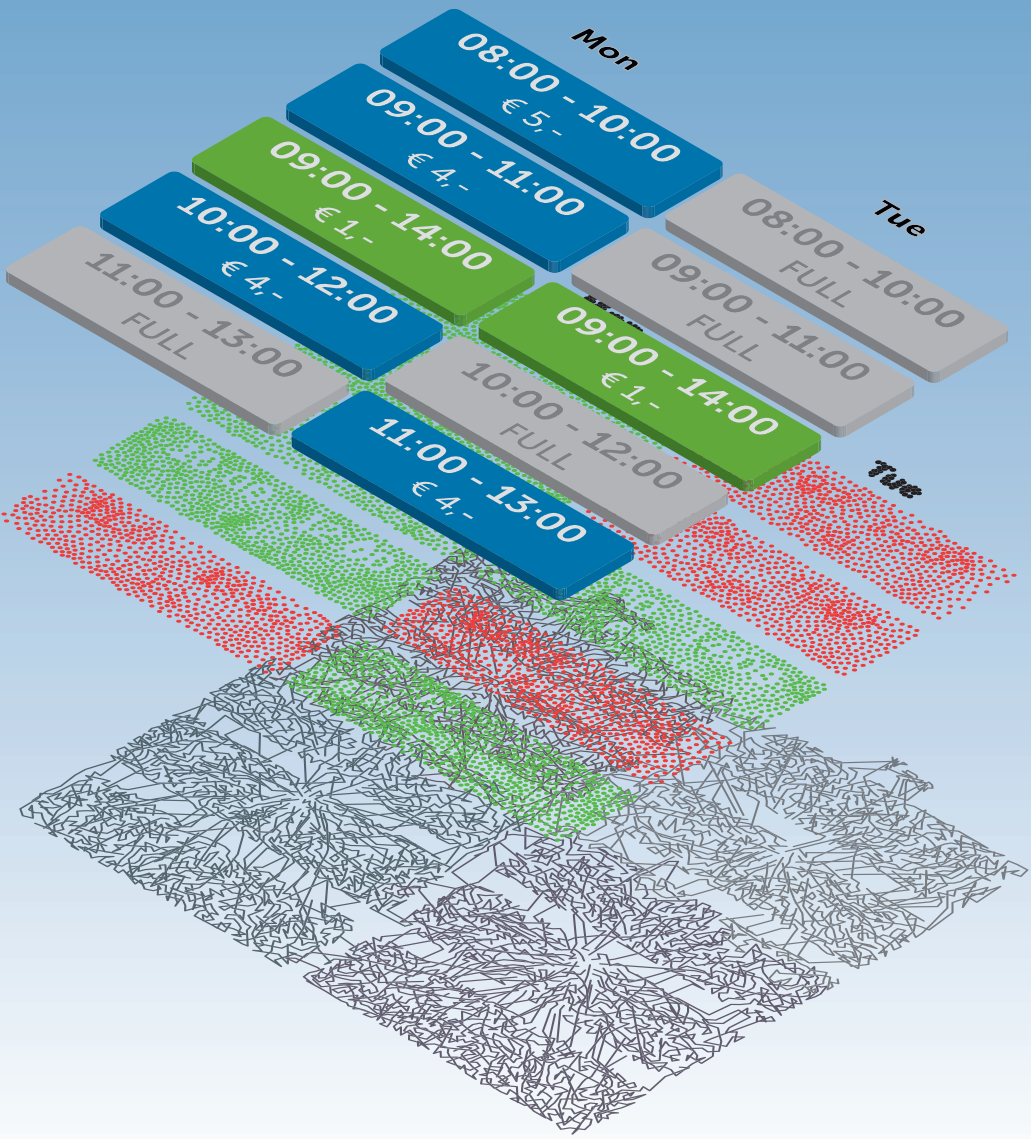
THOMAS VISSER

Vehicle Routing and Time Slot Management in Online Retailing



Vehicle Routing and Time Slot Management in Online Retailing

Vehicle Routing and Time Slot Management in Online Retailing

Voertuigroutering en bezorgbloksturing voor online bezorgservice

Thesis

to obtain the degree of Doctor from the
Erasmus University Rotterdam
by command of the
Rector Magnificus

Prof.dr. R.C.M.E. Engels

and in accordance with the decision of the Doctorate Board.

The public defence shall be held on

Thursday 14 November 2019 at 15:30 hours

by

Thomas Roemer Visser
born in Utrecht, The Netherlands.

Erasmus University Rotterdam

Doctoral committee

Promotor:	Prof.dr. A.P.M. Wagelmans
Other members:	Dr.ir. N.A.H. Agatz Prof.dr. S. Irnich Prof.dr. M.W.P. Savelsbergh
Copromotor:	Dr. R. Spliet

Erasmus Research Institute of Management – ERIM
The joint research institute of the Rotterdam School of Management (RSM)
and the Erasmus School of Economics (ESE) at the Erasmus University Rotterdam
Internet: www.erim.eur.nl

ERIM Electronic Series Portal: repub.eur.nl

ERIM PhD Series in Research in Management, 482
ERIM reference number: EPS-2019-482-LIS
ISBN 978-90-5892-580-0

Cover image: ©Thomas R. Visser
Cover design: PanArt, www.panart.nl

This publication (cover and interior) is printed by Tuijtel on recycled paper, BalanceSilk®.
The ink used is produced from renewable resources and alcohol free fountain solution.
Certifications for the paper and the printing production process: Recycle, EU Ecolabel, FSC®, ISO14001.
More info: www.tuijtel.com


Acknowledgements

This dissertation would not have been possible without the help and support of a number of people. First of all, I would like to thank my supervisors dr. Remy Spliet and prof.dr. Albert Wagelmans. Remy, thank you for your motivation, patience and help during the PhD project. I will miss our many fun and fruitful meetings, which typically resulted in new mathematical insights or in improving my scientific writing. Albert, thank you for always being available for advice, and providing your experience in publication strategy. I would also like to thank dr.ir. Niels Agatz, who I also considered being a supervisor within our Last-mile project. During my PhD, I was very happy to have continued the work you started in your PhD in a renewed collaboration with ORTEC and AH Online. Also, I will always keep fond memories of the – surprisingly many – meetings/discussions we had together with Remy inside a moving vehicle, particularly in trains. I would like to thank also Joydeep Paul, who was my fellow PhD colleague in the Last-mile delivery project, for all the fun and useful discussions we had over the years.

I like to thank prof.dr. Martin Savelsbergh and prof.dr. Stephan Irnich for serving in my inner committee. Moreover, thank you Martin for the amazing time I had during my research visits to Georgia Tech in Atlanta. Stephan, thank you for inviting me to come and visit you and your group in Mainz to present some of my work. I am grateful to prof.dr. Goos Kant, prof.dr. Jan Fabian Ehmke and prof.dr. Moritz Fleischmann for serving in my outer committee and joining the opposition during my defense. Goos, thank you also for your involvement and all efforts for the project from the ORTEC side. Thank you both Jan and Moritz for the nice (quite recently started) annual meetings of the last-mile community we had together.

I am very grateful to have Kevin Dalmeijer and Mathijs van Zon as my paranymphs. Kevin, thank you for being my brother 'vehicle router extraordinaire', as we: both did a PhD in vehicle routing with time windows, both were supervised by Remy and Albert, started and ended roughly at the same time, in the last two years shared an

office together, but mainly because of our shared interest in each other's research. I will deeply miss our (prime!) times in the office, which were awesome. The many things that we experienced together would just be too much – in printing costs – to write down. Furthermore, your optimism and positivism are unrivaled, which has really helped me over the years. I hope we can continue to help each other, and finally start on that list of new problems that need our attention. Mathijs, thank you for being my other brother in vehicle routing: also under Remy and Albert, but you started some years later. I am grateful for our extensive discussions on vehicle routing, especially concerning our shared interest in vehicle routing heuristics and how to make them fast. I wish you the best of luck the coming years finishing your research. As you know, being the most senior 'vehicle routing' PhD in the office now brings a lot of responsibilities, but I know you will perform this duty admirably.

I would like to thank Gertjan van den Burg, my first roommate, who not only showed me the tools Git and Gnuplot, the latter which I used to make movies of simulations which found their way in many presentations, but also showed me Factorio. I have gracefully kept this tradition by passing on the – only slightly – addictive game Factorio to my new roommate Kevin, who I believe has passed it on again. I would like to thank Rutger Kerkkamp, who shared this LaTeX–template. I will miss our afternoon talks in the office, which were mostly related to programming but could cover any topic. Moreover, thank you for your continuous interest and frequent visits to my classical concerts. Furthermore, I am grateful to Thomas Breugem for always being available for drinks, especially abroad. I also like to thank my other colleagues at the Erasmus University for the open atmosphere and useful discussions, and also the many game nights, nice dinners and the very fun conference visits we had together. In particular, thank you Judith, Harwin, Evelot, Charlie, Sha, Amy, Nemanja, Rowan, Naut, Rolf, Ymro, Utku, Jeroen, Rommert, Wilco, Dennis, Twan, Marieke, Diego, Alp, Gert-Jaap and Paul.

My PhD project was part of a larger collaboration with planning software firm ORTEC and the Dutch online grocery retailer AH Online. In the many times I visited ORTEC, before and during my PhD, I always felt very welcome and part of the team. I would like to thank all ORTEC-ers for creating such a positive work environment. Moreover, I like to thank Joaquim Dos Santos Gromicho and Leendert Kok for motivating me to pursue a PhD. Joaquim, thank you for recommending me this particular PhD project, and Leendert, thank you also for your continuous support during the project. Also, I am grateful to Laurien Verheijen for helping me with cloud developments during my time at ORTEC. During the project, we had a number of

different ORTEC contacts. I would like to thank Ineke Meuffels, Caroline Jagtenberg and Pim van 't Hof for their efforts for the project.

For any OR scientist, it is really inspiring to actually get into practice and experience the current challenges in logistics firsthand. Already from the start of my PhD, I had the opportunity to spend time at the main e-fulfillment center of AH Online. I would like to thank Richard Leveling and Coen Schlüter for giving me such a warm welcome inside the company, and for their support over the years. Richard, I am especially grateful for your enthusiasm and motivation during the project. I have learned a lot from that one early morning shift of delivering groceries around the small city of Leerdam, from that other shift of picking groceries in the fulfillment center, but mostly from spending many weeks with the delivery planning specialists. Richard Knoester and Maurice Poot, thank you for sharing some of your great expertise in managing the time slots and the vehicle planning. In the end, I am very proud to have contributed in some way to the creation of new tools to support and assist these specialists.

I would like to thank the people at Surfsara, in particular the Lisa Cluster, which was used extensively for the many computational experiments in this dissertation. Also, I would like to thank the developers involved in making and maintaining the following open-source tools: LaTeX, C++, Python, QGis, Gnuplot, OpenStreetMap, OSRM, StippleGen2, Git, Fork, and the following closed-source tools: Gurobi, GitHub, CLion and PyCharm. These tools were all invaluable to me during my dissertation project, either by making a particular task possible, or achieving a particular task much faster. Needless to say, I can highly recommend every one of these tools.

Making music together, either as (jazz) pianist, harpsichord player or by singing in various choirs, has always been my great passion. I am grateful to everyone with whom I have made music over the years. In particular, I would like to thank Gilles Michels, Michiel Meijer and Paulien Kostense, the artistic team of Collegium Musicum Traiectum, for their passion and drive to make beautiful Baroque music together.

Furthermore, I would like to thank my friends from university and high school, and in particular from the USKO, for the many nice moments we had together during these years, which really helped me to escape the 'PhD bubble' for a moment. I would like to thank Dennis Broeders and Bram Wage for our long friendships, before and during the PhD. Dennis, thank you for the many dinners we had over the years, which we usually spend with deep and enjoyable discussions on music theory. Bram, thank

you for always enlightening me with your one-of-a-kind humor, especially concerning the multiplication of matrices and vectors.

I like to thank my family for their support over the years. I am grateful to my parents, Maarten and Elmira, for all their help and continuous believe in me. Maarten, thank you also for our many useful discussions and the proofreading of my dissertation. Elmira, thank you for your care and for making sure the door was always open for me to come and visit, even on short notice. I am also grateful to Quirine, my sister, and Joshua for their great interest and support over the years. I am happy that you found a nice place to settle in Rotterdam, although I will miss being able to bike there from the office.

Finally, thank you Francine. Thank you for always being there for me, for always listening to my stories you might find 'completely trivial' (probably true), for your proofreading and suggestions for my dissertation, and all the countless other things you have done for me the past years. I could not have completed this dissertation without your endless help, love and support. For that I am eternally grateful.

Thomas R. Visser
September 2019

Contents

Chapter 1

Introduction

Many online retailers and logistics providers face the difficulties of efficiently managing the *last mile*, the last leg of the e-commerce supply chain in which purchased goods are transported to the customer's location. This is especially true for online retailers offering *attended home delivery*, for which the customer has to remain at home to receive their online purchased goods. Online grocery retailing is a primary example, but other examples include the delivery of large electronics and furniture. The immense growth of online purchases in recent years and the relatively high transportation costs and environmental impact of the last-mile home delivery services has motivated retailers and logistics providers to investigate various alternatives to attended home delivery. Still, customers heavily prefer attended home delivery for its high level of service.

In online retailing, *delivery failures*, for instance when the customer is not at home, are costly because products have to be returned and stored, and deliveries have to be re-scheduled. This is especially true in online grocery retailing, as most grocery products are perishable and might have to be replaced. To decrease the chances of delivery failures, many retailers (for instance online grocers such as Albert Heijn Online, Picnic (NL), Ocado, Waitrose (UK) and Peapod (USA)) offer customers to choose from a set of narrow delivery time windows, called *time slots*. The customers' choice of the time slot affects the efficiency of the delivery routes in terms of the number of customers that can be served and the transportation cost. It is crucial for retailers to manage the availability of their time slots. However, this can be challenging, as the customer demand can vary heavily, and the amount of available resources, for instance delivery vehicles and drivers, may be limited. Too many placed customer requests can cause major operational issues for the retailer. The

management of time slots during the ordering process is generally called *Dynamic Time Slot Management* (DTSM). In this dissertation, we focus on methods which use *vehicle routing* to help retailers manage the availability of time slots in real time.

1.1 Vehicle Routing

The vehicle routing problem (VRP) is a classic combinatorial optimization problem proposed originally by Dantzig and Ramser (1959). It consists of finding a set of routes serving all customers using a fleet of vehicles with limited capacity, while minimizing total travel costs. The problem belongs to the so-called class of *NP-hard* problems, which are notoriously difficult to solve, due to the exponential number of operations which might be required to find an optimal solution (unless P=NP). Because of its practical relevance, the problem has been studied intensively in the scientific literature, and many extensions have been proposed to model various real-world applications (Irnich et al., 2014). In this dissertation, we consider a number of extensions of the VRP which are used to model problem characteristics typically arising in online retailing. For instance, we use the vehicle routing problem with time windows (VRPTW), in which customers must be served within their time window, to incorporate the selected time slots of the customers (Desaulniers et al., 2014). Note that throughout this dissertation, we use *time window* to denote an interval of time which is known or set by the retailer, while we specifically use *time slot* in case such an interval is offered by the retailer to the customer.

While exact methods, many of which are based on *Branch-and-price*, have been proposed to solve the vehicle routing problem with many extensions to optimality, such methods generally require a lot of computation time (Desaulniers et al., 2014). In practice, when delivery schedules have to be made on a daily basis, the time available for computations is very limited (e.g., one hour). Moreover, when vehicle routing is used during the ordering process, the time available for computations might be even more limited. Therefore, in practice heuristic methods are mostly applied to find good delivery schedules. Examples of such heuristics include cheapest insertion and neighborhood search, which are both studied in this dissertation.

1.2 Time Slot Management in Online Retailing

The process from online purchase to home delivery typically employed by online retailers can be described as follows. During the *ordering process*, customers first

identify themselves on the retailer's website, fill their order basket and afterwards finalize their order. At that moment, the retailer constructs a time slot offer to be shown on the time slot webpage. The customer can select a time slot among the offered ones. Upon selection, the retailer updates the set of placed orders. The ordering process continues until the *cut-off time*, typically 12 to 18 hours before the execution of the corresponding delivery shift. At this time, a delivery schedule is constructed containing all placed customer requests for the delivery shift. Based on this delivery schedule, the production of the orders can start in the fulfilment center. Afterwards, the produced orders might first be transported to hub locations from which the delivery vehicles depart. Finally, the routes of the delivery schedule are executed and, if no operational issues arise, customers will receive their purchased goods within their chosen time slot.

The menu from which customers can pick their time slot can vary considerably between online retailers. To illustrate this, we briefly consider the time slot webpages of two competing Dutch online grocers: AH Online (`http://ah.nl`) and Picnic (`https://picnic.app/nl/`). At the time of writing, AH Online offers time slots for up to four weeks in advance. A single day can be selected, which then shows the time slot availability on that day. AH Online offers up to 15 different time slots per day, 7 in the morning and 8 in the afternoon/evening. These time slots are overlapping and vary in length, between 1 and 6 hours, and in delivery fee, €2.50 up to €11.95. In contrast, Picnic offers at most a single one-hour time slot each day without delivery fees, which can be selected up to one week in advance. Although the time slots differ from day to day, we note that there is a recurring weekly pattern. The same 1-hour time slot is offered each Monday, while a different one is offered each Tuesday. The differences in time slot menus used by the two companies probably coincide with their differences in service areas and type of delivery vehicles used. AH Online offers services in a large part of the Netherlands, including large cities, towns and rural areas, and uses large delivery vehicles, while Picnic currently only offers services within the large cities, and uses small electric vehicles. In this dissertation, we consider the dynamic management of time slots inspired by both these business cases.

In the scientific literature, two types of time slot management schemes are generally distinguished (Agatz et al., 2013). *Static Time Slot Management* concerns the tactical planning of available time slots for areas of customer locations (see for instance Agatz et al. (2011) and Hernandez et al. (2017)). For example, time slots offered to rural and other low demand customer areas can be limited, to aggregate their demand over only a few delivery moments. All decisions are made *before* customers

can actually order, and are fixed during the ordering process.

In contrast, *Dynamic Time Slot Management* (DTSM) concerns the dynamic construction of time slots offers *during* the ordering process, based on already placed customer orders. While some proposed methods use static time slot order limits for each area (see for instance Bruck et al. (2018)), most proposed methods exploit vehicle routing to asses the available vehicle capacity (see for instance Campbell and Savelsbergh (2005, 2006), Ehmke and Campbell (2014), Cleophas and Ehmke (2014), Yang et al. (2016) and Köhler et al. (2019)). These methods can help retailers avoid major operational issues, as the number of required delivery vehicles is limited (and can be monitored) in real time.

Although most of the literature on DTSM focuses on the benefits of using vehicle routing heuristics, little research has been done investigating their decision time, i.e., the computation time of the vehicle routing procedures, and their effects on the performance. In particular, the complexity of the underlying routing problem and the number of placed orders impact the decision times of the vehicle routing procedures. As time slot offers are constructed in real time, these decision times may result in unwanted waiting time for the customers on the webpage. Moreover, as customers typically do not select their time slot instantaneously, critical challenges arise as the interactions between customers and the DTSM system become simultaneous, which have not yet been studied in the literature. And finally, it might be helpful to move some of the time-consuming vehicle routing work to a strategic phase. However, within the classes of currently proposed time slot management methods, we are unaware of a method for the strategic phase which retains the useful properties of monitoring and limiting the required number of vehicles during the ordering process.

