this level
Distance function	A function which defines how the dissimilarity is computed based on the parameter values along the dimensions that are used for clustering. For example when a pickup location is expressed as a latitude and longitude coordinate set, the distance between requests could be computed using the great circle distance [64] between the two points on earth.
Threshold	The maximum allowable dissimilarity to use while clustering at this level. The threshold should be expressed in the same unit as the value which is computed by the distance function.

To set these parameters appropriately an analysis of the historical data and the context of the problem is required. Therefore, no statements regarding setting cluster parameters are presented in this general method description. However, a problem specific analysis, in which parameters are determined for the instances described in Section 7.1.1, is presented in Sections 7.1.2 and 7.1.3. This analysis may serve as inspiration for setting the described parameters appropriately in other problem contexts.

4.2. GENERATING REQUESTS

The process of generating requests using requests types consists of two parts. First, using a prediction model, for each request type, the number of occurrences during a future time frame is predicted. Second, this number of representations is generated based on the requests belonging to the requests type. To do this, a prediction model should first be chosen. How a prediction model is chosen is described in Section 4.2.1. The chosen prediction model is described in Sections 4.2.2 and 4.2.3. Finally, how representations are created is described in Section 4.2.4.

4.2.1. CHOOSING A PREDICTION MODEL

A prediction model is required to predict the total number of occurrences of requests, belonging to a certain request type, within a future time frame. To do this all requests which belong to a certain object type are grouped into comparable time frames. For example, when the goal is to predict the number of requests occurring tomorrow, requests are grouped into days which yields a table of occurrences for each date. For such data structures a variety of time-series forecasting methods are available to predict values for future time frames. Amongst these methods are:

- Naive approach (where $\hat{y}_{t+1} = y_t$)
- Simple average
- Moving averages
- Holt-Winters Method [65]
- Autoregressive Moving Average (ARMA) models [66]
- Autoregressive Integrated Moving Average (ARIMA) models [66]
- Neural Networks [67] [68]

As can be experienced during rush-hour in any city, the amount of transportation is not constant during the entire day. Something similar can be seen across the week, often, at least, the amount of transportation is different during weekends. This means for a prediction of requests to be accurate it should probably make use seasonality and should probably be able to deal with strongly varying time-series. Because of these reasons the first three methods listed above can be ruled out.

Therefore only the Holt-Winters and ARIMA (which also can represent an ARMA model when the differencing term is set to zero) are investigated further. When investigating seasonal ARIMA models it is found that the quality of the prediction is determined by the autoregressive (P), differencing (D), moving average (Q) terms and seasonality parameter (m). Even when the seasonality parameter is set to a weekly pattern which appears to be present within the problem instances (described in Section 7.1.1), the other three are still required to be chosen. Given that each request type requires its own model and thousands of request types may be present, these parameters are required to be chosen fully automatically within a relatively small amount of computation time. A method proposed by Hyndman and Khandakar [69] is used to do this by making use of the implementation in the `pmdarima` package available for Python. The same is done for the Holt-Winters model by making use of the exponential smoothing method available within the `statsmodels` package. It is observed that both methods predict approximately three times the amount of request that are actually made known. Even when rounding down the predicted amounts, in an attempt to make predictions more conservative, still more than double the number of requests is predicted.

Furthermore, neural networks are not used as these methods also often also require a significant amount of tuning and computation time before good results are obtained [68].

Therefore, even though the evaluated methods are being used very successfully in other problem contexts, it appears that results are not very promising when being used for the prediction of the number of occurrences of individual request types without any human interference. Because of this it is chosen to use a relatively straightforward prediction model which also has a parameter that allows for the conservativeness of the prediction to be chosen. The method, which makes use of the Relative Frequency of Occurrence of requests is described in the following sections.

4.2.2. RELATIVE FREQUENCY OF OCCURRENCE

The Relative Frequency of Occurrence ($\mathtt{rf}_o$), or also called empirical probability, is the ratio of the number of outcomes in which a specified event occurs to the total number of trials. In predicting requests the number of trials is determined by the number of comparable time frames available in historical data. For example, when the goal is to predict the number of requests occurring tomorrow, historical data can be grouped into days which would yield a table of occurrences for each date. The number of outcomes in which a specified event occurs is then defined by the number of days a request of a specific request type occurred. For example, if a request type occurred a single time on six days within a historical data set with a length of one week then the $\mathtt{rf}_o$ for the future time frame equals $\frac{6}{7} \approx 0{,}86$.

Furthermore, a different $\mathtt{rf}_o$ can be obtained by making use of assumed patterns within the data. For example, when a weekly occurrence is assumed and the future time frame is a Thursday, the number of trails may be limited to only contain previous Thursdays instead of all (week)days. If there is indeed a weekly recurrence present this yields a much higher $\mathtt{rf}_o$. When using multiple patterns, multiple $\mathtt{rf}_o$'s are obtained whereof the maximum is chosen for each predicted request.

$\mathbf{T_{-4}}$	$\mathbf{T_{-3}}$	$\mathbf{T_{-2}}$	$\mathbf{T_{-1}}$	$\mathbf{rf_o}$	$\mathbf{T_{-0}}$
2	3	2	1	**1.00**	**1**

$\mathbf{T_{-4}}$	$\mathbf{T_{-3}}$	$\mathbf{T_{-2}}$	$\mathbf{T_{-1}}$	$\mathbf{rf_o}$	$\mathbf{T_{-0}}$
2	3	2	1	**1.00**	**1**
1	2	1	0	**0.75**	**2**

$\mathbf{T_{-4}}$	$\mathbf{T_{-3}}$	$\mathbf{T_{-2}}$	$\mathbf{T_{-1}}$	$\mathbf{rf_o}$	$\mathbf{T_{-0}}$
2	3	2	1	**1.00**	**1**
1	2	1	0	**0.75**	**2**
0	1	0	0	**0.25**	**3**
0	0	0	0	**0.00**	**3**

Table 4.3: Example of how the $\texttt{rf}_o$ is determined based on historical time frames T_{-4} until T_{-1}. In the first four columns the number of occurrences of requests, belonging to a single request type, are listed. In the last column the number of expected requests within a certain future time frame T_{-0} is listed. On each row the $\texttt{rf}_o$ for the additionally predicted request is listed in the fifth column. Left: determining the $\texttt{rf}_o$ of the first predicted request. Middle: determining the $\texttt{rf}_o$ of the second predicted request. Right: determining the $\texttt{rf}_o$ of the third and final predicted request.

4.2.3. Predicting the Number of Occurrences

It may happen that historical time frames contain multiple occurrences of a single request type. When this is the case, multiple occurrences of that request type should, ideally, also be predicted for the future time frame. To enable this, the number of occurrences within historical time frames is updated each time an additional request is predicted. More specifically, after each prediction, the number of occurrences within each historical time frame is reduced by one. This means that the $\texttt{rf}_o$ for subsequent predicted requests belonging to the same request type will eventually reach zero. When the $\texttt{rf}_o$ reaches zero, no further requests are predicted. An example of this process has been shown in Table 4.3. In this example three requests are predicted for a future time frame T_{-0} based on the occurrences within four historical time frames T_{-4} until T_{-1}. It can be seen that each predicted request receives a different $\texttt{rf}_o$.

Therefore, by defining a minimum required $\texttt{rf}_o$ the number of predicted requests can be controlled. Having this threshold allows for the prediction of a small number of requests which all have a relatively high $\texttt{rf}_o$ or the prediction of a larger number of requests where some may have a relatively low $\texttt{rf}_o$. Generally, lowering this threshold allows to create predictions with more true positives but at the cost of having (even) more false positives.

4.2.4. Creating Representations

After clustering a set of requests into a request type and predicting the number of occurrences for that request type, actual predicted requests can be generated. Two simple methods for creating representations of request types are:

- Taking the mean of all requests belonging to the request type
- Random sampling from the requests belonging to the request type

Given that HACLC creates clusters whereof all requests are within a certain specified distance from each other, circular shaped clusters are created when using spatial data. As it can be argued that a circular shaped cluster can be best represented by its centroid, the mean of all requests belonging to a request type is used as a representation. However, as the obtained location might not be a node of the road network, the nearest node or path of the road network is chosen for route calculation.

4.3. Horizon and Frequency

The prediction horizon determines for how far into the future, requests should be predicted. The prediction frequency determines how often this set of predicted requests is updated. Determining the ideal prediction horizon and frequency depends on the specifics of the problem context. However, when choosing a long prediction horizon, the time frame for which requests are being predicted naturally becomes longer.

When the used time frame is longer the method will basically look further ahead. This means an attempt will be made to take into account requests which will become known in a relatively long time from now when creating solutions for requests which are currently known.

It makes sense to assume that the further away requests become known in the future, the less effect their presence will have on any routes in the near future. This means that, in general, increasing the prediction horizon beyond a certain amount might not have any beneficial effect.

Even more so, when the arrival rate of additional requests is relatively high, increasing the length of the prediction horizon means a relatively large amount of additional requests are taken into account simultaneously. If a large amount of requests is added to a vehicle routing problem, the complexity of solving the problem increases tremendously. When dealing with limited available computation time this might cause a solver to come up with worse solutions as compared to taking a smaller number of predicted requests into account. Predicting a smaller number of requests and thus having a shorter prediction horizon might therefore be beneficial (or might even be required to remain within solvable problem sizes in certain contexts).

For every problem context both the prediction horizon and prediction frequency should therefore be chosen so that, at any point, a limited amount of predicted requests reside within the problem that is being solved. This amount should roughly be in the order of a several hundred to a thousand requests given that the current limit for solving a pickup and deliver vehicle routing problem (even in a static context, without computation time constraints) is less than a few thousand requests.

5 INCORPORATING PREDICTED REQUESTS

This chapter will cover how the requests that were predicted, using the method described in Chapter 4, are used to alter the solution of a DVRP so that later actually appearing requests can be served more efficiently when they become known. As shown in Figure 3.2, this part of the method assumes that the current solution (S), the additional requests (R_a), the fulfilled requests (R_f) and the predicted requests (R_p) are available as input. Transforming this input into the output composed of a modified solution (S'), a set of modified additional requests (R'_a) and the unmodified set of requests which are (being) fulfilled (R_f) consists of three parts. These three steps will be described, more in detail, in the following sections.

5.1. ADDING PREDICTED REQUESTS

Each time requests are predicted these requests are all added to the set of additional requests to create the modified set of additional requests ($R'_a = (R_a, R_p)$). By doing so, predicted requests are mixed together with already made known requests which actually need to be fulfilled. Therefore, when the updated static problem is solved by a vehicle routing solver, the predicted requests will be seen as 'regular' requests. Because of this, during the optimisation process, simply by the presence of these predicted requests, routes will be adjusted so that these predicted requests can also be fulfilled efficiently.

5.2. REPLACING PREDICTED REQUESTS

The process of adding predicted requests works relatively straightforward as described in the previous section. The process of, at some point, removing these predicted requests to accommodate for additional requests which are actually made known is however less trivial.

Essentially, when the predicted requests would be removed after generating a solution, routes with room for servicing additional requests would be obtained. However, when requests are removed from the problem definition and any re-optimisation would be performed, this room and the structure of the previously obtained routes would almost definitely be lost in the process of finding a better solution. Therefore, predicted requests can not simply be removed at any time to accommodate for additional requests which are actually made known. To prevent losing the structure of the solution imposed by the presence of the predicted requests, each individual predicted request is preferably removed only when it can directly be replaced by a similar additional request which was made known.

This poses a challenge as each time an additional request is made known, it should be determined if a predicted request should be removed and if so, which one. In other words, it should be checked if a predicted request, which is 'similar', is part of the current solution and if so, this request should be removed. To enable this operation, a request classifier is introduced.

This classifier makes use of the generated request types and cluster levels which were described in Section 4.1. Given that all request types are distinguishable and all predicted requests are based upon them, they can be used to match additional requests to predicted requests.

The developed classifier, which tries to assign a request type label to an additional request, is listed in Algorithm 1. The procedure starts with all generated request types being a candidate for matching with the additional request. Then, while looping over all cluster levels, the procedure tries to narrow down the number of candidate request types by removing them when the additional request is too dissimilar to the centroid of a request type as defined by the distance function and threshold at each cluster level. It may happen that an additional request is dissimilar to all request types. When this happens the additional request is considered to not match with any predicted request and thus no predicted request is removed. When a single request type remains as a candidate, a predicted request belonging to that request type is removed if one resides within the current solution. If multiple request types remain as candidates, the one for which the centroid is closest to the additional request is chosen as the final candidate. Only a request belonging to that final request type is attempted to be removed.

Algorithm 1 Classifying additional requests

```
1:  procedure CLASSIFYINGADDTIONALREQUESTS(requestTypes, clusterLevels, additionalRequest)
2:      for clusterLevel ∈ clusterLevels do
3:          for requestType ∈ requestTypes do
4:              $d$ ← distance(centroid(requestType),additionalRequest, dimensions(clusterLevel))
5:              if d > 0.5 threshold(clusterLevel) then
6:                  requestTypes ← removeFrom(requestType, requestTypes)
7:              end if
8:          end for
9:      end for
10:     if length(requestTypes) == 0 then
11:         return 0
12:     else if length(requestTypes) == 1 then
13:         return label(requestType[0])
14:     else
15:         return label(closest(requestTypes, additionalRequest))
16:     end if
17: end procedure
```

If a predicted request is indeed removed, the corresponding additional request is directly inserted using sequential cheapest insertion (a common insertion heuristic of which the implementation is covered in Section A.1.2). This insertion creates a modified solution S'. Furthermore, the additional request is removed from the modified set of additional requests R'_a. Given that a predicted request which is similar to the corresponding additional request was removed, it is likely that its pickup and deliver tasks will be inserted at the locations whereof the predicted request was previously removed from. However, given that the two requests are almost never exactly the same, an insertion at the described locations might not be feasible due to time or capacity constraints. This means that even though a replacement is attempted, the inserted request might sometimes still end up at a different location within the solution.

5.3. REMOVING PREDICTED REQUESTS

When predicted requests are only being replaced, a certain portion of them is likely to never be removed from the created solution. This because it is unlikely that a prediction will, for each request type, contain not more requests than additionally will be made known. In other words, some more requests of a specific request type will often be predicted. Therefore, it is likely that some predicted requests will remain within the solution when only using the two previously described methods for adding and replacing predicted requests.

If vehicles are made to realise a proposed solution this means that, at some point, a vehicle will be directed to drive towards the pickup location of a predicted request. It can be both detrimental and beneficial to do this depending on the fact if a similar additional request will still be made known while driving towards that location.

When a similar additional request is made known, this request can directly be fulfilled which was concluded to be most efficient according to previously performed optimisations. Even more so, when a request has very narrow time constraints, being in the proximity of the pickup location might even be a requirement for being able to fulfil the request at all. In such a situation a vehicle is essentially being relocated to allow fulfilment of a future request. Deciding not to remove predicted requests can therefore be seen as a variation upon the relocation strategies introduced by Bent and Van Hentenryck [70].

When a vehicle is relocated and an appropriate request is not made known, additional distance might be driven for no reason. In this case the decision of not removing a predicted request results in behaviour that is contradictory to the objective of minimising the total distance driven by all vehicles. Furthermore, it can be argued that it makes little sense to prioritise a possibly occurring request over requests that might are already be known and can possibly also be fulfilled by the same vehicle. Postponing fulfilling these requests may in the end require additional vehicles to fulfil all requests within their time constraints. Again, the consequences of doing so might be contradictory to the objective of minimising the total distance driven by all vehicles.

Choosing whether remaining predicted requests should be removed therefore depends on the specifics of the problem context, the accuracy of the prediction and the reticence of the prediction. However, given that the urgency of requests in the evaluated problem context is generally not very high (as will be described in Section 7.1.2), it chosen to remove predicted requests. Both methods described in the following subsections are used to do so.

5.3.1. MADE KNOWN REMOVAL

First, predicted requests can be removed when the current time exceeds the time at which they were supposed to be made known ($T \geq k_i$). The reasoning behind this is as follows. A predicted request is supposed to be similar to an additional request. This means that the time at which a predicted request is made known (even though it is inserted earlier) is also similar to the time the additional request is estimated to be made known.

When no additional request will appear anymore it means the predicted request will be removed relatively early. This means that possible commitments, that may have been made, due to the structure of the solution imposed by the presence of the predicted request, may be kept to a minimum. Given that the solution is subject to constant re-optimisation this allows, relatively early, for a change in the solution structure if beneficial. Even when such a change is made and an additional request is still made known a little later, re-optimisation allows the solution to revert back to the previous structure. Naturally, this is only the case when no obstructing commitments have been made in the mean time.

5.3.2. TO BE REALISED REMOVAL

Even when using made known removal in might still happen that a vehicle will be directed to drive towards the pickup location of a predicted request. This may happen when the moment at which a request is made known is close to the start of the pickup time window and the travel time towards the pickup location is relatively long. To prevent this from happening, predicted requests are also removed when a vehicle is about to start travelling towards its pickup location. When some additional slack is introduced and predicted requests are removed even a little earlier, re-optimisation may still be possible so that requests sequenced after the to be removed request may be scheduled more favourably.

6 VEHICLE ROUTING SOLVER

This chapter describes how a vehicle routing solver is designed specifically for the purpose of this research. As shown in Figure 3.2 this solver represents one of the three components of the proposed method for solving the problem described in Chapter 3.1. First, some general design considerations are described in Section 6.1. Then, the actual design of the solver is described in Section 6.2. To check if the solver can be appropriately used for one the three components in the proposed method, the performance of the implementation is validated in Section 6.3. Finally, a developed visualisation tool for this vehicle routing solver is presented in Section 6.4.

6.1. CONSIDERATIONS

As described in Section 2.1.3, some of the best solutions ever found for a range of vehicle routing problem have been found by making use of Ruin & Recreate (R&R) methods [19, 29, 31, 32, 55]. It is therefore decided to base the design of the developed solver on this method.

Given that the idea of R&R methods is to make (educated) guesses on which part of the solution should be ruined and then reconstructed, there is generally no guarantee of obtaining a better solution during a single iteration. However, it has been shown that when many of these iterations are performed often small improvements can be found which gradually lead to some of the best solutions ever found for a range of problem instances. One goal for a good DVRP solver, which makes use of R&R methods, should therefore be to perform as many iterations as possible within the limited available computation time. As more iterations are the only way towards better solution quality, speed is important.

Furthermore, some of the latest record breaking solutions for static vehicle routing problems have been found by implementing a range of R&R heuristics [19, 29, 31, 32, 55]. In solving static problem variations it therefore appears to be beneficial to implement different kinds of heuristics to be able to increase the chances of being able to escape specific local minima. However, when solving dynamic problem variations, the available computation time is often in the range of seconds to several minutes instead of up to multiple hours. This means there is simply not enough time to extensively let a wide variety of heuristics attempt to improve a certain solution. Also, implementing additional heuristics, in a very computationally efficient manner, requires a lot of time. Because of these reasons, it is decided to implement a limited range of construction, LS and removal heuristics and to make these as fast as possible.

In that sense the design of the solver is chosen to be rather similar to the design as described by Ropke and Pisinger [19]. However, their approach is improved upon or deviated from in several ways. The specifics are covered in the following section which describes the design of the solver.

6.2. DESIGN

To solve VRP's the solver goes through three phases namely construction, LS and R&R. Furthermore, the R&R phase is parallelised to be able to perform as many iterations as possible on all modern hardware. Therefore, the general structure of the solver is as shown in Figure 6.1. In this figure each block represents a process and blocks stacked upon each other occur in parallel. Which methods are used during each individual process is described in the following subsections.

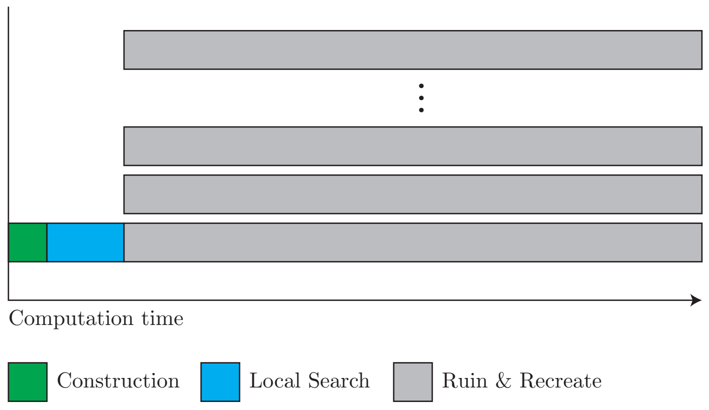


Figure 6.1: Phases of the developed vehicle routing solver. Each block represents a process. Vertically stacked blocks are processes that are executed in parallel. The Ruin & Recreate phase is parallelised to use modern hardware to its full potential.

6.2.1. CONSTRUCTION

A construction method is required to create an initial solution. All construction methods used by Ropke and Pisinger [19], namely sequential cheapest insertion, parallel cheapest insertion and regret insertion are implemented. Furthermore, a new construction heuristic is developed which makes use of the relatedness of requests and tries to insert requests which are clustered together into a single route. Therefore, this method is named cluster insertion. How this construction heuristic and the construction heuristics from the literature exactly work and are implemented is described in Section A.1. The often used Clarke and Wright Savings [71] heuristic is not implemented because it is not compatible with the problem variation defined in Section 3.1. The heuristic is not compatible because vehicles are allowed to have different starting- and end locations. Finally, when using sequential cheapest insertion, unscheduled requests are first sorted from longest to shortest travel distance (from the pickup to the deliver location) in an attempt to start with scheduling the 'hardest' requests first. The order of the unscheduled requests is not of importance for the other three construction heuristics.

To determine the most appropriate default construction heuristic, the methods are compared. Given that these methods are also required for the R&R procedure described in Section 6.2.3, spending a significant amount of time to efficiently implement these methods can be justified which makes a relatively good comparison possible. To compare the performance of the four construction heuristics both in terms of the quality of the obtained solutions as in the amount of computation time that they require, several experiments are ran on 58 instances with approximately 500 requests which require up to 100 vehicles. These instances and the hardware that these experiments are run on are described in Section 6.3.1. The results of the experiments are presented in Figures 6.2 and 6.3. The two sub figures in Figure 6.2 each show the distribution of the percent deviation from the corresponding best known solutions for one of the two objectives, for each construction heuristic. It can be seen that the sequential cheapest insertion heuristic performs worst in terms of both objectives. The parallel cheapest insertion heuristic performs best in terms of the number of obtained routes and distance travelled. However, cluster insertion closely matches the performance of parallel cheapest insertion in terms of the distance travelled. When looking at computation times it can be seen that sequential cheapest insertion and cluster insertion show comparable performance. The parallel cheapest insertion heuristic is approximately 10 times slower and two regret insertion heuristic is approximately a 100 times slower.

Given that the main objective of the instances that are evaluated in Chapter 7 is to minimise the distance travelled and the developed cluster insertion heuristic shows performance comparable to that of parallel cheapest insertion while requiring approximately 10 times less computation time, this heuristic is chosen for the construction phase. By merely using this construction heuristic solutions are obtained on average in 126 milliseconds. These solutions require, on average, 73,86% more distance to be travelled as compared to the corresponding best known solutions.

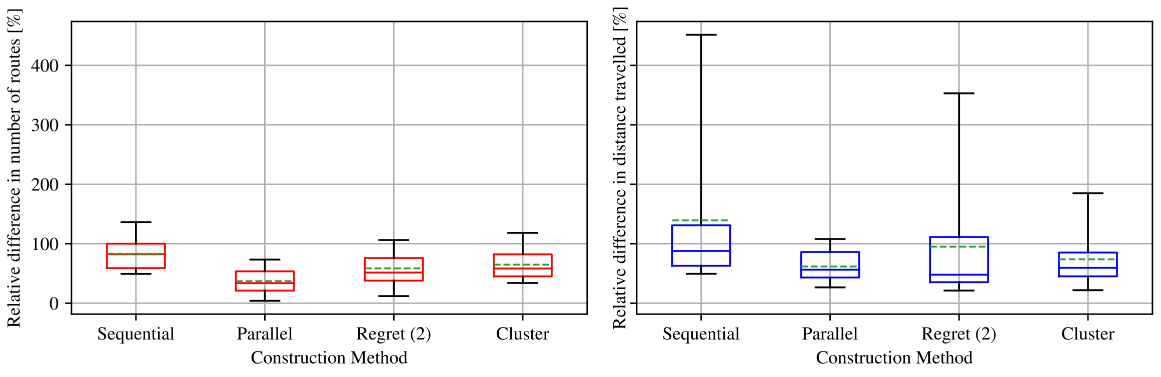


Figure 6.2: Performance comparison of different construction heuristics. On the left the obtained relative differences in the number of routes as compared to the best known solutions, for instances of approximately 500 requests, when only using different construction methods. On the right the obtained relative differences in the distance travelled as compared to the best known solutions, for instances of approximately 500 requests, when only using different construction methods.

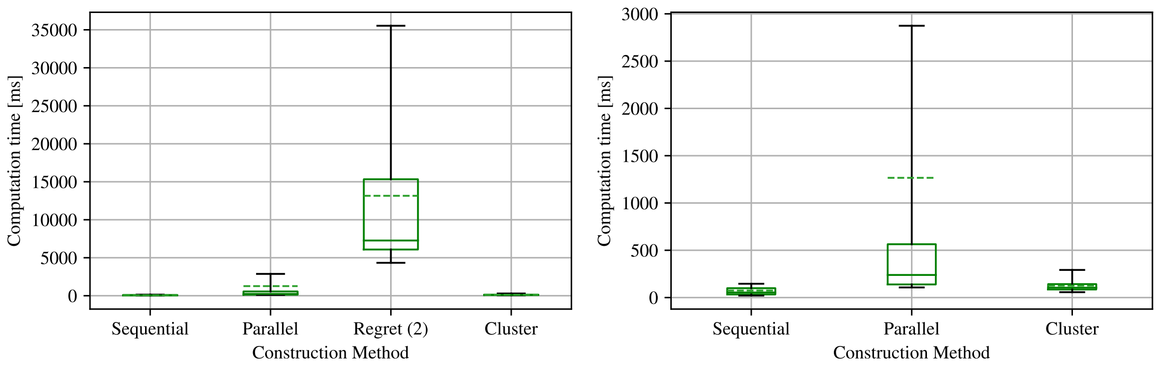


Figure 6.3: Performance comparison of different construction heuristics. Distributions of the required computation time, for instances of approximately 500 requests, when only using construction methods. On the left including regret insertion. On the right without regret insertion to better show the differences between the remaining methods.

6.2.2. Local Search

Initially it was decided to recreate the solver developed by Ropke and Pisinger [19] given that is showed good results while only using a relatively small range of heuristics. However, in an attempt to improve upon their results it is decided to implement some LS operators as well (given that they did not make use of LS). In choosing which LS operators should be implemented the required time for proper implementation and the effect that operators might have on the quality of the obtained solutions are weighed. It is estimated that the rearrange and shift operators (as visualised in Figures 2.5 and 2.4) may have a considerable impact on the quality of the obtained solutions. This is because the relocation of individual requests is something that the described R&R method is not able to do which means this operator may be able to improve certain solutions where R&R might not be able to do so. Furthermore, given that methods for unscheduling and rescheduling requests are also required to be efficiently implemented for the R&R procedure, this operator can be implemented relatively easily by reusing certain parts of the implementation. The same holds for the two-opt operator (as visualised in Figure 2.7) as only a section of a route is required to be reversed and feasibility of the solution needs to be checked. Furthermore, the reversal of a set of nodes is also something which the R&R procedure is not able to do except by unscheduling an entire route. The also popular exchange operator is not implemented because the number of possibilities that is required to be considered during each iteration is roughly n^2 where n is equal to number of requests. Given that, in roughly the same amount of computation time, thousands of R&R iterations may be be performed, which may result in the same exchange operations, instead it is decided to put additional effort into making the implementation as fast as possible. How the LS operators are exactly implemented is described in Section A.2.

To evaluate the effectiveness of the implemented LS operators the same experiments as described in Section 6.2.1 are run while now going through both the construction and LS phase. The results of the experiments are shown in Figures 6.4 and 6.5. For comparison, the results obtained using cluster insertion are also added. Again, the two sub figures in Figure 6.4 each show the distribution of the percent deviation from the corresponding best known solutions for one of the two objectives, after going through the two phases. It can be seen that by using LS operators, solutions are obtained on average in 1544 milliseconds. These solutions require, on average, 17,85% more distance to be travelled as compared to the corresponding best known solutions. This means that by adding roughly a second worth of LS, much better solutions can be obtained as compared to only using the slightly slower parallel cheapest insertion heuristic for construction (which would require approximately the same amount of computation time). These results therefore justify the chosen construction heuristic and appear to be worth the time used for the implementation of the local search operators.