In this dissertation, we try to fill these gaps in the literature. We investigate the complexity of vehicle routing problems encountered in online retailing, and study speed-up techniques. We model and investigate practice-inspired extensions to the DTSM model, and build a solution framework for these models. Furthermore, we propose a new variant of DTSM. Extensive computational studies highlight the characteristics of the models and the performance of the solution methods.

1.3 Outline

In this dissertation, we investigate the use of vehicle routing in Dynamic Time Slot Management (DTSM) to help retailers manage the availability of time slots in real time.

In Chapters 2 and 3, our aim is to study the real-time performance of DTSM procedures for a practice-inspired online retailing setting. Since DTSM makes use of vehicle routing heuristics in real time, the complexity and computation times of such methods are crucial. To this end in Chapter 2, we study speed-up techniques for commonly used vehicle routing heuristics such as cheapest insertion and neighborhood search. In practice, time-dependent travel times are typically used to model road congestion by using historic speed profiles. Also, route duration constraints are typically used to model labor agreements concerning the working hours of the drivers, and route duration can also appear in the objective as cost. The combination of time-dependent travel times and route duration increases the complexity of vehicle routing heuristics, and therefore impacts the computation times. We study the use of piecewise linear functions, and observe these can be evaluated in different orders. This leads us to propose a novel tree-based data structure, which improves the complexity of computations and memory use. We study the computation time and memory usage of the pre-calculation methods on benchmark instances with 1000 customers.

In Chapter 3, we investigate the real-time performance of Dynamic Time Slot Management. In practice, customers arrive over time on the retailer's website to select a time slot. Furthermore, time slot offers take (computation) time to construct, and customers typically do not choose their time slot instantaneously. As more and more customers arrive in a short time span, the interactions between customers and the system become simultaneous, which raises critical challenges which have not been studied in the current literature. In particular, a time slot offer can become invalidated by other customers that place an order in the meantime, and the waiting time experienced by customers might become (too) long. We model simultaneous interactions in DTSM by incorporating the time it takes a customer to select a time slot, and the decision time of the system needed to construct such an offer. We provide a DTSM framework of procedures that is suitable for the new model, and try to adapt the best performing DTSM methodology in current literature. We also consider additional procedures that can be run in the background. While we allow time slot offers to become invalidated, our framework still guarantees the existence of a feasible schedule of placed requests. We use the algorithms and speed-up techniques provided in Chapter 2, and generate problem instances inspired by current practice which include multiple depots, time-dependent travel times and route duration constraints. We simulate a real-time ordering process with up to 8000 customers arriving in a time span of as much as 80000 seconds or as little as 8 milliseconds.

To avoid time-consuming routing procedures in DTSM, it might be beneficial to move some of the work to a strategic phase, i.e., before the ordering process. In Chapter 4, we propose *Strategic Time Slot Management*, a novel variant of DTSM, which utilizes a priori routes and time slot assignment. It is inspired by current practice in online grocery retailing, where many customers tend to order quite regularly, for instance every week or every two weeks, and have their favorite delivery day and time slots. This information can be exploited in a strategic phase. In Strategic Time Slot Management, each customer location is assigned a single time slot for each day of the week and a priori delivery routes are used to guide time slot availability. This simplifies the management of time slots during the ordering process considerably, while still guaranteeing the existence of a feasible schedule of placed customer requests. It also allows smoothing of the fulfillment center operations and provides delivery consistency. We model the design problem and investigate the single vehicle (or single-route) case. We propose a 2-stage stochatic programming formulation for the design problem and develop a sample average approximation solution approach. The expected revenue of an a priori route is evaluated using an efficient dynamic program. An extensive computational study on random instances up to 12 customers provides insights in the benefits of Strategic Time Slot Management.

Finally, in Chapter 5 we summarize our main findings. Each chapter can be read individually and, consequently, some concepts and notation will be (re)introduced. However, note that in Chapter 3, we make use of the algorithms presented in Chapter 2.

1.4 Summary of main contributions

The main contribution of each individual chapter can be summarized as follows:

- Chapter 2 proposes a novel tree-based data structure to lower the computational and memory complexities of neighborhood search move evaluations for VRPs with time-dependent travel times and route duration constraints (Visser and Spliet, 2017). This chapter is based on joint work with Remy Spliet. At the time of writing, this work has been accepted for publication in *Transportation Science*.

- Chapter 3 models the simultaneous interactions arising in real-time Dynamic Time Slot Management, provides a solution framework, and studies the performance using real-time simulations. This chapter is based on joint work with

Niels Agatz and Remy Spliet. At the time of writing, this work is in preparation for journal submission.

- Chapter 4 presents a novel variant of Dynamic Time Slot Management which uses a priori routing and time slot assignment, which is called Strategic Time Slot Management (Visser and Savelsbergh, 2019). This chapter is based on joint work with Martin Savelsbergh. At the time of writing, this work is under review at *Transportation Science*.

Chapter 2

Efficient Move Evaluations for Time-Dependent Vehicle Routing Problems

Thomas R. Visser and Remy Spliet

2.1 Introduction

The classic Vehicle Routing Problem with Time Windows (VRPTW) consists of finding a set of routes satisfying all customer requests within their time windows using a homogeneous fleet of vehicles with limited capacity while minimizing total travel costs. Recently, much attention has been given to routing problems in which travel times are assumed to be time-dependent, see for instance Balseiro et al. (2011); Dabia et al. (2013); Donati et al. (2008); Figliozzi (2012); Hashimoto et al. (2008); Ichoua et al. (2003); Kok et al. (2011, 2010); Spliet et al. (2018) and see Gendreau et al. (2015) for a recent literature review. Also in recent works on various orienteering problems, time-dependent travel times appear (Garcia et al., 2013; Gavalas et al., 2015; Verbeeck et al., 2014), see Gunawan et al. (2016) for a recent survey. Time-dependent travel times are important in many real world applications, for instance to model road congestion or public transportation networks (Gendreau et al., 2015; Gunawan et al., 2016). In many studies, the total route duration, including waiting time, is minimized (see for example Balseiro et al. (2011); Dabia et al. (2013); Kok et al. (2011, 2010)) or constrained (see for example Garcia et al. (2013); Gavalas

This chapter is based on Visser and Spliet (2017).

et al. (2015); Verbeeck et al. (2014)). Route duration in the objective can model a driver's salary, while the constrained route duration can model the maximum allowed working time of a driver.

Although exact methods have been proposed in the literature to solve time-dependent routing problems, for instance Dabia et al. (2013) which solve some instances up to 100 requests to optimality, (meta-)heuristics are needed to obtain high quality solutions for real world instances with 1000+ requests. Many heuristics for solving various rich vehicle routing problems rely internally on some form of Neighborhood Search (Vidal et al., 2013), see for example Balseiro et al. (2011); Donati et al. (2008); Figliozzi (2012); Garcia et al. (2013); Gavalas et al. (2015); Hashimoto et al. (2008); Verbeeck et al. (2014). Typically, a family of neighboring solution schedules, generated by applying various moves on the current incumbent solution schedule, are iteratively checked for feasible improvements. In such algorithms, it is critical to quickly check feasibility and the objective value of these moves. It is known that given a route as sequence of requests, its feasibility in the duration constrained VRPTW (so without time-dependent travel times) (Campbell and Savelsbergh, 2004; Savelsbergh, 1992) can be checked in $\mathcal{O}(n)$ time without pre-calculations (Vidal et al., 2015). This is also the case for the time-dependent VRPTW (so without duration constraints and without duration objective) (Donati et al., 2008; Vidal et al., 2015). However, when route duration and time-dependent travel times must be simultaneously optimized, the problem becomes more difficult.

The use of precalculated values stored as global data can be effective to speed-up Neighborhood Search procedures by avoiding unnecessary re-calculations during move evaluations. Kindervater and Savelsbergh (1997) proposed a framework to store global variables related to time windows and capacity constraints of a route in memory. Moves are evaluated in a so-called lexicographic order such that these global variables can be updated efficiently during the search. Many authors have since published effective global route variables for many different routing and scheduling problems, including Campbell and Savelsbergh (2004) who proposed global data for many constraints including shift time-limits. Irnich (2008) proposed a framework for non-lexicographic search, and introduced a class of pre-calculation data structures which retains fast move evaluations for non-lexicographic evaluation order while requiring only slightly more memory. Recently, Vidal et al. (2015) surveyed and generalized the concept of pre-calculation for many timing subproblems. This generalization is called "Reoptimization by concatenation". Using this framework, move evaluations of the duration minimized or constrained VRPTW (Campbell and Savelsbergh, 2004;

Savelsbergh, 1992) and the time-dependent VRPTW (without duration constraints) (Donati et al., 2008) can be done in $\mathcal{O}(1)$ time. However, Vidal et al. (2015) note that to their knowledge no efficient method for reoptimization exists for the move evaluation of the duration constrained or minimizing time-dependent VRPTW.

Hashimoto et al. (2008) discuss efficient move evaluations for the time-dependent VRPTW with additionally time-dependent piecewise linear start of service costs. We notice that the time-dependent VRPTW with route duration constraints or minimization can be reformulated to the problem of Hashimoto et al. (2008), thus allowing efficient reoptimization techniques to be applied. In particular, this shows that the *earliest-* and *latest arrival time* global variables of Savelsbergh (1992); Campbell and Savelsbergh (2004) can be generalized to piecewise linear forward- and backward *ready time* functions, and that move evaluations using these stored functions in reoptimization can be done in $\mathcal{O}(np)$ time, which is an improvement over a naive approach which takes $\mathcal{O}\big(n^2p\big)$ time. Here n is the total number of customers and p the maximum number of breakpoints of a travel time function. However, detailed analysis of reoptimization methods specifically for the time-dependent VRPTW with route duration constraints or minimization reveals new insights to increase performance further, which is the main focus of this chapter.

The contributions in this chapter are the following. We prove that the feasibility and cost calculation of a route without precalculations can be done in $\mathcal{O}(np \log n)$ time, improving the previously known $\mathcal{O}\big(n^2p\big)$ time. Using these ideas, we propose a novel data structure which is small in memory, $\mathcal{O}(np \log n)$, but allows the move evaluation complexity to remain in $\mathcal{O}(np)$, even when the neighborhood is searched in non-lexicographic order. This turns out to be particularity useful for evaluating advanced neighborhoods such as k-exchange. We support our complexity results by presenting numerical results on large benchmark instances of 1000 customers. Furthermore, we illustrate the general applicability of the speed-up methods by discussing extensions such as Multiple Time Windows.

This chapter is structured as follows. Section 2.2 contains the Time-dependent VRPTW with route duration constraints and minimization, and a typical Neighborhood Search procedure for solving it. In Section 2.3, we discuss the concept of ready time functions, which are the main ingredient of the speed-up methods. In Section 2.4, we discuss a speed-up method using forward and backward ready time functions, while in Section 2.5 we introduce a new data structure consisting of ready time function trees. In Section 2.6 we discuss some additional methods and in Section 2.7 we provide a summary of all methods. Section 2.8 contains the results of

our computational experiments and in Section 2.9 we illustrate the general applicability of the methods by providing some applications. Section 2.10 contains our conclusions.

2.2 Time-dependent VRPTW

The Time-Dependent VRPTW (TDVRPTW), with route duration constraints and minimization, is defined on a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, with vertex set $\mathcal{V} = \mathcal{V}^{\mathrm{C}} \cup \{o, d\}$ consisting of a set of $n = \left|\mathcal{V}^{\mathrm{C}}\right|$ customer vertices and two vertices representing the depot: source vertex o and sink vertex d. There is a fleet of identical vehicles, R in total, each with a capacity Q, starting at depot vertex o and ending at depot vertex d. Let a vertex $i \in \mathcal{V}$ be characterized by its demand q_i, service time s_i and time window $[a_i, b_i]$, with a_i (b_i) the earliest (latest) time at which service can start at the location. Without loss of generality, the demands at the depot vertices satisfy $q_o = q_d = 0$. It is assumed that a vehicle arriving at a customer before the time window opens must wait at the customer's location. Furthermore, let the planning horizon be finite and given by $[0, T]$. Let $\tau_{ij}(t)$ denote the travel time function which gives the travel time from vertex i to vertex j when departing from vertex i at time $t \in [0, T]$. The travel time functions are piecewise linear, continuous and satisfy the first-in-first-out (FIFO) property, meaning that the arrival time functions $\alpha_{ij}(t) := t + \tau_{ij}(t)$ are all strictly increasing (Ichoua et al., 2003). Furthermore, each travel time function has at most p breakpoints with p a fixed integer, which corresponds to at most $\lceil p/2 \rceil$ speed zones, i.e., time periods of constant speed, see Ghiani and Guerriero (2014) and Ichoua et al. (2003). Travel times at any time t do not have to satisfy the triangle inequality. Let c_{ij} be the fixed distance cost for traversing arc $(i, j) \in \mathcal{A}$, for instance based on distance, and let c^{DUR} be the fixed duration cost per time unit a vehicle is away from the depot. Also, the fixed distance costs c_{ij} do not have to satisfy the triangle inequality. Let Δ denote the maximum route duration in case this is constrained. The goal of the problem is to find at most R feasible vehicle routes, i.e., od-paths in $\mathcal{G}$, covering all customer requests with lowest total cost. Here, the total cost consists of either the sum of all distance costs c_{ij} of the arcs used, the sum of all route duration costs of the routes, or both.

We consider the above TDVRPTW to be solved heuristically by a Neighborhood Search-based method. In this chapter, we study the timing subproblem of checking feasibility and objective value change of insertion- and exchange type of moves, used extensively by such Neighborhood Search methods to solve the above problem.

2.2.1 Neighborhood Search

A Neighborhood Search procedure typically starts with an initial solution S and considers the neighborhood of this solution, $\mathcal{N}(S)$, which contains all solutions S' which are in some sense close to S. Typical examples of such neighborhoods include insertion, swap, 2-Opt* and k-exchange (Desaulniers et al., 2014). By searching $\mathcal{N}(S)$, a new solution $S^* \in \mathcal{N}(S)$ is found that is feasibile and has the lowest cost. We *move* to this new solution by replacing S with S^*. This procedure typically repeats until the local optimum solution is found which contains no improving solution in its neighborhood. Metaheuristics provide ways to continue the search beyond a local optimum, see for instance Labadie et al. (2016), but still the most time consuming part of such methods is typically the evaluation of all the moves in a neighborhood. Common speed-up techniques include the use of *pre-checks* and the use of *pre-calculations*. Pre-checks are quick calculations to conclude infeasibility or inferiority of a move without having to perform the full time-consuming exact feasibilty or cost calculations. Examples include time window violation checks using bounds on travel time and cost evaluations using bounds on the exact cost. Pre-calculations try to speed up the exact move evaluation calculations by storing partial calculation results, related to the current solution S, in memory. The partial results in memory need to be updated between the Neighborhood Search iterations to reflect the new solution. This chapter focuses mainly on speed-up methods of the latter kind, although we will see that their efficiency can depend on the used pre-checks.

2.3 Ready time functions

Let us introduce some definitions and theorems regarding so-called ready time functions (Dabia et al., 2013), which are the key ingredient of our analysis.

First, let us define the *time window ready time function* $\theta_i(t)$ for each vertex $i \in \mathcal{V}$, which represents the earliest time a vehicle is ready after service when arriving at vertex i at time t.

Definition 2.1 (Time window ready time function)**.** *The time window ready time function* $\theta_i : [0, b_i] \to \mathbb{R}$ *of vertex* $i \in \mathcal{V}$ *is defined as*

$$\theta_i(t) := \begin{cases} a_i + s_i & \text{if } t < a_i, \\ t_i + s_i & \text{if } t \in [a_i, b_i]. \end{cases} \tag{2.1}$$

Time window ready time functions are a special case of the general *ready time*

functions, which are convenient for determining the minimum route duration of a given route of customers. Let us define these functions as follows.

Definition 2.2 (Ready time function)**.** *Given a route* $r = (\ldots, i, \ldots, j, \ldots)$ *as a path of vertices on* $\mathcal{G}$*, the ready time function* $\delta_{ij}^r(t)$ *is defined as the function that gives the (earliest) time when* service is completed *at vertex* $j \in \mathcal{V}$ *after an arrival at vertex* $i \in \mathcal{V}$ *at time* t *when executing route* r*.*

By the FIFO property, later departures yield later arrivals at all subsequent vertices in a route. Hence, service at a vertex must start as soon as possible to minimize route duration. Since no other time-dependent penalties or constraints occur in our TDVRPTW, this uniquely defines the value of $\delta_{ij}^r(t)$ by using that every activity between i and j must start as soon as possible. Therefore, the ready time function δ_{ij}^r between any two vertices i and j in route $r = (\ldots, i, \ldots, j, \ldots)$ can be computed as follows:

$$\delta_{ij}^r(t) = \left(\theta_j \circ \alpha_{j-1,j} \circ \cdots \circ \theta_{i+2} \circ \alpha_{i+1,i+2} \circ \theta_{i+1} \circ \alpha_{i,i+1} \circ \theta_i\right)(t), \tag{2.2}$$

using the function composition notation $(f \circ g)(t) := f(g(t))$. Given the o–d ready time function δ_{od}^r of route r, the minimum route duration Δ_r^* can be calculated as follows:

$$\Delta_r^* = \min_{t \in T} \left\{\delta_{od}^r(t) - t\right\}. \tag{2.3}$$

The corresponding optimal depot departure time t_o^{r*} for route r is given by:

$$t_o^{r*} \in \arg\min_{t \in T} \left\{\delta_{od}^r(t) - t\right\}. \tag{2.4}$$

By the FIFO property, all ready time functions are nondecreasing. It turns out that all ready time functions are also piecewise linear and continuous but generally not convex. We will prove this formally in the next section, when we analyze the complexity of computing compositions. The minimum route duration in Equation (2.3) is therefore attained by at least one breakpoint of $\delta_{od}^r(t)$, which can be found in polynomial time by enumerating over the breakpoints of $\delta_{od}^r(t)$.

2.3.1 Complexity Analysis

Let a piecewise linear, continuous and nondecreasing function $f : [0, T_f] \to \mathbb{R}$ with domain $\operatorname{dom}(f) := [0, T_f] \subseteq [0, T]$ be given. We define its *ordered set of breakpoints* $\mathcal{F}_f := \left\{(t_1, f(t_1)), (t_2, f(t_2)), \ldots, \left(t_{\phi_f}, f(t_{\phi_f})\right)\right\}$, with a number of $\phi_f := |\mathcal{F}_f|$ break-

points. Without loss of information, let us define $\mathcal{F}_f$ to have the special property that the first breakpoint of f, $(0, f(0))$, is omitted in the set $\mathcal{F}_f$ only if f starts with a horizontal segment, i.e., if $f(0) = f(t_1)$. For example, $\mathcal{F}_{\alpha_{ij}} = \{(0, t_{ij}), (T - t_{ij}, T)\}$ is the ordered set of breakpoints of a classical non-time dependent arrival time function $\alpha_{ij}(t)$ with constant travel time $t_{ij} < T$, while $\mathcal{F}_{\theta_i} = \{(a_i, a_i + s_i), (b_i, b_i + s_i)\}$ is the ordered set of breakpoints of a time window ready time function $\theta_i(t)$ given by Equation (2.1). Notice that the former function starts with a non-horizontal segment and thus its set of ordered breakpoints starts with a breakpoint at $t = 0$, while the latter function starts with a horizontal segment for $t \in [0, a_i]$, and thus its set of ordered breakpoints starts with a breakpoint at $t = a_i$. Throughout this chapter, any piecewise linear, continuous and nondecreasing function f is computationally directly associated with its ordered set of breakpoints $\mathcal{F}_f$.

Theorem 2.3. *Let $f_1(t)$ and $f_2(t)$ be two piecewise linear, continuous and nondecreasing functions. The following properties hold for the composition $f = f_2 \circ f_1$:*

1. *f is again a piecewise linear, continuous and nondecreasing function,*
2. *The number of breakpoints ϕ_f of f is at most $\phi_{f_1} + \phi_{f_2} - 2$. Moreover, f has at most 1 breakpoint if either $\phi_{f_1} = 1$ or $\phi_{f_2} = 1$, and $\phi_f = 0$ if either $\phi_{f_1} = 0$ or $\phi_{f_2} = 0$,*
3. *Calculation of f requires at most $\mathcal{O}(\phi_{f_1} + \phi_{f_2})$ operations.*

Proof. Proof of Property 1: The composition of two continuous functions is again continuous. Let $t \in \mathrm{dom}\,(f_1)$ be such that $f_1(t) \in \mathrm{dom}\,(f_2)$ and both t is not a breakpoint of f_1 and $f_1(t)$ is not a breakpoint of f_2. Since f_1 is differentiable at t and f_2 at $f_1(t)$, let f_1' and f_2' denote the derivatives of f_1 and f_2 respectively, between their breakpoints. By elementary calculus, it holds that the derivative f', which gives the slope of the composition, is given by $f'(t) = f_1'(t) \cdot f_2'(f_1(t))$ for any t such that both $f_1(t)$ and $f_2(f_1(t))$ are between breakpoints. Because both functions are piecewise linear, the derivatives f_1' and f_2' are constant between breakpoints of respectively f_1 and f_2. Their product is therefore constant as well between breakpoints. We conclude f is a piecewise linear function as well. Also, because both derivatives are nonnegative by the nondecreasing property of f_1 and f_2, the derivative f' is also nonnegative, and by additionally using the continuity of f it follows that f is nondecreasing.

Proof of Property 2: By the above observations, the value of the derivative f' does not change more than $\phi_{f_1} + \phi_{f_2}$ times on its domain. Therefore, f cannot

have more than $\phi_{f_1} + \phi_{f_2}$ breakpoints. Notice that the resulting domain of f will be $\text{dom}(f) = [0, \min\{f_1(T_{f_1}), T_{f_2}\}]$, given that $\text{dom}(f_1) = [0, T_{f_1}]$ and $\text{dom}(f_2) = [0, T_{f_2}]$. This effectively reduces the number of breakpoints f can have by at least one, since the breakpoint at $\max\{f_1(T_{f_1}), T_{f_2}\}$ falls outside the resulting domain. The bound on the number of breakpoints of f can be tightened further by using properties of our breakpoint representation. Let $(t_1^1, f_1(t_1^1))$ be the first breakpoint of f_1 and $(t_1^2, f_2(t_1^2))$ the first breakpoint of f_2. If $t_1^1 > 0$, then f_1 starts with a horizontal segment, i.e., zero slope and a similar condition holds for f_2. Therefore, the composition f will start a with horizontal segment on the first part of domain $[0, \max\{t_1^1, f_1^{-1}(t_1^2)\}]$, with f_1^{-1} the inverse of f_1. The breakpoint corresponding to $\min\{t_1^1, f_1^{-1}(t_1^2)\}$ falls inside this horizontal segment at the start and is removed from the resulting breakpoint representation. This still holds when either $t_1^1 = 0$ or $t_1^2 = 0$. The bound on the number of breakpoint of f is therefore $\phi_{(f_2 \circ f_1)} \leq \phi_{f_1} + \phi_{f_2} - 2$. Naturally, the following special cases hold: $\phi_f \leq 1$ if either $\phi_{f_1} = 1$ or $\phi_{f_2} = 1$, which means that at least one of the two functions corresponds with a fixed arrival time, and $\phi_f = 0$ if either $\phi_{f_1} = 0$ or $\phi_{f_2} = 0$.

Proof of Property 3: Given that both sets of breakpoints $\mathcal{F}_{f_1}$ and $\mathcal{F}_{f_2}$ of respectively f_1 and f_2 are sorted in time, the new set of sorted breakpoints $\mathcal{F}_f$ can be constructed from begin to end using at most $\phi_{f_1} + \phi_{f_2}$ comparisons. A single comparison is used to determine which of the first remaining breakpoint of both sets is the earliest and should be incorporated in $\mathcal{F}_f$ first. Since one comparison is responsible for incorporating one breakpoint from either $\mathcal{F}_{f_1}$ and $\mathcal{F}_{f_2}$, at most $\phi_{f_1} + \phi_{f_2}$ comparisons are needed in total. Furthermore, the calculation of the values of a new breakpoint $(t, f(t))$ of f can be done in $\mathcal{O}(1)$ time since either value t or $f(t)$ is part of a breakpoint in respectively $\mathcal{F}_{f_1}$ or $\mathcal{F}_{f_2}$ and the other value can be calculated by linear interpolation in $\mathcal{O}(1)$. Since the corresponding line segment is already known by the fact that the breakpoint sets are sorted, no additional operations are needed to locate the right line segment for interpolation. Therefore, the composition requires at most $\mathcal{O}(\phi_{f_1} + \phi_{f_2})$ operations. □

The proof of Property 3 of Theorem 2.3 suggests an efficient algorithm for calculating compositions of nondecreasing piecewise linear functions. Furthermore, the following corollary follows directly from Property 2 of Theorem 2.3.

Corollary 2.4. *The composition $f = f_2 \circ f_1$ can only have more breakpoints than either f_1 or f_2 if both $\phi_{f_1} \geq 3$ and $\phi_{f_2} \geq 3$.*

Throughout this chapter, it is assumed that for all arcs $(i, j) \in \mathcal{A}$, the number

of breakpoints of the arrival time function is bounded by some fixed $p \in \mathbb{N}$: i.e., $\phi_{\alpha_{ij}} \leq p$ for all $(i,j) \in \mathcal{A}$. Recall that the number of breakpoints $\phi_{\theta_i} = 2$ for any time window ready time function θ_i. By Corollary 2.4 and Equation (2.2), taking the composition of arrival time- and time window ready time functions repeatedly can only increase the number of breakpoints of the resulting composition if $p \geq 3$. This leads to the following Lemma.

Lemma 2.5. *The ready time function δ^r_{ij}, with $i,j \in r$ such that $i < j$ and a total of m vertices are visited, has $\mathcal{O}(mp)$ number of breakpoints, which can be calculated from scratch in at most $\mathcal{O}\big(m^2p\big)$ operations. In the special case of $p = 2$ breakpoints, the ready time function δ^r_{ij} has at most 2 breakpoints and requires at most $\mathcal{O}(m)$ operations to calculate from scratch.*

Proof. Repeated application of Properties 2 and 3 of Theorem 2.3 on the ready time function composition of Equation (2.2) gives the required number of breakpoints $\mathcal{O}(mp)$ and the number of operations $\mathcal{O}\big(m^2p\big)$. In the special case of $p = 2$, e.g., as in the case of classical non-time dependent travel times, all arrival- and time window ready time functions have at most two breakpoints and thus also the composition. Therefore, in this case, a ready time function is obtained by calculating at most $\mathcal{O}(m)$ function compositions and thus the total number of operations required is $\mathcal{O}(m)$. □

2.4 Forward and backward ready time functions

In this section, we describe a procedure that allows fast feasibility checks for insertion and exchange moves in local search procedures. The procedure is essentially a generalization of the *forward* (*backward*) *slack* variables introduced by Savelsbergh (1992). In the comprehensive survey of Vidal et al. (2015), the authors state that they are unaware of efficient feasibility checks for the time-dependent VRPTW with route duration costs. However, we notice that this problem can be reformulated as a time-dependent VRPTW with linear time-dependent costs, for which Hashimoto et al. (2008) provided fast feasibility checks. The method presented here can be found as a special case of the method of Hashimoto et al. (2008). Our presentation allows us later in Section 2.5 to introduce a new data structure which can decrease computation times further, in particular for more advanced moves like exchanges.

2.4.1 Insertion Moves

Let a route $r \in \mathcal{R}$ with m customers be given. Let us conveniently label the customers such that $r = (o, 1, 2, \ldots, m, d)$. We will prove that evaluating the insertion of another customer j directly before customer i into this route, resulting in route $\tilde{r} = (o, 1, 2, \ldots, i-1, j, i, \ldots, m, d)$, can be done in $\mathcal{O}(mp)$ time.

First, notice that both feasibility of the vehicle capacity and the new fixed arc cost component $\sum_{(i,j)\in\tilde{r}} c_{ij}$ can be determined in $\mathcal{O}(1)$ time, given that the used capacity and fixed arc cost component of the old route r are stored in memory. The difficult part is to determine feasibility of the time window constraints and the new route $\tilde{r}$ minimum route duration $\Delta^*_{\tilde{r}}$. The latter is needed to check feasibility of the route duration constraint and to determine the new route duration cost.

A direct consequence of Equation (2.2) are the following equations for the o–d ready time function $\delta^r_{od}(t)$ of the old route r and the o–d ready time function $\delta^{\tilde{r}}_{od}(t)$ of the new route $\tilde{r}$.

$$\begin{aligned}\delta^r_{od}(t) &= (\theta_d \circ \alpha_{md} \circ \cdots \circ \theta_i \circ \alpha_{i-1,i} \circ \theta_{i-1} \circ \cdots \circ \alpha_{12} \circ \theta_1 \circ \alpha_{o1} \circ \theta_o)(t) \\ &= (\delta^r_{id} \circ \alpha_{i-1,i} \circ \delta^r_{o,i-1})(t), \qquad (2.5)\\ \delta^{\tilde{r}}_{od}(t) &= (\theta_d \circ \alpha_{md} \circ \cdots \circ \theta_i \circ \alpha_{ji} \circ \theta_j \circ \alpha_{j,i-1} \circ \theta_{i-1} \circ \cdots \circ \alpha_{12} \circ \theta_1 \circ \alpha_{o1} \circ \theta_o)(t) \\ &= (\delta^r_{id} \circ \alpha_{ji} \circ \theta_j \circ \alpha_{j,i-1} \circ \delta^r_{o,i-1})(t). \qquad (2.6)\end{aligned}$$

Equation (2.6) shows that in order to calculate the new route o–d ready time function $\delta^{\tilde{r}}_{od}(t)$, it suffices to calculate a composition consisting of the old route r ready time functions $\delta^r_{o,i-1}(t)$ and $\delta^r_{id}(t)$, the arrival-time functions $\alpha_{i-1,j}(t)$ and $\alpha_{ji}(t)$, and the time window ready time $\theta_j(t)$. Ready time functions of the form $\delta^r_{o,i-1}(t)$ and $\delta^r_{id}(t)$ are called respectively the *forward* and *backward* ready time functions. The equation shows that if these forward and backward ready time function are in memory, the calculation of $\delta^{\tilde{r}}_{od}(t)$ can be done by calculating four function compositions and using Equation (2.3) on the resulting $\delta^{\tilde{r}}_{od}(t)$ to get the exact minimum duration of the new route $\tilde{r}$.

By using Lemma 2.5 and Theorem 2.3 on Equation (2.6) (bottom part), one can prove the following complexity result of the exact insertion move evaluation.

Theorem 2.6. *Given the forward- and backward ready time functions $\delta^r_{o,i-1}(t)$ and $\delta^r_{id}(t)$ of a route $r = (o, 1, \ldots, i-1, i, \ldots, m, d)$, the o–d ready time function $\delta^{\tilde{r}}_{od}(t)$ of a new route $\tilde{r} = (o, 1, \ldots, i-1, j, i, \ldots, m, d)$ can be calculated using at most $\mathcal{O}(mp)$ operations in which also the exact minimum duration $\Delta^*_{\tilde{r}}$ of route $\tilde{r}$ is calculated. In*

the special case of $p = 2$, only at most $\mathcal{O}(1)$ operations are required.

Proof. According to Lemma 2.5 the composition $\delta^{\tilde{r}}_{od}(t) = \left(\delta^{r}_{id} \circ \alpha_{ji} \circ \theta_j \circ \alpha_{i-1,j} \circ \delta^{r}_{o,i-1}\right)(t)$ can be calculated in $\mathcal{O}(mp)$ operations and has $\mathcal{O}(mp)$ breakpoints, if $p \geq 3$. Determining the minimum route duration $\Delta^*_{\tilde{r}}$ by means of Equation (2.3) requires additionally $\mathcal{O}(mp)$ operations, yielding a total time complexity of $\mathcal{O}(mp)$. In the special case of $p = 2$, both $\delta^{r}_{o,i-1}$ and δ^{r}_{id} have at most $p = 2$ breakpoints and thus the composition $\delta^{\tilde{r}}_{od}(t) = \left(\delta^{r}_{id} \circ \alpha_{ji} \circ \theta_j \circ \alpha_{i-1,j} \circ \delta^{r}_{o,i-1}\right)(t)$ can be calculated and used to determine minimum route duration in total of $\mathcal{O}(1)$ time. □

Notice that in the special case of $p = 2$, which includes the (classical) non-time dependent duration minimizing or constraint VRPTW, insertions can be checked in $\mathcal{O}(1)$ time using this method. This matches the well known result of Savelsbergh (1992) and Campbell and Savelsbergh (2004). Moreover, the forward- and backward ready time functions, which contain at most 2 breakpoints in this special case, can be directly related to the global variables used by Savelsbergh (1992) and Campbell and Savelsbergh (2004) to quickly evaluate moves.

Provided that the number of customers in a route is of $\mathcal{O}(n)$, with n the total number of customers, the composition can be calculated in $\mathcal{O}(np)$ time and has $\mathcal{O}(np)$ number of breakpoints. We believe it is unlikely that for our setting a method exists which can exactly check insertion moves faster than $\mathcal{O}(np)$. Going over the breakpoints of an o–d ready time function $\delta^{\tilde{r}}_{od}$, as required to determine the minimum route duration, already takes $\mathcal{O}(np)$ time by breakpoint enumeration, which seems necessary for general non-convex ready time functions.

2.4.2 Exchange Moves

Ready time functions can also be used to quickly evaluate more advanced moves than insertion, like the commonly used exchange moves. An exchange move takes two subsequences of customers from two routes and exchanges them. Usually, the size of the subsequences considered is $0, 1, 2, \ldots, k$ with a constant maximum size $k \in \mathbb{N}$, and subsequences of different length can be exchanged, but their orientation stays the same. This way, the exchange neighborhood consists of $\mathcal{O}\left(n^2k^2\right)$ possible moves. An example of an exchange move is given in Figure 2.1. The special case of an exchange move with $k = 1$ is generally called a swap move. Also insert and 2-opt* moves can be seen as special cases of an exchange move if some subsequences are allowed to be empty. Note that the exchange moves do not reverse the direction of a subsequence, since reversals typically result in time window infeasibilities. However, all methods

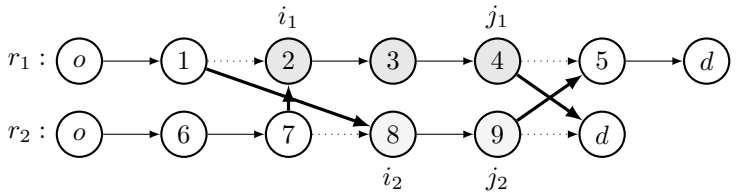


Figure 2.1: Illustration of the evaluation of a 3-exchange move $(i_1, j_1, i_2, j_2) = (2, 4, 8, 9)$, which exchanges customers 2, 3 and 4 from route r_1 with customers 8 and 9 from route r_2.

presented in this chapter can be straightforwardly extended to also support moves with subsequence reversals.

Let us denote the move $\mathcal{M}$ which exchanges customers $i_1, \ldots, j_1$ from route r_1 with customers $i_2, \ldots, j_2$ from route r_2 by $\mathcal{M} = (i_1, j_1, i_2, j_2)$. To evaluate the move (i_1, j_1, i_2, j_2), first the o–d ready time functions $\delta_{od}^{\tilde{r}_1}$, $\delta_{od}^{\tilde{r}_2}$ resulting from the exchange need to be calculated:

$$\begin{aligned}
\delta_{od}^{\tilde{r}_1}(t) &= \left(\delta_{j_1+1,d}^{r_1} \circ \alpha_{j_2,j_1+1} \circ \delta_{i_2 j_2}^{r_2} \circ \alpha_{i_1-1,i_2} \circ \delta_{o,i_1-1}^{r_1}\right)(t), \\
\delta_{od}^{\tilde{r}_2}(t) &= \left(\delta_{j_2+1,d}^{r_2} \circ \alpha_{j_1,j_2+1} \circ \delta_{i_1 j_1}^{r_1} \circ \alpha_{i_2-1,i_1} \circ \delta_{o,i_2-1}^{r_2}\right)(t).
\end{aligned} \tag{2.7}$$

Similar to Theorem 2.6, these compositions can be calculated and checked for minimum route durations in $\mathcal{O}((m_1 + m_2)p)$ time, with m_1 and m_2 the number of customers of routes r_1 and r_2 respectively, provided that all partial ready time functions, including the middle parts $\delta_{i_1 j_1}^{r_1}$ and $\delta_{i_2 j_2}^{r_2}$, are already available in memory. Supposing that the number of customers in the routes is of the order $\mathcal{O}(n)$, the composition can be calculated and checked for minimum route durations in $\mathcal{O}(np)$ time. However, if some functions are not in memory, then by Lemma 2.5 it requires $\mathcal{O}\big(n^2 p\big)$ operations to calculate the missing ready time functions from scratch. This increase illustrates the benefit of having the ready time functions available in memory.

2.4.3 Updates and Memory

After a neighborhood is searched and the best (improving) move is found, this move is executed. The global data structures need to be updated for the changed route(s). In general, most forward (δ_{oi}^{r}) and backward (δ_{id}^{r}) ready time functions of a changed route r need to be updated, requiring $\mathcal{O}\big(n^2 p\big)$ memory and time in total by Lemma 2.5. Would additionally all partial ready time functions (δ_{ij}^{r} for all

$i, j \in r \setminus \{o, d\}$, $i < j$) be stored in memory, to provide instant availability of the middle segment ready time function in the exchange neighborhood, then both the update time and memory complexity increases to $\mathcal{O}\Big(\sum_{j=1}^{n}\sum_{i=j}^{n} ip\Big) = \mathcal{O}\big(n^3 p\big)$. In practical settings this complexity is usually too computationally expensive. However, we can search an exchange neighborhood efficiently with only the forward and backward ready time functions in memory, requiring only $\mathcal{O}\big(n^2 p\big)$ memory and update time in total, by using Lexicographic Search (Kindervater and Savelsbergh, 1997) as explained in the next section.