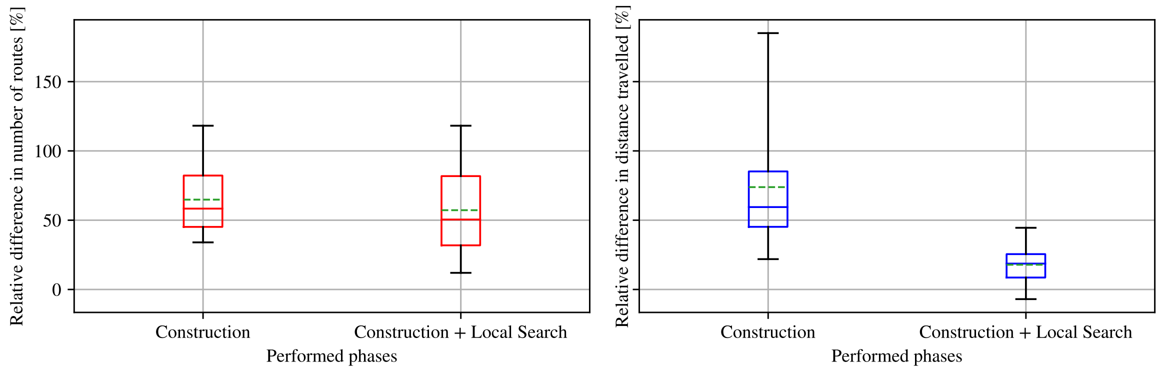


Figure 6.4: On the left the obtained relative differences in the number of routes as compared to the best known solutions, for instances of approximately 500 requests, when going through different phases. On the right the obtained relative differences in the distance travelled as compared to the best known solutions, for instances of approximately 500 requests, when going through different phases.

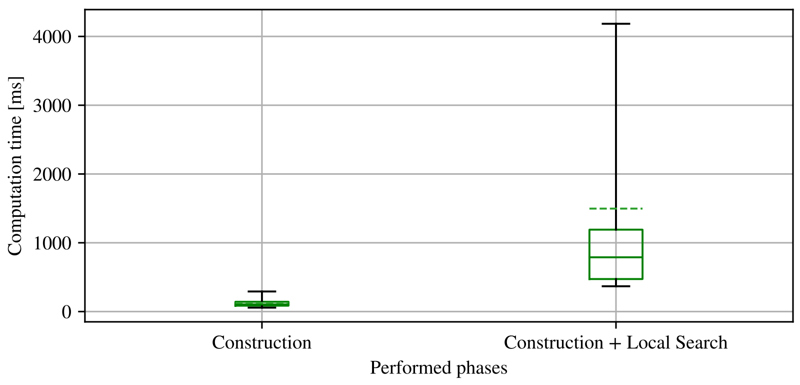


Figure 6.5: The required computation time, for instances of approximately 500 requests, when going through different phases.

6.2.3. RUIN-AND-RECREATE

The Ruin & Recreate (R&R) procedure is also inspired by the procedure described by Ropke and Pisinger [19]. However, in an attempt to improve upon their approach it is decided to deviate from their approach in several ways, namely:

- Several additional removal methods are implemented to be able to escape from a larger range of local minima. It is chosen to do this as Laporte et. al [55] have shown that this is beneficial. Because of this the method makes use of 8 removal methods and 4 insertion methods. Both sets of methods are listed in Tables 6.1 and 6.3.
- It is decided not to implement a procedure where certain removal methods are chosen more or less frequently based on their performance during previous iterations. Given that Ropke and Pisinger [19]

showed that their preference for a certain removal method was at most 7 times as large as compared to the least preferred removal method it is decided to instead put additional effort into making the implementation as fast as possible. Given that it is likely that an efficient implementation may be several orders of magnitude faster as compared to one which has not been implemented well, it appears it might be more appropriate to spend additional time on the implementation than to add additional complexity to the method.

- Similarly, amongst other reasons, it is also decided not to implement a SA approach which may accept worsening solutions. It is decided to do so even though promising results have been obtained by using this method on static VRP's, as was described in Section 2.1.3. One of the reasons for doing this is that when solving DVRP's the static problem definition may often change which makes it hard to set an appropriate rate by which the temperature should decrease and worsening solutions should be accepted. Furthermore, as the problem definition changes often it might not make much sense to explore the solution space around worse solutions when improvements may still be found near the current best solution. Finally, there is less time available to be able to explore the solution space around worse solutions because of the limited available computation time.
- It is decided to use a different procedure for the selection of insertion heuristics. Given that a difference of two orders of magnitude can be observed in the computation time used by different insertion heuristics it is decided to use the relatively slow ones less frequent. As the time required for the removal of requests is nearly negligible compared to the time required for the insertion of requests (using any insertion method), the used insertion method dictates how many R&R iterations can be performed within a certain amount of time. Therefore, for example, roughly a hundred R&R iterations can be performed using sequential cheapest insertion in the same time a single iteration can be performed using two-regret insertion. Because of this a method which adapatively selects insertion heuristics is used. How this method works is described below.

To improve the solution as quickly as possible, initially only the relatively fast sequential cheapest insertion heuristic (as described in Section A.1.2) is used. When no improvement can be found (for some iterations) more time-consuming insertion heuristics are also used. Simultaneously, the search neighbourhood is gradually enlarged with each iteration in which no solution improvement is found (or in other words, more requests may be removed during the following iterations). When a solution improvement is eventually found, there is no reason to assume that the obtained solution can also be seen as such a 'hard to get out of' local minima. Therefore, the procedure reverts back to the relatively fast sequential cheapest insertion heuristic and the small search neighbourhood. How this procedure is exactly implemented is described in Section A.3

Finally, the R&R procedure is parallelised by letting multiple procedures run simultaneously. When any of these procedures finds a solution improvement during an iteration, the improved solution is directly copied to a global best solution shared by all procedures. As each procedure creates a copy of this global solution during each iteration, the improved solution is immediately used to find further improvements. The inefficiency caused by the parallelised global solution update is thereby limited to, at most, a single iteration for each separate procedure. Given that each procedure is able to perform several hundreds of iterations per second (on problem instances with approximately 500 requests) the inefficiencies caused by parallelisation are relatively small, even when multiple solution improvements are found per second.

Table 6.1: Removal methods used in the Ruin & Recreate procedure

Name	**Description**
`randomRequests`	Sequentially unschedules n times a random request. The idea is to improve the solution by ruining a different part of the solution virtually every time.
`randomRoutes`	Sequentially unschedules n times all requests within a random route. The idea is to improve the solution by reconstructing a set of routes in a different way virtually every time. Even though the same requests may be removed, unscheduled requests are randomly ordered which means a different solution may be found when using a sequential insertion method.
`routesWithLeastRequests`	Sequentially unschedules n times the route with the least number of scheduled requests. The idea is to improve the solution by ruining routes which contain the least number of requests so that less routes may be used.
`mostCostlyRoutes`	Sequentially unschedules n times the route which has the largest travelling distance. The idea is to improve the solution by ruining the most costly routes so that a reconstruction may yield a less costly route.
`mostCostlyRequests`	Sequentially unschedules n times the request which by itself causes the largest increase in travelling distance. The cost is equal to the negative removal cost visualised in Figure 2.3.
`randomRequestAndRelatedToIt`	Unschedules a random request. Furthermore, the n requests which are most related to this request (according to the measure described in Section A.1.5) and at most n requests which are incompatible with this request are also unscheduled. The idea is to improve the solution around a certain request by ruining a relatively large neighbourhood around the request.
`randomRouteAndRelatedToIt`	Unschedules all requests within a random route. Furthermore for each unscheduled request also the 2 requests most related to it (according to the measure described in Section A.1.5) and up to 2 requests which are incompatible with it are unscheduled. The idea is to improve the solution around a single route by also ruining a small neighbourhood around it.
`mostCostlyRequestAndRelatedToIt`	Unschedules the requests which by itself causes the largest increase in travelling distance. The cost is equal to the negative removal cost visualised in Figure 2.3. Furthermore, the n requests which are most related to this request (according to the measure described in Section A.1.5) and at most n requests which are incompatible with this request are also unscheduled. The idea is to improve the solution around the most costly request by ruining a relatively large neighbourhood around the request.

Table 6.3: Insertion methods used in the Ruin & Recreate procedure

Name	**Description**
`sequentialCheapestInsertion`	Sequential cheapest insertion which is is implemented as described in Section A.1.2.
`parallelCheapestInsertion`	Parallel cheapest insertion which is implemented as described in Section A.1.3.
`twoRegretInsertion`	Regret insertion which is implemented as described in Section A.1.4 where $k = 2$.
`threeRegretInsertion`	Regret insertion which is implemented as described in Section A.1.4 where $k = 3$.

6.2.4. REDUCING ROUTES

A method for reducing the number of routes that is being used within a solution is also implemented. This method is a variation upon a method introduced by Nagata and Bräysy [72]. The method basically makes use of an ejection chain and tries to insert remaining unscheduled requests by removing easily insert-able requests.

The procedure starts by removing a random route from the solution. Then, while no stopping conditions is reached, each unscheduled request is attempted to be inserted using sequential cheapest insertion (as described in Section A.1.2). When it is not possible to insert a request using this method, an attempt is made to insert the request using least hardest removal insertion. Least hardest removal insertion is a method which attempts to make it possible to insert a certain request, into an existing solution, by removing a set of scheduled requests which together have the smallest sum of a 'hardness' parameter. To enable this, each request has an own hardness integer value which is stored and initialised at zero. Each time it is not possible to insert a request using both sequential cheapest insertion and least hardest removal insertion, the hardness value for that request is incremented by one. By doing this, eventually, requests which are hard to insert will get a relatively high hardness value and are therefore unlikely to be removed to allow the insertion of other requests. If eventually all requests are inserted, an additional route is removed and the process starts again.

It is worth mentioning that this algorithm is only used by the developed solver when the main objective is to reduce the number of routes. When this is the case the algorithm is used alternately with the Ruin & Recreate procedure. The different phases the solver goes through are then as illustrated by Figure 6.6. How this algorithm is implemented is described in Appendix A.4.

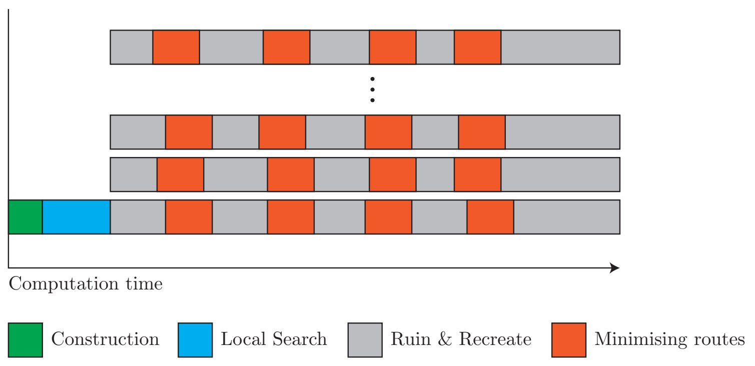


Figure 6.6: Phases of the developed vehicle routing solver when the main objective is to reduce the number of routes.

6.3. VALIDATION

To validate if the developed solver can be used as a tool to evaluate the proposed method to improve dynamic route optimisation, both in terms of solution quality and speed, a large number of experiments is ran. The setup of these experiments is covered in Section 6.3.1. The results of these experiments are presented in Section 6.3.2.

6.3.1. EXPERIMENTAL SETUP

The experimental setup for assessing the performance of the solver is covered in the following subsections. Given that currently no widely used benchmark instances for the dynamic PDPTW exist, static benchmark instances are used to evaluate performance. These instances, the stopping conditions and hardware used in these experiments are described in the following subsections.

INSTANCES

Experiments were ran on sets of instances consisting of approximately 50, 100, 200, 300, 400 and 500 requests. These sets were originally introduced by Li and Lim [30] and designed to "have a variety of distribution properties". These instances can currently be regarded as a research benchmark for the PDPTW, mainly due to the website of the Transportation Optimization Portal of SINTEF Applied Mathematics which holds frequently updated tables of the currently best known solutions together with references to the related research in which they were found. The primary objective of these instances (after scheduling all requests) is to minimise the number of routes. The secondary objective is to minimise the total travelling distance.

STOPPING CONDITIONS

Two stopping conditions for the termination of each experiment are used. First, a maximum computation time of 60 seconds is set. This to, in some degree, emulate the computation time constraints which have to be dealt when solving DVRP's. Second, the experiment is also terminated when the value, rounded to two decimal places, for each objective is less than or equal to the corresponding objective value of the best known solution. A strictly equal comparison is not used because the "SINTEF" website rounds objective values to two decimal places for the determination of the best known solution. Furthermore, for possible reproduction, the best known solutions which were available on September 17^{th} 2018 are used for comparison.

HARDWARE

These experiments are ran on an HP ZBook Studio x360 G5 Mobile Workstation with an Intel Core i7-8750H Coffee Lake processor. This processor supports 12 parallel threads which are used by the parallelised R&R and route reduction procedure.

6.3.2. RESULTS

In this section a summary of the results of the experiments for evaluating the performance of the implemented vehicle routing solver is presented. Given that the tables with the individual results take up a considerable amount of space these have been included in Appendix C. A summary of the performance in terms of the quality of the obtained solutions is presented in Figure 6.7. The two sub figures each show the distribution of the percent deviation from the corresponding best known solutions for one of the two objectives, for all sets of instances. In other words, these figures summarise the last two columns of all tables listed in Appendix C. It can be seen that for all instances with approximately 50 requests, all best known solutions are found. The same be said for roughly half of the instances with approximately 100 requests.

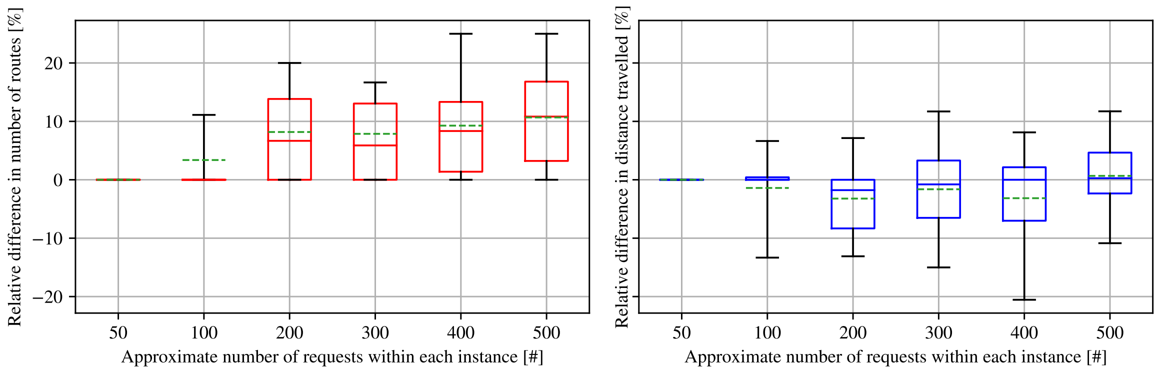


Figure 6.7: Summary of the results available in Appendix C. On the left the obtained relative differences in the number of routes as compared to the best known solutions, for all sets of instances, when solving for 60 seconds, are shown. On the right the obtained relative differences in the distance travelled as compared to the best known solutions, for all sets of instances, when solving for 60 seconds, are shown.

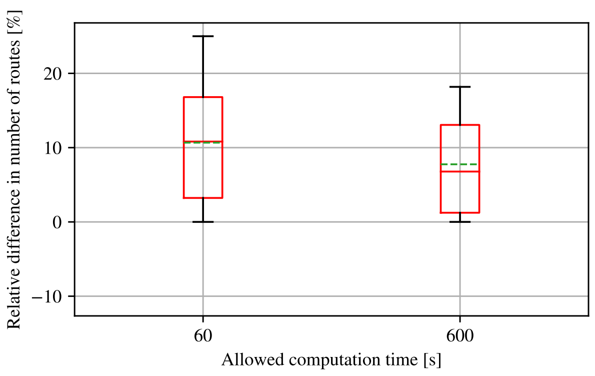
For larger problem instances often solutions are found which use more routes but require less distance to be travelled. Because of the hierarchical objective structure it can at least be concluded that the route reduction procedure (as described in Section 6.2.4) is not able to be on par with state-of-the-art methods when allowed only 60 seconds of computation time. Given that this algorithm is merely a relatively easy implementable variation upon a more extensive method described in the literature and is tested under severe time constraints, this is not surprising. Also, given that route reduction is not an objective of the instances described in Section 7.1.1, this is not seen as a problem.

Furthermore, it appears that solutions with fewer routes will generally require more distance to be travelled. Because of this effect, it is hard to draw conclusions about the ability of the solver to find solutions with a minimal distance travelled when a solution with the same number of routes is not obtained.

In an attempt to be able to draw such conclusions regarding solver performance, experiments on the instances with approximately 500 requests were performed again, now with 600 seconds of available computation time. A summary of the quality of the obtained solutions as compared to having 60 seconds of available computation is shown in Figure 6.8. Solutions with the same number of routes are now obtained for at least 20% of the problem instances. When only looking at these problem instances it can be seen that the relative differences in terms of the distance travelled are more or less between 0 and 10% as compared to the best known solutions which were found without any restrictions on the amount of available computation time. In presenting these results it is worth mentioning that a large part of the 600 seconds is spent trying to reduce routes. The R&R procedure, which actually minimises distance, only operates during a relatively small portion of the available computation time. Furthermore, on the majority of the instances the relative difference in distance travelled ranges roughly between +10 and -10% with a mean around zero. Therefore, when looking at minimising the distance travelled, even though no hard conclusions may be drawn, it appears that when using the developed solver at least solutions within 10% of the best known solutions can be found on relatively large problem instances within the order of magnitude of seconds. Furthermore, every best known solution found by Ropke and Pisinger [19] is found within 60 seconds of computation time.

Also, when specifically looking at computation times, it can be seen that all best known solutions for instances with approximately 50 requests are, on average, found in 1052 milliseconds. When four outliers, in terms of computation time, are neglected the average even becomes 139 milliseconds. This means that the same or better results are obtained over 7400 times faster than reported by Li & Lim [30] and over 470 times faster than reported by Ropke and Pisinger [19]. In doing so it can be concluded that the developed implementation is able to significantly outperform both implementations, even when Moore's law is taken into account which suggests that computational power must have grown by a factor 64 (2^6) over the past 12 years.

Finally, in presenting these results it is worth mentioning that the developed solver does not produce fully deterministic results. This is caused by the parallelised R&R procedure which, depending on the run time of individual threads, may sometimes find certain solutions improvements faster. Given that any solution improvement causes a global solution update which is used by all threads during following iterations, obtaining a (slightly) different solution can steer the solver into a different direction within the solution space. This often causes the solver to get stuck or terminate at a different location within the solution space. As at least hundreds of solution improvements are generally found during each optimisation and hundreds of iterations are performed per second, this behaviour is almost certain to occur which means that, with relative certainty, it can be said that no two solutions will be the same when the best known (or optimal) solution is not found. As in the literature the best results, after running the same experiment multiple times, are often presented [33], the conclusions drawn in this section still hold even as each experiment is run only once.



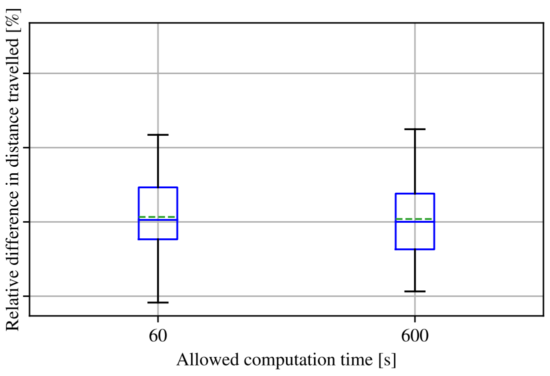


Figure 6.8: Summary of the results available in Appendix C. On the left the obtained relative differences in the number of routes as compared to the best known solutions, for instances with approximately 500 requests, when solving for 60 and 600 seconds, are shown. On the right the obtained relative differences in the distance travelled as compared to the best known solutions, for instances with approximately 500 requests, when solving for 60 and 600 seconds, are shown.

6.4. VISUALISATION

A visualisation tool is developed to be able to visually inspect solutions. This tool is not only able to visualise solutions after the optimisation is performed but is also able to dynamically visualise the global best solution while an optimisation or simulation is actually running. The tool consists of two views namely the *timetable* view and the *routes* view. Both views are visualised in Figures 6.9 and 6.10. It is worth mentioning that these views, besides them being dynamic, can also be interacted with.

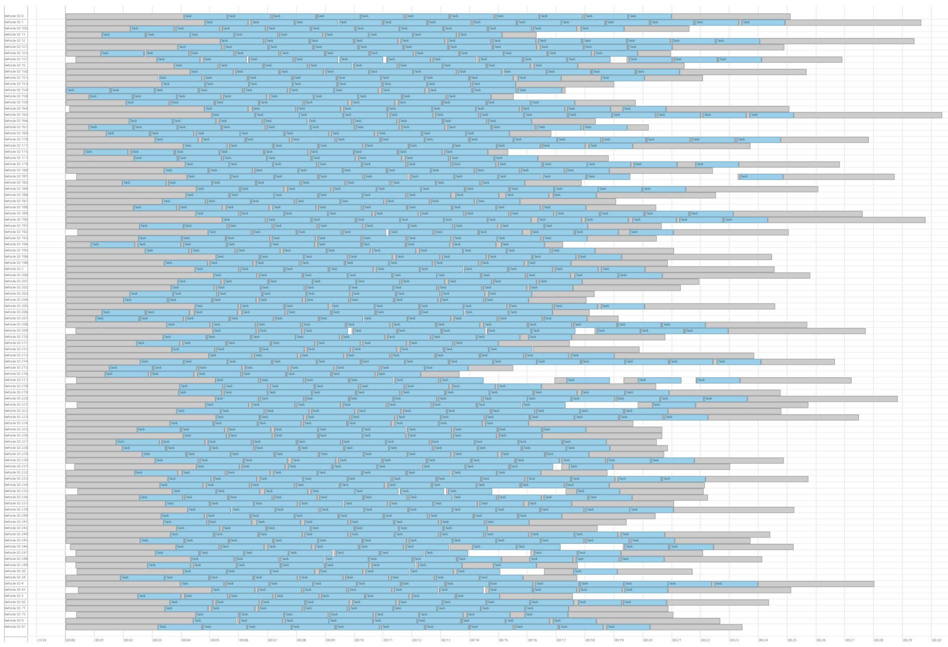


Figure 6.9: Zoomed-out *timetable* view of the visualisation tool. Time is listed on the x-axis. Each row represents a vehicle. The grey blocks indicate that a vehicle is travelling. The blue blocks indicate that a vehicle is fulfilling a task. The visualised solution is the best known (found) solution belonging to the $lc1\_10\_1$ problem instance which belongs to the set of SINTEF benchmark instances described in Section 6.3.1. This instance contains 527 requests and the solution requires 100 vehicles.

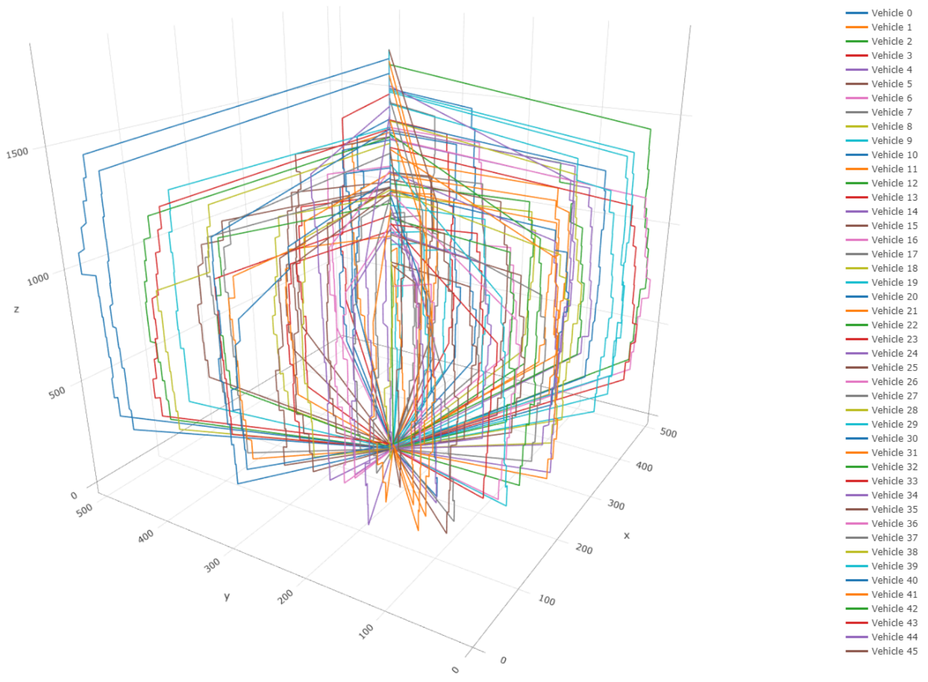


Figure 6.10: *Routes* view of the visualisation tool. Time is represented by the vertical z axis. Space is represented by the x and y axes. Each line represents the route of a vehicle through space and time. The vertical line segments indicate that a vehicle is stationary for a certain amount of time which suggests that it is fulfilling a task. The visualised solution is the best known (found) solution belonging to the $lc1\_10\_1$ problem instance which belongs to the set of SINTEF benchmark instances described in Section 6.3.1. This instance contains 527 requests and the solution requires 100 vehicles. The depot location of this instance (where all vehicles start and end) is located in the middle of the x and y axes.

7 EVALUATION

To evaluate the anticipatory method, a variety of experiments is required to be performed. To do this in a controlled environment, where the contribution of the anticipatory method can be isolated, it is decided to use numerical simulations. How these simulations are set-up, and which problem instances will be solved while simulating, will be described in Section 7.1. Given that real-time simulations take a considerable amount of time and running a variety of experiments is desired, the effect of the simulation speed is first investigated, in Section 7.2, to determine if time-consuming experiments can be run using less computation time. The prediction model of the anticipatory method has several parameters that are required to be set. To investigate the sensitivity of the anticipatory method to these parameters two additional sets of experiments are covered in Section 7.3 and Section 7.4. Finally, to meet both research goals and investigate how multiple strategies perform on a range of problem instances, four strategies are compared in Section 7.5.

7.1. EXPERIMENTAL SET-UP

In this section the experimental set-up used to perform numerical simulations is covered. First, the real-world problem instances and historical data that is used are described in Section 7.1.1. Second, an analysis of the instances and the historical data is performed in Section 7.1.2. How, based on this analysis, several method parameters are set is described in Section 7.1.3. Other optimisation strategies that are used to compare the performance of the anticipatory method to, are described in Section 7.1.4. Then, how the multiple required components for evaluation are implemented is described in Section 7.1.5. How these components are combined in order to perform simulations is described in Section 7.1.6. Finally, the hardware used for running these experiments is described in Section 7.1.7.

7.1.1. INSTANCES

Obtaining problem instances that meet all conditions required to be used for evaluation of the anticipatory method is challenging. This is because not only a set of dynamic requests is required but also a much larger set of historical requests is required. Because of the latter, the few synthesised Solomon-based benchmark instances [73] available for dynamic pickup and deliver vehicle routing can not be used for evaluation.

This means either a large amount of effort should be put into synthesising data or data should be found elsewhere. Given that the significance and objectivity of self synthesised data will always be questionable, especially when the goal is to find patterns in the occurrence of requests, it is decided not to synthesise problem instances.

Instead it was decided, prior to this research, to collaborate with industry so that real-world instances could be obtained. Therefore, this research is performed in collaboration with ORTEC. This collaboration has made it possible to make use of data of one of their customers. The customer in question is a transportation company who deals with the relatively urgent delivery of flowers to and from growers, distributors and auctions in The Netherlands and Belgium. The company will be referred to as "the transportation company" for the remainder of this thesis.

In creating problem instances using data from the transportation company, a significant amount of time is spent on approximating the real-world problem as close as possible. Measures that are taken to do so are described below.

REQUESTS

Hundreds of thousands of actual historical requests are extracted from copies of production databases. Almost all request parameters are literally being used. The only parameters that are synthesised are the duration of the pickup tasks and the duration of the deliver tasks. These durations are set to equal the request quantity, only then in minutes. Given that the quantity of requests ranges from 1.0 to 43.0 (a full vehicle load), unloading a full vehicle is estimated to take 43 minutes. These values are confirmed to roughly coincide with reality by an ORTEC consultant. Finally, both the pickup and deliver time windows of all requests are required to end later than 06:00:00 AM and should start before 18:00:00 PM to ensure that all requests can be fulfilled while vehicles are available. Requests that do not meet these constraints are neglected.