2.4.4 Lexicographic Search

Lexicographic Search entails searching a neighborhood in such an order that calculations done for evaluating a move can be used for efficient evaluation of the next move. For example, an exchange neighborhood can be searched in such an order that only relatively small computations are needed to update readily calculated middle segment ready time functions (and forward segment ready time functions) between moves. To illustrate this, let us consider two consecutive moves, $\mathcal{M}_1$ and $\mathcal{M}_2$, in an exchange neighborhood and let $\mathcal{M}_1$ be given by (i_1, j_1, i_2, j_2) in notation used earlier. Suppose we restrict the next move $\mathcal{M}_2$ to be *near* $\mathcal{M}_1$, meaning it is obtained by only extending one of the middle segments by one customer, i.e., $(i_1, j_1 + 1, i_2, j_2)$ or $(i_1, j_1, i_2, j_2 + 1)$, or by starting a new middle segment of zero or one vertex. In the first case, the middle segment ready time function $\delta^{r_1}_{i_1,j_1+1}$ can be obtained in $\mathcal{O}(np)$ time by extending the previous middle segment ready time function with one vertex: $\delta^{r_1}_{i_1,j_1+1} = \big(\theta_{j_1+1} \circ \alpha_{j_1,j_1+1} \circ \delta^{r_1}_{i_1,j_1}\big)$, which requires $\mathcal{O}(np)$ operations. In the last case of starting a new middle segment of zero or one vertex, the middle segment ready time function can also be obtained in $\mathcal{O}(np)$ operations. Together with the forward and backward ready time functions in memory, the total time required for evaluating an exchange move is still $\mathcal{O}(np)$, without needing to pre-calculate and store all partial ready time functions δ^{r}_{ij}. Since such an lexicographic ordering exists to cover the full exchange neighborhood, it can be searched efficiently. Notice that this does require us to keep the middle segment updated between exchange moves, which requires some computation time. Also notice that some other neighborhoods, like exchange with fixed subsequence length $k \geq 3$ of both segments, cannot be searched lexicographically. Therefore, moves in such neighborhoods generally cannot be evaluated in $\mathcal{O}(np)$ time. In Section 2.5 we introduce a special data structure of ready time functions which can overcome this issue.

2.4.5 Summary

We have seen that storing the forward- and backward ready time functions δ^r_{oi} and δ^r_{id} of all routes requires $\mathcal{O}(n^2p)$ memory and operations to update. This enables us to do fast insertion move evaluations in $\mathcal{O}(np)$ time. However, exchange moves also require the middle segment ready time functions to be available. By searching the exchange neighborhood in lexicographic order, the middle segments are updated gradually between moves which keeps the time of the move evaluation of $\mathcal{O}(np)$. Searching the exchange neighborhood in a non-lexicographic order increases the move evaluation time to $\mathcal{O}(n^2p)$, or requires us to store all partial ready time functions which cost $\mathcal{O}(n^3p)$ memory and operations to update.

2.5 Ready time function tree

In this section, we show that o–d ready time functions can actually be calculated from scratch in $\mathcal{O}(np \log n)$ operations, instead of the previous $\mathcal{O}(n^2p)$ operations. This insight leads to a new data structure. By storing specific partial ready time functions in a balanced binary search tree data structure, any partial ready time function can be obtained in $\mathcal{O}(np)$ operations without the need for a lexicographic order. Furthermore, we show that such trees require only $\mathcal{O}(np \log n)$ memory and operations to update, which is less than the $\mathcal{O}(n^2p)$ memory and operations needed for the forward and backward ready time functions. Although different, our tree data structure achieves the same goals as the class of l-level Hierarchy data structures of Irnich (2008) for non-time dependent routing problems: to retain fast move evaluations, even for a non-lexicographic evaluation order, while only slightly increasing both the complexity of memory and operations needed to update, which we explain next.

2.5.1 Motivation

Our motivation for an efficient tree data structure comes from a simple observation. Although the *order* in which the ready time function compositions are calculated obviously does not influence the final result, the order does significantly impact the (worst-case) number of operations required. Let us illustrate this by the following example. Suppose the o–d ready time function δ^r_{od} of a route $r = (o, 1, 2, 3, d)$ containing 5 vertices, $m = 3$ customers, needs to be calculated from scratch. Let us

calculate it in two ways:

$$\delta^r_{od} = (\theta_d \overset{8}{\circ} (\alpha_{3d} \overset{7}{\circ} (\theta_3 \overset{6}{\circ} (\alpha_{23} \overset{5}{\circ} (\theta_2 \overset{4}{\circ} (\alpha_{12} \overset{3}{\circ} (\theta_1 \overset{2}{\circ} [\alpha_{o1} \overset{1}{\circ} \theta_o])))))))) \tag{2.8}$$

$$= ([(\theta_d \overset{5}{\circ} \alpha_{3d}) \overset{7}{\circ} (\theta_3 \overset{4}{\circ} \alpha_{23})] \overset{8}{\circ} [(\theta_2 \overset{3}{\circ} \alpha_{12}) \overset{6}{\circ} (\theta_1 \overset{2}{\circ} \alpha_{o1} \overset{1}{\circ} \theta_o)]), \tag{2.9}$$

in which the number above the composition symbol represents the order of evaluation (composition 1 is evaluated first, composition 2 second, etc.). Equation (2.8) starts by calculating $\delta^r_{o1} = (\theta_1 \circ (\alpha_{o1} \circ \theta_o))$ and then extends this function by forward compositions to form δ^r_{o2}, then to δ^r_{o3}, etc., and repeats this process until δ^r_{od} is obtained. Equation (2.9) uses a different evaluation order. First, all functions of the form $\delta^r_{i,i+1} = (\theta_{i+1} \circ \alpha_{i,i+1})$ are calculated. Then, two neighboring functions are combined into functions of the form $\delta^r_{2i,2i+2}$, and then these latter functions are combined to functions of the form $\delta^r_{4i,4i+4}$, etc., and this is repeated until the only two remaining functions are combined to form δ^r_{od}.

Both Equations (2.8) and (2.9) require an equal number of compositions and produce the same *o–d* ready time function with at most $4p - 6$ breakpoints, but for $p \geq 3$, the first equation requires much more operations in the worst case than the second. This is due to the favorable order in which the second equation evaluates the compositions. Each time subsequently the composition involving functions with the least amount of breakpoints is evaluated, while the first equation keeps evaluating the composition involving the largest function. This is reflected by the order of the number of operations required. Using Equation (2.8), $\mathcal{O}(ip)$ operations are required to extend $\delta^r_{o,i-1}$ to δ^r_{oi}. Therefore, in total $\mathcal{O}\left(\sum_{i=1}^{m+1} ip\right) = \mathcal{O}(m^2p)$ number of operations are required to obtain δ^r_{od}. Using Equation 2.9, $\mathcal{O}(p)$ operations are required for evaluating each lowest-level compositions, compositions $1, 2, \ldots, 5$, which is in total $\mathcal{O}(mp)$. The higher level compositions 6 and 7 each require $\mathcal{O}(2p)$ operations, which is in total again $\mathcal{O}(mp)$. The highest level composition 8 requires $\mathcal{O}(4p) = \mathcal{O}((m+1)p)$ operations. In this way, the compositions are grouped in a number of $\mathcal{O}(\log m)$ levels each requiring a total of $\mathcal{O}(mp)$ operations. Overall, using Equation (2.9) thus requires $\mathcal{O}(mp \log m)$ operations. This is an improvement over using Equation (2.8) requiring $\mathcal{O}(m^2p)$ operations.

Besides lowering the complexity of calculating an *o–d* ready time function from scratch, this favorable order of composition evaluation also gives rise to a new data structure. The intermediate ready time functions obtained during the evaluation using Equation (2.9), can be stored in memory. These functions form a balanced binary search tree data structure of ready time functions of a route. This data

structure can be used to quickly obtain partial ready time functions.

In the following, we will formally define the ready time function tree and provide construction and memory complexity results. Furthermore, we derive complexity results for obtaining partial ready time functions and show how this is useful for checking insertion moves and the more advanced exchange moves. We conclude by elaborating its benefits in terms of non-lexicographical neighborhood searches.

2.5.2 Tree definition and construction

Let a route $r = (o, 1, 2, \ldots, m, d)$ be given. The ready time function tree $\mathcal{T}^r$ of route r consists of all intermediate partial ready time functions resulting from calculating δ^r_{od} in the efficient way illustrated by Equation (2.9). It is convenient to use the following slightly different ready time function definition ${\delta'}^r_{ij}$:

$${\delta'}^r_{ij} = \begin{cases} (\theta_j \circ \alpha_{j-1,j} \circ \cdots \circ \theta_1 \circ \alpha_{o,1} \circ \theta_o) & \text{if } i = o, \\ (\theta_j \circ \alpha_{j-1,j} \circ \cdots \circ \theta_{i+1} \circ \alpha_{i,i+1}) & \text{otherwise.} \end{cases} \tag{2.10}$$

This ensures the nice property that ${\delta'}^r_{ij} = {\delta'}^r_{lj} \circ {\delta'}^r_{il}$ for all $i < l < j$. Notice that ${\delta'}^r_{od} = \left({\delta'}^r_{md} \circ \cdots \circ {\delta'}^r_{12} \circ {\delta'}^r_{o1}\right) = \delta^r_{od}$.

Tree $\mathcal{T}^r$ is a balanced binary tree. Each leaf node of $\mathcal{T}^r$ contains a ready time function between two consecutive vertices: the leftmost leaf node contains ${\delta'}^r_{o1}$, the one next ${\delta'}^r_{12}$, etc. Leaf node i contains ${\delta'}^r_{i,i+1}$ for $i \in r \setminus \{d\}$. Each internal node of $\mathcal{T}^r$ consists of the composition of its two child nodes, for instance internal node ${\delta'}^r_{o2}$ consists of the composition of its children ${\delta'}^r_{o1}$ and ${\delta'}^r_{12}$. By construction, the root node of tree $\mathcal{T}^r$ contains the o–d ready time function $\delta^r_{od} = {\delta'}^r_{od}$. Figure 2.2 provides an example of the balanced binary search tree of ready time functions for a route of 15 customers.

The tree is most efficiently constructed from bottom to top, like in the example of Equation (2.9). First, all ready time functions $\delta'^r_{i,i+1}$ concerning only two adjacent route vertices are calculated and put into the leaf nodes. Next, two leaf nodes are combined by composition to form their parent node. When all parents of the leaf nodes are calculated, they are combined to form their parents. This process repeats until the last two remaining nodes are combined to form the root node, which corresponds to ${\delta'}^r_{od}$. The speed and memory performance of the construction process are summarized by the following theorem.

Theorem 2.7. *Without any pre-calculations, the construction of the ready time function tree $\mathcal{T}^r$ of a route r with m customers requires $\mathcal{O}(mp \log m)$ operations and*

equal memory to store, in case $p \geq 3$.

Proof. Let m be the number of customers on route r. The calculation of each leaf node ready time function ${\delta'}^r_{i,i+1}$ requires at most $p+2$ operations, except for ${\delta'}^r_{0,1}$ which requires at most $p+2+p+2$ operations. Each leaf node has a ready time function with at most p breakpoints and there are $m+1$ leaf nodes. Thus a total of at most $\mathcal{O}((m+2)p)$ operations are needed for the lowest level of the tree. By construction, the binary tree has a height of $\lceil \log_2(m+1) \rceil + 1 = \mathcal{O}(\log m)$ levels. Let the lowest level containing only the leaf nodes be denoted by level 0 and the highest level containing the root node be level $L = \lceil \log_2(m+1) \rceil$. The tree has at most 2^{L-l} nodes at level $l \in \{0, 1, \ldots, L\}$ and each node has a ready time function of at most $2^l p$ breakpoints. To calculate the ready time function of a node at level l requires the composition of its two childs in level $l-1$, which requires at most $2 \cdot 2^{l-1} p = 2^l p$ operations. Thus in total for level l at most $2^{L-l} 2^l p = \mathcal{O}((m+1)p) = \mathcal{O}(mp)$ operations are needed. Since there are $\lceil \log_2(m+1) \rceil + 1$ levels, the total construction time of the tree is bounded by $(\lceil \log_2(m+1) \rceil + 1) \cdot (m+2)p = \mathcal{O}(mp \log m)$. This is also the amount of memory needed by a very similar argument. □

Notice that the *o–d* ready time function δ^r_{od} is always the root node in the ready time function tree and the minimum duration can be obtained from it by examining all its $\mathcal{O}(mp)$ breakpoints. Therefore, by Theorem 2.7, we can calculate the minimum duration of a route r with m nodes from scratch in $\mathcal{O}(mp \log m)$ operations by constructing the ready time function tree. This improves the previously best known methods requiring $\mathcal{O}(m^2 p)$ operations. Also, the memory and update time required to store and update the data structure is $\mathcal{O}(mp \log m)$, which is again lower than the memory and update time required for storing and updating all forward and backward ready time functions δ^r_{oi} and δ^r_{id}, which is $\mathcal{O}(m^2 p)$. Moreover, a tree in memory is particularly useful in obtaining any partial ready time function of a route quickly, which is the topic of the next section.

2.5.3 Obtaining partial ready time functions using the tree

Fast move evaluations of both insertion and exchange moves require us to calculate or obtain some partial ready time functions (forward, backward or middle segment ready time functions). If a particular ready time function is in a ready time function tree in memory, it can be obtained immediately. For partial ready time functions not in the tree, we will show that the tree nodes can be used to calculate *any* partial

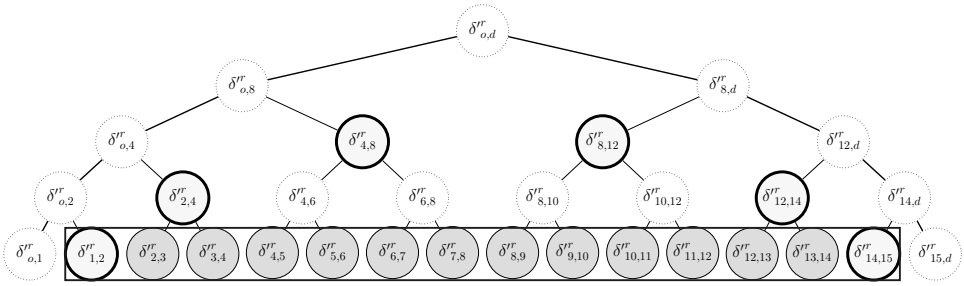


Figure 2.2: Example of a ready time function tree of route $(o, 1, 2, \ldots, 14, 15, d)$.

ready time function δ^r_{ij} in $\mathcal{O}(\bar{m}p)$ time, with $\bar{m}$ the number of vertices between i and j.

Let us first consider the following example. Suppose we have a route $r = (o, 1, 2, \ldots, m-1, m, d)$ with $m = 15$ customers and have obtained its ready time function tree, which is illustrated in Figure 2.2. Suppose that to evaluate an exchange move, partial ready time function ${\delta'}^r_{1,15}$ needs to be calculated. It can been seen in Figure 2.2 that this ready time function is not already in the tree itself. Instead of calculating it from scratch, which requires the composition of all the nodes in the rectangle in Figure 2.2, tree nodes at a higher level can be used to reduce the number of total operations required: ${\delta'}^r_{1,15} = \left({\delta'}^r_{15,14} \circ {\delta'}^r_{12,14} \circ {\delta'}^r_{8,12} \circ {\delta'}^r_{4,8} \circ {\delta'}^r_{2,4} \circ {\delta'}^r_{1,2}\right)$. In Figure 2.2 these corresponding tree nodes are illustrated by the thick circles. The composition of these thick nodes is most efficiently calculated using the following composition evaluation order:

$$ {\delta'}^r_{1,15} = \left(\left[\left({\delta'}^r_{14,15} \overset{2}{\circ} {\delta'}^r_{12,14}\right) \overset{4}{\circ} {\delta'}^r_{8,12}\right] \overset{5}{\circ} \left[{\delta'}^r_{4,8} \overset{3}{\circ} \left({\delta'}^r_{2,4} \overset{1}{\circ} {\delta'}^r_{1,2}\right)\right]\right). \tag{2.11} $$

It turns out that by properties of balanced search trees, the above example is among the configurations requiring most operations. Analysis of these configurations leads to the following theorem regarding the worst-case number of operations needed for obtaining *any* partial ready time functions using the tree in memory.

Theorem 2.8. *Given the ready time function tree $\mathcal{T}^r$ of a route r, any partial route ready time function δ^r_{ij} can be obtained in at most $\mathcal{O}(\bar{m}p)$ operations, with $\bar{m}$ the number of vertices between i and j, inclusive, on route r.*

Proof. Suppose δ^r_{ij} needs to be obtained using the tree $\mathcal{T}^r$, with $\bar{m}$ vertices between i and j in route r. By general properties of balanced search trees (see de Berg et al. (2008, p. 96–99)), the ready time functions stored in the nodes of $\mathcal{T}^r$ which are most

efficient for composing δ'^{r}_{ij} (the thick nodes in Figure 2.1) can be found as follows. Leaf node $\delta'^{r}_{i-1,i}$ in $\mathcal{T}^r$ corresponds to the node directly left of the required interval and leaf node $\delta'^{r}_{j,j+1}$ directly right next to the interval. Both these leaf nodes have a unique search path to the root node δ'^{r}_{od}. At some node, which we denote by $\delta'^{r}_{\text{split}}$, both search paths will be merged. In Figure 2.1, the leaf nodes are δ'^{r}_{o1} and $\delta'^{r}_{15,d}$ and their search path merges at $\delta'^{r}_{\text{split}} = \delta'^{r}_{od}$. The most efficient nodes in the tree for the composition can now be found to be all *right* child nodes of the nodes along the (left) search path of $\delta'^{r}_{i-1,i}$ and all the *left* child nodes of the nodes along the (right) search path of $\delta'^{r}_{j,j+1}$. Here, we denote the two children of a non-leaf node in the tree as being left or right, with the left child having the ready time function with lower indices. Let us denote the composition of the right children along the left search path by δ'^{L} and likewise the composition of the left children among the right search path by δ'^{R}. Now the required ready time function can be found by calculating the composition of these two parts: $\delta'^{r}_{ij} = (\delta'^{\text{R}} \circ \delta'^{\text{L}})$. It can be proven (see de Berg et al. (2008, p. 96–99)) that for each level in the tree, at most two nodes with the same level will be part of the required composition. In case two nodes of the same level are present in the composition, one node will be contained in δ'^{L} and the other must be in δ'^{R}. Furthermore, nodes in the composition δ'^{L} increase in level with larger indices while the nodes in the composition δ'^{R} decrease in level with larger indices. Therefore, given the interval $[i, j]$ of the required ready time function and its corresponding split node $\delta'^{r}_{\text{split}}$, the composition consists in worst-case of one node of *each* level below $\delta'^{r}_{\text{split}}$ in δ'^{L} and also one node of *each* level below $\delta'^{r}_{\text{split}}$ in δ'^{R}. There are $\bar{m}$ vertices between i and j, inclusive. Let $\bar{l}$ be highest level in the tree of which nodes can appear in the composition, i.e., one level below the split node. It can be seen that $\bar{l} \leq \lceil \log(\bar{m} + 2) \rceil - 1$. We have seen that the order of evaluating compositions is most efficient when iteratively selecting the composition involving the smallest functions. Therefore, the compositions in δ'^{L} are evaluated from the lowest level nodes up to level $\bar{l}$, in Figure 2.1 from left to right, and likewise the compositions in δ'^{R} are evaluated, in Figure 2.1 from right to left, and finally $\delta'^{r}_{ij} = (\delta'^{\text{R}} \circ \delta'^{\text{L}})$ is evaluated. The total number of operations required to calculate

${\delta'}^r_{ij}$ in the worst case using this procedure is given by:

$$\begin{aligned} & \mathcal{O}\left(2\sum_{l=1}^{\bar{l}-1}\sum_{k=0}^{l}2^k p + 2\sum_{k=0}^{\bar{l}-1}2^k p\right) \qquad (2.12)\\ = \; & \mathcal{O}\left(2\sum_{l=2}^{\bar{l}}\left[(2^l-1)p\right] + 2(2^{\bar{l}}-1)p\right)\\ = \; & \mathcal{O}\Big(2\left(2^{\bar{l}+1}-4-\bar{l}-1\right)p + 2\left(2^{\bar{l}}-1\right)p\Big)\\ = \; & \mathcal{O}\Big(3\cdot 2^{\bar{l}+1}p - \left(\bar{l}+12\right)p\Big)\\ = \; & \mathcal{O}(3\bar{m}p)\\ = \; & \mathcal{O}(\bar{m}p). \end{aligned}$$

In Equation (2.12), the double summation in the first term represents the worst-case number of operations needed to construct ${\delta'}^{\mathrm{L}}$ and is counted twice for also constructing ${\delta'}^{\mathrm{R}}$. The last term in Equation (2.12) corresponds to the worst-case number of operations needed to evaluate the composition ${\delta'}^r_{ij} = ({\delta'}^{\mathrm{R}} \circ {\delta'}^{\mathrm{L}})$. Since ${\delta'}^r_{ij}$ contains at most $\mathcal{O}(\bar{m}p)$ breakpoints, δ^r_{ij} can be obtained from ${\delta'}^r_{ij}$ in at most $\mathcal{O}(\bar{m}p)$ operations. Therefore, using the tree $\mathcal{T}^r$ in memory, δ^r_{ij} can be obtained in at most $\mathcal{O}(\bar{m}p)$ operations. □

In the special case of $p = 2$, which includes the case of classical non-time dependent travel times, we have seen that all forward, backward and partial ready time functions contain at most $p = 2$ breakpoints. Therefore, the ready time tree $\mathcal{T}^r$ of a route r with m customers consists of $\mathcal{O}(m)$ tree nodes (ready time functions) of at most two breakpoints. Thus, the tree can be stored using $\mathcal{O}(m)$ memory. Also, it can be shown that construction can be done in $\mathcal{O}(m)$ time and calculation of a partial ready time function visiting $\bar{m}$ customers using the tree can be done in $\mathcal{O}(\log \bar{m})$ time. This is lower than the $\mathcal{O}(\bar{m})$ operations required when calculating such partial ready time function for $p = 2$ from scratch.

2.5.4 Insertion Moves

Evaluating an insertion move of inserting customer j between $i-1$ and i in route r with m customers requires calculation of $\delta^r_{o,i-1}$ and δ^r_{id}. By Theorem 2.8, both ${\delta'}^r_{o,i-1}$ and ${\delta'}^r_{id}$ can be calculated in $\mathcal{O}(mp)$ operations using the ready time function tree $\mathcal{T}^r$ of r. Then $\delta^r_{o,i-1} = {\delta'}^r_{o,i-1}$ and $\delta^r_{id} = \left(\theta_i \circ {\delta'}^r_{id}\right)$, with the latter requiring

$\mathcal{O}(mp)$ operations. Now, the composition of the new o–d ready time function of Equation (2.6) is used, which again requires $\mathcal{O}(mp)$ time. In total, this method requires $\mathcal{O}(mp)$ time to evaluate an insertion move and therefore does not increase the overall complexity of the move evaluation compared to the method of Section 2.4.

2.5.5 Exchange Moves

Evaluating an exchange move which exchanges customers $i_1, \ldots, j_1$ of route r_1 with customers $i_2, \ldots, j_2$ of route r_2 requires the calculation of the following six ready time functions: $\delta^{r_1}_{o,i_1-1}$, $\delta^{r_1}_{i_1,j_1}$, $\delta^{r_1}_{j_1+1,d}$, $\delta^{r_2}_{o,i_2-1}$, $\delta^{r_2}_{i_2,j_2}$ and $\delta^{r_2}_{j_2+1,d}$. Suppose both routes have $\mathcal{O}(m)$ number of customers. By a similar argument to the insertion moves, all these ready time functions can be obtained in $\mathcal{O}(mp)$ time using the ready time function trees $\mathcal{T}^{r_1}$ and $\mathcal{T}^{r_2}$. Thus, the total complexity of evaluating an exchange move remains $\mathcal{O}(mp)$. However, this complexity does not rely on the storage of calculated middle segments for previous moves. Therefore, using the ready time function trees retains the total move evaluation complexity $\mathcal{O}(mp)$ even if the exchange neighborhood is searched in non-lexicographic order.

2.5.6 Updates and Memory

By Theorem 2.7, the tree $\mathcal{T}^r$ of a route r with m customers requires $\mathcal{O}(mp \log m)$ memory to store and an equal amount of operations to construct from scratch. Each time a move is executed which changes route r, we update the memory by reconstructing the corresponding tree $\mathcal{T}^r$ from scratch in $\mathcal{O}(mp \log m)$ operations. This is needed, because the efficiency of obtaining partial ready time functions from the tree relies heavily on the property of the tree being balanced, i.e., having maximum height of at most $\mathcal{O}(\log m)$. It is difficult to maintain this property after the number of customers of a route has changed, without re-building the tree from scratch. Although there exist so-called *self-balancing* binary tree data structures, which can re-balance themselves after an update, we are yet to discover such a structure that truly lowers the update complexity of $\mathcal{O}(mp \log m)$ in case a move changes the number of customers in a route.

2.5.7 Non-lexicographic and Lexicographic Neighborhood Search

We have seen that in case of exchange moves the ready time function trees can be used to efficiently evaluate moves without requiring a particular evaluation order. This opens up possibilities for efficient non-lexicographic neighborhood searches for our

problem, such as (parts of) the *Sequential Search* framework of Irnich (2008) for example or a simple Variable Neighborhood Search in which the exchange subsequence length k is increased dynamically during the search.

Moreover, also in the case of Lexicographic Search the use of ready time function trees potentially decreases the average computation times. Let us illustrate this in the setting of lexicographic k-exchange. Although we have seen in Section 2.4.4 that the particular evaluation order of moves allows us to update middle segment ready time functions between moves with minimal effort, in practice such updates between moves are usually done in a *lazy* fashion, meaning only when this is actually needed for evaluating the new move. Quick pre-checks typically conclude infeasibility or inferiority of the new move without the need of these ready time functions. Suppose multiple consecutive moves are deemed infeasible or inferior by pre-checks. Then no time consuming exact feasibility or cost calculations are necessary and no updates of ready time functions between moves are done. When subsequently a new move passes all pre-checks, the ready time functions have to be updated, which, using regular forward extension, require a quadratic number of operations in the number of previous infeasible moves. However, when using the ready time tree these updates require only a linear number of operations in the number of previous infeasible moves. Hence, the tree method utilizes the pre-checks much better.

2.6 Additional methods

In this section we present two other pre-calculation methods related to the Forward and Backward (F/B) method of Section 2.4 and the method of Ready time function Tree (TREE) presented in Section 2.5: the Ready time function Tree + Forward/Backward Hybrid (TREE+F/B) and the All in memory method (ALL).

2.6.1 Ready time function Tree + Forward/Backward Hybrid

The combination of the forward and backward ready time functions in memory with the ready time function trees in memory is particularly useful for searching advanced Neighborhoods such as k-exchange. The needed forward and backward ready time functions are pre-calculated in memory, while the ready time function trees can be used to update middle segment ready time function efficiently. The move evaluation complexity remains $\mathcal{O}(mp)$, with m the number of customers in the affected routes. Similar to using only the ready time trees, this move complexity is maintained even if the neighborhood is searched in non-lexicographic order. The memory requirement

and update complexity between iterations is $\mathcal{O}(m^2p)$ per route, due to the complexity of the Forward and Backward method.

2.6.2 All ready time functions in memory

Another method to ensure even quicker move evaluations for advanced Neighborhoods such as k-exchange is to keep *every* partial ready time function δ^r_{ij} in memory. For each move, forward/backward and middle segment ready time functions are all pre-calculated, so only some composition of these ready time functions needs to be calculated and minimized. This still requires $\mathcal{O}(mp)$ operations with m number of customers in the affected routes, but the worst-case number of operations is reduced by a constant factor compared to the above TREE + F/B hybrid method. The price is however an increase in memory and update operations needed to maintain all these partial ready time functions: $\mathcal{O}(m^3p)$ memory and update operations are now needed per route, which is a factor m higher than the Forward and Backward method.

2.7 Summary of the methods

We give a brief summary of the methods considered in this chapter by providing their computation time and memory complexities of insertion and k-exchange neighborhood search. Table 2.1 contains complexities of the following methods, with n the total number of customers and p the highest number of breakpoints among the arrival time functions.

- **non-TD**
  For comparison, we include the known complexity results of the efficient forward (backward) slack methods (Campbell and Savelsbergh, 2004; Savelsbergh, 1992) for the classical non-time dependent, duration constrained or minimized VRPTW. In case of non-lexicographic k-exchange neighborhood search, note that the 1-level Hierarchy data structure of Irnich (2008) can be used to lower the single move evaluation complexity to $\mathcal{O}(1)$ while increasing the update/memory complexity to $\mathcal{O}(n^{4/3})$. Higher level Hierarchies may decrease the update/memory complexity further: Irnich (2008) conjectured that an l-level Hierarchy has an update/memory complexity of $\mathcal{O}\left(n^{2^{l+1}/(2^{l+1}-1)}\right)$.

- **NAIVE**
  This methods re-calculates the complete ready time function $\delta^{\tilde{r}}_{od}$ from scratch

for each new move by iterative forward composition. Besides from the arrival time functions, no (other) ready time functions are stored in memory. To provide a fair comparison, this method is allowed to keep only the last calculated forward-, middle and backward segment ready time functions of the previous move in memory.

- **TREE**
  Fast insertion checks by storing a ready time function tree for each route. Forward, backward and partial ready time functions are calculated using the ready time function trees. Exchange moves can also be evaluated efficiently by using non-Lexicographic Search.

- **F/B**
  Faster insertion checks by storing forward and backward ready time functions. Exchange moves can only be evaluated efficiently by using Lexicographic Search, since the middle segment ready time functions are not in memory.

- **TREE+F/B**
  Even faster insertion checks by both storing a ready time function tree and forward and backward ready time functions for each route and calculating the needed partial ready time functions. Exchange moves can also be evaluated efficiently using non-Lexicographic Search.

- **ALL**
  Fastest insertion checks by storing all partial ready time functions. Exchange moves can also be efficiently evaluated in case of non-Lexicographic Search, but requires a higher memory and update time complexity.

The investigated methods, NAIVE, TREE, F/B, TREE+F/B and ALL, are presented in this order to illustrate their increasing amount of pre-calculation done (e.g., NAIVE doing no pre-calculations while ALL does the most), so increasing in expected move evaluation efficiency as well as needed update time and memory.

The table gives the complexities of the methods in case of *Insertion*, *k-Exchange* (Lexicographic) and *k-Exchange non-Lexicographic* Neighborhood Searches, for which we report the complexity of evaluating a single move, the total time for searching the entire neighborhood and the update time and memory complexity needed to update and store the data structures used by the methods between neighborhood searches. Furthermore, we included the total time needed by a cheapest insertion construction heuristic (*Full constr.*), which uses the Insertion Neighborhood iteratively to route

Table 2.1: Complexities of the investigated methods during insert and exchange neighborhood evaluations.

Neighborhood Operation		non-TD	NAIVE	TREE	F/B	TREE+F/B	ALL
Insert	single move	$\mathcal{O}(1)$	$\mathcal{O}\left(n^2p\right)$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$
	total	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(n^4p\right)$	$\boldsymbol{\mathcal{O}\left(n^3p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3p\right)}$
	update/mem.	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log n)}$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^2p\right)$
Full constr.	total	$\mathcal{O}\left(n^3\right)$	$\mathcal{O}\left(n^5p\right)$	$\boldsymbol{\mathcal{O}\left(n^4p\right)}$	$\boldsymbol{\mathcal{O}\left(n^4p\right)}$	$\boldsymbol{\mathcal{O}\left(n^4p\right)}$	$\boldsymbol{\mathcal{O}\left(n^4p\right)}$
k-Exchange	single move	$\mathcal{O}(1)$	$\mathcal{O}\left(n^2p\right)$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$
Lex.	total	$\mathcal{O}\left(n^2k^2\right)$	$\mathcal{O}\left(n^4k^2p\right)$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$
	update/mem.	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log n)}$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^3p\right)$
k-Exchange	single move	$\mathcal{O}(n)$	$\mathcal{O}\left(n^2p\right)$	$\boldsymbol{\mathcal{O}(np)}$	$\mathcal{O}\left(n^2p\right)$	$\boldsymbol{\mathcal{O}(np)}$	$\boldsymbol{\mathcal{O}(np)}$
Non-lex.	total	$\mathcal{O}\left(n^3k^2\right)$	$\mathcal{O}\left(n^4k^2p\right)$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$	$\mathcal{O}\left(n^4k^2p\right)$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$	$\boldsymbol{\mathcal{O}\left(n^3k^2p\right)}$
	update/mem.	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log n)}$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^2p\right)$	$\mathcal{O}\left(n^3p\right)$

all customers from scratch (using at most n iterations of insertion). In the table, we further assume that the number of customers m of any route is $\mathcal{O}(n)$. Complexities in bold indicate the lowest complexity among the investigated methods, where we disregard non-TD as it is not suited for the TDVRPTW.

2.7.1 Bounded customers per route and static move evaluations

An important feature of most large real-world routing problems is the fact the maximum number of customers in a route is implicitly bounded such that it does not scale with the total number of customers n, e.g., due to capacity constraints. Often this occurs due to limited vehicle capacity, but other constraints can also play a role. Table 2.2 shows the complexity results similar to Table 2.1, so with n the number of total customers and p the highest number of breakpoints among the arrival time functions, but now using the assumption that the maximum number of customers in a route is bounded by a constant M. The table also assumes the use of so-called *static move descriptors* (Zachariadis and Kiranoudis, 2010), which, in short, means that in some neighborhood search iteration except the first, only new moves concerning at least one route which was just changed need to be actively checked, provided the best moves concerning the other routes are kept in memory. Also, when executing a move, only the data structures of the affected routes need to be updated. This strategy is particularly efficient when the maximum number of customers in a route M is small compared to the total number of customers n. We use this strategy for all our computational experiments.

Table 2.2: Complexities of the investigated methods during insert and exchange neighborhood evaluations, when the maximum number of customers in a route is bounded by M and static move evaluations are used.

Neighborhood	Operation	non-TD	NAIVE	TREE	F/B	TREE+F/B	ALL
Insert	single move	$\mathcal{O}(1)$	$\mathcal{O}\left(M^2p\right)$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$
	total	$\mathcal{O}(Mn)$	$\mathcal{O}\left(M^3np\right)$	$\boldsymbol{\mathcal{O}\left(M^2np\right)}$	$\boldsymbol{\mathcal{O}\left(M^2np\right)}$	$\boldsymbol{\mathcal{O}\left(M^2np\right)}$	$\boldsymbol{\mathcal{O}\left(M^2np\right)}$
	update	$\mathcal{O}(M)$	–	$\boldsymbol{\mathcal{O}(Mp\log M)}$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^2p\right)$
	memory	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log M)}$	$\mathcal{O}(Mnp)$	$\mathcal{O}(Mnp)$	$\mathcal{O}(Mnp)$
Full constr.	total	$\mathcal{O}\left(Mn^2\right)$	$\mathcal{O}\left(M^3n^2p\right)$	$\boldsymbol{\mathcal{O}\left(M^2n^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2n^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2n^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2n^2p\right)}$
k-Exchange	single move	$\mathcal{O}(1)$	$\mathcal{O}\left(M^2p\right)$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$
Lex.	total	$\mathcal{O}\left(Mnk^2\right)$	$\mathcal{O}\left(M^3nk^2p\right)$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$
	update	$\mathcal{O}(M)$	–	$\boldsymbol{\mathcal{O}(Mp\log M)}$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^3p\right)$
	memory	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log M)}$	$\mathcal{O}(Mnp)$	$\mathcal{O}(Mnp)$	$\mathcal{O}\left(M^2np\right)$
k-Exchange	single move	$\mathcal{O}(M)$	$\mathcal{O}\left(M^2p\right)$	$\boldsymbol{\mathcal{O}(Mp)}$	$\mathcal{O}\left(M^2p\right)$	$\boldsymbol{\mathcal{O}(Mp)}$	$\boldsymbol{\mathcal{O}(Mp)}$
Non-lex.	total	$\mathcal{O}\left(M^2nk^2\right)$	$\mathcal{O}\left(M^3nk^2p\right)$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$	$\mathcal{O}\left(M^3nk^2p\right)$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$	$\boldsymbol{\mathcal{O}\left(M^2nk^2p\right)}$
	update	$\mathcal{O}(M)$	–	$\boldsymbol{\mathcal{O}(Mp\log M)}$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^2p\right)$	$\mathcal{O}\left(M^3p\right)$
	memory	$\mathcal{O}(n)$	–	$\boldsymbol{\mathcal{O}(np\log M)}$	$\mathcal{O}(Mnp)$	$\mathcal{O}(Mnp)$	$\mathcal{O}\left(M^2np\right)$

2.8 Computational Experiments

In this section, we present the results of numerical experiments in which we apply the presented methods on several benchmark instances. The methods are tested in a construction heuristic and a neighborhood search improvement heuristic, which are both implemented in C++11 and compiled using GCC version 6.3. All runs are executed as a single thread on an Intel® Xeon® E5-2650 v2 with 2.6 GHz (Turbo Boost up to 3.6 GHz) and 32 GB of RAM running Debian Linux version 9. All CPU times were measured using `std::chrono::high_resolution_clock`. Only one thread was run at any time on the CPU.

We test the speed-up methods in a parallel cheapest insertion construction heuristic and a k-exchange neighborhood search improvement heuristic. The pseudo-code of these algorithms can be found in the Appendix and next we give a short summary. Each iteration, the best feasible move is found and executed. In case of the construction heuristic, a move is an insertion of an unplanned customer into a route, while in case of the improvement method a move is a k-exchange. After the first iteration, the best move for each route–customer combination (route–route combination for the improvement heuristic) is kept in memory and only moves involving changed routes are re-calculated (static move descriptors). During move evaluation, we use the following pre-checks in this order: (1) capacity; (2) non-TD time window feasibilty; (3) cost lower bound. The capacity pre-check simply checks whether the vehicle capacity of the new route is violated. The non-TD time window feasibilty

uses the non-time dependent earliest and latest arrival times based on the highest speed of each arc to check time window feasibilty. The cost lower bound uses either exact new route distances, in case of distance only based objective, or estimates the new route duration by using only highest speed along each arc and service times, in case of duration only based objective, to check inferiority of the current move compared to the best move seen so far. The exact feasibility and cost of a move are only calculated when all pre-checks are passed.

2.8.1 Instances

We use the Gehring and Homberger (Gehring and Homberger, 2001) instances with 1000 customers for the VRPTW for our experiments. These instances are currently the largest commonly used VRPTW instances and can be found online on the VRP-REP: vehicle routing problem repository (Mendoza et al., 2014, `http://vrp-rep.org`). Each of these 60 instances are split into 6 groups of 10 instances: C1, RC1, R1, C2, RC2, R2. The first letter denotes the geographic spread of the customers, with C: clustered, RC: random-clustered, R: random. The number represents the instance type, with 1: short routes and tight time windows, 2: long routes and wide time windows.

Time-dependent travel times are added to these instances by means of the speed-profiles introduced by Figliozzi (2012). For each instance, the planning horizon $[0, T] = [0, b_d]$, with b_d the depot time window end time, is partitioned into a number of speed zones each with equal duration. Each speed zone has a constant speed factor which modifies the classical (nominal) travel times based on Euclidean distance. Table 2.3 shows the speed-profiles used in our computational experiments. For each speed-profile TDx, it shows the maximum number of breakpoints p, and for each of the zone end times $t \cdot b_d$ the speed factor. Speed-profile TD0 corresponds to the classical non-time dependent travel times and result in travel time functions with only $p = 2$ breakpoints. Speed-profiles TD1, TD2 and TD3 each result in travel time functions with at most $p = 10$ breakpoints and decrease the travel times on average over the whole planning horizon by 25%, 50% and 75% respectively. As in Figliozzi (2012), we use the same speed profile for all arcs. Zone start and end times were rounded to the nearest integer, while arrival time functions were calculated using 5 decimal precision to avoid numerical instabilities.