VEHICLES

A sufficient number of a 100 vehicles is assumed to be available at each one of the five actual depot locations of the transportation company. Each vehicle is required to return to the same depot location as it originated from. Furthermore, vehicles are assumed to have a capacity of 43.0 in the unit of the to be transported goods, just as is the case in reality. Finally, all vehicles are assumed to be available, each day, between 06:00:00 AM and 23:59:59 PM.

DISTANCE AND TRAVEL TIME VALUES

To obtain realistic distance and travel time values, hundreds of thousands of addresses are geocoded using ORTEC cloud services to obtain actual coordinates for all locations. Using these coordinates, tens of millions of fastest path calculations are performed using ORTEC cloud services to obtain actually used distance and travel time matrices specifically for the type of vehicles that is used by the transportation company.

DATES

Given that the available data sources contain requests up to 2018, all instances are created using request data belonging to the year 2017. Furthermore, given that the transportation company dispatches vehicles every single day, each day is seen as a separate possible problem instance. To circumvent having to deal with historical data that is inconveniently stored in either winter or summer time, dates of instances were chosen so that all data within the same historical data-set makes use of the same time zone. Furthermore, it was chosen to use 8 weeks of historical data for each instance. The created problem instances are listed in Table 7.1. For the remainder of this thesis, these instances will be referred to by their name as listed in the first column of the table. Based on the start of the summer time (March 26th in 2017) and the fact that two months worth of historical data are required, instances A through E are chosen to represent each day of the work week of week 22. To have no overlapping data with the first set of instances, the instances F through J are chosen to represent each day of the work week of week 32. The DOD of the described instances ranges from 0,70 to 0,93.

Table 7.1: Created problem instances

Name	Date	Weekday	Requests [#]	Historical data used for prediction
A	2017-05-29	Monday	1.655	2017-04-02 UTAI 2017-05-28, 59.633 requests
B	2017-05-30	Tuesday	1.399	2017-04-03 UTAI 2017-05-29, 59.900 requests
C	2017-05-31	Wednesday	1.227	2017-04-04 UTAI 2017-05-30, 59.965 requests
D	2017-06-01	Thursday	1.179	2017-04-05 UTAI 2017-05-31, 60.498 requests
E	2017-06-02	Friday	949	2017-04-06 UTAI 2017-06-01, 61.489 requests
F	2017-08-07	Monday	1.251	2017-06-12 UTAI 2017-08-06, 38.699 requests
G	2017-08-08	Tuesday	821	2017-06-13 UTAI 2017-08-07, 38.400 requests
H	2017-08-09	Wednesday	840	2017-06-14 UTAI 2017-08-08, 38.147 requests
I	2017-08-10	Thursday	722	2017-06-15 UTAI 2017-08-09, 38.147 requests
J	2017-08-11	Friday	474	2017-06-16 UTAI 2017-08-10, 37.788 requests

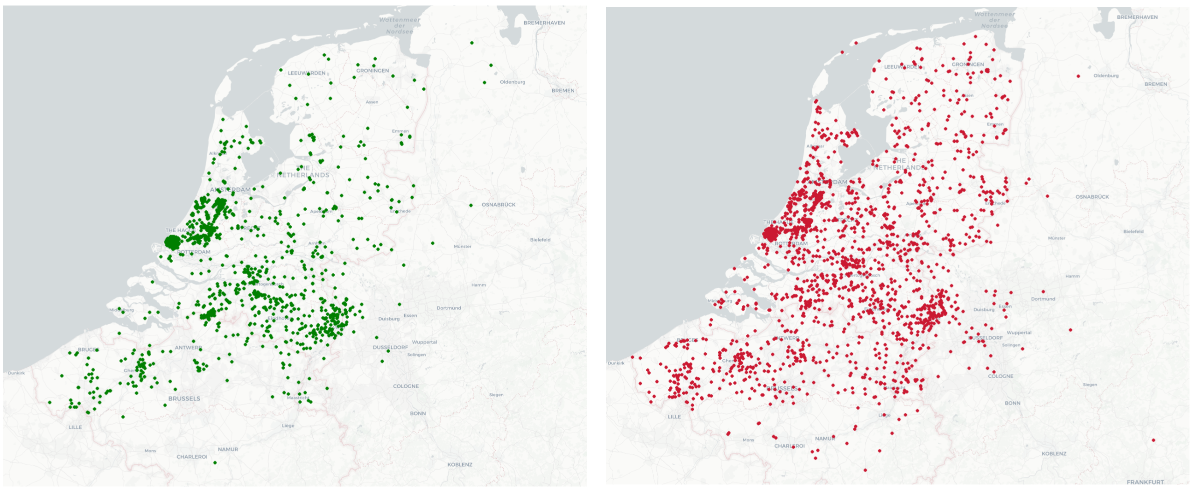
7.1.2. HISTORICAL DATA ANALYSIS

After processing the raw data to create the instances described in Section 7.1.1, several tables of historical data are obtained. The contents of these tables is as listed in Table 7.2. In an attempt to better understand the data that needs to be dealt with, an analysis of the data is performed.

Table 7.2: Available columns in historical data

Column	Description
`pickupLocationX`	The longitude of the pickup location
`pickupLocationY`	The latitude of the pickup location
`deliverLocationX`	The longitude of the deliver location
`deliverLocationY`	The latitude of the deliver location
`quantity`	The quantity of load needed to be transported
`pickupStart`	The start of the pickup time window
`pickupEnd`	The end of the pickup time window
`deliverStart`	The start of the deliver time window
`deliverEnd`	The end of the deliver time window
`madeKnown`	The moment at which a request was made known
`pickupDuration`	The duration of the pickup task
`deliverDuration`	The duration of the deliver task

First of all, to explore the spatial properties of the data, all pickup and deliver locations of all requests belonging to the historical data set of instances A and F are plotted on a map. These plots are shown in Figure 7.1. It can be seen that both the pickup and deliver locations are scattered across the Netherlands and Belgium. Furthermore, it can be seen that there are more unique deliver locations than there are pickup locations.



(a) Pickup locations (b) Deliver locations

Figure 7.1: Pickup and deliver locations of all requests within the combined historical data-set of instances A and F (as described in Section 7.1.1)

Second, to have an indication of the amount of requests that need to be dealt with every day, the number of occurring requests per day is plotted in Figure 7.2. It can be seen that during each workday between 400 and 2000 requests need be dealt with. Not entirely surprising, it can be seen that the number of occurring requests is much higher during working days as compared to the weekend. Furthermore, a clear weekly pattern can be observed. Besides for a single exception (2^{nd} Easter Day), Mondays are the busiest days of the week and towards the weekend the workload generally decreases each day. Also, it can be seen that instances F till J represent days during a less busy period of the year as only about two-thirds of the requests need to be dealt with as compared to the days which are represented by instances A till E.

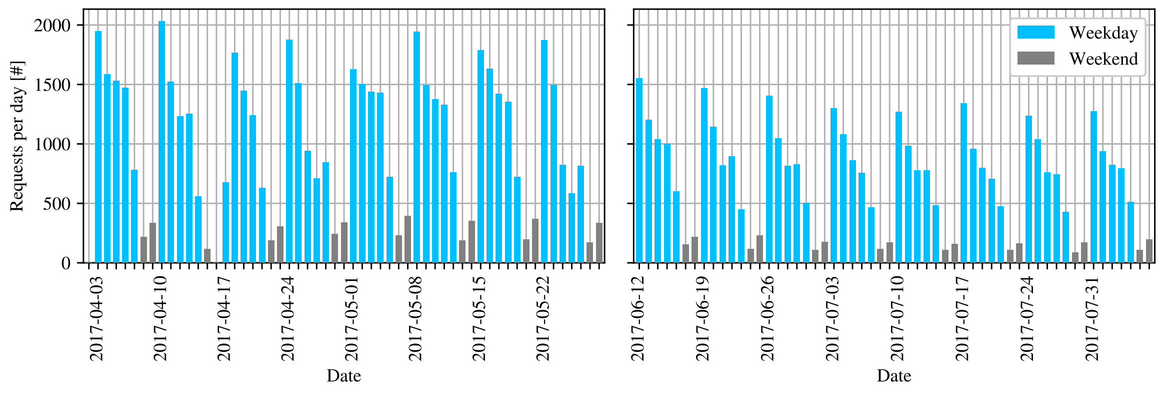


Figure 7.2: Total number of requests occurring each day within the historical data-sets of instances A and F (as described in Section 7.1.1). Left: historical data belonging to instance A. Right: historical data belonging to instance F.

Histograms of the time of the day at which requests become known are shown in Figure 7.3. It can be seen that only a relatively small portion of the requests is already known at 06:00 AM when vehicles may start driving. Furthermore, requests keep arriving almost the entire workday with peaks at around 09:00 AM and 15:00 PM. Based on these histograms it can already be concluded that any initial solution that may be created at 06:00 AM will most likely look a lot different as compared to a solution which has been created (and is assumed to be realised) at the end of the day at 23:59:59 PM.

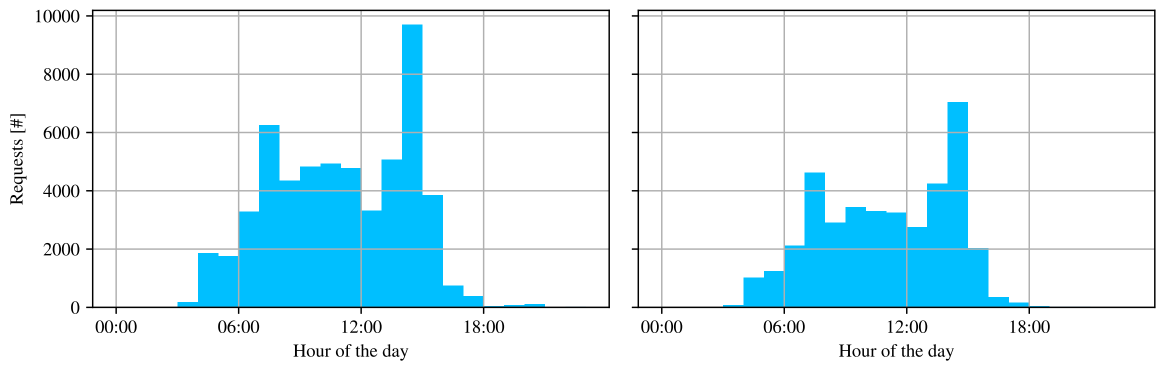


Figure 7.3: Histogram of the time of the day at which requests are made known within the historical data-sets of instances A and F (as described in Section 7.1.1). Left: historical data belonging to instance A. Right: historical data belonging to instance F.

Histograms of the length of the pickup time windows of requests are shown in Figure 7.4. It can be seen that time windows range from less than an hour to almost 24 hours. Furthermore, a majority of them is either around 6 or 14 hours long. The fact that these time windows are relatively long severely increases the complexity of the problem given that these time windows allow for tasks to be scheduled in almost any order. Finally, histograms, of the amount of time that is present between the moment requests become known and the moment their pickup time windows starts, are shown in Figure 7.4. By using this figure together with the histogram about time window length, conclusions about the urgency of requests can be drawn. Given that most requests can directly be pickup up after they have become known and time windows are multiple hours long, requests should generally be acted upon within several hours.

Finally, histograms of the request quantity are shown in Figure 7.4. It can be seen that the quantity of requests ranges from 1.0 to 43.0. Furthermore, the majority of the requests has a quantity smaller or equal to 4.0 which means often 10 or more requests can be transported by the same vehicle simultaneously.

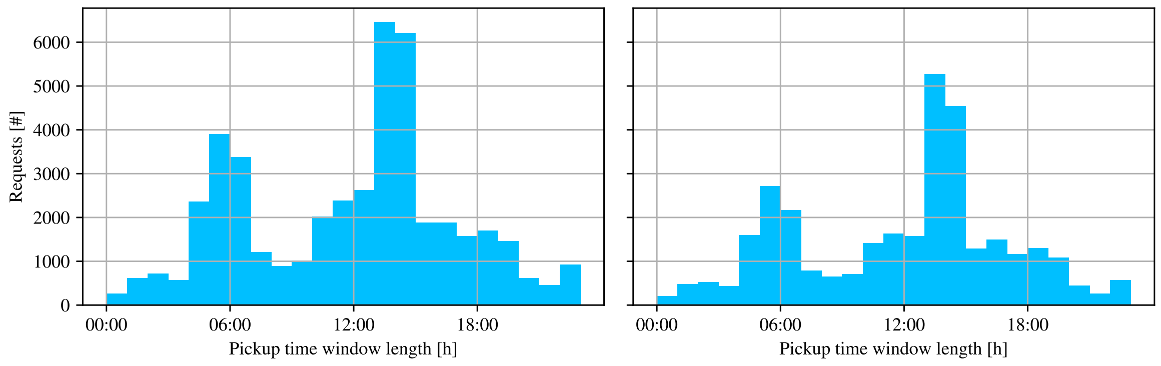


Figure 7.4: Histogram of the length of the pickup time windows of requests within the historical data-sets of instances A and F (as described in Section 7.1.1). Left: historical data belonging to instance A. Right: historical data belonging to instance F.

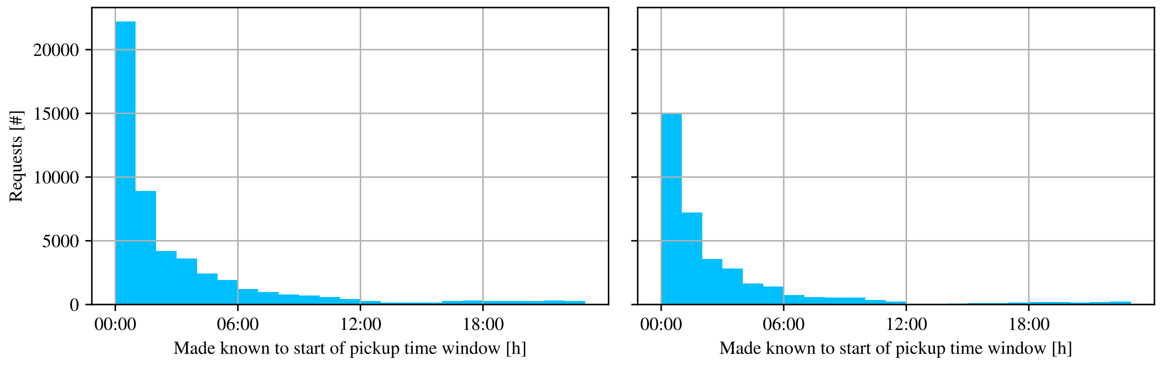


Figure 7.5: Histogram of the amount of time present between the moment requests become known and the moment their pickup time windows start. These requests belong to the historical data-sets of instances A and F (as described in Section 7.1.1). Left: historical data belonging to instance A. Right: historical data belonging to instance F.

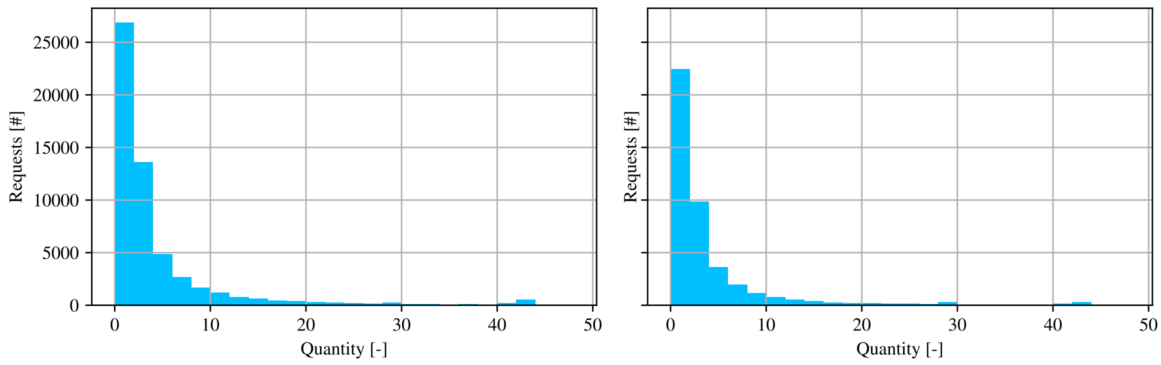


Figure 7.6: Histogram of the request quantity within the historical data-sets of instances A and F (as described in Section 7.1.1). Left: historical data belonging to instance A. Right: historical data belonging to instance F.

7.1.3. METHOD PARAMETERS

As described in Chapter 4, both the chosen clustering method as the prediction model require certain parameters to be set for the prediction of requests. How these parameters are set is described below.

CLUSTERING PARAMETERS

To be able to fulfil additional requests efficiently, it is desired to drive as little additional distance as possible. Vehicles should therefore be at the right place, within the defined time windows, while still having enough remaining capacity. As it was shown, in Section 7.1.2, that time windows are generally multiple hours long, it appears to be most important to be at the right place while having enough remaining capacity. Also, it was shown that the number of unique pickup locations is much smaller than the number of unique deliver locations. Therefore, by first separating the data-set based on the pickup location, relatively little computational effort is required while relatively important dimensions are addressed first.

Because of similar reasons it is chosen to then further separate requests based on the deliver location and request quantity. Only after these three levels, the data is separated further based on time dimensions. The used cluster levels and their parameters are therefore as listed in Table 7.4. The second column lists the dimensions that are used for clustering on each level. The third column describes the function which determines the (dis)similarity based on the values of requests along these dimensions. The fourth column describes how the threshold is computed based on the data-set. Finally, to give an indication of the value of these thresholds, the computed thresholds for instance A are listed in the fifth column. The sensitivity of the anticipatory method to these thresholds is investigated in Section 7.4. The minimum number of requests which are required for a request type to remain valid at each clustering level is set to four. This number was chosen as, with eight weeks of historical data and a weekly pattern, theoretically, at least an $\texttt{rf}_o$ higher than 0.50 can be obtained for each predicted request.

It is worth mentioning that when attempting to cluster a relatively small amount of approximately 60.000 requests, on their pickup location, while making use of a great circle distance function, several hours of computation time are required. However, when only considering unique locations, as proposed, merely seconds are required by the used Python based implementation. Also, even though the (dis)similarity matrix for the number of unique locations can probably be stored within random access memory it is still decided not to compute this matrix so that the implementation may possibly also be used for clustering larger sets of requests.

Finally, while clustering on multiple levels will probably not have any beneficial effects on the used computation time after the data has already been separated multiple times, it is still decided to do so for the sake of the simplicity of the implementation. This choice appears to be justifiable by the fact that very little additional request types are formed at lower clustering levels which suggest that only after a few levels, requests within the same request type are already very similar. The latter suggests that not a significantly different clustering result would be obtained when multiple dimensions would be clustered on simultaneously or when clustering levels would be used in a different order.

Table 7.4: Levels and their parameters used in clustering requests. The last column states the calculated threshold for Instance A (as described in Section)

Level	Dimension(s)	Distance function	Threshold	Inst. A
1	`pickupLocationX`, `pickupLocationY`	The great circle distance between both coordinate pairs.	10% of the median trip distance (the great circle distance between the pickup and the deliver location) of all requests.	5.48 km
2	`deliverLocationX`, `deliverLocationY`	The great circle distance between both coordinate pairs.	10% of the median trip distance (the great circle distance between the pickup and the deliver location) of all requests.	5.48 km
3	`quantity`	Absolute difference	10% of the median vehicle capacity	4.3
4	`pickupEnd`	Absolute difference in time of the day (in such a way that 23:59 PM is also close to 00:01 AM)	10% of the median time pickup time window length in seconds	4787 s
5	`deliverEnd`	Absolute difference in time of the day (in such a way that 23:59 PM is also close to 00:01 AM)	10% of the median time deliver time window length in seconds	4860 s
6	`pickupStart`	Absolute difference in time of the day (in such a way that 23:59 PM is also close to 00:01 AM)	20% of the median time pickup time window length in seconds	9574 s
7	`deliverStart`	Absolute difference in time of the day (in such a way that 23:59 PM is also close to 00:01 AM)	20% of the median time deliver time window length in seconds	9720 s
8	`madeKnown`	Absolute difference in time of the day (in such a way that 23:59 PM is also close to 00:01 AM)	10% of the median time pickup time window length in seconds	4787 s
9	`pickupDuration`	Absolute difference	10% of the median vehicle capacity in minutes	4.3 min
10	`deliverDuration`	Absolute difference	10% of the median vehicle capacity in minutes	4.3 min

PREDICTION MODEL PARAMETERS

Four parameters are required to be chosen for the used prediction model. These are the $\texttt{rf}_o$, the assumed patterns within the data for which the $\texttt{rf}_o$ is determined and the prediction frequency and horizon. The $\texttt{rf}_o$ is initially set equal to 0.66 based on some very simple intial experiments. The sensitivity of the anticipatory method to this parameter is investigated in Section 7.3. As a weekly pattern is found within the number of occurring requests per day, a daily and a weekly pattern are assumed for the determination of the maximum $\texttt{rf}_o$ of each request. Finally, given that the number of occurring requests per day lies within the bounds of the problem sizes that can currently be solved, the entire problem can be solved at once and no separation into multiple regions or time slices appears to be required. The number of predictions of requests can therefore be minimised to a single prediction every single day. The prediction horizon is therefore set to 24 hours.

7.1.4. STRATEGIES

To perform a qualitative analysis of the performance of the anticipatory method (**Anticipatory (A)**), it is decided to compare the implementation to three other optimisation strategies for solving DVRP's. The implementations of these strategies make use of parts of the same solver as described in Chapter 6. This allows for a relatively fair comparison which is not subject to quality of the implementation. The three other strategies are:

- **Reactive + Cheapest Insertion (CI)** This strategy is added to serve as an upper bound (in terms of distance travelled) on the solutions which may be obtained. The strategy consists of creating an initial solution based on the requests which are known at 06:00 AM. In creating this initial solution the solver goes through the phases as visualised in Figure 6.1. To incorporate additional requests only the sequential cheapest insertion heuristic (as described in Section A.1.2) is used to insert requests into the solution when they become known. No further re-optimisation is performed. This strategy is often also referred to as a *greedy* approach.
- **Reactive + Ruin & Recreate (RR)** This strategy is added to serve as a close competitor to the anticipatory

method. The strategy consists of creating an initial solution based on the requests which are known at 06:00 AM. In creating this initial solution the solver goes through the phases as visualised in Figure 6.1. To incorporate dynamic requests the sequential cheapest insertion heuristic (as described in Section A.1.2) is also used. However, in parallel R&R procedures are run which constantly try to re-optimise the part of the global solution which has not yet been realised or is being realised. The entire approach is therefore as visualised in Figure 7.7. This strategy is also referred to as the *reactive* approach.

- **Full information (FI)** This strategy is added to serve as a lower bound (in terms of distance travelled) on the solutions which may be obtained. During this strategy it is assumed that all additional requests are already known at 06:00 AM. The initial and final solution is therefore created based on all requests. In creating this solution the solver goes through the phases as visualised in Figure 6.1.

7.1.5. IMPLEMENTATION

The components introduced in Section 3.2 and described in Chapters 4 till 6 are implemented in `Python` and `C++`. Everything related to creating problem instances and the prediction of requests is implemented in `Python` requiring over 5000 lines of code. The vehicle routing solver and problem simulator are implemented in `C++` using 43 classes consisting of 9200 lines of code.

7.1.6. SIMULATION

To evaluate the performance of the strategies (described in Section 7.1.4), simulations are run. For both the reactive approach and the anticipatory method the schematic overview of the simulation is as shown in Figure 7.7. In this figure each block represents a process and blocks stacked on top of each other occur in parallel. Simulations consist of two steps namely creating an initial solution and handling dynamic requests. What happens during both phases is described below. Finally, it is worth mentioning that when using the greedy approach a similar simulation structure is used only no R&R procedures are run parallel to handling the dynamic requests.

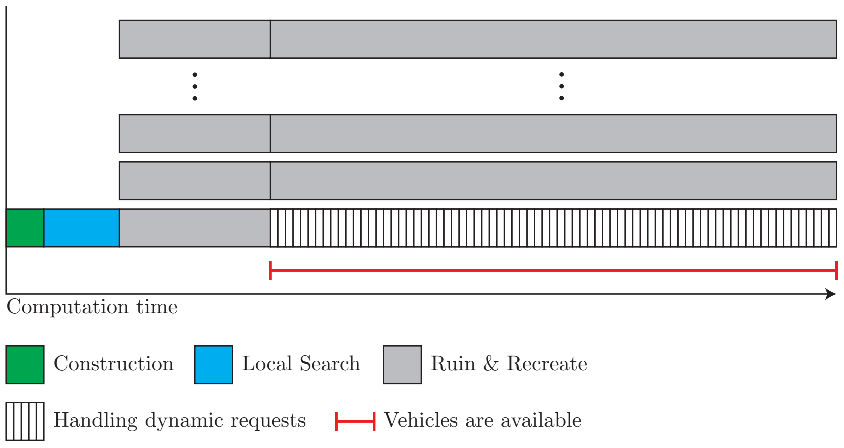


Figure 7.7: Schematic overview of the structure of the vehicle routing solver and simulation process

CREATING AN INITIAL SOLUTION

As a portion of the requests is already known at 06:00 AM, an initial solution can be created before vehicles become available. The amount of computation time for creating an initial solution is chosen to be in the order of magnitude of a few minutes to allow the solver to find relatively good solutions for problem instances with up to several hundreds of requests. However, it is estimated that this amount of time might not be sufficient to find very good solutions for problem instances with over a 1000 requests. As the anticipatory method adds additional predicted requests to the problem, such larger problem instances are required to be solved. Therefore, to evaluate the method in a situation where the odds are slightly stacked against the anticipatory method, it is assumed that requests are already known at 5:55 AM and only 300 seconds are available to create an initial solution based on the requests which were made known before 06:00 AM. In creating this initial solution the solver goes through the phases shown in Figure 6.1. These steps are also visualised in the entire simulation overview shown in Figure 7.7. Finally, in creating the initial (and final) solution for the full information strategy, 15 minutes of computation time are allowed for creating a solution.

HANDLING DYNAMIC REQUESTS

As described in the dynamic problem definition in Section 3.1.2, additional requests are required to be added to the problem definition when they have become known. To simulate this, every 30 seconds, within the simulation, it is checked if additional requests have become known and if so, these are added to the problem definition. By doing so the difference in time at which requests are added to the problem definition and when they actually have become known is at most 30 seconds.

Furthermore, during the same 30 second interval, the solution is fixed up to the current simulation time incremented with 30 seconds. This means that at the start of each step, the solution is at least fixed up to the end of that step. By doing so it is ensured that the part of the solution that is realised during the step will not be affected by re-optimisation that runs in parallel to the step. Which part of the solution specifically becomes fixed when time progresses is described in the problem definition covered in Section 3.1.2.

Finally, distance and travel time matrices are loaded into memory in advance which means that additional distance and travel time values are assumed to be available instantaneously when additional requests become known. Even though this will not be the case in any real-world application, it is estimated that, with additional engineering, these values may be made available within several hundreds of milliseconds. As this would only slightly delay the moment at which requests become known it assumed that this deviation from reality is negligible in evaluating the anticipatory method.

7.1.7. HARDWARE

All experiments are run on an HP ZBook Studio x360 G5 Mobile Workstation with an Intel Core i7-8750H Coffee Lake processor. This processor supports 12 parallel threads which are used by the parallelised R&R procedure. However, as handling dynamic requests requires a single thread, only 11 threads are effectively being used.

7.2. SENSITIVITY TO SIMULATION SPEED

Simulations of an entire instance (or day) are required to be performed to assess the performance of a certain optimisation strategy. As vehicles are available for 18 hours, each simulation requires 18 hours of computation time when simulations are run in real-time. However, it is desired to:

- Simulate multiple strategies to be able to compare performance.
- Simulate on multiple problem instances to ensure that possible differences in performance can be attributed to the used strategy instead of to the structure of a particular problem instance.
- Simulate each strategy on each problem instance multiple times as the developed solver does not produce deterministic results and confidence intervals might overlap.
- Simulate with different parameter values in an attempt to determine the sensitivity to the parameters on the considered problem instances.

This poses a problem as simply simulating all strategies on ten instances, all for ten times, already requires over 200 days of computation time. It is therefore decided to first investigate the effect of the simulation speed on the quality of the obtained solutions.

By increasing the simulation speed, time within the simulation actually runs faster than real-time. This means each simulation can be performed using less computation time. As time progresses faster within the simulation, amongst other things, vehicles will drive faster and requests are made known after a smaller amount of time. However, the only thing that can not be scaled accordingly is the time in which the solver is able to find better solutions. The solver is not able to do this as it runs in parallel with the scaled process and it can not be made to work faster than it is already doing in real-time. This means that the solver has relatively less time available to find better solutions before additional information becomes known and also a larger part of the solution becomes fixed. As this means that relatively less R&R iterations can be performed, on average, only worse solutions can be obtained.