Table 2.3: Speed Profiles

		Zones $\cdot b_d$				
	p	$[0, 0.2]$	$[0.2, 0.4]$	$[0.4, 0.6]$	$[0.6, 0.8]$	$[0.8, 1]$
TD0	2	1.00	1.00	1.00	1.00	1.00
TD1	10	1.00	1.60	1.05	1.60	1.00
TD2	10	1.00	2.00	1.50	2.00	1.00
TD3	10	1.00	2.50	1.75	2.50	1.00

2.8.2 Insertion Experiments

The speed-up of using TREE and F/B over NAIVE during insertion move evaluations is tested using a parallel cheapest insertion construction heuristic which iteratively builds multiple routes from scratch (Potvin and Rousseau, 1993). Note that insertion moves are simple and only require forward and backward ready time functions. Therefore, methods TREE+F/B and ALL will not reduce computation times further and thus are not tested here.

Table 2.4 shows the results of the cheapest insertion construction heuristics on the Gehring and Homberger 1000 customer instances when using insertion costs based on distance only. The table shows the instance groups with the used speed profile. The column #routes shows the number of routes of the constructed solutions, respectively, averaged over the ten instances in a group. The columns #mov., #PCfeasmov. and #feasmov. show the total number of moves evaluated, number of moves that passed all pre-checks and number of moves that were found feasible by the exact move evaluation, respectively, again averaged over the instances. The Average CPU columns show the total CPU time needed for the construction heuristic in seconds, averaged over the ten instances. The column Speed-up over NAIVE shows the CPU time speed-up factor of using TREE and F/B over NAIVE, respectively, with speed-up factor 1.00 being equal in speed.

We see that the TREE and F/B methods speed-up the construction heuristic in every instance group and every speed-profile including the non-time dependent profile TD0. Most speed-up is gained on the time-dependent instances with long routes and large time windows (C2, RC2 and R2 instances), although the differences in speed-up between the speed-profiles TD1, TD2 and TD3 are minimal.

Table 2.5 shows the result of the same construction heuristic experiment but with route duration as objective. Again most speed-up is gained on time-dependent instances with long routes and large time windows, but the amount of speed-up is much larger compared to the distance insertion costs used in Table 2.4. Using distance costs, average speed-ups of up to 5.09 are observed, while using duration

Table 2.4: Construction heuristic results – distance costs – $n = 1000$ customers.

						Average CPU (s)			Speed-up o. NAIVE	
		#routes	#mov.	#PCfeasmov.	#feasmov.	NAIVE	TREE	F/B	TREE	F/B
TD0	C1	100.5	3,426,065.3	205,084.4	205,084.4	1.79	1.73	1.63	1.04	1.10
	RC1	95.5	3,588,281.3	246,653.6	246,653.6	1.84	1.75	1.63	1.06	1.13
	R1	95.4	3,595,665.4	322,749.4	322,749.4	1.94	1.82	1.67	1.07	1.16
	C2	33.9	8,956,573.0	371,908.8	371,908.8	2.76	2.08	1.80	1.33	1.54
	RC2	24.9	14,211,415.5	485,851.7	485,851.7	3.86	2.32	1.94	1.67	1.99
	R2	21.8	14,292,815.3	609,510.8	609,510.8	4.38	2.47	2.01	1.77	2.17
TD1	C1	99.2	3,451,070.5	276,156.8	241,109.7	2.40	2.14	1.83	1.12	1.31
	RC1	94.6	3,594,157.5	334,627.1	271,475.3	2.46	2.13	1.82	1.16	1.35
	R1	95.2	3,603,186.3	408,761.7	355,976.2	2.84	2.39	1.94	1.19	1.46
	C2	33.2	9,129,195.8	468,515.3	398,546.2	4.18	2.70	2.05	1.55	2.04
	RC2	24.7	14,300,147.0	690,603.4	510,306.1	7.81	3.58	2.41	2.18	3.24
	R2	21.6	14,264,599.8	843,368.2	668,166.0	9.24	4.09	2.56	2.26	3.61
TD2	C1	99.0	3,471,849.1	303,541.0	270,376.0	2.52	2.21	1.88	1.14	1.34
	RC1	94.7	3,595,981.6	370,811.0	283,539.2	2.53	2.18	1.84	1.16	1.37
	R1	95.0	3,601,883.0	445,726.9	380,396.7	2.93	2.47	1.98	1.19	1.48
	C2	32.4	9,213,043.1	523,403.2	428,760.9	4.73	2.90	2.13	1.63	2.22
	RC2	24.8	14,346,930.9	831,875.0	524,698.9	9.47	4.41	2.74	2.15	3.46
	R2	21.3	14,348,174.2	999,680.8	711,870.6	13.95	5.02	3.05	2.78	4.57
TD3	C1	98.3	3,486,147.5	337,896.2	293,191.5	2.58	2.27	1.90	1.14	1.35
	RC1	94.7	3,591,158.7	415,358.2	298,731.7	2.63	2.26	1.88	1.17	1.40
	R1	94.5	3,612,366.1	481,004.6	398,440.5	3.04	2.54	2.01	1.20	1.51
	C2	32.1	9,286,906.1	594,376.6	454,312.8	4.75	3.00	2.18	1.58	2.18
	RC2	24.7	14,314,163.0	1,015,197.1	537,624.1	17.34	5.90	3.41	2.94	5.09
	R2	21.3	14,334,770.9	1,142,693.3	758,262.8	8.86	4.92	2.85	1.80	3.10

costs average speed-ups of up to 8.89 are observed. Note that the cost LB pre-check is weaker in case of duration costs than for distance costs. Therefore, the fraction of moves passing the pre-checks in case of duration costs is higher and the heuristics must spend more time on the exact feasibilty and cost calculations which both the TREE and the F/B method speed up. In both tables, the F/B method outperforms the TREE method by up to 2 times and on average by 1.4. The TREE method needs to evaluate some compositions to construct the forward- and backward ready time functions, while in the F/B method these are readily available in memory. The increases in update times for both methods are much lower than the overall decreases in time needed for each iteration.

2.8.3 Lexicographic Exchange Experiments

The next experiments compare the average computation times of NAIVE, TREE, F/B, TREE+F/B and ALL methods in a k-exchange improvement heuristic. The heuristic is run on all Gehring and Homberger 1000 customer instances. A value of maximum subsequence length $k = 8$ is selected and the heuristic is run iteratively until the local optimum is reached. The heuristic starts with the solution obtained by the construction heuristic with distance costs as described in Section 2.8.2. We choose to start with these solutions over the ones constructed with duration costs,

Table 2.5: Construction heuristic results – duration costs – $n = 1000$ customers.

						Average CPU (s)			Speed-up o. NAIVE	
		#routes	#mov.	#PCfeasmov.	#feasmov.	NAIVE	TREE	F/B	TREE	F/B
TD0	C1	101.3	3,423,521.2	270,889.6	270,889.6	1.90	1.82	1.69	1.05	1.13
	RC1	106.0	3,533,413.7	307,888.9	307,888.9	2.01	1.86	1.72	1.08	1.17
	R1	103.9	3,553,110.7	477,005.9	477,005.9	2.26	2.05	1.83	1.10	1.24
	C2	35.1	8,947,100.9	540,249.8	540,249.8	3.18	2.29	1.94	1.39	1.64
	RC2	34.7	12,496,541.9	708,577.0	708,577.0	4.72	2.65	2.13	1.78	2.21
	R2	29.6	13,075,602.8	1,148,350.8	1,148,350.8	6.79	3.30	2.41	2.06	2.82
TD1	C1	102.9	3,451,578.7	407,555.8	355,700.2	2.82	2.70	2.24	1.04	1.26
	RC1	109.1	3,524,992.4	475,308.3	404,702.8	3.46	3.00	2.35	1.15	1.47
	R1	116.6	3,529,151.9	661,467.3	614,328.0	4.28	3.58	2.67	1.19	1.61
	C2	34.8	9,038,582.8	858,311.5	763,742.0	12.49	5.88	3.76	2.12	3.32
	RC2	36.6	12,200,695.9	1,208,980.3	1,051,252.1	49.15	13.71	6.34	3.59	7.76
	R2	35.5	12,493,647.4	1,488,960.8	1,411,922.9	74.81	17.10	8.54	4.38	8.76
TD2	C1	109.5	3,456,519.0	478,188.4	414,159.1	3.10	3.00	2.43	1.03	1.27
	RC1	116.1	3,517,931.8	508,909.9	405,680.0	3.62	3.07	2.43	1.18	1.49
	R1	122.0	3,537,531.0	667,778.4	612,971.7	4.56	3.71	2.78	1.23	1.64
	C2	35.2	9,107,428.3	1,094,623.0	978,532.8	13.94	6.51	4.12	2.14	3.38
	RC2	38.1	11,914,694.5	1,569,033.9	1,317,501.2	70.30	19.76	8.86	3.56	7.93
	R2	36.3	12,663,590.7	1,386,494.6	1,308,252.1	48.13	13.41	6.49	3.59	7.42
TD3	C1	114.6	3,452,527.3	552,267.4	465,432.2	3.39	3.21	2.51	1.05	1.35
	RC1	119.7	3,500,667.4	563,992.7	430,201.7	3.81	3.25	2.51	1.17	1.52
	R1	129.9	3,520,679.9	698,052.6	622,257.3	4.78	3.83	2.81	1.25	1.70
	C2	35.2	9,060,139.2	1,345,638.7	1,200,492.9	17.03	8.22	4.86	2.07	3.51
	RC2	40.0	11,909,922.2	1,658,617.5	1,317,798.6	72.03	19.96	8.92	3.61	8.07
	R2	37.3	12,605,706.6	1,538,136.5	1,444,501.2	72.90	16.17	8.20	4.51	8.89

because the latter tend to have a lot more and smaller routes which limits possibilities of exchanging large subsequences of customers.

Table 2.6 shows the results of these runs for the different speed-profiles. The column #It presents the number of iterations of the exchange heuristic used to obtain the local minimum, averaged over the group of ten instances. The columns #mov., #PCfeasmov. and #feasmov. again show the total number of moves evaluated, number of moves that passed all pre-checks and number of moves that were found feasible by the exact move evaluation, respectively, again averaged over the instances. The columns Average CPU show the CPU time in seconds needed to obtain the local optimum and the columns Speed-up over NAIVE give the factor CPU time was reduced compared to NAIVE, each averaged over the instance group. As with the construction experiments, most speed-up of F/B, TREE and ALL methods over NAIVE is gained on instances C2, RC2 and R2 with large number of customers in the routes. Although speed-ups of up to 2.54 times occur for the classical non-time dependent profile TD0, the methods are particularly able to speed-up the time-dependent speed-profiles, showing speeds-ups up to 3.61 with the F/B method, up to 3.94 with the TREE+F/B method and up to 5.48 times with the ALL method, although differences between TD1, TD2 and TD3 are minimal.

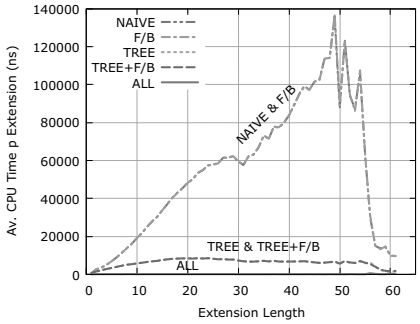
One cause of the speed-up of the TREE, TREE+F/B and ALL methods over

Table 2.6: Lexicographic 8-exchange results – $n = 1000$ customers.

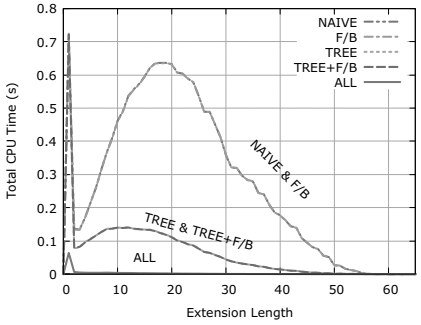
						Average CPU (s)					Speed-up over NAIVE			
		#It	#mov.	#PCfeasmov.	#feasmov.	NAIVE	TREE	F/B	TREE +F/B	ALL	TREE	F/B	TREE +F/B	ALL
TD0	C1	243.4	133,245,009.7	33,557.8	11,426.7	6.31	6.31	6.28	6.30	6.33	1.00	1.00	1.00	1.00
	RC1	488.8	309,583,214.7	307,664.2	123,642.0	13.38	13.14	12.96	12.93	12.87	1.02	1.03	1.03	1.04
	R1	418.6	265,539,145.5	618,157.8	174,454.5	12.05	11.65	11.39	11.27	11.08	1.03	1.06	1.07	1.09
	C2	142.8	403,395,382.2	296,601.1	18,116.4	22.19	21.78	21.60	21.58	21.82	1.02	1.03	1.03	1.02
	RC2	178.5	845,004,596.0	22,548,303.1	1,962,978.5	105.17	71.06	58.88	56.03	50.71	1.48	1.79	1.88	2.07
	R2	181.3	890,270,828.2	36,885,040.5	2,410,307.5	146.60	90.38	71.39	66.73	57.73	1.62	2.05	2.20	2.54
TD1	C1	265.2	151,428,561.3	1,100,200.1	150,226.1	12.14	11.33	10.34	9.87	8.97	1.07	1.17	1.23	1.35
	RC1	534.4	344,519,537.6	1,229,367.2	355,145.9	20.27	18.27	17.45	16.66	15.66	1.11	1.16	1.22	1.29
	R1	481.8	308,158,527.9	2,001,607.0	484,879.3	22.75	19.57	18.57	16.91	15.07	1.16	1.23	1.35	1.51
	C2	178.7	509,306,192.6	18,963,511.9	461,662.9	124.91	92.88	69.79	64.31	49.94	1.34	1.79	1.94	2.50
	RC2	205.4	987,288,907.5	48,963,091.0	4,822,667.3	929.13	386.68	257.51	235.82	170.16	2.40	3.61	3.94	5.46
	R2	218.1	1,089,008,156.0	57,897,750.6	3,173,019.5	594.36	353.03	222.63	191.10	115.50	1.68	2.67	3.11	5.15
TD2	C1	289.2	165,125,066.9	1,794,314.5	268,090.6	15.97	14.55	13.03	12.21	10.73	1.10	1.23	1.31	1.49
	RC1	542.7	354,288,954.3	1,879,741.2	534,412.2	24.57	21.61	20.27	19.02	17.38	1.14	1.21	1.29	1.41
	R1	574.2	367,765,351.5	2,903,340.6	857,239.0	31.10	26.47	24.32	21.99	19.19	1.17	1.28	1.41	1.62
	C2	188.3	563,514,893.2	33,443,587.0	609,260.5	211.88	146.74	106.99	97.05	70.36	1.44	1.98	2.18	3.01
	RC2	223.9	1,105,241,530.7	46,213,980.3	5,742,931.5	513.78	348.56	201.33	178.53	116.17	1.47	2.55	2.88	4.42
	R2	259.2	1,314,357,671.3	83,270,404.5	7,135,950.5	916.22	515.75	318.28	276.55	170.12	1.78	2.88	3.31	5.39
TD3	C1	298.5	174,860,286.8	2,515,959.7	346,823.2	19.74	17.65	15.63	14.35	12.17	1.12	1.26	1.38	1.62
	RC1	590.0	383,723,779.3	2,751,181.5	810,596.5	29.75	25.84	23.72	22.05	19.73	1.15	1.25	1.35	1.51
	R1	603.0	393,329,002.4	3,526,909.0	1,061,401.1	36.04	30.06	27.52	24.61	21.10	1.20	1.31	1.46	1.71
	C2	184.1	562,132,609.0	52,675,904.2	917,470.8	312.82	219.27	154.38	138.38	93.75	1.43	2.03	2.26	3.34
	RC2	230.9	1,121,656,354.6	57,124,211.0	6,800,568.9	498.24	395.30	215.88	191.79	120.50	1.26	2.31	2.60	4.13
	R2	280.0	1,437,934,819.2	116,147,438.8	9,984,865.4	1,213.36	763.68	445.26	383.56	221.46	1.59	2.73	3.16	5.48

NAIVE and F/B during lexicographic exchange is the decreased CPU time needed to update the middle segment ready time functions with a number of new customers. This happens in Lexicographic Search when between two feasible moves a number of moves are found to be infeasible by the pre-checks and the middle segment ready time function is updated in a lazy fashion. To illustrate the differences in middle segment extension times, Figure 2.3(a) shows the average CPU times needed in nanoseconds per middle segment ready time function extension by a certain number of customers during the first iteration of M-exchange on instance R2_10_04 with speed-profile TD3. Also the total CPU times in seconds spent on middle segment ready time function extension of a certain length during the iteration and the cumulative total CPU times are shown in Figure 2.3(b) and Figure 2.3(c), respectively. Recall that both NAIVE and F/B methods extend middle segment ready time functions by successive forward composition, resulting in a quadratic time complexity in number of extension customers (see Lemma 2.5), while the TREE and TREE+F/B methods only need linear time (see Theorem 2.8) and ALL no time at all. Figure 2.3(a) shows these differences empirically, especially for the lower extension lengths. The TREE method even seems to perform better on average than its worse-case predicted linear time. For large extension lengths, the average CPU times for most methods become noisy due to the low number of such large extensions observed in our experiment.

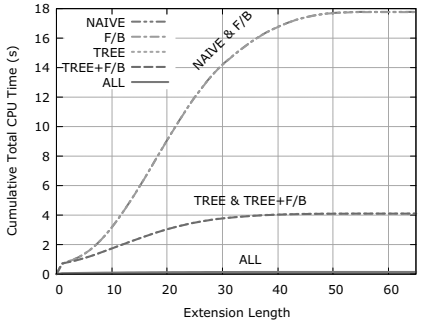
The results of the segment extension measurements as illustrated in Figure 2.3 suggest that the speed-up of TREE compared to F/B will increase as the maximum segment length k increases. Table 2.7 shows the results of the k-exchange runs with



(a) Average CPU time per extension (ns).



(b) Total extension time (s).



(c) Cumulative total extension time (s).

Figure 2.3: Middle segment extension CPU times during the first iteration of M-exchange on instance TD3_R2_10_04.

maximum subsequence lengths $k = 2, 4, 8, 16, 32, M$. Here, only the long-route instance groups C2, RC2, and R2 are shown, because the routes of the other groups are too short, on average 10 customers, to apply higher maximum exchange subsequence lengths. Notice that in this table the speed-up is given compared to F/B to illustrate the additional benefits of TREE+F/B and ALL over F/B when subsequence lengths k are large. For large k, TREE+F/B is able to offer an additional speed-up of up to 1.56 on top of F/B, while ALL offers a speed-up of up to 2.83 over F/B (1.87 over TREE+F/B) at the cost of more memory needed.

2.8.4 Non-lexicographic Exchange Experiments

We have seen that the addition of ready time function trees in the TREE+F/B method compared to only forward/backward ready time functions in the F/B method give a speed-up during lexicographic exchange. To show the benefits of the ready

Table 2.7: Lexicographic k-exchanges results – $n = 1000$ customers – Speed-profile TD3.