To determine how much worse solutions become when the simulation speed is severely sped up a large number of simulations is performed for both the anticipatory method and the reactive approach. Specifically, instance A (as described in Section 7.1.1) is simulated 10 times using both strategies at simulation speeds 128, 64, 32, 16, 8 and 4. The method parameters are set as described in Section 7.1.3. The required computation time for running these simulations is roughly 8 days. The total distance that is required to be travelled when using both strategies at different simulation speeds is summarised in Figure 7.8.

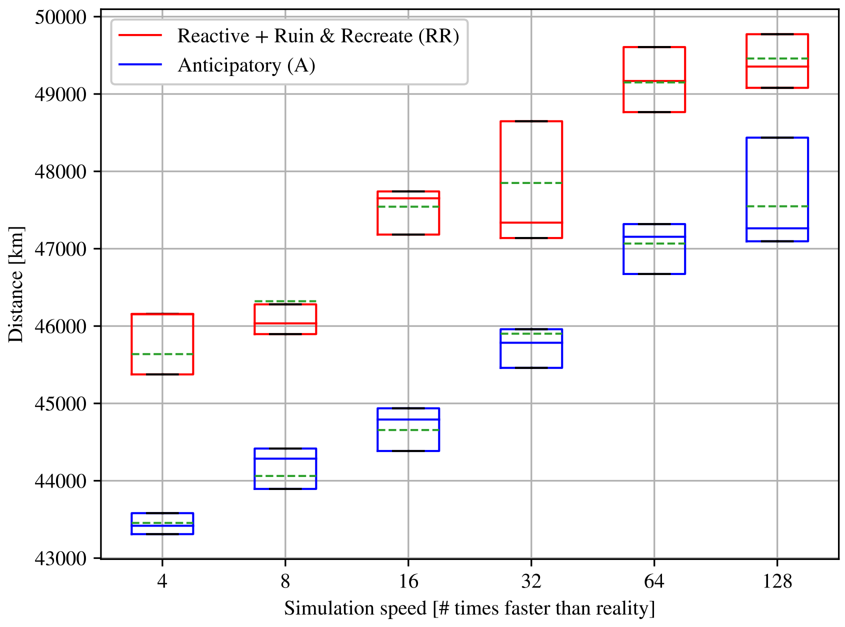


Figure 7.8: Distributions of the final objective value (the total distance travelled by all vehicles) when solving instance A (as described in Table 7.1) using both the reactive- and the anticipatory strategy, at different simulation speeds

First of all, it can be seen that the anticipatory method significantly outperforms the reactive approach, regardless of the used simulation speed. Also, the relative difference in performance between the two methods appears to be similar and in the range of 3 to 10%. Furthermore, it can be seen that both methods perform approximately 8% worse when the simulation speed is set to 128 as compared to using a simulation speed equal to four. As the differences between both methods, in terms of the quality of the obtained solutions, appear to be be relatively consistent and the absolute difference in the quality of the obtained solutions is in the order of magnitude of a several percent, using a higher simulation speed to draw conclusions about relative performances appears to be justifiable. It is therefore decided to use a simulation speed of 32 for further experiments. In doing so it is assumed that the described effects are similar across all instances. Furthermore, it is assumed that the absolute performance of both methods will be a few percent better when simulating in real-time but the relative difference in performance will be similar. Finally, these assumptions are later validated again in Section 7.5 as a small number of experiments will also be run in real-time (with a simulation speed equal to one).

7.3. SENSITIVITY TO MINIMUM RELATIVE FREQUENCY OF OCCURRENCE

In an attempt to investigate whether better results may be obtained by using a different $\mathtt{rf}_o$ (other than 0.66 as mentioned in Section 7.1.3), an additional experiment is run. Again on instance A, with a simulation speed equal to 32 and parameters values set as described in Section 7.1.3, four values for the $\mathtt{rf}_o$ are evaluated. Naturally, only the proposed strategy is used in these simulations. Finally each simulation is repeated 10 times to obtain relatively trustworthy confidence intervals. The total required computation time is approximately 27 hours. The total distance that is required to be travelled when using different $\mathtt{rf}_o$'s is summarised in Figure 7.9.

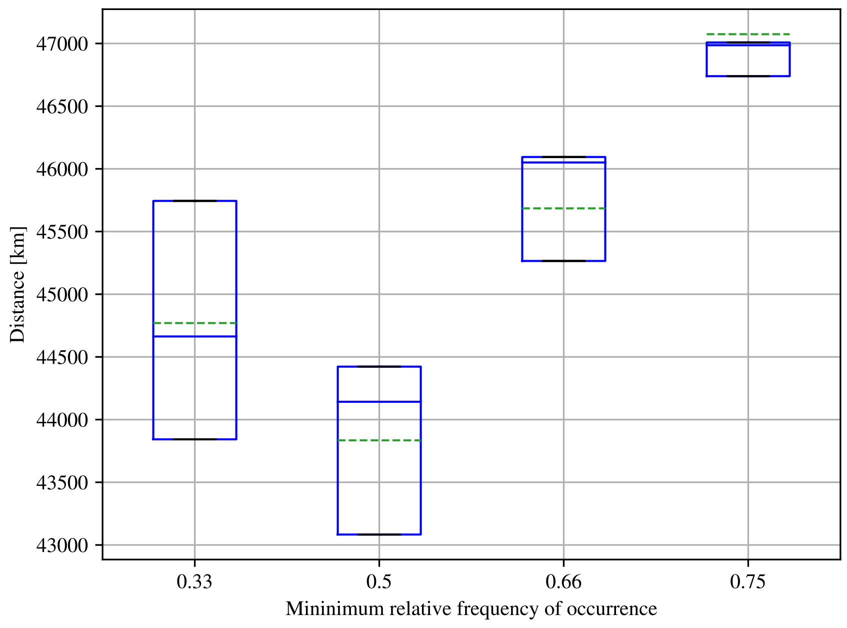


Figure 7.9: Distributions of the final objective value (the total distance travelled by all vehicles) when solving instance A (as described in Table 7.1) using the anticipatory method when different $\texttt{rf}_o$'s are used.

It can be seen that using a $\texttt{rf}_o$ equal to 0.50 actually appears to best when having to choose from the four evaluated options. This value therefore appears to result in the optimal balance in creating a prediction with more true positives at the cost of having (even) more false positives (as also mentioned in Section 4.2.3).

It is decided not to evaluate more values as it is very likely that the $\texttt{rf}_o$ of each predicted request will be a multiple of 0.125 given that there are eight weeks of historical data and a weekly pattern is assumed. It is unlikely that the daily pattern will yield the highest $\texttt{rf}_o$ given that much less requests need to be dealt with during the weekends (as shown in Figure 7.2). Furthermore, even if this would not be the case it can be argued that determining the $\texttt{rf}_o$ up to two or three decimal places may be seen as overfitting a parameter to a single problem instance. An $\texttt{rf}_o$ equal to 0.50 is therefore used in all remaining experiments.

7.4. SENSITIVITY TO CLUSTERING PARAMETERS

In an attempt to investigate whether better results may be obtained by using different clustering parameters (other than those mentioned in Section 7.1.3), an additional experiment is run. Again on instance A, with a simulation speed equal to 32, an $\texttt{rf}_o$ equal to 0.50 and clustering parameters set as listed in Table 7.5, multiple simulations are run. Naturally, only the proposed strategy is used in these simulations. Finally each simulation is repeated 10 times to obtain relatively trustworthy confidence intervals. The total required computation time is approximately 34 hours. The total distance that is required to be travelled when using different clustering parameters is summarised in Figure 7.10.

Table 7.5: Different sets of clustering parameters that are used in the experiment to determine the sensitivity of the anticipatory method to different clustering parameters. The listed percentages at each level replace the percentages listed in Table 7.4 which are used in the calculation of the threshold at each level. The results of the experiment are shown in Figure 7.10.

Level	**Extra small**	**Small**	**Medium**	**Large**	**Larger**
1	5%	10%	15%	20%	25%
2	5%	10%	15%	20%	25%
3	5%	10%	15%	20%	25%
4	5%	10%	15%	20%	25%
5	5%	10%	15%	20%	25%
6	10%	20%	30%	40%	50%
7	10%	20%	30%	40%	50%
8	5%	10%	15%	20%	25%
9	5%	10%	15%	20%	25%
10	5%	10%	15%	20%	25%

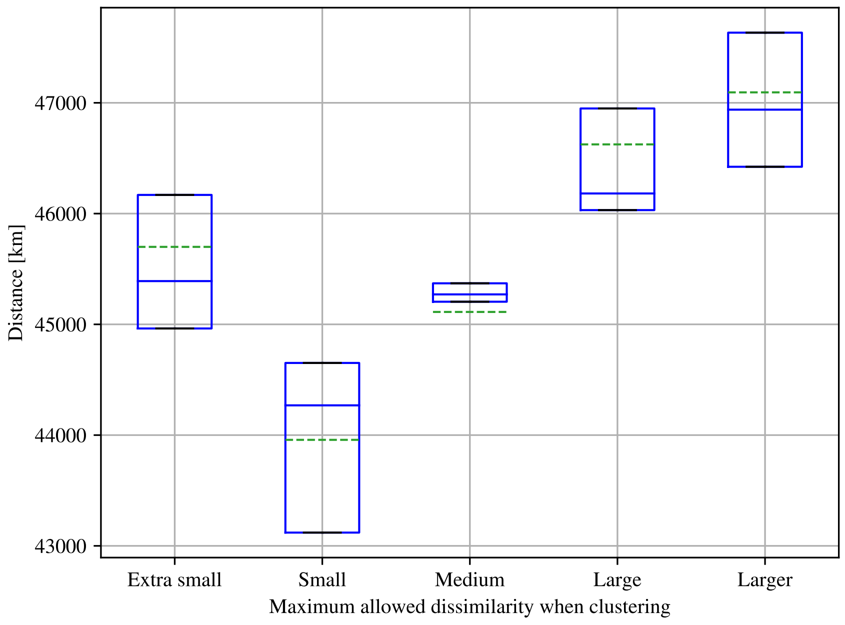


Figure 7.10: Distributions of the final objective value (the total distance travelled by all vehicles) when solving instance A (as described in Table 7.1) using the anticipatory method when the clustering parameters are used as listed in Table 7.5 are used.

It can be seen that when using the *small* clustering parameters, the best results are obtained. This means that the initial set of values was chosen relatively well (after some small initial experiments). It is observed that when using the *extra small* clustering parameters, the number of request types and predicted requests is smaller as compared to using the *small* clustering parameters. Furthermore, fewer request replacements (as described in Section 5.2) are observed. This suggests that more requests types are ignored because these contain to few requests which results in fewer predicted requests. Furthermore request types which are created appear to be so specific that additional requests are often to dissimilar for a replacement to be attempted. When using wider cluster parameters it appears that replacements are attempted more often but these appear to be less successful, more frequently, as a significant amount of additional distance is still required to be travelled. It is decided not to evaluate more values as it appears that a local minima is found. The clustering parameters as listed in Table 7.4 are therefore used in all remaining experiments.

7.5. COMPARING STRATEGIES

To investigate how the anticipatory method performs on different problem instances and compares to the other optimisation strategies (described in Section 7.1.4), a large number of experiments is run. Specifically, all four strategies are simulated 10 times on the 10 instances (described in Section 7.1.1). All simulations are performed using a simulation speed equal to 32. In these simulations the anticipatory method makes use of the parameter values as described in Section 7.1.3 and those defined in the previous two sections. The required computation time for running these simulations is approximately 10 days.

All results on each instance are summarised in separate figures. The results on the instances where the anticipatory method performs best (instance F) and performs worst (instance I) as compared to the reactive approach are shown in Figures 7.11 and 7.12. The remaining results are included in Appendix D. In each of these figures, on the left, it is shown how the amount of distance that is planned to be travelled during the entire day evolves during the day, for each strategy. Each line represents a single simulation. On the right the distribution of the total distance that is required to be travelled by all vehicles is summarised for each strategy.

It can be seen that the full information approach always performs best as it requires the least amount of distance to be travelled. The greedy approach always performs worst as the most distance is required to be travelled. As only a relatively small portion of the requests is known at 06:00 AM, solutions of both the reactive and greedy approach require relatively little distance to be travelled initially. As predicted requests are added to the problem in the anticipatory method, the initial solution requires more distance to be travelled initially.

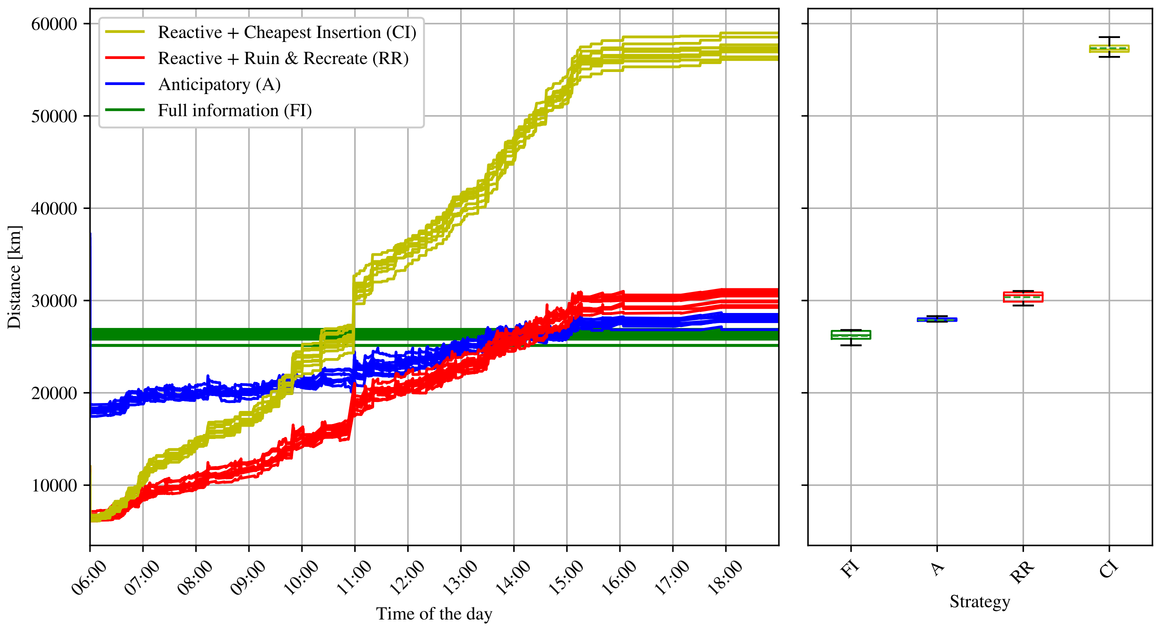


Figure 7.11: Performance of different optimisation strategies on instance F. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

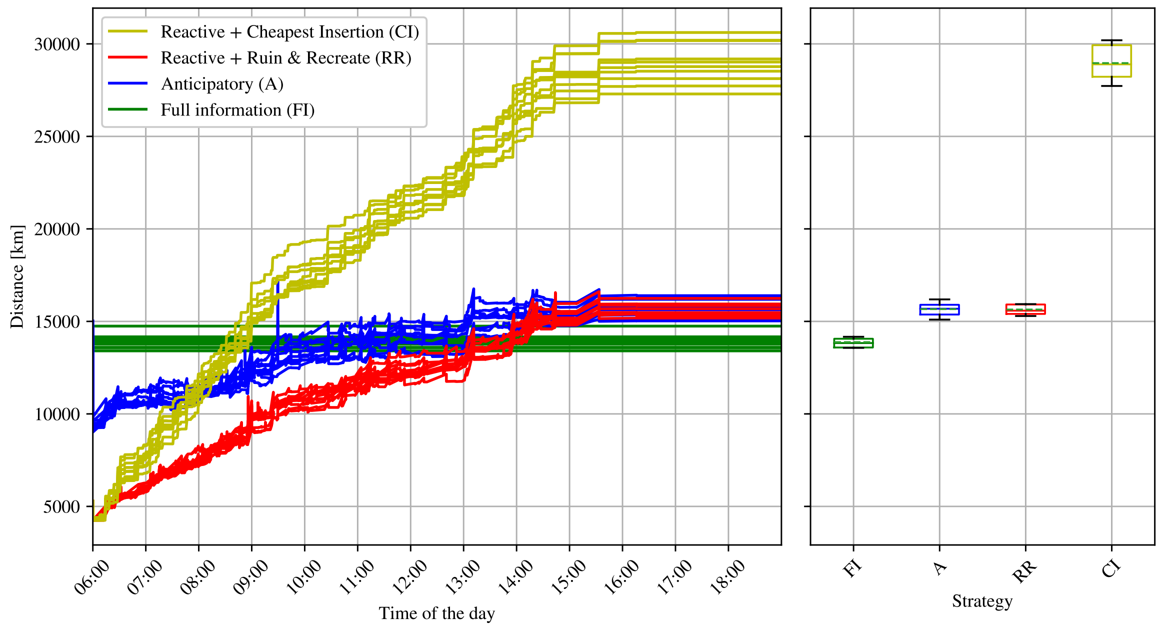


Figure 7.12: Performance of different optimisation strategies on instance I. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

As the described problem instances can currently not be served to optimality, the results belonging to the full information strategy are used as a benchmark. How the other three remaining strategies perform compared to this benchmark, on average, on all instances, is shown in Figure 7.13. When using the greedy approach, on average, 109,32 ± 9,09 % more distance is required to be travelled as compared to the full information strategy. When using the reactive approach, on average, 15,40 ± 2,75 % more distance is required to be travelled as compared to the full information strategy. When using the anticipatory method, on average, only 10,07 ± 3,52 % more distance is required to be travelled as compared to the full information strategy.

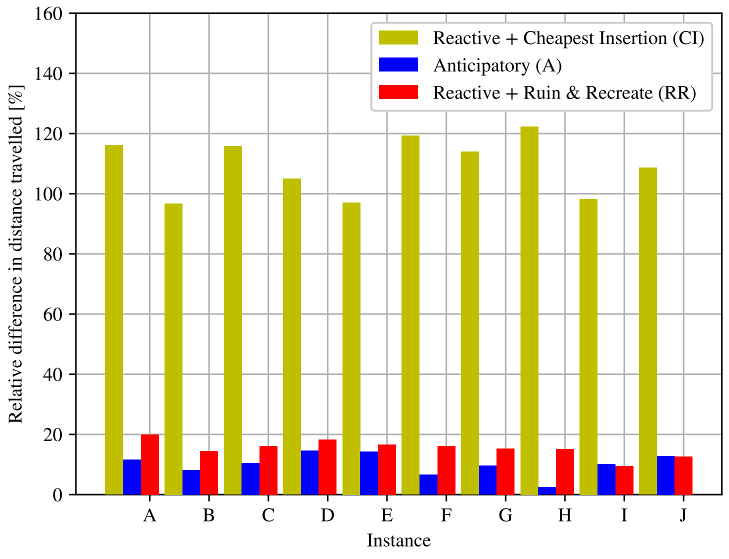


Figure 7.13: Relative difference in average distance that is required to be travelled for three strategies as compared to the full information strategy, on all instances.

In comparing the performance of the reactive approach (with R&R) and the anticipatory method it can be seen (from Figure 7.12) than even when the anticipatory method performs worst, on average, as compared to the reactive approach, the confidence intervals appear to overlap completely and only a small difference in the average distance travelled can be observed. It can therefore be concluded that, on the evaluated instances, and when making use of the described solver, on average, the anticipatory method does not significantly perform worse in any case. When the anticipatory method performs best however, on average, as compared to the reactive approach, it can be seen (from Figure 7.11) that confidence intervals do not overlap at all and the anticipatory method significantly outperforms the reactive approach. How the anticipatory method, on average, performs as compared to the reactive approach is listed in Table 7.6. It can be seen that when using the anticipatory method, on average, 4,58± 3,48% less distance is required to be travelled.

Table 7.6: Relative difference in the average distance that is required to be travelled and the number of vehicles that are utilised when using the anticipatory method as compared to the reactive approach, on all instances.

Instance	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**Avg.**
Distance [%]	-7,00	-5,50	-4,88	-3,05	-1,99	-8,19	-4,98	-11,04	0,69	0,14	**-4.58**
Vehicles [%]	-3.82	-2.37	-1.87	-2.68	-3.94	-3.82	-3.87	-3.95	-3.87	-3.18	**-3.34**

Additionally, it is worth mentioning that the anticipatory method also requires less vehicles to fulfil all requests. The relative difference in the number of vehicles that are required when using the anticipatory method as compared the reactive approach is listed in Table 7.6. It can be seen that, on average, 3,34 ± 0,74% less vehicles are required.

Finally, to validate again if comparable results are obtained when using a simulation speed equal to one, some additional experiments are ran. Specifically, each strategy is simulated a single time on instances B and H using a simulation speed equal to one. The required computation time for these experiments is roughly 5 days. The results are shown in Figures D.9 and D.10. It can be seen that both the reactive and the anticipatory method produce results which are several percent better than the results which were obtained when simulating 32 times faster. Still, the anticipatory method outperforms the reactive approach, in terms of the distance travelled, on instances B and H, by 8,14% and 5,95% respectively.

7.6. DISCUSSION

In this section, the obtained results are discussed. First, a difference in the added value of the anticipatory method is observed, in Table 7.6, among instances A through E which represent days belonging to the same work week. The added value of the anticipatory method decreases throughout the week just as the size of the problem instances also decreases throughout the week. However, even though correlation may be observed this naturally does not necessarily imply causality. For example, this effect may also be caused by the set of requests, which has to be dealt with, becoming less predictable throughout the week. To validate this the quality of the prediction is determined by labelling the additional requests using the classifier described in Section 5.2. The quality of the prediction is then quantified by determining the overlap between the labels of the predicted requests and the labels of the additional requests. For instance A the results are plotted against the $\mathtt{rf}_o$ of each predicted request in Figure 7.14. It can be seen that, for the majority of the predicted requests a unique additional request, belonging to the same request type, actually becomes known. It can also be seen that when the $\mathtt{rf}_o$ is relatively high a larger portion of the predicted requests appears to be predicted correctly. According to this measure 78% of the predicted requests for instance A are predicted correctly. For instance E this is only 49%. It is observed that the number decreases throughout the week. The percentages for all instances are listed in Table 7.7.

Table 7.7: Derived prediction quality on all instances. The *actual* row states the number of requests that are actually made known. The *predicted* row states the number of predicted requests. The *overlap* indicates what percentage of the predicted requests are also actually made known according to the measure derived using the request classifier. This percentage gives an indication of the quality of the prediction.

Instance	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Actual [#]	1377	1159	1095	1000	493	1068	821	753	717	430
Predicted [#]	876	845	630	510	403	782	549	442	388	336
Overlap [%]	78	68	57	54	49	80	73	78	41	40

A similar effect in the quality of the prediction is observed for instances F till J. However, in addition to this, it was found that the official summer holiday for the central and southern parts of the Netherlands also started after the week which is represented by instances F till J. This may explain an even larger difference in the obtained added value during the end of the week.

The difference in the added value of the anticipatory method therefore appears to be explainable by a difference in the prediction quality. The prediction method is based on the assumption that similar requests which previously have occurred according to certain patterns will also occur according to the same patterns in the future. Therefore, when reality deviates from observed patterns, regardless of the cause, a relatively worse prediction is generated.

The obtained percentages for the quality of the prediction also appear to be explaining why the anticipatory method works at all. It can be concluded that a significant amount of information of relatively high accuracy is provided to the solver in advance. It also shown that when all unknown information is provided in advance, as is the case in the full information approach, even better results can be obtained.

Finally, the reduction in the number of used vehicles can partly be explained by the fact that less distance is required to be travelled. However, after visually inspecting the solutions using the tool described in Section 6.4, it is observed that the presence of predicted requests also pushes fulfilling already known requests more towards the beginning of the day whereby the workload of fulfilling requests appears to be distributed more evenly along the workday. To illustrate this, how the number of vehicles that are planned to be used during the entire day, evolves throughout the day, when using the different strategies (on instance F) is shown in Figure 7.15.

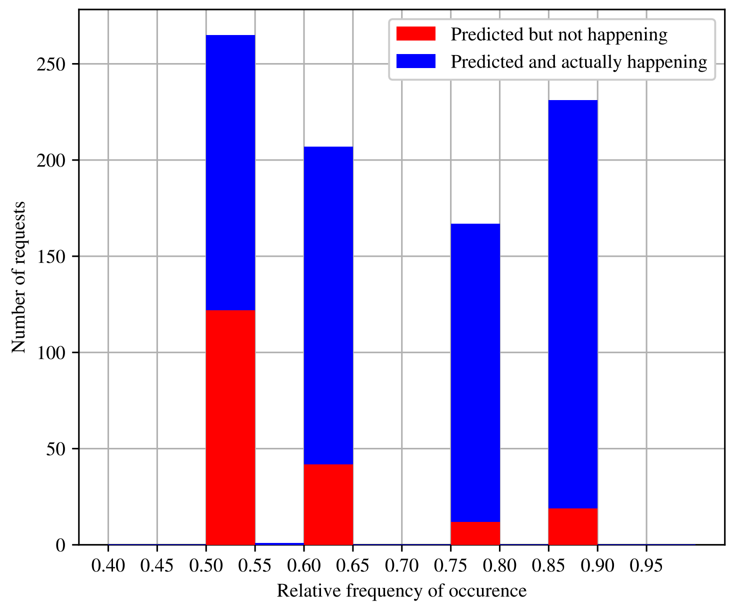


Figure 7.14: Visualising the quality of the prediction for instance A which is obtained using the anticipatory method.

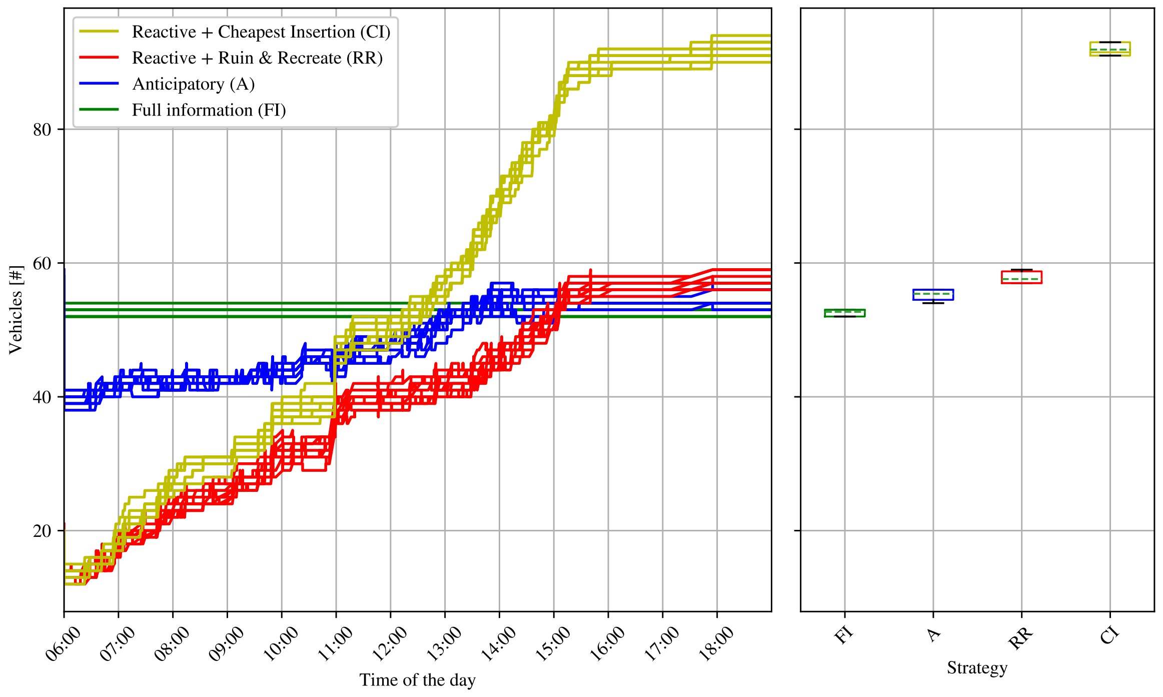


Figure 7.15: Performance of different optimisation strategies on instance F. On the left it is shown how the number of vehicles that is planned to be used during the entire day evolves throughout the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the number of used vehicles during the day for each strategy are shown.

8 CONCLUSIONS

The objective of this research was to develop an algorithm which is able to more efficiently control a fleet of vehicles in a situation where additional customers may request service while vehicles are already driving. The focus was put on attempting to improve the performance of the fleet of vehicles by making use of historical data. To do this a new anticipatory insertion method was developed. This approach consists of three parts.