						Average CPU (s)			Sp.-up o. F/B	
		#It	#mov.	#PCfeasmov.	#feasmov.	F/B	TREE +F/B	ALL	TREE +F/B	ALL
$k = 2$	C2	147.4	34,413,792.5	4,556,501.6	131,673.7	7.93	7.81	6.53	1.01	1.21
	RC2	160.6	56,255,515.9	4,302,823.9	813,716.1	12.85	12.79	11.72	1.00	1.10
	R2	211.2	77,322,050.0	8,939,513.8	1,086,952.8	17.55	17.45	14.77	1.01	1.19
$k = 4$	C2	167.9	147,397,603.0	17,235,212.8	417,412.0	34.56	32.90	24.15	1.05	1.43
	RC2	194.5	263,205,979.9	16,446,865.9	2,839,082.7	50.27	48.39	37.62	1.04	1.34
	R2	258.3	360,998,852.4	34,209,933.8	4,668,740.8	93.75	88.81	65.54	1.06	1.43
$k = 8$	C2	184.1	562,132,609.0	52,675,904.2	917,470.4	154.38	138.38	93.75	1.12	1.65
	RC2	230.9	1,121,656,354.6	57,124,211.0	6,800,568.9	215.88	191.79	120.50	1.13	1.79
	R2	280.0	1,437,934,819.2	116,147,438.8	9,984,865.4	445.26	383.56	221.46	1.16	2.01
$k = 16$	C2	220.9	1,962,306,197.3	126,298,069.5	1,914,591.0	315.06	257.69	173.43	1.22	1.82
	RC2	256.1	4,025,788,272.9	203,148,876.0	24,781,583.0	801.11	704.17	469.89	1.14	1.70
	R2	306.7	5,312,619,679.9	269,700,418.7	10,710,454.3	906.12	681.79	346.35	1.33	2.62
$k = 32$	C2	218.2	3,528,300,535.3	171,578,568.4	2,418,781.9	486.21	355.27	243.36	1.37	2.00
	RC2	243.5	9,438,707,760.0	447,270,487.9	46,460,316.6	1,588.52	1,309.84	872.85	1.21	1.82
	R2	284.2	12,481,222,179.8	525,720,866.4	26,408,131.0	2,643.13	1,721.28	924.76	1.54	2.86
$k = M$	C2	210.2	3,447,702,250.0	174,267,323.3	2,159,624.5	475.70	345.98	234.16	1.37	2.03
	RC2	273.4	14,696,107,214.8	484,354,526.4	48,907,276.7	1,740.77	1,340.06	876.40	1.30	1.99
	R2	276.3	17,005,163,425.9	640,998,884.0	32,736,498.9	2,599.30	1,670.21	918.00	1.56	2.83

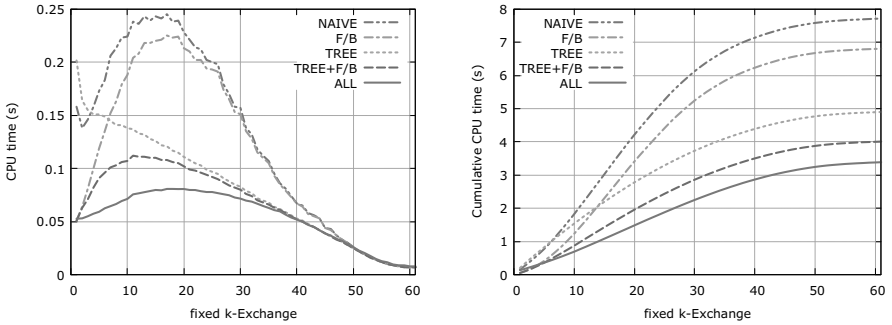
time function tree in non-lexicographic neighborhood search, we measure CPU times of executing single iterations of a fixed k-exchange neighborhood. In such a neighborhood, only exchanges of two subsequences with exactly k customers are evaluated. Notice that such neighborhoods cannot be searched lexicographically (or do not benefit from this).

Table 2.8 shows the total computation times, averaged over the instance groups, needed to do a single iteration of fixed k-exchanges from $k = 1$ up to $k = M$ and the corresponding speed-up over the NAIVE method. Only the long-route large-TW instance groups, C2, RC2 and R2, are shown, because the other instances had too short routes. Both the TREE and TREE+F/B perform better than the F/B method. This illustrates the improved non-lexicographic move complexity of the TREE-based methods over the F/B method.

Table 2.8: Non-lexicographic fixed k-exchanges results – first iteration – $n = 1000$ customers.

		Average CPU (s)					Speed-up over NAIVE			
		NAIVE	TREE	F/B	TREE +F/B	ALL	TREE	F/B	TREE +F/B	ALL
TD3	C2	9.80	5.79	8.10	4.54	3.30	1.69	1.21	2.16	2.97
	RC2	5.61	4.19	4.87	3.73	3.48	1.34	1.15	1.51	1.61
	R2	6.41	4.91	5.58	4.49	4.34	1.30	1.15	1.43	1.48

Figure 2.4(a) shows the CPU time for running one iteration of fixed k-exchange



(a) Total CPU time (s). (b) Cumulative total CPU time (s).

Figure 2.4: Non-lexicographic fixed k-exchange results – CPU-times in seconds for the first iteration of instance TD3 R2_10_04 for each fixed subsequence length k.

for varying k in the construction heuristic solution of instance R2_10_04 with speed-profile TD3. Also the cumulative total CPU times are shown in Figure 2.4(b). The figures show that the total CPU times of TREE and TREE+F/B follow those of ALL rather than following F/B when increasing k. Although the F/B method provides a high speed-up for low k, the TREE-based methods provide a high speed-up for high subsequence length k. This is in line with the non-lexicographic move complexities presented in Tables 2.1 and 2.2. The figures also illustrate the benefits of combining the TREE and F/B methods into TREE+F/B to decrease computation times.

2.8.5 Memory usage

The investigated pre-calculation methods TREE, F/B, TREE+F/B and ALL have each shown to speed-up insertion and exchange neighborhood search running times over NAIVE. However, this speed-up comes at a cost of increased memory usage, as shown in worst-case complexities in Tables 2.1 and 2.2. To verify this empirically, we measured the additional memory used by each method during the lexicographic 8-exchange runs reported in Table 2.6. Table 2.9 shows the maximum memory, in number of breakpoints, that was used by each method during these 8-exchange runs, averaged over the ten instances per instance group. In this table, we only show results of the non-time dependent speed-profile TD0 and speed-profile TD3. While the TREE and F/B methods seem to require an equal amount of memory and TREE+F/B twice that amount, the ALL method needs up to 10 times the amount of memory used by TREE+F/B and up to 20 times the amount used by

Table 2.9: Additional memory usage during 8-exchange.

		Additional Memory Needed (#BreakPoints)				
		NAIVE	TREE	F/B	TREE+F/B	ALL
TD0	C1	–	4,073.0	5,831.4	9,895.1	13,393.9
	RC1	–	3,993.2	5,202.7	9,176.3	12,116.4
	R1	–	3,869.4	4,737.8	8,596.5	10,647.6
	C2	–	4,266.2	5,685.6	9,946.8	32,592.0
	RC2	–	4,176.1	4,667.6	8,836.4	37,308.0
	R2	–	4,155.7	4,543.5	8,689.4	36,737.4
TD3	C1	–	9,589.1	11,361.3	20,886.0	29,000.3
	RC1	–	8,277.3	8,483.4	16,705.4	23,448.5
	R1	–	10,152.9	8,952.0	18,988.7	26,720.4
	C2	–	9,745.3	10,270.0	19,917.6	76,340.1
	RC2	–	8,752.9	9,292.9	18,001.3	126,385.9
	R2	–	12,004.9	10,340.9	21,741.7	207,990.5
TD3 usa13509		–	273,460.0	1,349,116.0	1,622,576.0	61,350,064.0

either TREE and F/B in case of speed-profile TD3. Note that the speed-up of ALL on the 1000 customer instances with $k = 8$ over the TREE+F/B method is up to 1.75 times and for $k = M$ up to 1.86 times (see Table 2.7), while the required memory is much more. Furthermore, to empirically illustrate the memory complexities presented in Tables 2.1 and 2.2, we include the memory usage for a much larger instance, the TD3 usa13509 instance. We created this instance since by our knowlegde no VRPTW instances with more than 1000 customers are commonly used currently in the literature. This instance has $n = 13508$ customers based on the well-known TSP-LIB (Reinelt, 1991) instance *usa13509*, using additionally vertex 1 as depot, 150 vehicles with capacity $Q = 250$, time horizon $b_d = 1000000$, customers each require 1 quantity and have a time window either $[0, b_d/2]$, $[b_d/4, 3b_d/4]$, or $[b_d/2, b_d]$, and using speed-profile TD3. On this instance the 8-exchange is run for one iteration after the cheapest insertion construction heuristic. The lower complexity of TREE compared to F/B can be seen empirically as the latter requires almost 5 times more memory. The ALL method requires almost 38 times more memory than the TREE+F/B method. In all experiments, as indicated by the complexities, the memory requirement of ALL grows much faster than the other methods. This would make ALL an unsuitable method when memory is limited, for instance when using fast CPU caches or parallel processes on GPUs.

2.9 Other applications

In this section, we briefly illustrate the general applicability of the presented methods by considering two other applications of the presented speed-up methods.

2.9.1 Multiple Time Windows

The presented methods in this chapter can easily be adapted to solve the TDVPRTW with route duration constraints and objective and with additionally that customers have multiple time windows. For this, only the time window ready time functions of Definition 2.1 need to be modified. For a customer $i \in \mathcal{V}^{\mathrm{C}}$ with a number of w time windows $\left[a_i^1, b_i^1\right], \left[a_i^2, b_i^2\right], \ldots, [a_i^w, b_i^w]$, the time window ready time function θ_i is given by:

$$\theta_i(t) := \begin{cases} a_i^1 + s_i & \text{if } t < a_i^1, \\ t_i + s_i & \text{if } t \in \left[a_i^1, b_i^1\right], \\ a_i^2 + s_i & \text{if } b_i^1 < t < a_i^2, \\ t_i + s_i & \text{if } t \in \left[a_i^2, b_i^2\right], \\ \vdots & \vdots \\ a_i^m + s_i & \text{if } b_i^{w-1} < t < a_i^w, \\ t_i + s_i & \text{if } t \in [a_i^w, b_i^w]. \end{cases} \tag{2.13}$$

This function has $\mathcal{O}(w)$ number of breakpoints. Figure 2.5 shows an example of a time window ready time function in case of a customer with $w = 2$ time windows. This multiple time window ready time function has very similar properties as the arrival time functions. Note that this function is not continuous but lower semi-continuous, however function composition can be shown to preserve semi-lower continuity. Therefore, all of the presented complexity results in Tables 2.1 and 2.2 only the p changes to $p + w$ or, equivalently, to $\max\{p, w\}$.

2.9.2 Pre-checks

Although the presented speed-up methods enable fast move evaluations for our TDVRPTW, they can also be used as pre-checks for move evaluations of more difficult routing problems. For instance, consider the time-dependent vehicle routing problem with driver legislations (see for instance Kok et al. (2010)), which includes additional constraints such as driving- and working time breaks a driver needs to take during his shift. Since our problem with route duration constraints is a relaxation of this

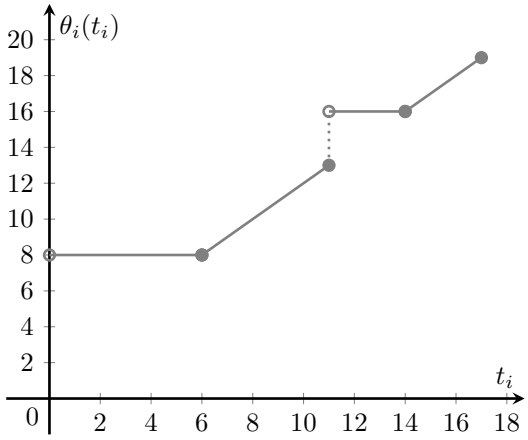


Figure 2.5: An example of a multiple time window ready time function $\theta_i(t_i)$ for TWs $[a_i^1, b_i^1] = [6, 11]$, $[a_i^2, b_i^2] = [14, 17]$ and $s_i = 2$.

problem, the presented methods can serve as pre-check before conducting the more time-consuming exact feasibility check. Notice that pre-checks based on our TD-VRPTW are generally stronger for the more difficult problem than pre-checks based on the classical vehicle routing problem with duration constraints, which can result in speed-ups.

2.10 Conclusion

In this chapter, we investigate the time-dependent vehicle routing problem with time windows, route duration constraints and duration minimization, and propose methods to speed-up common Neighborhood Search based heuristics by decreasing move evaluation CPU times. The inclusion of both time-dependent travel times and route duration constraints and objective are important to model real-world problem features such as road congestion and driver's maximum working shift duration, but increase the neighborhood move evaluation complexity.

The analysis of ready time functions leads to promising preprocessing methods to increase the efficiency of move evaluations. We discussed the F/B method which stores forward and backward ready time functions in memory to reduce the move evaluation complexity from quadratic in number of customers to linear. Empirical results on 1000 customer benchmark instances show speed-ups of up to 8.89 times during construction and up to 3.61 times during exchange with limited subsequence

length. Speed-ups increase significantly as the number of customers in a route or as the max exchange subsequence length increase.

To decrease the exchange move evaluation times further, we developed a new tree-based data structure of ready time functions. It allows move evaluation complexity to remain linear even when the search is conducted using a non-lexicographical order. We presented a method using only the tree-based data structure, TREE, and a combined method, TREE+F/B. Empirically, benefits are also observed in Lexicographic Searches, where speed-ups of the TREE+F/B method of up to 1.56 are achieved on top of the F/B method. This speed-up is attained without increasing the order of memory required. The most speed-up was observed using a method storing all partial ready time functions, ALL, up to 1.87 on top of the TREE+F/B method, while the required memory increased cubically.

Finally, we presented two other applications of the speed-up methods including Multiple Time Windows. These illustrate the general applicability of the investigated speed-up techniques.

We have seen that the tree-based data structure shares the goals of fast feasibility checks and low update/memory complexities, even for non-lexicographic searches, with the class of l-level Hierarchy data structures proposed by Irnich (2008) for non-time dependent routing problems. The analysis of the l-level Hierarchy data structures by Irnich (2008) relies on segment concatenations which can be done in constant time. In our case, the number of operations required for ready time function concatenation depends on the number of breakpoints, but can be bounded by $\mathcal{O}(np)$. This analysis results in fast $\mathcal{O}(np)$ feasibility checks, while update/memory requires $\mathcal{O}\big(n^{7/3}p\big)$ in case of 1-level Hierarchy (conjectured $\mathcal{O}\Big(n^{1+(2^{l+1})/(2^{l+1}-1)}p\Big)$ for l-level Hierarchy). These update/memory complexities are still higher than the $\mathcal{O}(np\log n)$ of the tree-based data structure. However, such analysis does not include the fact that the order in which ready time functions are concatenated influences the complexity, and that the number of breakpoints is not fixed: smaller segments require less memory and operations to update. Therefore, as a future research direction, we believe it is interesting to see what the exact complexities of the l-level Hierarchy for ready time functions are and how these data structures perform empirically compared to our presented methods.

Appendix

We outline the parallel cheapest insertion heuristic in Algorithm 2.1 and the lexicographic k-exchange improvement heuristic in Algorithm 2.2, which are used in the computational experiments of Section 2.8.

2.A Algorithm 2.1: Parallel Cheapest Insertion

Algorithm 2.1 Parallel Cheapest Insertion Construction Heuristic

Input: set of unplanned customers $\mathcal{U}$
Output: feasible solution S

1: $S \leftarrow$ empty solution
2: $\mathcal{M}_1, \ldots, \mathcal{M}_R \leftarrow$ empty move
3: $c_1^*, \ldots, c_R^* \leftarrow 0$
4: $\tilde{\mathcal{R}} \leftarrow \{1, \ldots, R\}$
5: $\mathcal{D} \leftarrow$ InitializeReadyTimeFunctions(S)
6: $\boldsymbol{\delta} \leftarrow \{\}$
7: **while** $|\mathcal{U}| \geq 1$ **do**
8: **for each** customer $v \in \mathcal{U}$ **do**
9: **for each** route $r \in \tilde{\mathcal{R}}$ **do**
10: **if** PreCheckFeasibleInsertInRoute(v, r, S) **then**
11: **for each** position p in route r **do**
12: **if** PreCheckFeasibleInsert(v, r, p, S) **then**
13: **if** PreCheckInsertCostLB(v, r, p, S) $< c_r^*$ **then**
14: $\boldsymbol{\delta} = \{\delta_\mathrm{F}, \delta_\mathrm{B}\} \leftarrow$GetReadyTimeFunctions($r$, p, $\mathcal{D}$, $\boldsymbol{\delta}$)
15: **if** IsFeasibleInsert(v, r, p, $\boldsymbol{\delta}$, S) **then**
16: $c \leftarrow$ CalcInsertCost(v, r, p, $\boldsymbol{\delta}$, S)
17: **if** $c < c_r^*$ **then**
18: $c_r^* \leftarrow c$, $\mathcal{M}_r \leftarrow (v, r, p)$, $v_r^* \leftarrow v$
19: $\mathcal{M}^* \leftarrow$ empty move, $c^* \leftarrow 0$
20: **for each** route $r \in \{1, \ldots, R\}$ **do**
21: **if** $c_r^* < c^*$ **then**
22: $c^* \leftarrow c_r^*$, $\mathcal{M}^* \leftarrow \mathcal{M}_r$, $v^* \leftarrow v_r^*$, $r^* \leftarrow r$
23: **if** $c^* < 0$ **then**
24: $S \leftarrow$ ExecuteMoveOnSolution($\mathcal{M}^*$, S)
25: $\mathcal{D} \leftarrow$UpdateReadyTimeFunctions($\mathcal{M}^*$, S, $\mathcal{D}$)
26: $\mathcal{U} \leftarrow \mathcal{U} \setminus \{v^*\}$
27: $\tilde{\mathcal{R}} \leftarrow \{r^*\}$
28: $c_{r^*}^* \leftarrow 0$
29: **else**
30: **break**
31: **return**

Lines 2–4: The algorithm keeps the best move for each route r in memory. Therefore, for each route r, the best move in memory $\mathcal{M}_r$ is initialized as an empty move

and the corresponding best move cost c_r^* is initialized to zero. The set of routes to check $\tilde{\mathcal{R}}$ is initialized to all routes. These best moves per route, their cost and the set of routes to check, are part of the Static Move Descriptors (see Section 2.7.1). Line 5: The set of ready time functions in memory $\mathcal{D}$ is initialized by InitializeReadyTimeFunctions. Depending on the preprocessing method used (see Section 2.7), this set $\mathcal{D}$ contains: (i) nothing in case of NAIVE (Section 2.7), (ii) all δ_{ij}^r in each tree $\mathcal{T}^r$ in case of TREE (Section 2.5), or (iii) all δ_{oi}^r (forward) and δ_{id}^r (backward) for each vertex i along each route r in case of F/B (Section 2.4). We have implemented the set of ready time functions $\mathcal{D}$ as continuous arrays of ready time functions for each route, even for each tree $\mathcal{T}^r$ of ready time functions. In the latter case, binary operators can be efficiently used to determine parent and child node indices in such continuous array (as needed for obtaining partial ready time functions, see Section 2.5.3). Lines 8–9: All possible insertion moves are checked concerning the routes in the set of routes to check $\tilde{\mathcal{R}}$. In the first iteration this set contains all routes. Line 10: PreCheckFeasibleInsertInRoute checks if insertion of customer v in route r does not violate the capacity restriction. Line 11: Moves are checked in lexicographic order by gradually increasing position p in each route. Line 12: PreCheckFeasibleInsert checks if insertion of customer v at this position p does not violate the time window restrictions, using non-time dependent earliest and latest arrival times based on the (non-time dependent) highest speed of each arc (see Section 2.8). Line 13: A lower bound on the cost is calculated by PreCheckInsertCostLB to check if the move can possibly improve the current best move $\mathcal{M}_r$ in memory. This cost lower bound is based on exact new route distances or estimated using the (non-time dependent) highest speed of each arc and service times (see Section 2.8). Line 14: To prepare the exact feasibility check and cost calculation, the necessary ready time functions are obtained in GetReadyTimeFunctions using memory $\mathcal{D}$, calculated from scratch or calculated using the previously obtained ready time functions, contained in set $\boldsymbol{\delta}$ to shorten notation. In case of NAIVE, the forward ready time function is obtained by extending the previous forward ready time function stored in $\boldsymbol{\delta}$, while the backward ready time function must be calculated from scratch. In case of TREE, the forward and backward ready time functions are obtained using the tree in memory $\mathcal{D}$ (see Section 2.5.4) or, in case of F/B, they are directly obtained from the memory $\mathcal{D}$ (see Section 2.4.1). Lines 15–16: Using the ready time functions stored in $\boldsymbol{\delta}$, exact feasibility of the move is checked by IsFeasibleInsert and its cost is calculated by CalcInsertCost, using Equation (2.6) and then Equation (2.3). Lines 17–18: If the current move is feasible and improves the current best move $\mathcal{M}_r$ of route r

in memory, then the current move replaces the best move in memory. Lines 20–22: After all possible moves are checked, the overall best move $\mathcal{M}^*$ is found among the best move of each route. Lines 23–30: If the overall best move $\mathcal{M}^*$ improves the solution, i.e., $c^* < 0$, this move is executed on the current feasible solution S with ExecuteMoveOnSolution and the set of ready time functions in memory $\mathcal{D}$ is updated with UpdateReadyTimeFunctions (see Sections 2.4.3 and 2.5.6). The set of routes to check next, $\tilde{\mathcal{R}}$, is updated to contain only the route that was changed by the executed move and best move cost $c^*_{r^*}$ of route r^* is reset to zero. The algorithm now repeats to check all new moves. Note that now only moves that affect the changed route in $\tilde{\mathcal{R}}$ need to be checked, since all best moves of the non-changed routes are already stored in memory (Static Move Descriptors, see Section 2.7.1). Lines 30 and 31: The algorithm stops when no improving insertion move can be found or when all customers have been inserted in the solution.