First, a new method for predicting requests based on historical data was introduced. This prediction method assumes the presence of patterns within the occurrence of similar requests. An easily extendable clustering method for grouping similar requests was introduced. By making use of intuitive clustering parameters at multiple levels this method can be tuned to a specific problem context with relative ease. In the design of this clustering method a large scale application is kept in mind and several measures are taken to improve the performance in terms of computational efficiency. As a result the proposed method is able to handle at least hundreds of thousands, and possibly millions, of requests. An intuitive prediction model which allows for the usage of multiple specific patterns was proposed. The prediction model furthermore contains a parameter which allows for control over the degree by which patterns should be followed before requests are actually predicted, essentially allowing control over the reticence of the prediction.

Second, methods for adding, replacing and removing predicted requests are introduced. These methods ensure that any structure within the solution of a dynamic vehicle routing problem, imposed by the presence of predicted requests, is preserved while re-optimisation can be performed to let the solution adapt to new information. Especially the novel replacement method, which makes use of parts of the clustering method used for predicting requests, is not seen in the literature before.

Third, a parallelised dynamic vehicle routing solver, which makes use of heuristics, was developed. It was shown that this solver is able to find best known solutions on benchmark problem instances with up to several hundreds of requests within seconds. In doing so the implementation is able to significantly outperform one of the most cited methods in the literature [19] both in terms of solution quality and computation time. Furthermore, by making use of this solver the research is also one of the first works which combines anticipatory routing with an adaptive variable large neighbourhood search approach instead of merely using local search methods which are known to produce significantly lesser solution quality.

Fourth, the entire method is tuned and extensively evaluated on 10 real-world problem instances with up to 1.655 requests per day. These instances are multiple times larger than the instances which are generally solved in the literature. It is shown that, on average, it possible to improve upon a competitive reactive approach by 4,58% in terms of the distance that is required to be travelled. Additionally, 3,35% fewer vehicles are required to fulfil the same set of requests. It is worth mentioning that the competitive reactive approach already performs only 15,40 % worse as compared to the also evaluated full information approach. Therefore, by using the proposed method, merely 10,07% additional distance is required to be travelled as compared to knowing about all requests in advance.

While doing so it can be concluded that it is indeed possible to improve upon a competitive reactive approach by making use of historical data. Furthermore, as simultaneously both the number of vehicles as the total distance which is travelled by these vehicles is reduced, it can be concluded that this can be done without any detrimental side effects, besides having to deal with a relatively more complicated implementation.

8.1. RECOMMENDATIONS

The presented results have shown that significant theoretical improvements can be achieved in terms of operational efficiency for controlling a fleet of vehicles operated by a transportation company. These improvements could contribute to lowering the cost of transportation and specifically the cost of last-mile delivery over the road by requiring less distance to be travelled while also requiring fewer vehicles to fulfil the same set of requests. However, even though this work may be seen a solid proof-of-concept, several shortcomings and remaining opportunities for further research can still be identified.

A first opportunity is related to the evaluated problem instances. In this research 10 different problem instances belonging to a transportation company were evaluated. Even though these instances show some variance in terms of size and days of the week and months of the year that they represent, they still belong to the same single transportation company. It would be interesting to see how the presented results generalise to different comparable transportation companies and other problem contexts where the spatial and temporal properties of requests may be very different. It is worth mentioning that even though a significant amount of effort has been put into acquiring additional data-sets, only a single problem context could be obtained as acquiring multiple consecutive weeks or months of request data was found to be relatively hard due to privacy legislation.

Second, multiple opportunities for further improvement can also be identified related to the used prediction model. First, as only relatively simple patterns were used, more complicated patterns such customers requesting goods every other week or customers requesting goods every several days could be explored. Also a prediction model which works on address level, in an attempt to predict individual customer behaviour, may also yield better results in certain problem contexts. Essentially, further research should focus on designing a more accurate prediction model, possibly by making use of neural networks and other tools related to machine learning.

Third, the proposed methods for replacing and removing predicted requests could possibly also be improved. For example, when replacing requests, neighbouring request types could also be explored when a replacement initially appears not be possible. Also the removal of predicted requests, could perhaps be delayed by a certain amount so that an actual replacement might be made possible when there is a slight deviation in the time at which requests become known. Additionally, a modified request classifier may be developed which also takes the current solution into account and allows for the replacement of a slightly different predicted request when the solution severely benefits from doing this. However, as all these suggestions are essentially methods to better deal with an inaccurate prediction, it is believed that the emphasis should lie on creating a more accurate prediction.

Fourth, a shortcoming is found in the vehicle routing solver which, despite being very fast, appears to get stuck in different local minima. Therefore, even though it was specifically decided not to do so, implementing a method which also allows for deteriorating solution quality while trying to find a better solution (such as simulated annealing) may be beneficial when the related challenges described in Section 6.1 can be addressed.

Finally, a comparison of the proposed method with the Multiple Scenario Approach may also be pursued. It would be interesting to see whether dividing and dedicating the available computational resources to separate scenarios, to better incorporate stochastic information, weighs up against being able to dedicate more resources to the optimisation of a single scenario to achieve better solution quality. Before doing this it worth mentioning that in Section 7.2 it was seen that when the simulation speed is increased by a factor 30 the solution quality deteriorates by approximately 10%. This suggests that when optimising 30 scenarios in parallel, with the same amount of computational resources, using the MSA, the solution quality of each individual scenario would also be approximately 10% worse as compared to the single scenario which would be used by the proposed method.

9

Acronyms

$\texttt{rf}_o$ Relative Frequency of Occurrence. x, xiii, 23, 24, 23, 24, 46, 47, 50, 51, 55

ARIMA Autoregressive Integrated Moving Average. 23

ARMA Autoregressive Moving Average. 23

CVRP Capacitated Vehicle Routing Problem. 3, 4

DOD Degree of Dynamism. 8, 42

DVRP Dynamic Vehicle Routing Problem. 3, 8, 9, 17, 27, 31, 35, 37, 47

eDOD Effective Degree of Dynamism. 8

GA Genetic Algorithms. 7, 9

GSA Global Stochastic Assessment. 10

HACLC Hierarchical Agglomerative Complete Linkage Clustering. 20, 21, 24

LNS Large Neighbourhood Search. 8, 69

LS Local Search. 6, 7, 8, 9, 10, 31, 33, 69, 74, 77

MSA Multiple Scenario Approach. 10, 60

NRR Normalised Request Relatedness. 72, 73

PDPTW Capacitated Pickup and Deliver Vehicle Routing Problem with Time Windows. 8, 37

PDVRP Pickup and Delivery Vehicle Routing Problem. 3, 4

R&R Ruin & Recreate. ix, xiii, 8, 31, 32, 33, 34, 35, 36, 37, 38, 39, 47, 48, 49, 54, 69, 74, 75, 77

SA Simulated Annealing. 7, 8, 10, 35

TS Tabu Search. 7, 8, 10

TSP Travelling Salesman Problem. 3, 4, 10

UTAI Up to and including. 42

VRP Vehicle Routing Problem. 3, 9, 16, 31, 35, 71

VRPTW Vehicle Routing Problem with Time Windows. 3, 4

BIBLIOGRAPHY

[1] H. Zou and M. M. Dessouky, *A look-ahead partial routing framework for the stochastic and dynamic vehicle routing problem,* Journal on Vehicle Routing Algorithms **1**, 73 (2018).

[2] T. Rokicki, *E-Commerce Market in Europe in B2C,* Information Systems in Management **7**, 133 (2018).

[3] M. Parry, O. Canziani, J. Palutikof, P. van der Linden, and C. Hanson, *Climate Change 2007: Impacts, Adaptation and Vulnerability*, Tech. Rep. (2007).

[4] M. Barth and K. Boriboonsomsin, *Real-World Carbon Dioxide Impacts of Traffic Congestion,* Transportation Research Record **2058**, 163 (2008).

[5] G. B. Dantzig and J. H. Ramser, *The Truck Dispatching Problem,* Management Science **6**, 80 (1959).

[6] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, MOS-SIAM Series on Optimization (SIAM, 2014).

[7] Z. Borčinova, *Two models of the capacitated vehicle routing problem,* Croatian Operational Research Review **8**, 463 (2017).

[8] M. M. Solomon and J. Desrosiers, *Survey Paper — Time Window Constrained Routing and Scheduling Problems,* Transportation Science **22** (1988), 10.1287/trsc.22.1.1.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Computers and Intractability **24**, 90 (1979).

[10] M. W. P. Savelsbergh, *Local search in routing problems with time windows,* Annals of Operations Research **4**, 285 (1985).

[11] D. Bertsimas, P. Jaillet, and S. Martin, *Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications,* Operations Research (2018).

[12] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa, *Improved branch-cut-and-price for capacitated vehicle routing,* Mathematical Programming Computation **9**, 61 (2017).

[13] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, *On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,* Proceedings of the National Academy of Sciences **114**, 462 (2017).

[14] G. Laporte, H. Mercure, and Y. Nobert, *An exact algorithm for the asymmetrical capcitated vehicle routing problem,* Networks **16**, 33 (1986).

[15] S. Martello, M. Minoux, C. Ribeiro, and G. Laporte, *Surveys in Combinatorial Optimization,* North-Holland Mathematics Studies **31** (1987).

[16] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet, *Classical and modern heuristics for the vehicle routing problem,* International Transactions in Operational Research **7**, 285 (2000).

[17] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, *The vehicle routing problem: State of the art classification and review,* Computers and Industrial Engineering **99**, 300 (2016).

[18] J. Naoum-Sawaya, R. Cogill, B. Ghaddar, S. Sajja, R. Shorten, N. Taheri, P. Tommasi, R. Verago, and F. Wirth, *Stochastic optimization approach for the car placement problem in ridesharing systems,* Transportation Research Part B: Methodological **80**, 173 (2015).

[19] S. Ropke and D. Pisinger, *An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows,* Transportation Science **40**, 393 (2006).

[20] J. F. Cordeau, G. Laporte, M. W. Savelsbergh, and D. Vigo, *Vehicle Routing,* in *Handbooks in Operations Research and Management Science*, Vol. 14 (2007) pp. 367–428, arXiv:arXiv:1011.1669v3 .

[21] O. Bräysy and M. Gendreau, *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms,* Transportation Science **39**, 104 (2005).

[22] C. Groër, B. Golden, and E. Wasil, *A library of local search heuristics for the vehicle routing problem,* Mathematical Programming Computation **2**, 79 (2010).

[23] S. Kirkpatrick, C. D. J. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing,* Science **220**, 671 (1983).

[24] F. Glover, *Tabu Search - Part I,* ORSA Journal on Computing **1**, 135 (1989), arXiv:arXiv:1011.1669v3 .

[25] R. Moretti Branchini, V. Amaral Armentano, and A. Løkketangen, *Adaptive granular local search heuristic for a dynamic vehicle routing problem,* Computers and Operations Research **36**, 2955 (2009).

[26] J. H. Holland, *Genetic Algorithms,* Scientific American **267**, 66 (1992).

[27] F. T. Hanshar and B. M. Ombuki-Berman, *Dynamic vehicle routing using genetic algorithms,* Applied Intelligence **27**, 89 (2007).

[28] P. Shaw, *A new local search algorithm providing high quality solutions to vehicle routing problems,* (1997).

[29] G. Schrimpf, J. Schneider, H. Stamm-wilbrandt, and G. Dueck, *Record Breaking Optimization Results Using the Ruin and Recreate Principle,* Journal of Computational Physics **159**, 139 (2000).

[30] H. Li and A. Lim, *A Metaheuristic for the Pickup and Delivery Problem with Time Windows,* in *13th IEEE International Conference on Tools with Artificial Intelligence* (2001).

[31] R. Bent and P. V. Hentenryck, *A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows,* Computers and Operations Research **33**, 875 (2006).

[32] J. Christiaens and G. V. Berghe, *A Fresh Ruin & Recreate Implementation for the Capacitated Vehicle Routing Problem*, Tech. Rep. (2016).

[33] J. Christiaens and G. V. Berghe, *Slack Induction by String Removals for Vehicle Routing Problems*, Tech. Rep. (2018).

[34] S. Dreiseitl and L. Ohno-Machado, *Logistic regression and artificial neural network classification models: A methodology review,* Journal of Biomedical Informatics **35**, 352 (2002).

[35] W. W. M. Kool, H. C. Van Hoof, and M. Welling, *Attention Solves Your TSP,* (2018), arXiv:1803.08475 .

[36] H. N. Psaraftis, M. Wen, and C. A. Kontovas, *Dynamic vehicle routing problems: Three decades and counting,* Networks **67**, 3 (2016).

[37] N. H. M. Wilson and N. J. Colvin, *Computer control of the Rochester dial-a-ride system* (Massachusetts Institute of Technology, Center for Transportation Studies, 1977).

[38] K. Lund, B. G. M. Oli, and J. M. Rygaard, *Vehicle routing problems with varying degrees of dynamism,* Lyngby, Denmark: IMM, The Department of Mathematical Modelling, Technical University of Denmark (1996).

[39] R. Baldacci, M. Battarra, and D. Vigo, *Operations Research / Computer Science Interfaces Series*, Vol. 43 (2008) pp. 3–27.

[40] G. Berbeglia, J. F. Cordeau, and G. Laporte, *Dynamic pickup and delivery problems,* European Journal of Operational Research **202**, 8 (2010).

[41] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update rules,* in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, Vol. 28 (1984) pp. 488–492.

[42] M. Gendreau, G. Laporte, and R. Seguin, *Stochastic vehicle routing,* European Journal of Operational Research .

[43] G. Ghiani, E. Manni, A. Quaranta, and C. Triki, *Anticipatory algorithms for same-day courier dispatching,* Transportation Research Part E: Logistics and Transportation Review **45**, 96 (2009).

[44] S. Ichoua, M. Gendreau, and J. Potvin, *Exploiting Knowledge About Future Demands for Real-Time Vehicle Dispatching,* Transportation Science **40**, 211 (2006).

[45] A. Larsen, O. B. G. Madsen, and M. M. Solomon, *The A Priori Dynamic Traveling Salesman Problem with Time Windows,* Transportation Science **38**, 459 (2004).

[46] J. van Hemert and J. La Poutre, *Dynamic Routing Problems with Fruitful Regions: Models and Evolutionary Computation,* Parallel Problem Solving from Nature - PPSN VIII , 692 (2004).

[47] R. W. Bent and P. Van Hentenryck, *Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers,* Operations Research **52**, 977 (2004).

[48] G. Ghiani, E. Manni, and B. W. Thomas, *A Comparison of Anticipatory Algorithms for the Dynamic and Stochastic Traveling Salesman Problem,* Transportation Science **46**, 374 (2012).

[49] M. Saint-Guillain, Y. Deville, and C. Solnon, *A Multistage Stochastic Programming Approach to the Dynamic and Stochastic VRPTW - Extended version,* in *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming* (2015) pp. 357–374.

[50] B. W. Thomas, *Waiting Strategies for Anticipating Service Requests from Known Customer Locations,* Transportation Science **41**, 319 (2007).

[51] J. Branke, M. Middendorf, G. Noeth, and M. Dessouky, *Waiting Strategies for Dynamic Vehicle Routing,* Transportation Science **39**, 298 (2005).

[52] T. Andersson and P. Värbrand, *Decision support tools for ambulance dispatch and relocation,* Journal of the Operational Research Society **58**, 195 (2007).

[53] A. Wallar, M. V. D. Zee, J. Alonso-Mora, and D. Rus, *Vehicle Rebalancing for Mobility-on-Demand Systems with Ride-Sharing,* in *IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)* (2018).

[54] G. Laporte and J.-F. Cordeau, *The dial-a-ride problem: models and algorithms,* Annals of Operations Research **153**, 29 (2007).

[55] G. Laporte, R. Musmanno, F. Vocaturo, G. Laporte, R. Musmanno, and F. Vocaturo, *An Adaptive Large Neighbourhood Search Heuristic for the Capacitated Arc-Routing Problem with Stochastic Demands,* Transportation Science **44**, 125 (2010).

[56] H.-h. Bock, *Origins and extensions of the k -means algorithm in cluster analysis,* Electronic Journal for History of Probability and Statistics **4**, 1 (2008).

[57] J. Ward, *Hierarchical Grouping to Optimize an Objective Function,* Journal of the American Statistical Association **58**, 236 (1963).

[58] S. Zhou, A. Zhou, W. Jin, Y. Fan, and W. Qian, *FDBSCAN: a fast DBSCAN algorithm,* Ruan Jian Xue Bao **11**, 735 (2000).

[59] L. McInnes, J. Healy, and S. Astels, *hdbscan: Hierarchical density based clustering,* The Journal of Open Source Software **2**, 205 (2017).

[60] E. Jones, T. Oliphant, P. Peterson, and Others, *SciPy: Open source scientific tools for Python,* .

[61] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: an efficient data clustering method for very large databases,* in *ACM Sigmod Record*, Vol. 25 (ACM, 1996) pp. 103–114.

[62] D. Defays, *An efficient algorithm for a complete link method,* The Computer Journal **20**, 364 (1977).

[63] F. Cox and A. Cox, *Multidimensional Scaling,* 2 (200).

[64] *Admiralty Manual of Navigation*, BR Series No. 1 (Stationery Office, 1997) pp. 10–11.

[65] C. Chatfield, *The Holt-Winters Forecasting Procedure,* Journal of the Royal Statistical Society **27**, 264 (1978).

[66] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis* (2015).

[67] N. Davey, S. P. Hunt, and R. J. Frank, *Time Series Prediction and Neural Networks,* Journal of intelligent and robotic systems **1**, 91 (2001).

[68] R. Adhikari, *A neural network based linear ensemble framework for time series forecasting,* Neurocomputing **157**, 231 (2015).

[69] R. J. Hyndman and Y. Khandakar, *Automatic Time Series Forecasting : The forecast Package for R,* Journal of Statistical Software **27** (2008).

[70] R. Bent and P. Van Hentenryck, *Waiting and relocation strategies in online stochastic vehicle routing,* in *Proceedings of IJCAI International Joint Conference on Artificial Intelligence* (2007) pp. 1816–1821.

[71] G. Clarke and J. W. Wright, *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points,* Operations Research **12**, 568 (1964).

[72] Y. Nagata and O. Bräysy, *A powerful route minimization heuristic for the vehicle routing problem with time windows,* Operations Research Letters **37**, 333 (2009).

[73] R. Krishnamurti and G. Laporte, *Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows,* Transportation Research Part B: Methodological **38**, 669 (2004).

[74] M. M. Solomon, *Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints,* Operations Researchr **35**, 254 (1987).

[75] V. Jakubiuk and S. Popovic, *Implementation and Performance Analysis of Hash Functions and Collision Resolutions,* (2012).

Appendices

A
Implementation of the Vehicle Routing Solver

This chapter will describe how the vehicle routing solver covered in Chapter 6 is implemented in C++, generally considered to be the fastest programming language available. As described there, to solve vehicle routing problems the solver goes through three phases namely construction, Local Search and Ruin & Recreate (or Large Neighbourhood Search). How each individual phase is implemented is covered in the following sections. Finally, all symbols in this chapter are defined as listed in Table 3.1.

A.1. Insertion

Insertion heuristics are used to insert unscheduled requests into an existing (empty) solution. In this section first some general components, which are used by all these insertion heuristics, will be covered in Section A.1.1. Then three insertion heuristics are covered in sections A.1.2 till A.1.4. Finally, the self-developed insertion heuristic, named "cluster insertion" is covered in Section A.1.5.

A.1.1. General Components

All insertion heuristics make use of two general components. First, all insertion possibilities are computed. Second, the feasibility of such an insertion possibility is required to be checked. How these two components are implemented is described below.

Computing Insertion Possibilities

All insertion heuristics make use of lists of possibilities where a request can be inserted into an existing solution. Such a list is created by combining the lists of all possibilities for inserting the request into every existing route. The number of possibilities for inserting a request into a route w_k depends on the number of tasks z_k which are already being fulfilled by that route. The number of insertion possibilities for a pickup task, belonging to a single request, within route w_k is equal to $z_k + 1$. The number of insertion possibilities for a deliver task depends on the insertion location of the pickup task as the deliver task is required to be fulfilled after its corresponding pickup task. For each insertion possibility of a pickup task at location $z_{k,i}$ the number of insertion possibilities for the corresponding deliver task is therefore equal to $(z_k + 1) - i$. The number of insertion possibilities $|P_k|$ for a request into a route is therefore as defined by Equation 3.2a. For example, a route which is already fulfilling a 100 requests (or contains 200 tasks), therefore has over 20 thousand insertion possibilities for a single request.

$$|P_k| = \sum_{i \in 0,1,\ldots(z_k-1)} (z_k + 1) - i \quad \approx \quad \frac{1}{2} z_k^2 \tag{3.2a}$$

As the objective of the problem defined in Section 3.1 is to minimise the distance travelled by all vehicles, the cost of each insertion possibility should reflect the corresponding change in this objective. To achieve this the cost of each insertion possibility is defined by the sum of the distance of the additional arcs that need to be traversed minus the sum of the distance of the arcs that do not have to be traversed any more after inserting a request. For clarity, this sum has been visualised in Figure 2.2. It is worth mentioning that when both the pickup task and the deliver task are inserted at the same location, only three additional arcs are traversed and only one arc does not have to be traversed any more. This means that for the calculation of the cost of a single request insertion possibility, only 4 to 6 distance values need to be summed and subtracted. As this is computationally a very cheap operation, all insertion possibilities are first computed (also unfeasible possibilities) where after all possibilities are sorted and feasibility is only checked for possibilities that are desired to be realised, minimising the required computational effort.

CHECKING FEASIBILITY OF AN INSERTION POSSIBILITY

For an insertion possibility to be feasible, no capacity and time constraints should be violated. To efficiently check if time constraints are not violated a "push-forward" and "push-backward" method first described by Solomon [74] is used. By looping forward over all tasks already scheduled within a route the earliest possible start $es_{k,l}$ of each task $z_{k,l}$ at position l in task sequence Z_k of route w_k is computed using the procedure listed in Algorithm 2. The corresponding latest start $ls_{k,l}$ is computed using the procedure listed in Algorithm 3. In these algorithms are symbols are as defined in Table 3.1. Using the earliest and latest start of two consecutive scheduled tasks surrounding an insertion location it can be determined if the two tasks can be shifted apart far enough to accommodate one of the to-be inserted tasks belonging to a request. Furthermore it is checked if the to-be inserted task can still start within the bounds of its own time window. To check if no time constraints are violated when inserting the entire request $r_{i,us}$, consisting of both a pickup task and a deliver task, into route w_k, the procedure listed in Algorithm 4 is used. It is worth mentioning that checking the feasibility of an insertion possibility is computationally a relatively expensive operation and should therefore be performed as little as possible. To reduce the number of times it is performed, pre-checks can be used. These are introduced in Section A.4.1.

Algorithm 2 Determining the earliest possible start of a scheduled task

```
1: procedure EARLIESTPOSSIBLESTART($z_{k,l}$ , $w_k$ )
2:     $j \leftarrow$ getNodeIndexOf( $z_{k,l}$ )
3:     if $l = 0$ then                                          ▷ If it is the first task in the route
4:         $v_1 \leftarrow e_{2n+k} + t_{2n+k,j}$   ▷ Start of the vehicle time window plus the required travel time for travelling from the vehicle start location to the task location
5:     else
6:         $i \leftarrow$ getNodeIndexOf( $z_{k,l-1}$ )
7:         $v_1 \leftarrow es_{k,l-1} + d_i + t_{i,j}$   ▷ The earliest start of the previous task plus the duration of that task plus the required travel time for travelling from the previous task location to the current task location
8:     end if
9:     $v_2 \leftarrow e_j$                                     ▷ The start of the time window of the task
10:    return max($v_1, v_2$)
11: end procedure
```

Algorithm 3 Determining the latest possible start of a scheduled task

```
1: procedure LATESTPOSSIBLESTART($z_{k,l}$ , $w_k$ )
2:     $i \leftarrow$ getNodeIndexOf( $z_{k,l}$ )
3:     if $l = z_k - 1$ then                                    ▷ If it is the last task in the route
4:         $v_1 \leftarrow l_{2n+k} - t_{i,2n+m+k}$   ▷ End of the vehicle time window minus the required travel time for travelling from the task location to the vehicle end location
5:     else
6:         $j \leftarrow$ getNodeIndexOf( $z_{k,l+1}$ )
7:         $v_1 \leftarrow es_{k,l+1} - t_{i,j} - d_i$   ▷ The latest start of the next task in the sequence $Z_k$ minus the required travel time for travelling from the current task to the next task minus the duration of the current task
8:     end if
9:     $v_2 \leftarrow l_i$                                     ▷ The end of the time window of the task
10:    return min($v_1, v_2$)
11: end procedure
```

Algorithm 4 Checking time constraints for an insertion possibility

```
1: procedure EARLIESTPOSSIBLESTART($z_{k,l}$ , $w_k$ )
2:     $j \leftarrow$ getNodeIndexOf( $z_{k,l}$ )
3:     if $l = 0$ then                                        ▷ If it is the first task in the route
4:         $v_1 \leftarrow e_{2n+k} + t_{2n+k,j}$   ▷ Start of the vehicle time window plus the required travel time for travelling from the vehicle start location to the task location
5:     else
6:         $i \leftarrow$ getNodeIndexOf( $z_{k,l-1}$ )
7:         $v_1 \leftarrow es_{k,l-1} + d_i + t_{i,j}$   ▷ The earliest start of the previous task plus the duration of that task plus the required travel time for travelling from the previous task location to the current task location
8:     end if
9:     $v_2 \leftarrow e_j$                                   ▷ The start of the time window of the task
10:    return max($v_1$, $v_2$)
11: end procedure
```

A.1.2. SEQUENTIAL CHEAPEST INSERTION

The sequential cheapest insertion heuristic, often also referred to as greedy insertion, is one of the simplest and fastest heuristics to construct a solution to a VRP. For each unscheduled request it computes all insertion possibilities within the current solution and realises the best possibility if there is one. As this is done sequentially, for all unscheduled requests, the initial order of the unscheduled requests may impact the finally obtained solution. The procedure is listed in Algorithm 5. In this algorithm all symbols are as defined in Section 3.1.

Algorithm 5 Sequential Cheapest Insertion

```
1: procedure SEQUENTIALCHEAPESTINSERTION( $S$ , $R_{us}$ )
2:     for $r_{i,us} \in R_{us}$ do
3:         $P \leftarrow$ computeInsertionPossibilities( $r_{i,us}$ , $S$ )   ▷ Returns a set of all possibilities
4:         $P \leftarrow$ sortBySmallestToLargestCost( $P$ )
5:         $p \leftarrow$ findFirstFeasiblePossibility( $P$ )              ▷ Returns a single possibility
6:         $S \leftarrow$ realiseIfFoundPossibility( $p$ , $S$ )   ▷ Returns a solution with the inserted request, if possible
7:     end for
8:     return $S$
9: end procedure
```

A.1.3. PARALLEL CHEAPEST INSERTION

The parallel cheapest insertion heuristic expands on the sequential cheapest insertion heuristic by making the initial order of the unscheduled requests not of importance. This is done by computing all insertion possibilities for all unscheduled requests before each insertion. This ensures that each time the cheapest feasible insertion possibility across all unscheduled requests is realised. This comes at the cost of having to compute and handle a much larger list of insertion possibilities before each insertion. The procedure is listed in Algorithm 6. In this algorithm all symbols are as defined in Section 3.1. It is worth mentioning that after an insertion possibility has been realised into a single route, all insertion possibilities for all other unscheduled requests, in all other routes, remain unaffected. Because of this, an efficient implementation should only recompute possibilities for the remaining unscheduled requests in the affected route.

Algorithm 6 Parallel Cheapest Insertion

```
1: procedure PARALLELCHEAPESTINSERTION( $S$ , $R_{us}$ )
2:     while $n_{us} > 0$ do
3:         $P \leftarrow \{\}$                                                ▷ Initialise an empty set
4:         for $r_{i,us} \in R_{us}$ do
5:             $P \leftarrow P \cup$ computeInsertionPossibilities( $r_{i,us}$ , $S$ )   ▷ Append all possibilities for a request
6:         end for
7:         $P \leftarrow$ sortBySmallestToLargestCost( $P$ )
8:         $p \leftarrow$ findFirstFeasiblePossibility( $P$ )                     ▷ Returns a single possibility
9:         $S \leftarrow$ realiseIfFoundPossibility( $p$ , $S$ )   ▷ Returns a solution with the inserted request, if possible
10:    end while
11:    return $S$
12: end procedure
```

A.1.4. REGRET INSERTION

The regret insertion heuristic tries to improve upon the parallel cheapest insertion heuristic by 'looking ahead' and not simply choosing the cheapest insertion possibility for the current solution. The heuristic does this by, for each unscheduled request, besides computing the cheapest feasible insertion possibility, also computing subsequent feasible insertion possibilities. For each unscheduled request, the cost of each cheapest feasible insertion possibility is then defined as the cost difference between that possibility and one of its subsequent feasible insertion possibilities. Basically the cost for inserting each unscheduled requests is set equal to the possible increase in cost which may be encountered when the first possibility is not realisable any more after another unscheduled request has been inserted. This explains the name of the heuristic as the request with the largest possible regret, from not inserting at that moment, is inserted. The procedure of this heuristic is listed in Algorithm 7. In this algorithm all symbols are as defined in Section 3.1.

Algorithm 7 Regret Insertion

```
1: procedure KREGRETINSERTION( $S$ , $R_{us}$ , $k$ )
2:     while $n_{us} > 0$ do
3:         $P \leftarrow \{\}$                                                ▷ Initialise an empty set
4:         for $r_{i,us} \in R_{us}$ do
5:             $Q \leftarrow$ computeInsertionPossibilities( $r_{i,us}$ , $S$ )      ▷ Compute all possibilities for this request
6:             $Q \leftarrow$ sortBySmallestToLargestCost( $Q$ )
7:             $p \leftarrow$ findFirstFeasiblePossibility( $Q$ )                 ▷ Returns a single possibility
8:             $p_{regret} \leftarrow$ findKthFirsFeasiblePossibility( $Q$ , $k$ )      ▷ Returns the $k$-th feasible possibility
9:             $c_{regret} \leftarrow$ computeDifferenceInCostBetween( $p$ , $p_{regret}$ )
10:            $p \leftarrow$ changeCostOfInsertionPossibilityTo( $p$ , $c_{regret}$ )
11:            $P \leftarrow P \cup p$
12:        end for
13:        $P \leftarrow$ sortBySmallestToLargestCost( $P$ )
14:        $p \leftarrow$ getFirst( $P$ )                                      ▷ Returns a single possibility
15:        $S \leftarrow$ realisePossibility( $p$ , $S$ )             ▷ Returns a solution with the inserted request
16:    end while
17:    return $S$
18: end procedure
```

A.1.5. CLUSTER INSERTION

A new construction heuristic named cluster insertion is developed. Even though initial solutions may be constructed using more conventional heuristics such as sequential or parallel cheapest insertion, an attempt is made at improving upon these heuristics by making use of a measure for the relatedness of requests. Before the procedure is described several concepts are first introduced. First, the Normalised Request Relatedness (NRR) measure is introduced. Second, how seed requests are selected is described. Then, how cluster sizes for seed requests are determined using the NRR measure is described.

DETERMINING REQUEST RELATEDNESS

A variety of methods can be used to determine the relatedness of two requests. It may be said that two requests are related when both their pickup and deliver locations are quite similar. However, when the goal is to find requests which may be served efficiently by the same vehicle, other options arise as well. In the case of having two requests R_1 and R_2 consisting of a pickup tasks P_1 and P_2 and deliver tasks D_1 and D_2 there are in fact six possibilities for the ordering of these tasks are visited so that R_2 may be considered as being related to R_1. These possibilities, together with their requirements for R_2 to be considered as being related to R_1, are listed in Table A.1. However, given that all four tasks also have time windows, some orderings may not be feasible in certain cases. This feasibility is checked by making use of the methods described in Section A.1.1. For the feasible orderings the relatedness is defined as the additionally required travelling distance as compared to only travelling from P_1 to D_1. The relatedness of request a R_2 to R_1 is defined as the minimum additionally required travelling distance of all feasible orderings.

By computing this request relatedness of all requests, towards all other requests within the problem definition, a request relatedness matrix can be obtained. This matrix is then normalised to only contain values between zero and one. Finally by subtracting each value from one, a Normalised Request Relatedness (NRR) measure is obtained which equals one when two requests are very related and zero when two requests are unrelated. When the ordering of the tasks is not feasible for any of the six possibilities, the normalised relatedness of two requests is defined as zero, which states that they are incompatible.

Table A.1: Possibilities for the ordering of the tasks of two requests to determine if R_2 is related to R_1

Order of tasks	**Requirements for request R_2 considered to be related to R_1**
P_1, P_2, D_1, D_2	The travelling distance from P_1 to P_2 to D_1 should not be much larger than the travelling distance from P_1 to D_1. The the travelling distance from D_1 to D_2 should be small.
P_1, P_2, D_2, D_1	The travelling distance from P_1 to P_2 to D_2 to D_1 should not be much larger than the travelling distance from P_1 to D_1.
P_1, D_1, P_2, D_2	The the travelling distance from D_1 to P_2 should be small. The the travelling distance from P_2 to D_2 should be small.
P_2, P_1, D_1, D_2	The the travelling distance from P_2 to P_1 should be small. The the travelling distance from D_1 to D_2 should be small.
P_2, P_1, D_2, D_1	The the travelling distance from P_2 to P_1 should be small. The travelling distance from P_1 to D_2 to D_1 should not be much larger than the travelling distance from P_1 to D_1.
P_2, D_2, P_1, D_1	The the travelling distance from P_2 to D_2 should be small. The the travelling distance from D_2 to P_1 should be small.

SELECTING SEED REQUESTS

The request which has the highest average relatedness to a certain number of unscheduled requests and is least related to any of the already scheduled requests is chosen as a seed request. For each request the five requests which have the highest normalised relatedness are selected. The normalised relatedness of these requests is averaged. Then, the highest relatedness to any of the already scheduled is determined. This value is subtracted from the average highest relatedness to the five selected unscheduled requests. This final value is calculated for each unscheduled request and the request which has the highest value is chosen as the next seed request. Additionally, when this the value falls below a threshold (for example 0.05), no further seed requests may selected.

DETERMINING CLUSTER SIZE

The cluster size of a seed request is determined by making use of the normalised request matrix which was covered in Section A.1.5. The cluster size is defined as the number of unscheduled requests which have a normalised request relatedness, towards this request, which is larger than a certain relatively high threshold (for example 0.95). Furthermore, to prevent clusters becoming to small or to big, the number of requests can constrained to be within a certain range (for example larger than 5 requests and smaller than 5% of the total number of requests within the problem definition).

PROCEDURE

The entire procedure for cluster insertion is listed in Algorithm 8. The algorithm first tries to find a seed request. It then inserts this request into a new route and then inserts the related requests (or in other words, the cluster of the seed request) to fill up the created route. A new seed request is then chosen and the process is repeated. When no new seed request is found the remaining unscheduled requests are inserted using sequential cheapest insertion as described in Section A.1.2.

Algorithm 8 Cluster Insertion

```
1: procedure CLUSTERINSERTION( $S$ , $R_{us}$ , $n$ )
2:     while $n_{us} > 0$ do
3:         if there are no empty routes remaining then
4:             return $S$
5:         else
6:             $w_k \leftarrow$ getEmptyRoute( $S$ )                                ▷ Returns an empty route
7:         end if
8:         $r_{i,us} \leftarrow$ getSeedRequestWhichIsMostRelatedToFirstNButLeastRelatedTo( $R_{us}$ , $R_s$, $n$ )
9:         if $r_{i,us}$ was not found then
10:            $S \leftarrow$ sequentialCheapestInsertion($S$, $R_{us}$)
11:            return $S$
12:        end if
13:        $P \leftarrow$ computeInsertionPossibilitiesInRoute( $r_{i,us}$ , $w_k$ )      ▷ Returns a set of all possibilities
14:        $P \leftarrow$ sortBySmallestToLargestCost( $P$ )
15:        $p \leftarrow$ findFirstFeasiblePossibility( $P$ )                         ▷ Returns a single possibility
16:        $S \leftarrow$ realisePossibility( $p$ , $S$ )            ▷ Returns a solution with the inserted request
17:        $n_2 \leftarrow$ determineClusterSizeOf( $r_{i,us}$ )                               ▷ Returns a number
18:        $R_{i,us_{related}} \leftarrow$ getNUnscheduledRequestsMostRelatedTo( $r_{i,us}$ , $n_2$ )  ▷ Returns a set of requests
19:        for $r_{i,us_{related}} \in R_{i,us_{related}}$ do
20:            $P \leftarrow$ computeInsertionPossibilitiesInRoute( $r_{i,us}$ , $w_k$ )  ▷ Returns a set of all possibilities
21:            $P \leftarrow$ sortBySmallestToLargestCost( $P$ )
22:            $p \leftarrow$ findFirstFeasiblePossibility( $P$ )                     ▷ Returns a single possibility
23:            $S \leftarrow$ realisePossibility( $p$ , $S$ )        ▷ Returns a solution with the inserted request
24:        end for
25:    end while
26:    return $S$
27: end procedure
```

A.2. LOCAL SEARCH

Three local search operators are implemented. The shift and rearrange operators have a shared implementation. The two-opt operator is implemented seperatly. The three implementations are described in the following subsections.

COMBINED SHIFT AND REARRANGE OPERATOR

To implement the combined shift and rearrange operator, methods for scheduling and unscheduling requests are required. These methods are essentially the same methods that need to be implemented for Ruin & Recreate iterations. It therefore makes sense to reuse these methods when they do not perform much worse, in terms of computational efficiency, as compared to methods which may be implemented specifically for the purpose of Local Search. The implementation for the combined shift and rearrange operator is therefore as listed in Algorithm 9. The algorithm sequentially unschedules each scheduled request and reschedules it at the best location at that moment, within any of the routes. During a single iteration each requests is rescheduled a single time. The operator continues with more iterations until not a single solution improvement was found during the previous iteration.

Algorithm 9 Combined Shift and Rearrange Local Search Operator

```
1: procedure SHIFTREARRANGEOPERATOR( $S$ )
2:     while $S$ improved during the previous iteration do
3:         for $r_{i,s} \in R_s$ do
4:             $S \leftarrow$ unscheduleRequest($S$, $r_{i,s}$)                ▷
5:             $S \leftarrow$ sequentialCheapestInsertion($S$, [$r_{i,us}$])
6:         end for
7:     end while
8:     return $S$
9: end procedure
```

2-Opt Operator

The 2-opt operator is implemented by attempting to reversing each possible set of tasks within each route. When reversing a set of tasks yields a solution improvement this solution is used for further modifications. The implementation continues to try every set of tasks in a route until no solution improvement was found during a previous iteration. The entire procedure is thereby as listed in Algorithm 10.

Algorithm 10 2-Opt Local Search Operator

```
1: procedure 2OPTOPERATOR( $S$ )
2:     for $w_k \in W \in S$ do                                   ▷ For each route
3:         while $S$ improved during the previous iteration do
4:             for $z_{k,l} \in Z_k$ do                            ▷ For each task in this route
5:                 for $z_{k,m} \in Z_k$ do                        ▷ For each task in this route
6:                     if $m \le l$ then
7:                         continue
8:                     end if
9:                     $S_{\texttt{iteration}} \leftarrow$ reverseTasksFromToIn($z_{k,l}$, $z_{k,m}$, $S$)
10:                    if $S_{\texttt{iteration}} < S$ then
11:                        $S \leftarrow$ move($S_{\texttt{iteration}}$)
12:                    end if
13:                end for
14:            end for
15:        end while
16:    end for
17:    return $S$
18: end procedure
```

A.3. Ruin & Recreate

As described in Section 6.2.3 the R&R procedure works by removing requests and then reinserting them. The procedure is implemented as listed in Algorithm 11.

The procedure makes use of a variable search neighbourhood. In the implementation this behaviour is controlled by making use of a variable t which is defined as the number of iterations since the last solution improvement was found.

Similarly, the use of local search operators also depends on this variable. Initially, no local search operators are used. After a 100 iterations without improvement the relatively fast two-opt operator is used and only after a 1000 iterations without improvement the much slower shift and rearrange operators are used.

The size of the search neighbourhood is also determined based on the number of iterations since the last solution improvement was found. The number of requests and routes that are removed during each iteration is as defined by Equations 3.2b and 3.2c. These equations state that initially only a small part of the solution may be ruined. After a larger number of iterations without a solution improvement (t) a larger part of the solution may be ruined. However, this part is limited to be at most equal to 30% of the scheduled requests or 30% of the used routes. Furthermore, to increase randomness, actually a random number between these upper and lower bounds is chosen for the number of request or routes that are actually being removed.

$$n = \mathtt{rand}(5, \min(\mathtt{round}(0.30|R_s|), 50 + t)) \tag{3.2b}$$

$$n = \mathtt{rand}(1, \min(\mathtt{round}(0.30|W|), 3 + \mathtt{round}(t/10))) \tag{3.2c}$$

Algorithm 11 Ruin and Recreate

```
procedure ruinAndRecreate( $S$ )
    t ← 0                    ▷ The number of iterations since the last solution improvement was found
    while no stopping condition has been reached do
        removalMethod ← random(allRemovalMethods)
        if t < 1000 then
            insertionMethod ← sequentialCheapestInsertion
        else
            insertionMethod ← random(allInsertionMethods)
        end if

        $S_{\mathtt{iteration}}$ ← copy($S$)
        $n$ ← determineNumberOfRequestsOrRoutesToRemove($S_{\mathtt{iteration}}$, t)
        $S_{\mathtt{iteration}}$ ← removalMethod($S_{\mathtt{iteration}}$, $n$)
        $S_{\mathtt{iteration}}$ ← randomlyOrderUnscheduledRequests($S_{\mathtt{iteration}}$)
        $S_{\mathtt{iteration}}$ ← insertionMethod($S_{\mathtt{iteration}}$, $n$)

        if t > 100 or $S_{\mathtt{iteration}} < S$ then
            if t > 1000 then
                $S_{\mathtt{iteration}}$ ← shiftRearrangeOperator($S_{\mathtt{iteration}}$)
                $S_{\mathtt{iteration}}$ ← twoOptOperator($S_{\mathtt{iteration}}$)
            else
                $S_{\mathtt{iteration}}$ ← twoOptOperator($S_{\mathtt{iteration}}$)
            end if
        end if
        if $S_{\mathtt{iteration}} < S$ then
            $S$ ← copy($S_{\mathtt{iteration}}$)
            $t$ ← 0
        else
            t++
        end if
    end while
    return $S$
end procedure
```

A.4. Reducing Routes

The algorithm used for the reduction of routes is as listed in Algorithm 12.

Algorithm 12 Reducing Routes

```
procedure REDUCINGROUTES($S$)
    $S_{\texttt{procedure}} \leftarrow$ copy($S$)
    $S_{\texttt{procedure}} \leftarrow$ removeRandomRoute($S_{\texttt{procedure}}$)
    while no stopping conditions is reached do
        for $r_{i,us} \in R_{us}$ do                                    ▷ For each unscheduled request
            $S_{\texttt{procedure}} \leftarrow$ sequentialCheapestInsertion($S_{\texttt{procedure}}$, [$r_{i,us}$])
            if $r_{i,us}$ could not be inserted then
                $S_{\texttt{procedure}} \leftarrow$ leastHardestRemovalInsertion($S_{\texttt{procedure}}$, [$r_{i,us}$])
                if $r_{i,us}$ could not be inserted then
                    increaseHardnessOf($r_{i,us}$)
                    $S_{\texttt{procedure}} \leftarrow$ removeRandomRequest($S_{\texttt{procedure}}$)
                end if
            end if
            if $|R_{us}| = 0$ then                           ▷ If no more unscheduled requests remain
                $S \leftarrow$ copy($S_{\texttt{procedure}}$)
                $S_{\texttt{procedure}} \leftarrow$ removeRandomRoute($S_{\texttt{procedure}}$)
            end if
        end for
    end while
    return $S$
end procedure
```

A.4.1. PERFORMANCE IMPROVEMENTS

A significant amount of time is spent on improving the implementation and general performance of the solver. Several measures which are taken to improve the performance are described in the following sections.

HANDLING MEMORY

As C++ allows any implementation to interact with random access memory on a relatively low level, significant performance improvements can be achieved by doing this as efficiently as possible. Numerous measures are taken to do this. For example, lists of request insertion possibilities (as described in Section A.1.1) are actually sorted from largest to smallest cost. By sorting in this way it is possible to delete the first insertion possibility (if it was found to be infeasible) without having to move up all other possibilities one position in memory. Also by constructing objects in vectors as often as possible "in place" and using pointers as often as possible, entire objects are copied and moved as little as possible.

PRE-CHECKS

A pre-check is a computationally cheap operation which checks, in advance, if a specific constraint is not violated when a to be generated insertion possibility (as described in Section A.1.1) would be realised. By making use of these checks, the number of actually generated (infeasible) insertion possibilities can be heavily reduced. This generally means much smaller lists of insertion possibilities are required to be stored, handled and to be fully checked for feasibility when trying to find the best feasible insertion possibility at any moment. Multiple pre-checks are used at various locations to ensure as many infeasible insertion possibilities are disregarded as quickly as possible.

REQUEST INSERTION POSSIBILITY CACHE

When performing a significant amount of Ruin & Recreate iterations (in the order of 10^5 - 10^7) the process of inserting a certain request into a certain route has often already occurred during a previous iteration. To make use of this knowledge and significantly speed up the process of inserting these requests into these routes, the request insertion possibility cache was introduced. This cache eliminates the computation of insertion possibilities for each request that was previously inserted into the same route. This is done by, each time a request is inserted, storing the realised insertion possibility (which is feasible and best according to the objectives) together with a hashed key of the request and route combination. By doing so a single best possibility for inserting a request into a certain route can directly be presented when the situation occurs again. This cache was implemented using a custom implementation of a map in C++, named hopscotch map

[75]. This custom map implementation is required for performance reasons as look ups occur millions of times a second given that millions of insertion possibilities are evaluated each second. It is worth mentioning that even though this cache has the most impact during many Ruin & Recreate iterations, it is also used during Local Search and construction.

B
Scalability of Clustering Implementations

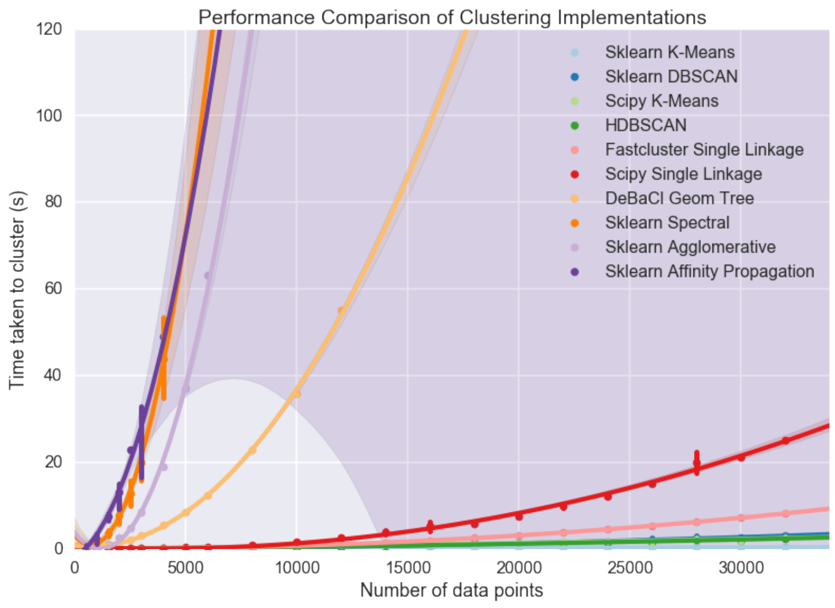


Figure B.1: Scalability of clustering implementations (2019; `https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html`)

C
Results on Li & Lim PDPTW instances

C.1. Instances with approximately 50 requests

Table C.1: Results on Li & Lim PDPTW benchmark instances with approximately 50 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/100-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	**R [#]**	**V [#]**	**D [-]**	t_c **[ms]**	$\mathbf{V_{bk}}$ **[#]**	$\mathbf{D_{bk}}$ **[-]**	**ΔV [%]**	**ΔD [%]**
LC101	53	**10**	**828.94**	7	10	828.94	0.00	0.00
LC102	53	**10**	**828.94**	5	10	828.94	0.00	0.00
LC103	52	**9**	**1035.35**	3648	9	1035.35	0.00	0.00
LC104	53	**9**	**860.01**	228	9	860.01	0.00	0.00
LC105	53	**10**	**828.94**	11	10	828.94	0.00	0.00
LC106	53	**10**	**828.94**	19	10	828.94	0.00	0.00
LC107	53	**10**	**828.94**	9	10	828.94	0.00	0.00
LC108	53	**10**	**826.44**	17	10	826.44	0.00	0.00
LC109	53	**9**	**1000.60**	15766	9	1000.60	0.00	0.00
LC201	51	**3**	**591.56**	27	3	591.56	0.00	0.00
LC202	51	**3**	**591.56**	64	3	591.56	0.00	0.00
LC203	51	**3**	**591.17**	84	3	591.17	0.00	0.00
LC204	51	**3**	**590.60**	92	3	590.60	0.00	0.00
LC205	51	**3**	**588.88**	69	3	588.88	0.00	0.00
LC206	51	**3**	**588.49**	70	3	588.49	0.00	0.00
LC207	51	**3**	**588.29**	92	3	588.29	0.00	0.00
LC208	51	**3**	**588.32**	77	3	588.32	0.00	0.00
LR101	53	**19**	**1650.80**	66	19	1650.80	0.00	0.00
LR102	55	**17**	**1487.57**	145	17	1487.57	0.00	0.00
LR103	52	**13**	**1292.68**	114	13	1292.68	0.00	0.00
LR104	52	**9**	**1013.39**	415	9	1013.39	0.00	0.00
LR105	53	**14**	**1377.11**	101	14	1377.11	0.00	0.00
LR106	52	**12**	**1252.62**	64	12	1252.62	0.00	0.00
LR107	52	**10**	**1111.31**	75	10	1111.31	0.00	0.00
LR108	50	**9**	**968.97**	53	9	968.97	0.00	0.00
LR109	53	**11**	**1208.96**	238	11	1208.96	0.00	0.00
LR110	52	**10**	**1159.35**	11621	10	1159.35	0.00	0.00
LR111	54	**10**	**1108.90**	166	10	1108.90	0.00	0.00
LR112	53	**9**	**1003.77**	321	9	1003.77	0.00	0.00
LR201	51	**4**	**1253.23**	85	4	1253.23	0.00	0.00

Continued on next page

Table C.1 - *Continued from previous page*

Instance	R [#]	V [#]	D [-]	t_c [ms]	V$_{bk}$ [#]	D$_{bk}$ [-]	ΔV [%]	ΔD [%]
LR202	50	**3**	**1197.67**	156	3	1197.67	0.00	0.00
LR203	51	**3**	**949.40**	168	3	949.40	0.00	0.00
LR204	50	**2**	**849.05**	228	2	849.05	0.00	0.00
LR205	51	**3**	**1054.02**	143	3	1054.02	0.00	0.00
LR206	50	**3**	**931.63**	154	3	931.63	0.00	0.00
LR207	51	**2**	**903.06**	203	2	903.06	0.00	0.00
LR208	50	**2**	**734.85**	234	2	734.85	0.00	0.00
LR209	51	**3**	**930.59**	295	3	930.59	0.00	0.00
LR210	51	**3**	**964.22**	134	3	964.22	0.00	0.00
LR211	50	**2**	**911.52**	20638	2	911.52	0.00	0.00
LRC101	53	**14**	**1708.80**	77	14	1708.80	0.00	0.00
LRC102	53	**12**	**1558.07**	160	12	1558.07	0.00	0.00
LRC103	53	**11**	**1258.74**	98	11	1258.74	0.00	0.00
LRC104	54	**10**	**1128.40**	100	10	1128.40	0.00	0.00
LRC105	54	**13**	**1637.62**	215	13	1637.62	0.00	0.00
LRC106	53	**11**	**1424.73**	163	11	1424.73	0.00	0.00
LRC107	53	**11**	**1230.14**	183	11	1230.14	0.00	0.00
LRC108	52	**10**	**1147.43**	421	10	1147.43	0.00	0.00
LRC201	51	**4**	**1406.94**	61	4	1406.94	0.00	0.00
LRC202	51	**3**	**1374.27**	474	3	1374.27	0.00	0.00
LRC203	51	**3**	**1089.07**	176	3	1089.07	0.00	0.00
LRC204	51	**3**	**818.66**	165	3	818.66	0.00	0.00
LRC205	51	**4**	**1302.20**	140	4	1302.20	0.00	0.00
LRC206	51	**3**	**1159.03**	79	3	1159.03	0.00	0.00
LRC207	51	**3**	**1062.05**	115	3	1062.05	0.00	0.00
LRC208	51	**3**	**852.76**	163	3	852.76	0.00	0.00
Average	**51.86**			**1052**			**0.00**	**0.00**
Standard deviation	**1.22**			**3690**			**0.00**	**0.00**

C.2. Instances with approximately 100 requests

Table C.2: Results on Li & Lim PDPTW benchmark instances with approximately 100 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/200-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	R [#]	V [#]	D [-]	t_c [ms]	V$_{bk}$ [#]	D$_{bk}$ [-]	ΔV [%]	ΔD [%]
LC1_2_1	106	**20**	**2704.57**	36	20	2704.57	0.00	0.00
LC1_2_2	105	**19**	**2764.56**	125	19	2764.56	0.00	0.00
LC1_2_3	103	17	3134.08	60264	17	3127.78	0.00	0.20
LC1_2_4	105	17	2724.87	60148	17	2693.41	0.00	1.17
LC1_2_5	107	**20**	**2702.05**	99	20	2702.05	0.00	0.00
LC1_2_6	107	**20**	**2701.04**	107	20	2701.04	0.00	0.00
LC1_2_7	107	**20**	**2701.04**	92	20	2701.04	0.00	0.00
LC1_2_8	105	20	2689.83	60122	19	3354.27	5.26	-19.81
LC1_2_9	105	**18**	**2724.24**	7240	18	2724.24	0.00	0.00
LC1_2_10	104	17	3180.83	60250	17	2942.13	0.00	8.11
LC2_2_1	102	**6**	**1931.44**	72	6	1931.44	0.00	0.00
LC2_2_2	102	**6**	**1881.40**	103	6	1881.40	0.00	0.00
LC2_2_3	101	**6**	**1844.33**	305	6	1844.33	0.00	0.00
LC2_2_4	102	**6**	**1767.12**	782	6	1767.12	0.00	0.00
LC2_2_5	101	**6**	**1891.21**	56	6	1891.21	0.00	0.00

Continued on next page

Table C.2 - *Continued from previous page*

Instance	**R** [#]	**V** [#]	**D** [-]	t_c **[ms]**	$\mathbf{V_{bk}}$ [#]	$\mathbf{D_{bk}}$ [-]	**ΔV [%]**	**ΔD [%]**
LC2_2_6	101	**6**	**1857.78**	215	6	1857.78	0.00	0.00
LC2_2_7	101	**6**	**1850.13**	197	6	1850.13	0.00	0.00
LC2_2_8	102	**6**	**1824.34**	160	6	1824.34	0.00	0.00
LC2_2_9	101	**6**	**1854.21**	207	6	1854.21	0.00	0.00
LC2_2_10	101	**6**	**1817.45**	199	6	1817.45	0.00	0.00
LR1_2_1	105	**20**	**4819.12**	174	20	4819.12	0.00	0.00
LR1_2_2	105	17	4666.09	60117	17	4621.21	0.00	0.97
LR1_2_3	104	15	3823.95	60141	14	4402.38	7.14	-13.14
LR1_2_4	105	11	2880.65	60216	10	3027.06	10.00	-4.84
LR1_2_5	106	16	4785.38	60130	16	4760.18	0.00	0.53
LR1_2_6	107	14	4231.58	60159	13	4800.94	7.69	-11.86
LR1_2_7	103	12	3617.50	60144	12	3543.36	0.00	2.09
LR1_2_8	103	10	2689.43	60160	9	2759.32	11.11	-2.53
LR1_2_9	105	14	4411.30	60202	13	5050.75	7.69	-12.66
LR1_2_10	104	11	4049.23	60191	11	3664.08	0.00	10.51
LR2_2_1	101	**5**	**4073.10**	65	5	4073.10	0.00	0.00
LR2_2_2	101	**4**	**3796.00**	1067	4	3796.00	0.00	0.00
LR2_2_3	101	**4**	**3098.36**	667	4	3098.36	0.00	0.00
LR2_2_4	101	3	2644.65	60113	3	2486.00	0.00	6.38
LR2_2_5	102	**4**	**3438.39**	236	4	3438.39	0.00	0.00
LR2_2_6	100	4	3201.54	60120	3	4518.93	33.33	-29.15
LR2_2_7	101	3	3176.26	60205	3	3098.35	0.00	2.51
LR2_2_8	100	3	2123.78	60091	2	2450.47	50.00	-13.33
LR2_2_9	100	3	4182.41	60361	3	3922.11	0.00	6.64
LR2_2_10	101	3	3361.78	60249	3	3254.83	0.00	3.29
LRC1_2_1	106	**19**	**3606.06**	107	19	3606.06	0.00	0.00
LRC1_2_2	103	15	3711.37	60113	15	3671.02	0.00	1.10
LRC1_2_3	105	13	3182.35	60121	13	3154.92	0.00	0.87
LRC1_2_4	106	**10**	**2631.82**	42730	10	2631.82	0.00	0.00
LRC1_2_5	107	**16**	**3715.81**	851	16	3715.81	0.00	0.00
LRC1_2_6	105	17	3368.66	60137	16	3572.16	6.25	-5.70
LRC1_2_7	106	15	3491.44	60183	14	3666.34	7.14	-4.77
LRC1_2_8	104	14	3086.66	60117	13	3145.74	7.69	-1.88
LRC1_2_9	104	14	3120.24	60136	13	3157.34	7.69	-1.17
LRC1_2_10	105	13	2873.84	60127	12	2928.90	8.33	-1.88
LRC2_2_1	101	6	3690.10	60201	6	3595.18	0.00	2.64
LRC2_2_2	102	5	3467.61	60189	5	3158.25	0.00	9.80
LRC2_2_3	101	4	2886.49	60201	4	2881.99	0.00	0.16
LRC2_2_4	101	4	2184.53	60076	3	2849.43	33.33	-23.33
LRC2_2_5	101	**5**	**2776.93**	173	5	2776.93	0.00	0.00
LRC2_2_6	101	**5**	**2707.96**	153	5	2707.96	0.00	0.00
LRC2_2_7	101	4	3028.34	60222	4	3016.53	0.00	0.39
LRC2_2_8	101	**4**	**2399.89**	1322	4	2399.89	0.00	0.00
LRC2_2_9	101	**4**	**2208.49**	2197	4	2208.49	0.00	0.00
LRC2_2_10	101	3	2546.93	60245	3	2437.88	0.00	4.47
Average	**103.08**			**33086**			**3.38**	**-1.40**
Standard deviation	**2.20**			**29456**			**8.88**	**6.86**

C.3. INSTANCES WITH APPROXIMATELY 200 REQUESTS

Table C.3: Results on Li & Lim PDPTW benchmark instances with approximately 200 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/400-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	**R [#]**	**V [#]**	**D [-]**	t_c **[ms]**	$\mathbf{V_{bk}}$ **[#]**	$\mathbf{D_{bk}}$ **[-]**	**ΔV [%]**	**ΔD [%]**
LC1_4_1	211	**40**	**7152.06**	66	40	7152.06	0.00	0.00
LC1_4_2	211	40	7151.26	60209	38	8007.79	5.26	-10.70
LC1_4_3	210	35	8191.99	60141	32	9163.09	9.38	-10.60
LC1_4_4	208	31	6877.66	60119	N/A	N/A	N/A	N/A
LC1_4_5	211	**40**	**7150.00**	165	40	7150.00	0.00	0.00
LC1_4_6	211	**40**	**7154.02**	372	40	7154.02	0.00	0.00
LC1_4_7	211	**40**	**7149.43**	6098	40	7149.43	0.00	0.00
LC1_4_8	208	**39**	**7111.16**	457	39	7111.16	0.00	0.00
LC1_4_9	209	37	7403.24	60237	36	7451.20	2.78	-0.64
LC1_4_10	208	36	7187.79	60143	35	7325.01	2.86	-1.87
LC2_4_1	203	**12**	**4116.33**	82	12	4116.33	0.00	0.00
LC2_4_2	202	**12**	**4144.29**	938	12	4144.29	0.00	0.00
LC2_4_3	203	12	5065.11	60132	12	4407.71	0.00	14.91
LC2_4_4	203	12	4038.00	60122	N/A	N/A	N/A	N/A
LC2_4_5	203	**12**	**4030.63**	387	12	4030.63	0.00	0.00
LC2_4_6	203	**12**	**3900.29**	469	12	3900.29	0.00	0.00
LC2_4_7	204	**12**	**3962.51**	902	12	3962.51	0.00	0.00
LC2_4_8	203	**12**	**3844.45**	53053	12	3844.45	0.00	0.00
LC2_4_9	205	**12**	**4188.93**	6257	12	4188.93	0.00	0.00
LC2_4_10	202	**12**	**3828.44**	2011	12	3828.44	0.00	0.00
LR1_4_1	208	**40**	**10639.75**	6704	40	10639.75	0.00	0.00
LR1_4_2	209	32	10220.29	60248	30	11026.32	6.67	-7.31
LR1_4_3	208	24	9148.32	60126	22	9291.25	9.09	-1.54
LR1_4_4	210	18	6913.01	60128	15	7546.42	20.00	-8.39
LR1_4_5	206	30	10775.42	60127	28	11374.06	7.14	-5.26
LR1_4_6	211	27	9623.20	60236	24	9872.59	12.50	-2.53
LR1_4_7	208	21	7996.53	60273	18	8999.97	16.67	-11.15
LR1_4_8	211	16	6222.67	60125	14	5848.60	14.29	6.40
LR1_4_9	209	26	10289.25	60222	24	9862.65	8.33	4.33
LR1_4_10	209	21	8292.11	60213	20	8364.66	5.00	-0.87
LR2_4_1	201	**8**	**9726.88**	28325	8	9726.88	0.00	0.00
LR2_4_2	201	7	10115.11	60185	7	9440.93	0.00	7.14
LR2_4_3	202	7	8877.10	60117	5	10658.64	40.00	-16.71
LR2_4_4	202	5	5882.67	60081	4	6403.06	25.00	-8.13
LR2_4_5	202	7	8903.20	60225	6	10084.44	16.67	-11.71
LR2_4_6	201	6	7971.40	60192	5	9044.03	20.00	-11.86
LR2_4_7	203	5	7518.80	60120	4	8263.26	25.00	-9.01
LR2_4_8	202	4	6028.08	60088	4	5303.74	0.00	13.66
LR2_4_9	202	6	9121.43	60190	6	7930.55	0.00	15.02
LR2_4_10	202	6	7095.69	60289	5	7786.13	20.00	-8.87
LRC1_4_1	208	37	8971.61	60168	36	9124.52	2.78	-1.68
LRC1_4_2	209	33	7996.08	60141	31	8346.06	6.45	-4.19
LRC1_4_3	206	25	7516.09	60103	24	7819.90	4.17	-3.88
LRC1_4_4	207	19	5848.39	60066	19	5804.47	0.00	0.76
LRC1_4_5	207	35	8636.49	60177	32	8867.38	9.38	-2.60
LRC1_4_6	208	32	8142.96	60180	30	8423.70	6.67	-3.33
LRC1_4_7	211	31	7870.98	60173	28	8037.87	10.71	-2.08
LRC1_4_8	208	30	7543.31	60102	26	7982.45	15.38	-5.50

Continued on next page

Table C.3 - *Continued from previous page*

Instance	R [#]	V [#]	D [-]	t_c [ms]	V_{bk} [#]	D_{bk} [-]	ΔV [%]	ΔD [%]
LRC1_4_9	209	28	8108.34	60162	25	8145.44	12.00	-0.46
LRC1_4_10	209	27	7228.23	60251	23	7222.97	17.39	0.07
LRC2_4_1	203	12	7503.01	60215	11	10155.25	9.09	-26.12
LRC2_4_2	203	10	7657.69	60171	10	7214.99	0.00	6.14
LRC2_4_3	201	9	5634.10	60081	8	6483.48	12.50	-13.10
LRC2_4_4	203	6	4780.10	60100	5	5300.42	20.00	-9.82
LRC2_4_5	203	11	6120.13	60216	10	7404.23	10.00	-17.34
LRC2_4_6	203	10	5919.18	60155	9	6337.08	11.11	-6.59
LRC2_4_7	202	9	5684.00	60198	8	6322.35	12.50	-10.10
LRC2_4_8	201	8	5197.74	60106	7	5778.36	14.29	-10.05
LRC2_4_9	203	7	6143.11	60105	6	6529.63	16.67	-5.92
LRC2_4_10	203	7	5252.68	60173	6	5551.07	16.67	-5.38
Average	**205.72**			**46438**			**8.18**	**-3.22**
Standard deviation	**3.48**			**24326**			**8.58**	**7.35**

C.4. INSTANCES WITH APPROXIMATELY 300 REQUESTS

Table C.4: Results on Li & Lim PDPTW benchmark instances with approximately 300 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/600-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	R [#]	V [#]	D [-]	t_c [ms]	V_{bk} [#]	D_{bk} [-]	ΔV [%]	ΔD [%]
LC1_6_1	315	**60**	**14095.64**	162	60	14095.64	0.00	0.00
LC1_6_2	314	59	14161.46	60206	57	15048.16	3.51	-5.89
LC1_6_3	314	52	14440.58	60084	49	15543.34	6.12	-7.09
LC1_6_4	314	49	14904.97	60081	47	13343.69	4.26	11.70
LC1_6_5	314	**60**	**14086.30**	293	60	14086.30	0.00	0.00
LC1_6_6	314	**60**	**14090.79**	2989	60	14090.79	0.00	0.00
LC1_6_7	314	**60**	**14083.76**	7547	60	14083.76	0.00	0.00
LC1_6_8	317	59	14625.38	60074	58	14880.70	1.72	-1.72
LC1_6_9	316	56	15172.14	60072	54	14652.14	3.70	3.55
LC1_6_10	319	55	14813.63	60078	52	14899.31	5.77	-0.58
LC2_6_1	304	**19**	**7977.98**	160	19	7977.98	0.00	0.00
LC2_6_2	302	19	8253.67	60157	18	9914.10	5.56	-16.75
LC2_6_3	304	18	7738.75	60112	17	8718.22	5.88	-11.23
LC2_6_4	306	18	9816.20	60131	17	7902.66	5.88	24.21
LC2_6_5	304	**19**	**8047.37**	24275	19	8047.37	0.00	0.00
LC2_6_6	305	19	8109.27	60118	18	8859.78	5.56	-8.47
LC2_6_7	304	19	8010.02	60109	19	7997.96	0.00	0.15
LC2_6_8	306	18	7738.05	60112	18	7579.93	0.00	2.09
LC2_6_9	307	19	8140.14	60110	18	8864.29	5.56	-8.17
LC2_6_10	304	18	7484.04	60124	17	7965.41	5.88	-6.04
LR1_6_1	317	59	22962.19	60117	59	22821.65	0.00	0.62
LR1_6_2	315	45	21360.01	60086	45	20144.05	0.00	6.04
LR1_6_3	317	37	19413.11	60073	37	17945.97	0.00	8.18
LR1_6_4	313	28	13776.27	60097	28	13191.79	0.00	4.43
LR1_6_5	313	42	22129.43	60107	37	25225.93	13.51	-12.28
LR1_6_6	315	34	22157.83	60106	31	22188.80	9.68	-0.14
LR1_6_7	312	28	16190.77	60108	24	18076.96	16.67	-10.43
LR1_6_8	314	21	12318.10	60118	18	12255.29	16.67	0.51
LR1_6_9	312	35	21376.73	60111	31	23239.79	12.90	-8.02

Continued on next page

Table C.4 - *Continued from previous page*

Instance	R [#]	V [#]	D [-]	t_c [ms]	V_{bk} [#]	D_{bk} [-]	ΔV [%]	ΔD [%]
LR1_6_10	314	30	18193.46	60146	26	19028.25	15.38	-4.39
LR2_6_1	304	11	22862.32	60264	11	21904.86	0.00	4.37
LR2_6_2	304	10	19810.58	60123	9	22310.56	11.11	-11.21
LR2_6_3	301	9	16086.77	60126	7	18337.46	28.57	-12.27
LR2_6_4	301	6	11604.34	60072	6	10792.55	0.00	7.52
LR2_6_5	303	9	20544.69	60188	9	19411.73	0.00	5.84
LR2_6_6	302	8	19412.71	60174	7	20702.64	14.29	-6.23
LR2_6_7	303	7	16360.85	60128	6	15526.81	16.67	5.37
LR2_6_8	303	6	11197.90	60076	4	12739.25	50.00	-12.10
LR2_6_9	302	9	18983.39	60180	8	18718.63	12.50	1.41
LR2_6_10	302	8	16495.39	60183	7	16805.06	14.29	-1.84
LRC1_6_1	313	54	17789.13	60228	52	18293.72	3.85	-2.76
LRC1_6_2	315	46	16093.73	60076	43	16576.53	6.98	-2.91
LRC1_6_3	314	36	14472.69	60094	36	13987.02	0.00	3.47
LRC1_6_4	317	25	11203.75	60119	25	10852.31	0.00	3.24
LRC1_6_5	315	49	16847.47	60246	45	17678.51	8.89	-4.70
LRC1_6_6	315	47	16987.57	60172	42	17079.65	11.90	-0.54
LRC1_6_7	315	42	15892.70	60105	37	16053.13	13.51	-1.00
LRC1_6_8	311	38	15405.24	60109	33	15812.61	15.15	-2.58
LRC1_6_9	314	38	14788.82	60231	33	15788.24	15.15	-6.33
LRC1_6_10	314	33	13867.90	60106	29	14584.41	13.79	-4.91
LRC2_6_1	303	16	15969.35	60138	16	14665.50	0.00	8.89
LRC2_6_2	303	13	15808.99	60116	13	13958.91	0.00	13.25
LRC2_6_3	303	10	15662.63	60092	10	12741.64	0.00	22.92
LRC2_6_4	302	8	11190.23	60143	7	10536.79	14.29	6.20
LRC2_6_5	303	14	13902.90	60133	13	14733.31	7.69	-5.64
LRC2_6_6	303	13	12642.68	60219	12	15200.75	8.33	-16.83
LRC2_6_7	305	11	12042.10	60108	10	15032.16	10.00	-19.89
LRC2_6_8	304	10	12619.63	60130	9	13836.07	11.11	-8.79
LRC2_6_9	302	10	12821.34	60195	9	13455.55	11.11	-4.71
LRC2_6_10	303	9	12019.17	60173	7	14141.16	28.57	-15.01
Average	**308.97**			**54708**			**7.87**	**-1.62**
Standard deviation	**5.78**			**16495**			**8.88**	**8.48**

C.5. Instances with approximately 400 requests

Table C.5: Results on Li & Lim PDPTW benchmark instances with approximately 400 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/400-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	R [#]	V [#]	D [-]	t_c [ms]	V_{bk} [#]	D_{bk} [-]	ΔV [%]	ΔD [%]
LC1_8_1	420	**80**	**25184.38**	482	80	25184.38	0.00	0.00
LC1_8_2	423	80	25489.83	60054	77	26864.13	3.90	-5.12
LC1_8_3	417	70	27135.93	60058	63	27261.46	11.11	-0.46
LC1_8_4	416	61	24488.03	60088	60	22744.15	1.67	7.67
LC1_8_5	421	**80**	**25211.22**	4803	80	25211.22	0.00	0.00
LC1_8_6	421	**80**	**25164.25**	2602	80	25164.25	0.00	0.00
LC1_8_7	420	**80**	**25158.38**	2298	80	25158.38	0.00	0.00
LC1_8_8	419	79	25871.52	60073	78	25348.45	1.28	2.06
LC1_8_9	418	75	25993.03	60076	72	26085.76	4.17	-0.36
LC1_8_10	419	75	27222.75	60128	69	29227.07	8.70	-6.86

Continued on next page

Table C.5 - *Continued from previous page*

Instance	**R** [#]	**V** [#]	**D** [-]	t_c **[ms]**	$\mathbf{V_{bk}}$ [#]	$\mathbf{D_{bk}}$ [-]	**ΔV** [%]	**ΔD** [%]
LC2_8_1	405	**24**	**11687.06**	231	24	11687.06	0.00	0.00
LC2_8_2	408	25	12581.14	60076	24	13990.57	4.17	-10.07
LC2_8_3	407	25	13547.50	60133	24	13085.26	4.17	3.53
LC2_8_4	407	24	14377.37	60148	23	13297.66	4.35	8.12
LC2_8_5	405	25	12329.80	60097	25	12298.33	0.00	0.26
LC2_8_6	406	25	12063.73	60081	24	12645.72	4.17	-4.60
LC2_8_7	408	**25**	**11854.44**	16845	25	11854.44	0.00	0.00
LC2_8_8	406	24	11953.85	60119	24	11454.33	0.00	4.36
LC2_8_9	404	**24**	**11629.52**	4577	24	11629.41	0.00	0.00
LC2_8_10	406	25	12285.38	60122	23	12459.43	8.70	-1.40
LR1_8_1	421	80	40424.84	60103	80	39291.32	0.00	2.88
LR1_8_2	417	59	36762.53	60128	59	34313.36	0.00	7.14
LR1_8_3	418	44	32445.39	60121	44	29637.24	0.00	9.48
LR1_8_4	417	27	21694.40	60110	25	20877.41	8.00	3.91
LR1_8_5	419	55	38386.62	60099	48	41284.95	14.58	-7.02
LR1_8_6	420	45	34582.39	60134	39	37276.52	15.38	-7.23
LR1_8_7	416	34	27967.35	60127	30	28180.30	13.33	-0.76
LR1_8_8	419	24	20156.02	60137	20	20037.07	20.00	0.59
LR1_8_9	417	45	36258.66	60250	40	38592.65	12.50	-6.05
LR1_8_10	420	35	29707.88	60122	31	30769.55	12.90	-3.45
LR2_8_1	405	15	36906.58	60182	14	46452.00	7.14	-20.55
LR2_8_2	404	13	31312.14	60125	11	40855.72	18.18	-23.36
LR2_8_3	404	11	26646.58	60089	9	30151.50	22.22	-11.62
LR2_8_4	405	8	21298.38	60068	6	21956.26	33.33	-3.00
LR2_8_5	403	12	38300.04	60199	11	36980.88	9.09	3.57
LR2_8_6	403	10	32704.95	60117	9	29832.94	11.11	9.63
LR2_8_7	404	8	29232.07	60088	7	27499.87	14.29	6.30
LR2_8_8	403	7	18475.34	60075	N/A	N/A	N/A	N/A
LR2_8_9	403	11	31360.33	60200	N/A	N/A	N/A	N/A
LR2_8_10	405	10	29662.88	60169	8	32588.26	25.00	-8.98
LRC1_8_1	411	69	32604.32	60076	66	32300.25	4.55	0.94
LRC1_8_2	415	59	28331.50	60100	56	28004.68	5.36	1.17
LRC1_8_3	421	50	25200.71	60112	48	24575.40	4.17	2.54
LRC1_8_4	418	34	18984.36	60151	34	18405.31	0.00	3.15
LRC1_8_5	418	65	31758.76	60094	58	31406.11	12.07	1.12
LRC1_8_6	422	60	28790.56	60089	54	29407.52	11.11	-2.10
LRC1_8_7	415	54	28409.02	60109	50	29378.53	8.00	-3.30
LRC1_8_8	422	50	26816.80	60127	45	26606.54	11.11	0.79
LRC1_8_9	418	51	25806.07	60144	44	25286.53	15.91	2.05
LRC1_8_10	420	45	24918.41	60111	40	24388.40	12.50	2.17
LRC2_8_1	406	21	21459.41	60282	20	23074.75	5.00	-7.00
LRC2_8_2	405	19	19981.17	60121	17	22686.62	11.76	-11.93
LRC2_8_3	404	16	17612.84	60073	14	20939.66	14.29	-15.89
LRC2_8_4	404	11	17146.05	60089	11	15260.02	0.00	12.36
LRC2_8_5	405	18	20277.02	60177	16	24404.69	12.50	-16.91
LRC2_8_6	402	17	19338.97	60119	15	22992.93	13.33	-15.89
LRC2_8_7	401	16	19579.82	60118	13	29162.55	23.08	-32.86
LRC2_8_8	403	14	19249.33	60119	11	25179.67	27.27	-23.55
LRC2_8_9	403	13	19922.69	60124	10	24786.05	30.00	-19.62
LRC2_8_10	404	11	19154.07	60145	9	21005.25	22.22	-8.81
Average	**411.60**			**53413**			**9.27**	**-3.15**
Standard deviation	**7.31**			**18196**			**8.45**	**9.01**

C.6. INSTANCES WITH APPROXIMATELY 500 REQUESTS

Table C.6: Results on Li & Lim PDPTW benchmark instances with approximately 500 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/1000-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	R [#]	V [#]	D [-]	t_c [ms]	V_{bk} [#]	D_{bk} [-]	ΔV [%]	ΔD [%]
LC1_10_1	527	**100**	**42488.66**	2117	100	42488.66	0.00	0.00
LC1_10_2	523	97	43026.54	62697	94	44622.39	3.19	-3.58
LC1_10_3	524	88	42767.65	60140	79	45623.05	11.39	-6.26
LC1_10_4	519	78	42062.97	62628	74	37649.21	5.41	11.72
LC1_10_5	529	**100**	**42477.41**	1713	100	42477.41	0.00	0.00
LC1_10_6	527	**101**	**42838.39**	3929	101	42838.39	0.00	0.00
LC1_10_7	526	100	42856.95	62446	100	42854.99	0.00	0.00
LC1_10_8	526	99	42711.60	62072	98	42949.56	1.02	-0.55
LC1_10_9	523	95	43478.19	62531	91	42663.13	4.40	1.91
LC1_10_10	523	94	43939.44	63804	87	45661.01	8.05	-3.77
LC2_10_1	507	**30**	**16879.24**	588	30	16879.24	0.00	0.00
LC2_10_2	508	32	19179.13	61753	30	21515.47	6.67	-10.86
LC2_10_3	508	30	20482.41	60855	30	17765.65	0.00	15.29
LC2_10_4	509	30	19967.17	60116	29	17994.30	3.45	10.96
LC2_10_5	507	32	18162.13	60733	31	17137.53	3.23	5.98
LC2_10_6	509	32	18003.10	60642	31	17194.13	3.23	4.70
LC2_10_7	509	32	18147.22	60915	31	18749.10	3.23	-3.21
LC2_10_8	507	31	17839.77	60730	30	17015.41	3.33	4.84
LC2_10_9	511	33	20175.24	60133	30	18429.66	10.00	9.47
LC2_10_10	504	31	19171.67	62281	29	17222.05	6.90	11.32
LR1_10_1	527	100	59069.52	62116	100	56744.91	0.00	4.10
LR1_10_2	520	80	53380.35	60128	80	49452.07	0.00	7.94
LR1_10_3	523	55	46283.96	60093	54	41768.24	1.85	10.81
LR1_10_4	519	32	33096.87	60112	27	31677.08	18.52	4.48
LR1_10_5	524	67	59321.33	60209	58	63134.76	15.52	-6.04
LR1_10_6	524	54	50248.67	62814	47	51246.67	14.89	-1.95
LR1_10_7	523	41	41492.09	60177	35	40439.63	17.14	2.60
LR1_10_8	523	29	30720.68	60125	24	30678.26	20.83	0.14
LR1_10_9	523	54	51696.31	62595	48	54812.46	12.50	-5.69
LR1_10_10	519	44	47312.74	61225	38	47945.15	15.79	-1.32
LR2_10_1	503	19	45420.21	60794	17	64486.92	11.76	-29.57
LR2_10_2	504	16	52497.36	60203	14	55369.13	14.29	-5.19
LR2_10_3	505	12	41904.50	60107	10	44925.60	20.00	-6.72
LR2_10_4	504	9	30159.10	60092	8	28171.94	12.50	7.05
LR2_10_5	504	15	56567.37	60523	13	58411.09	15.38	-3.16
LR2_10_6	506	13	47326.37	60107	11	47796.20	18.18	-0.98
LR2_10_7	505	11	38824.21	60091	8	44342.46	37.50	-12.44
LR2_10_8	504	8	28208.51	60118	6	27755.19	33.33	1.63
LR2_10_9	504	15	52186.18	60450	12	53508.74	25.00	-2.47
LR2_10_10	502	12	48522.24	60110	10	47288.78	20.00	2.61
LRC1_10_1	527	86	49914.71	60114	82	49175.44	4.88	1.50
LRC1_10_2	523	74	46513.67	60126	71	45583.72	4.23	2.04
LRC1_10_3	524	54	37744.53	60083	53	35831.40	1.89	5.34
LRC1_10_4	521	41	28676.91	60042	40	27487.53	2.50	4.33
LRC1_10_5	526	80	51707.65	62524	72	50628.04	11.11	2.13
LRC1_10_6	523	75	45395.58	62528	67	45154.07	11.94	0.53
LRC1_10_7	523	69	42270.42	62160	60	41890.14	15.00	0.91

Continued on next page

Table C.6 - *Continued from previous page*

Instance	R [#]	V [#]	D [-]	t_c [ms]	$\mathbf{V_{bk}}$ [#]	$\mathbf{D_{bk}}$ [-]	ΔV [%]	ΔD [%]
LRC1_10_8	522	65	42195.41	62440	55	42101.70	18.18	0.22
LRC1_10_9	524	64	41498.98	60182	53	39477.68	20.75	5.12
LRC1_10_10	522	56	37223.11	61834	47	37667.36	19.15	-1.18
LRC2_10_1	507	23	35624.01	60619	22	34767.39	4.55	2.46
LRC2_10_2	505	21	38316.70	60125	19	39496.45	10.53	-2.99
LRC2_10_3	507	18	31572.85	60055	16	28055.82	12.50	12.54
LRC2_10_4	506	13	26721.34	60110	11	23806.88	18.18	12.24
LRC2_10_5	504	19	33515.74	60195	16	41944.26	18.75	-20.09
LRC2_10_6	504	18	30802.28	60509	17	31003.36	5.88	-0.65
LRC2_10_7	505	17	34706.78	60658	15	34596.92	13.33	0.32
LRC2_10_10	504	14	30243.70	60982	11	30181.34	27.27	0.21
Average	**514.98**			**56902**			**10.67**	**0.67**
Standard deviation	**9.18**			**14955**			**8.75**	**7.49**

Table C.7: Results on Li & Lim PDPTW benchmark instances with approximately 500 requests. The column R [#] states the number of requests within the problem instance. The column V [#]states number of vehicles that are used in the found solution. The column D [-] states the distance travelled by all vehicles in the found solution. A **bold** type face indicates that the best known solution was found. The columns V_{bk} [#] and D_{bk} [-] state the same objective values for the best known solution as of 17-09-2018 gathered from `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/1000-customers/`. The final two columns state the percent deviations of both objective values of the found solutions as compared to the best known solutions.

Instance	R [#]	V [#]	D [-]	t_c [ms]	$\mathbf{V_{bk}}$ [#]	$\mathbf{D_{bk}}$ [-]	ΔV [%]	ΔD [%]
LC1_10_1	527	**100**	**42488.66**	1276	100	42488.66	0.00	0.00
LC1_10_2	523	97	42969.97	603749	94	44622.39	3.19	-3.70
LC1_10_3	524	86	42725.86	603203	79	45623.05	8.86	-6.35
LC1_10_4	519	76	40338.53	602640	74	37649.21	2.70	7.14
LC1_10_5	529	**100**	**42477.41**	1372	100	42477.41	0.00	0.00
LC1_10_6	527	**101**	**42838.39**	2780	101	42838.39	0.00	0.00
LC1_10_7	526	**100**	**42854.99**	250119	100	42854.99	0.00	0.00
LC1_10_8	526	99	42711.60	602357	98	42949.56	1.02	-0.55
LC1_10_9	523	94	42694.07	603091	91	42663.13	3.30	0.07
LC1_10_10	523	93	43654.73	602768	87	45661.01	6.90	-4.39
LC2_10_1	507	**30**	**16879.24**	528	30	16879.24	0.00	0.00
LC2_10_2	508	32	17598.61	601781	30	21515.47	6.67	-18.20
LC2_10_3	508	31	21706.83	604484	30	17765.65	3.33	22.18
LC2_10_4	509	30	20297.52	604852	29	17994.30	3.45	12.80
LC2_10_5	507	**31**	**17137.53**	306696	31	17137.53	0.00	0.00
LC2_10_6	509	**31**	**17194.13**	336152	31	17194.13	0.00	0.00
LC2_10_7	509	33	18989.83	601394	31	18749.10	6.45	1.28
LC2_10_8	507	31	17811.69	601833	30	17015.41	3.33	4.68
LC2_10_9	511	31	17807.98	602129	30	18429.66	3.33	-3.37
LC2_10_10	504	30	17548.98	600800	29	17222.05	3.45	1.90
LR1_10_1	527	100	57147.84	601026	100	56744.91	0.00	0.71
LR1_10_2	520	80	52285.19	604978	80	49452.07	0.00	5.73
LR1_10_3	523	54	46980.35	604011	54	41768.24	0.00	12.48
LR1_10_4	519	31	32924.57	604318	27	31677.08	14.81	3.94
LR1_10_5	524	63	58936.15	602673	58	63134.76	8.62	-6.65
LR1_10_6	524	53	49483.24	602050	47	51246.67	12.77	-3.44
LR1_10_7	523	38	41139.19	602825	35	40439.63	8.57	1.73
LR1_10_8	523	28	30896.85	606715	24	30678.26	16.67	0.71
LR1_10_9	523	52	52144.79	602119	48	54812.46	8.33	-4.87
LR1_10_10	519	43	46186.81	602077	38	47945.15	13.16	-3.67
LR2_10_1	503	18	58663.50	600995	17	64486.92	5.88	-9.03
LR2_10_2	504	15	54591.39	601786	14	55369.13	7.14	-1.40

Continued on next page

Table C.7 - *Continued from previous page*

Instance	**R [#]**	**V [#]**	**D [-]**	t_c **[ms]**	$\mathbf{V_{bk}}$ **[#]**	$\mathbf{D_{bk}}$ **[-]**	**ΔV [%]**	**ΔD [%]**
LR2_10_3	505	12	39786.25	601902	10	44925.60	20.00	-11.44
LR2_10_4	504	9	30150.57	600263	8	28171.94	12.50	7.02
LR2_10_5	504	15	54208.77	601102	13	58411.09	15.38	-7.19
LR2_10_6	506	12	51260.14	601950	11	47796.20	9.09	7.25
LR2_10_7	505	10	38672.96	600380	8	44342.46	25.00	-12.79
LR2_10_8	504	7	29440.62	600168	6	27755.19	16.67	6.07
LR2_10_9	504	14	53526.07	601174	12	53508.74	16.67	0.03
LR2_10_10	502	12	44609.52	601174	10	47288.78	20.00	-5.67
LRC1_10_1	527	87	49574.08	601194	82	49175.44	6.10	0.81
LRC1_10_2	523	73	46127.36	605962	71	45583.72	2.82	1.19
LRC1_10_3	524	54	37569.48	607187	53	35831.40	1.89	4.85
LRC1_10_4	521	40	28719.89	600197	40	27487.53	0.00	4.48
LRC1_10_5	526	80	51771.93	602997	72	50628.04	11.11	2.26
LRC1_10_6	523	75	46243.46	601240	67	45154.07	11.94	2.41
LRC1_10_7	523	70	42212.48	602342	60	41890.14	16.67	0.77
LRC1_10_8	522	64	41387.93	603014	55	42101.70	16.36	-1.70
LRC1_10_9	524	61	40110.95	602290	53	39477.68	15.09	1.60
LRC1_10_10	522	55	37199.98	601573	47	37667.36	17.02	-1.24
LRC2_10_1	507	22	35962.26	600992	22	34767.39	0.00	3.44
LRC2_10_2	505	21	35800.54	603447	19	39496.45	10.53	-9.36
LRC2_10_3	507	16	32880.54	605519	16	28055.82	0.00	17.20
LRC2_10_4	506	12	26671.48	600133	11	23806.88	9.09	12.03
LRC2_10_5	504	18	39118.11	600999	16	41944.26	12.50	-6.74
LRC2_10_6	504	17	34007.31	600789	17	31003.36	0.00	9.69
LRC2_10_7	505	17	32491.90	601034	15	34596.92	13.33	-6.08
LRC2_10_10	504	13	28393.91	602066	11	30181.34	18.18	-5.92
Average	**514.98**			**545184**			**7.76**	**0.39**
Standard deviation	**9.18**			**162792**			**6.74**	**7.02**

D

Remaining results on Comparing Strategies

D.1. Instance A

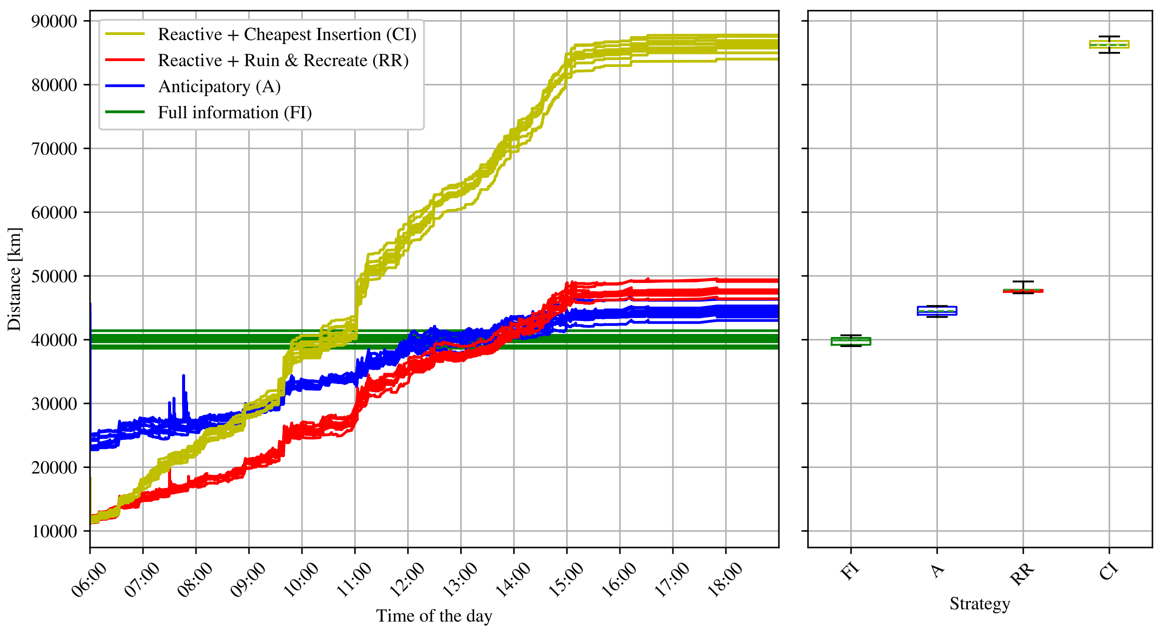


Figure D.1: Performance of different optimisation strategies on instance A. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.2. Instance B

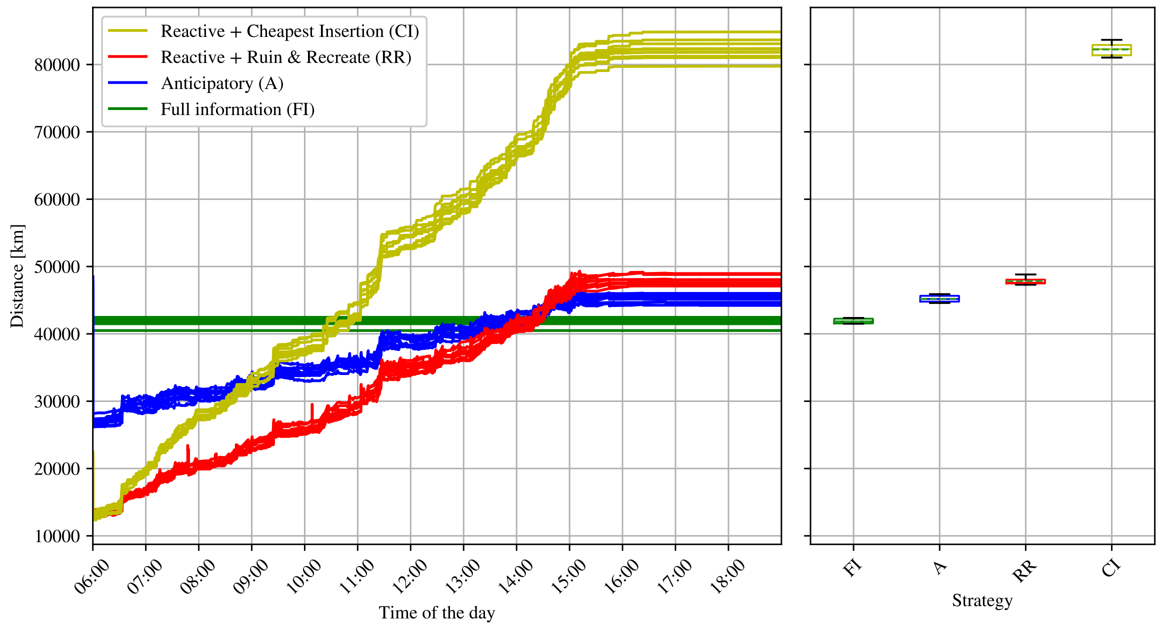


Figure D.2: Performance of different optimisation strategies on instance B. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.3. Instance C

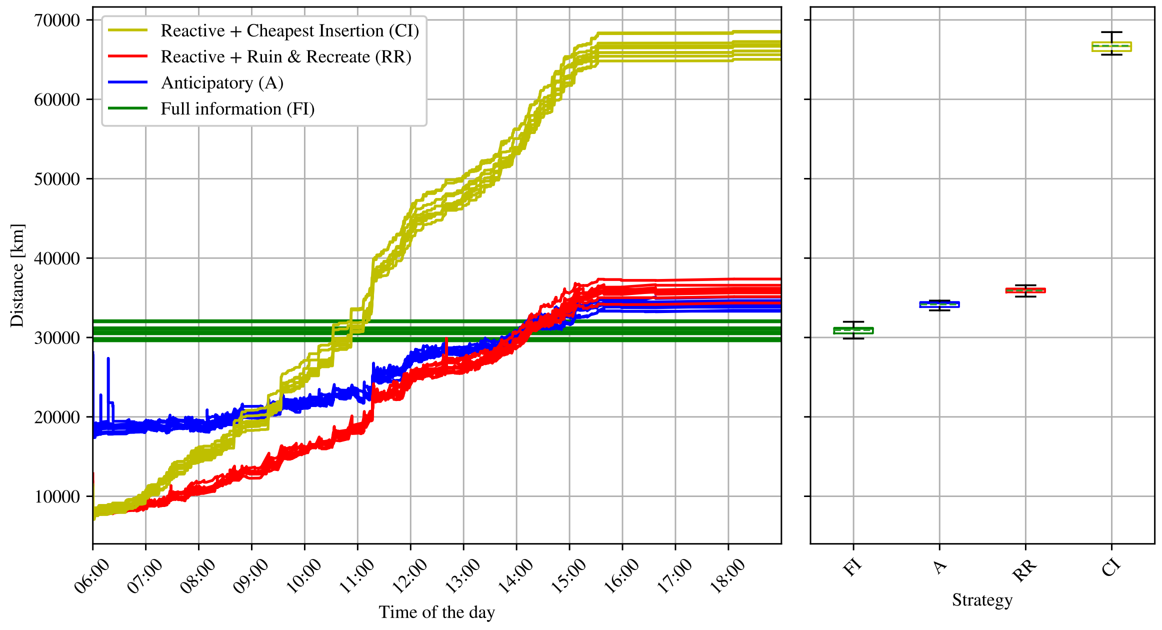


Figure D.3: Performance of different optimisation strategies on instance C. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.4. Instance D

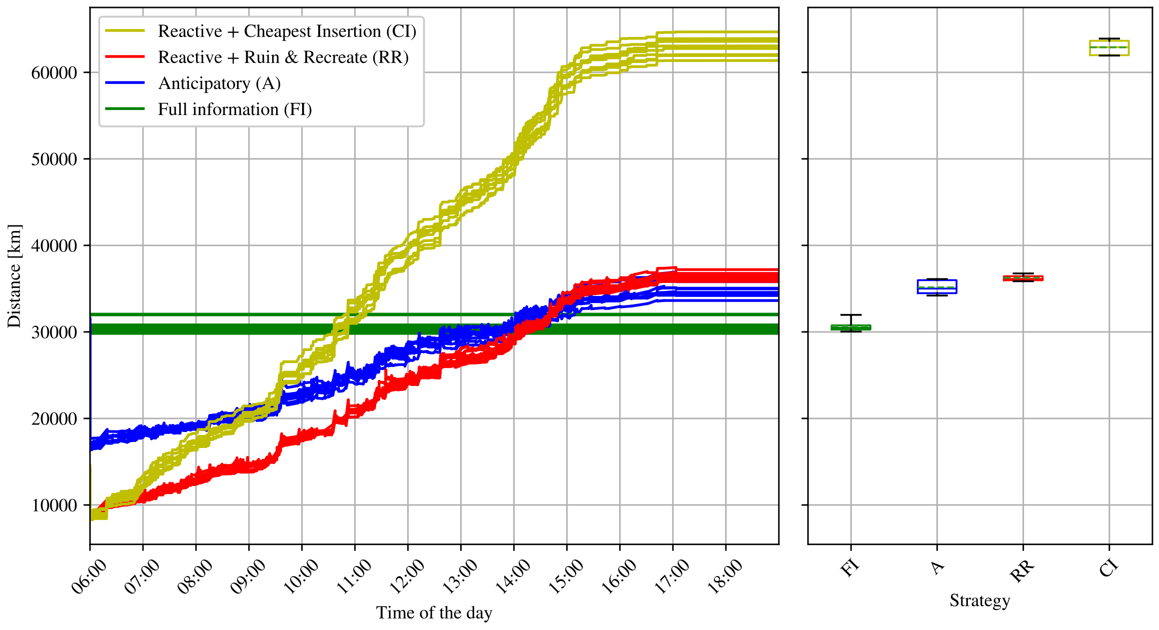


Figure D.4: Performance of different optimisation strategies on instance D. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.5. Instance E

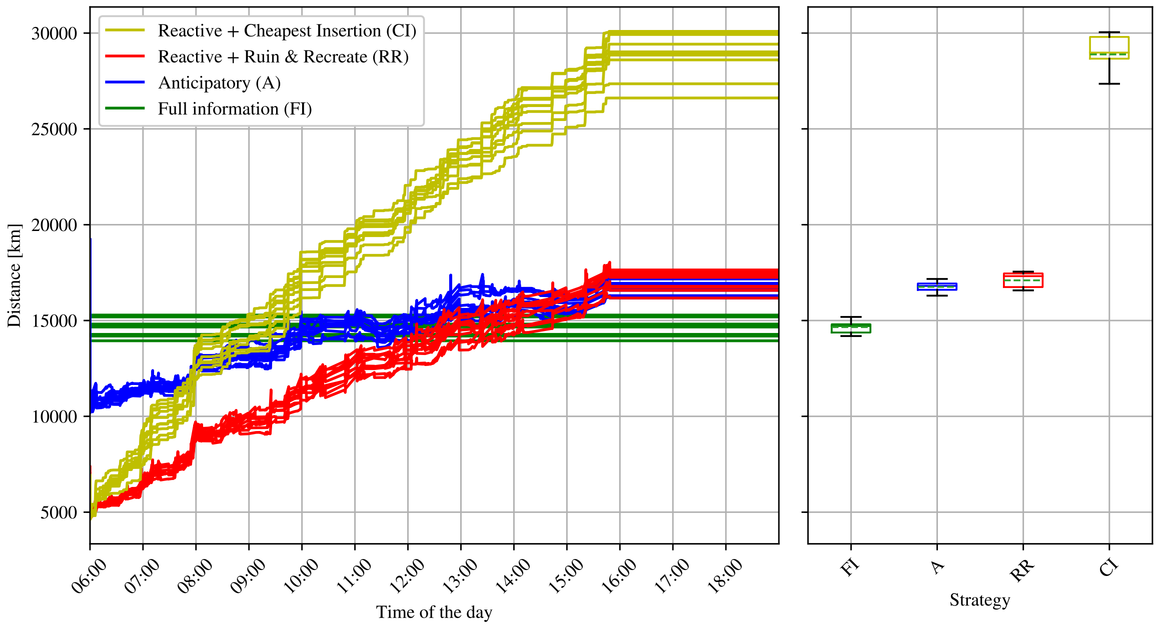


Figure D.5: Performance of different optimisation strategies on instance E. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.6. Instance G

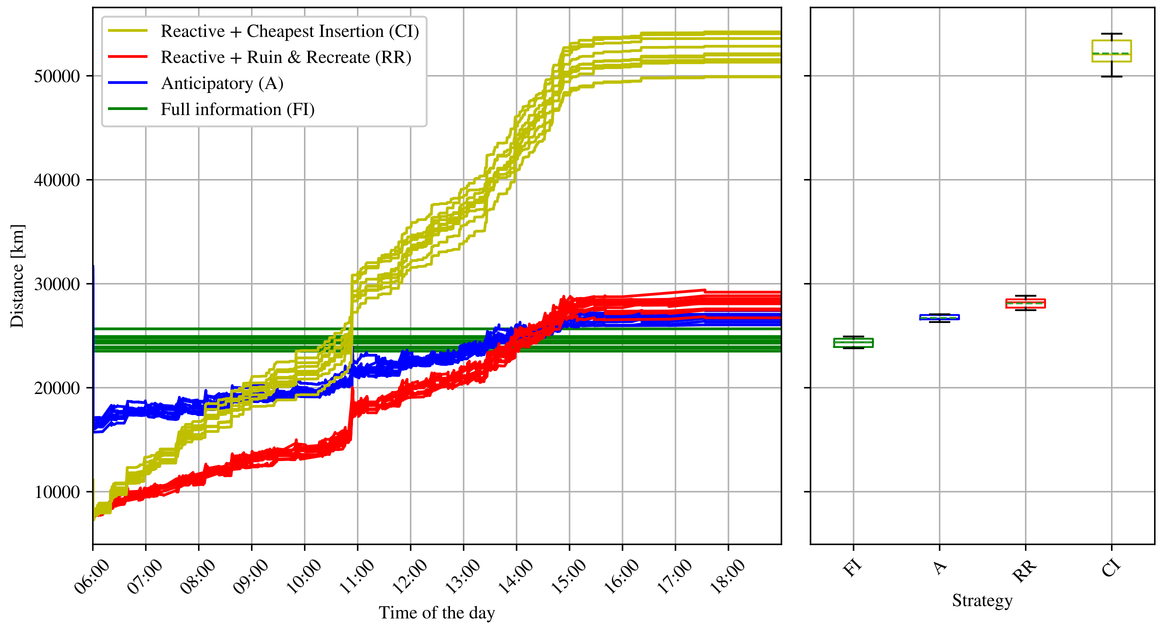


Figure D.6: Performance of different optimisation strategies on instance G. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.7. Instance H

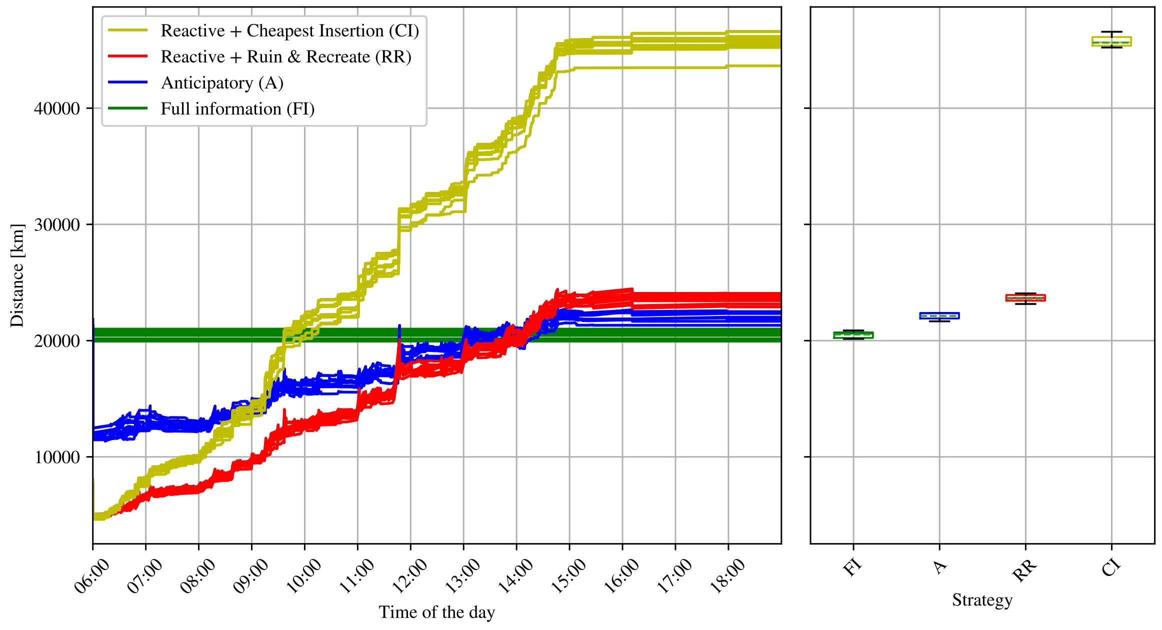


Figure D.7: Performance of different optimisation strategies on instance H. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.8. INSTANCE J

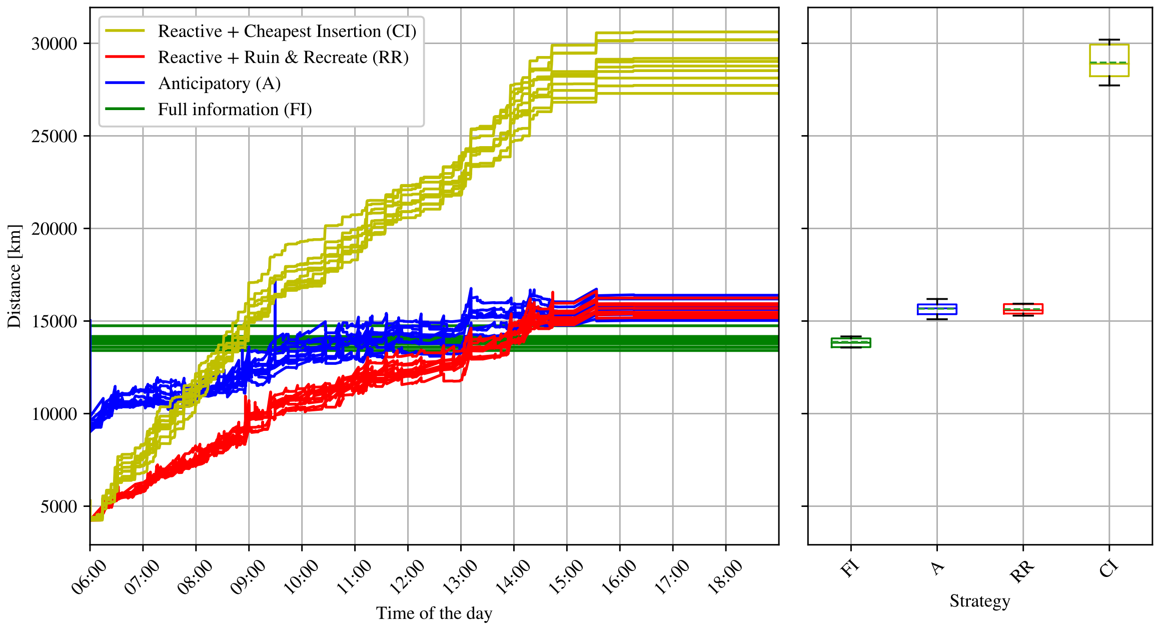


Figure D.8: Performance of different optimisation strategies on instance J. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.9. INSTANCE B - REAL-TIME

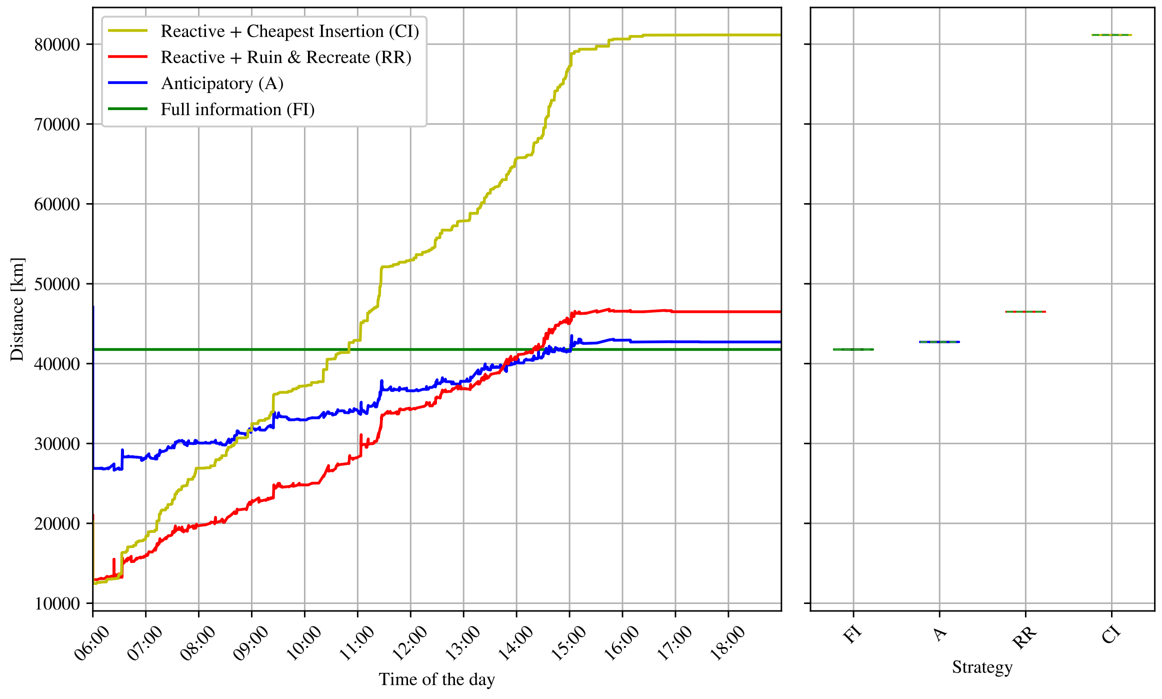


Figure D.9: Performance of different optimisation strategies on instance B when using a simulation speed equal to 1. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.

D.10. INSTANCE H - REAL-TIME

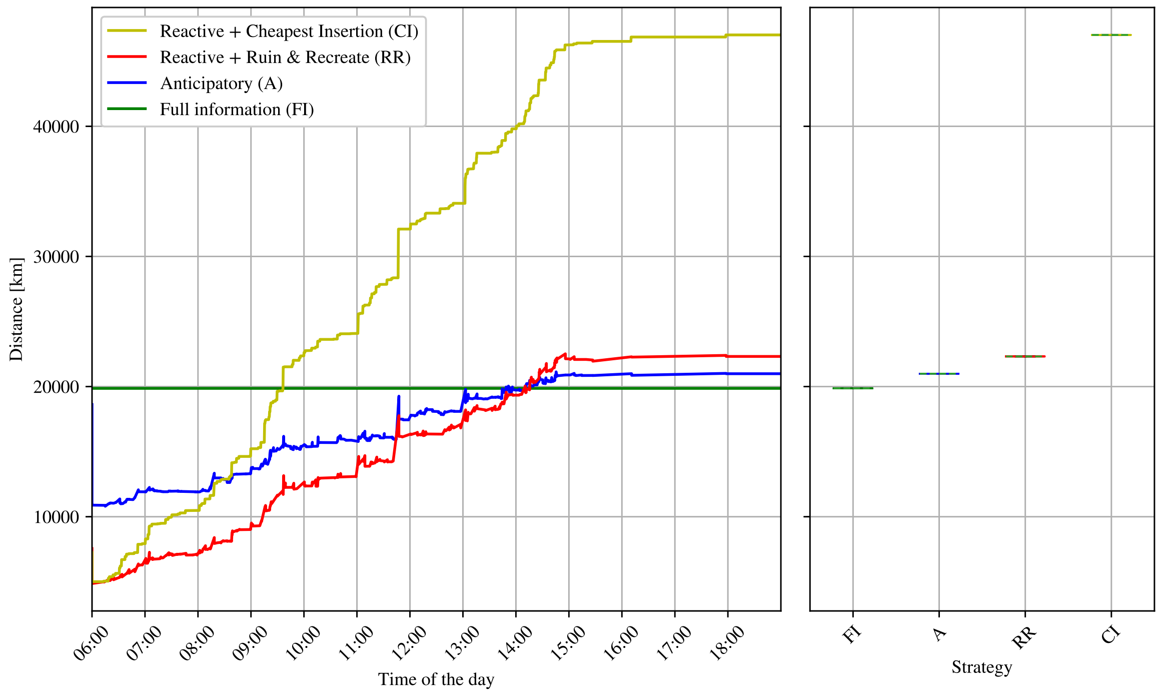


Figure D.10: Performance of different optimisation strategies on instance H when using a simulation speed equal to 1. On the left it is shown how the distance that is planned to be travelled during the entire day evolves during the day for each strategy. Each separate line represents a single simulation. On the right the distributions of the total distance travelled at the end of the day for each strategy are shown.