2.B Algorithm 2.2: Lexicographic k-Exchange

The lexicographic k-exchange improvement heuristic in Algorithm 2.2 is similar to Algorithm 2.1. We highlight the main differences. Lines 1–3: For Static Move Descriptors, the algorithm keeps the best move for each route pair (r_1, r_2) in memory. Therefore, the best moves per route pair $\mathcal{M}_{r_1 r_2}$ are initialized as empty moves and their best cost $c^*_{r_1 r_2}$ are initialized to zero. Line 4: The set of ready time functions in memory $\mathcal{D}$ in case of NAIVE, TREE and F/B contains ready time functions as in Algorithm 2.1 line 5 cases (i–iii) above, while in the additional cases contains: (iv) all δ^r_{ij} in each tree $\mathcal{T}^r$ and δ^r_{oi} and δ^r_{id} for each vertex i along each route r in case of TREE+F/B (Section 2.6.1), or (v) all δ^r_{ij} for all vertices i, j along each route r in case of ALL (Section 2.6.2). Lines 7–11: All possible k-exchange moves are checked which affect at least one route in the set of routes to check $\tilde{\mathcal{R}}$ (in the first iteration this set contains all routes). Moves are checked in lexicographic order by gradually extending the middle segments, i.e., by gradually increasing the position of customers j_1 and j_2. Line 12: PreCheckFeasibleExchange checks if the capacities of the routes modified by the k-exchange move are not violated and if the time window restrictions are not violated, similarly to the pre-checks used in Algorithm 2.1. Line 14: The necessary forward, middle and backward ready time functions of both routes affected by the move are obtained in GetReadyTimeFunctions using the memory $\mathcal{D}$, calculated from scratch or calculated using the previously obtained ready time functions (in set $\boldsymbol{\delta}$). In case of NAIVE, the forward and middle segments are ob-

Algorithm 2.2 Lexicographic k-Exchange Improvement Heuristic

Input: feasible solution S, max. segment length k
Output: feasible improved solution S (local optimum)

$\mathcal{M}_{1,2}, \mathcal{M}_{1,3}, \ldots, \mathcal{M}_{R-1,R} \leftarrow$ empty move
$c^*_{1,2}, c^*_{1,3} \ldots, c^*_{R-1,R} \leftarrow 0$
$\tilde{\mathcal{R}} \leftarrow \{1, \ldots, R\}$
$\mathcal{D} \leftarrow$InitializeReadyTimeFunctions(S)
$\boldsymbol{\delta} \leftarrow \{\}$
repeat
  for each routes $r_1, r_2 \in \{1, \ldots, R\}$ with $r_2 > r_1$ and $r_1 \in \tilde{\mathcal{R}}$ or $r_2 \in \tilde{\mathcal{R}}$ **do**
    for each customer i_1 in route r_1 **do**
      for each customer j_1 in route r_1 with $\text{Pos}(i_1) \leq \text{Pos}(j_1) \leq \text{Pos}(i_1) + k - 1$ **do**
        for each customer i_2 in route r_2 **do**
          for each customer j_2 in route r_2 with $\text{Pos}(i_2) \leq \text{Pos}(j_2) \leq \text{Pos}(i_2) + k - 1$ **do**
            if PreChecksFeasibleExchange(i_1, j_1, i_2, j_2, S) **then**
              if PreCheckExchangeCostLB(i_1, j_1, i_2, j_2, S) $< c^*_{r_1 r_2}$ **then**
                $\boldsymbol{\delta} = \{\delta^{r_1}_{\text{F}}, \delta^{r_1}_{\text{M}}, \delta^{r_1}_{\text{B}}, \delta^{r_2}_{\text{F}}, \delta^{r_2}_{\text{M}}, \delta^{r_2}_{\text{B}}\} \leftarrow$GetReadyTimeFunctions($i_1$, j_1, i_2, j_2, $\mathcal{D}$, $\boldsymbol{\delta}$)
                if IsFeasibleExchange(i_1, j_1, i_2, j_2, $\boldsymbol{\delta}$, S) **then**
                  $c \leftarrow$ CalcExchangeCost(i_1, j_1, i_2, j_2, $\boldsymbol{\delta}$, S)
                  if $c < c^*_{r_1 r_2}$ **then**
                    $c^*_{r_1 r_2} \leftarrow c$, $\mathcal{M}_{r_1 r_2} \leftarrow (i_1, j_1, i_2, j_2)$
  $\mathcal{M}^* \leftarrow$ empty move, $c^* \leftarrow 0$
  for each routes $r_1, r_2 \in \{1, \ldots, R\}$ with $r_2 > r_1$ **do**
    if $c^*_{r_1 r_2} < c^*$ **then**
      $c^* \leftarrow c^*_{r_1 r_2}$, $\mathcal{M}^* \leftarrow \mathcal{M}_{r_1 r_2}$, $r^*_1 \leftarrow r_1$, $r^*_2 \leftarrow r_2$
  if $c^* < 0$ **then**
    $S \leftarrow$ ExecuteMoveOnSolution($\mathcal{M}^*$, S)
    $\mathcal{D} \leftarrow$UpdateReadyTimeFunctions($\mathcal{M}^*$, S, $\mathcal{D}$)
    $\tilde{\mathcal{R}} \leftarrow \{r^*_1, r^*_2\}$
    $c^*_{r_1 r_2} \leftarrow 0 \; \forall r_1, r_2 \in \{1, \ldots, R\}$ with $r_1 < r_2$ and $r_1 \in \tilde{\mathcal{R}}$ or $r_2 \in \tilde{\mathcal{R}}$
until $c^* = 0$
return

tained by extending the previously obtained ready time functions stored in set $\boldsymbol{\delta}$, while the backward ready time functions are calculated from scratch. In case of F/B, TREE+F/B and ALL, the forward and backward ready time functions are obtained directly from the memory $\mathcal{D}$ (see Section 2.4.2 and Section 2.6.2), while in case of TREE they are obtained from the trees in memory $\mathcal{D}$ (see Section 2.5.5). In case of

NAIVE and F/B, the middle segments are obtained by extending the previously obtained middle ready time functions in set $\boldsymbol{\delta}$, while in case of TREE and TREE+F/B these are obtained from the trees in memory $\mathcal{D}$ (see Section 2.5.5), or, in case of ALL, are obtained directly from memory $\mathcal{D}$ (see Section 2.6.2). Lines 15–16: Using the ready time functions stored in $\boldsymbol{\delta}$, exact feasibility of the move is checked by IsFeasibleExchange and its cost is calculated by CalcExchangeCost, using Equation (2.7) and then Equation (2.3). Lines 17–18: If the current move is feasible and improves the current best move $\mathcal{M}_{r_1 r_2}$ of route pair (r_1, r_2) in memory, then the current move replaces the best move in memory. Lines 26–27: The set of routes to check next, $\tilde{\mathcal{R}}$, is updated to contain only the routes that were changed by the executed move and the best cost $c^*_{r_1 r_2}$ of all route pairs affected by the executed move are reset to zero. The algorithm now repeats to check all new moves. Note that now only moves that affect a changed route in $\tilde{\mathcal{R}}$ are checked, since all best moves of the non-changed route pairs are already stored in memory (Static Move Descriptors, see Section 2.7.1). Line 28: The algorithm stops when no improving k-exchange move can be found.

Chapter 3

When microseconds add up: On the real-time performance of Dynamic Time Slot Management

Thomas R. Visser, Niels Agatz and Remy Spliet

3.1 Introduction

Online retail sales continue to grow, and as e-commerce is becoming more mature, customers expect more control over how, when and where to get their purchased goods delivered. At the same time, online retailers want to minimize costly delivery failures. This is especially the case for attended home delivery, where the customer needs to be at home to receive the goods. This type of delivery is common in online grocery, but also for the delivery of furniture and home appliances. Here, retailers typically let customers select a narrow (e.g., 2-hour) time slot from a menu of delivery time slot options. The design of the time slot offer and the processing of placed orders is commonly referred to as dynamic time slot management (DTSM).

In this setting, we distinguish between the periods before and after a cut-off time. Before the cut-off time, customers continuously arrive over time and interact with the system to select a time slot for delivery. After the cut-off time, a delivery schedule is made for all accepted customer requests. The customer interaction with the system

This chapter is a working paper.

typically consists of the following phases i) the customer shares his or her delivery location, ii) the retailer offers a set of time slots, iii) the customer selects one time slot. In most applications, it might not be possible to accommodate all delivery requests, for instance due to a limited number of delivery vehicles. Therefore, usually a *valid* time slot offer is made, which for each time slot guarantees that if the customer would select it, there exists a feasible delivery schedule. A valid time slot offer could be empty, effectively rejecting the customer. Although it is a choice to only make valid time slot offers, most DTSM in the scientific literature adhere to this design principle.

DTSM for attended home delivery has recently received an increasing amount of attention in the scientific literature, e.g., Campbell and Savelsbergh (2005), Ehmke and Campbell (2014) and Köhler et al. (2019). Most of the literature focuses on quantifying the potential benefits of making dynamic time slot offers based on vehicle routing instead of simple static order limits per time slot. However, there has been little attention for the challenges related to the real-time application of DTSM. It is commonly recognized that the time it takes to make a time slot offer, i.e., *decision time*, should be sufficiently short, since customers may be lost if they have to wait. Although this is a necessary condition for DTSM in practice, it is not sufficient for many applications.

Current DTSM models do not consider the simultaneous interaction of multiple customers with a DTSM system. Two cases of simultaneous interaction are: i) a new customer arrives while the system is still processing a previous customer, which occurs if interarrival times are short compared to the decision time, and ii) customers require time to select a time slot, i.e., *selection time*, during which new customers may arrive. The simultaneous interaction of customers leads to the following complications: i) additional waiting time might be experienced when a customer needs to wait on a previous customer, and ii) a valid time slot offer might be invalidated. In this chapter, we describe why additional waiting time might be encountered, and demonstrate this in simulation experiments. To illustrate why a time slot offer might be invalidated, consider a valid time slot offer presented to a certain customer. While this customer has not yet made a selection, another customer selects a time slot. Now, it might no longer be guaranteed that if a time slot is selected from the offer then a feasible delivery schedule exists.

These complications arise in our collaboration with the Dutch e-grocer AH Online, and we believe they are encountered in many other cases as well. AH Online divides its operations in regions and multiple shifts per day with each more than 2,000

deliveries on average, resulting in numerous customers interacting with the system simultaneously at peak moments. Even more so, the Chinese e-tailer Alibaba receives 325,000 orders per second during peak selling periods (Russell and Liao, 2018). The invalidation of a time slot offer could clearly lead to both operational difficulties as well as decreased customer satisfaction when not dealt with properly. Also the effects of additional waiting time might be severe. A 2017 study suggests that every 100-millisecond delay in website load time can hurt sales conversion rates by 7% (Akamai, 2017).

The main contributions of this chapter are the following: i) we identify and model the simultaneous interactions between multiple customers arising in DTSM systems in the context of attended home delivery, ii) we build a framework of DTSM procedures that is suitable for the new model, and iii) we present real-time simulation experiments to demonstrate the incurred waiting times and assess the effects of invalidated time slot offers. In our framework, we first make a time slot offer, then a customer makes a selection, and finally we accept this offer if the selection is still valid and reject it otherwise. For the design of our framework, we try to stay as close as possible to the state-of-the-art in DTSM methodology. This means that we only make a valid time slot offer, and a customer is only accepted when we have a *feasibility guarantee*, i.e., a guarantee that a feasible delivery schedule exists to accommodate the new customer. We propose procedures for making a time slot offer and for searching for a feasibility guarantee, and even consider additional procedures that can be run in the background to support the former two procedures. In the design of all these procedures, we mostly try to adapt the best performing DTSM methodology in the current scientific literature to the new model, although these adaptations are outperformed in our experiments when using methodology that has not yet been applied in DTSM.

In Section 3.2, we discuss what is, in our view, the state-of-the-art in DTSM. In Section 3.3 we present our DTSM model and in Section 3.4 we provide our DTSM framework. We provide real time simulation experiments in Section 3.5 to measure the performance of our procedures. In Section 3.6, we outline some relevant new questions that would arise from the adoption of our model which have not been considered before. Finally, we provide concluding remarks in Section 3.7.

3.2 Related Literature

In this section, we highlight DTSM literature to provide an overview of state-of-the-art methodology. This methodology will serve as building blocks for the procedures used in our DTSM framework.

The way a time slot offer is made in Campbell and Savelsbergh (2005) can be summarized as follows. When a delivery request is made, an attempt is made to construct a delivery schedule of all previous placed requests. Then, for each potential time slot, the new customer is tried for insertion in the schedule to receive a delivery during that time slot. If this is successful, the time slot is offered, otherwise not. In their case, a delivery schedule is a solution to the classical vehicle routing problem with time windows. The VRPTW is a notoriously difficult problem to solve, Desaulniers et al. (2014), while only a limited time is available to search for a feasible delivery schedule. To limit computation time, they employ a greedy construction heuristic.

Similar approaches have been applied by Ehmke and Campbell (2014) and Cleophas and Ehmke (2014), while incorporating time dependent travel times to model congestion. Alternatively, Campbell and Savelsbergh (2006) and Yang et al. (2016) keep track of a current delivery schedule for all placed orders, or actually multiple alternatives. To make a valid time slot offer, they perform a cheapest insertion search. Moreover, when a customer selects a time slot, they perform a greedy randomized adaptive search procedure, GRASP, to construct multiple new current delivery schedules. Note that both Campbell and Savelsbergh (2006) and Yang et al. (2016) also dynamically determine prices of time slots which they include in their time slot offer. Pricing time slots is not only a means to increase revenue, but could also be used to steer demand towards other time slots. This could result in a less costly delivery schedule or the accommodation of more delivery requests.

Observe that the time to make a time slot offer is limited. For instance, in an online setting, a customer might only be willing to wait as long as it takes to load a web page. In Campbell and Savelsbergh (2005) the computation time of constructing a valid time slot offer is referred to as decision time. They report a maximum decision time of typically less than one second, and Yang et al. (2016) report an average decision time of 0.18 seconds. Note, that we believe the maximum decision time is a better indicator of whether or not any customer had an unacceptably long waiting time, as the decision time will typically increase with the number of placed orders.

Most DTSM studies in the current literature perform experiments with relatively small instances. For example, Campbell and Savelsbergh (2005) used instances of

100 requests, Campbell and Savelsbergh (2006) used instances which contain at most 30 requests, Ehmke and Campbell (2014) used instances of at most 60 requests, Cleophas and Ehmke (2014) used instances with the number of requests generated randomly with a mean of 130, and Köhler et al. (2019) used instances of at most 200 requests. A notable exception are the experiments presented by Yang et al. (2016), which include up to 3530 delivery requests. The delivery schedule is typically modeled as a solution to the VRPTW. Although Ehmke and Campbell (2014) and Cleophas and Ehmke (2014) include time-dependent travel times which increases the computational complexity of finding a feasible delivery schedule, they do not report decision times. Furthermore, on the increased computational complexity to make a time slot offer when pricing, Yang et al. (2016) report that the time to determine the prices is negligible with respect to determining which time slots can be offered. Given the above, we argue that it is not clear if state-of-the-art DTSM methods are able to make a time slot offer fast enough in applications of substantial size, while incorporating real world features in the delivery schedule beyond that of a VRPTW.

It not uncommon to maximize the number of expected deliveries, like for instance in Ehmke and Campbell (2014). Especially in online retail, attended home delivery services are still developing. In this industry it is not unreasonable to focus on gaining market share. In the future it will be more important to consider other objectives, like minimization of expected costs as in Agatz et al. (2011) or maximization of expected profit as in Campbell and Savelsbergh (2006).

Finally, we wish to point out the following. In most studies on DTSM, a time slot is only offered when the selection would result in a feasible delivery schedule after the cut-off time. There are other studies which do not impose this, like Bruck et al. (2018). The proposed procedures in these studies could be characterized as using static order limits per time slot. Here, the delivery region is partitioned and a maximum number of orders per partial region is predetermined. It is immediately clear that when applying such methods, a delivery request can be processed fast enough. However, as a result it could occur that no feasible delivery schedule exists after the cut-off time, which can cause major operational issues in practice.

3.3 Dynamic Time Slot Management Model

In this section we present our DTSM model. The ordering process takes place during the time period $[0, T]$, where T is the cut-off time. That is, during $[0, T]$ customers make delivery request while after T a delivery schedule is made and executed.

Let $\mathcal{C}$ be a collection of customers. At any moment in time it is unknown which customers will request a delivery in the future. For any customer $i \in \mathcal{C}$ that places a delivery request, let t_i be the time at which customer i arrives to make a delivery request of size q_i and known required service duration u_i.

Next, a time slot offer is made. Let $\mathcal{T}$ be a set of time slots, where each time slot is an interval of time later than T. The time it takes to make a time slot offer $\mathcal{T}_i \subseteq \mathcal{T}$ is denoted by r_i^{tso} and is referred to as a response time. This response time is at least as large as the decision time d_i^{tso}, which is the time it takes to construct the time slot offer, while the response time might also include an additional delay. Which time slots are included in the time slot offer is a decision which is part of the optimization problem. Therefore, we do not consider d_i^{tso} and r_i^{tso} as inputs to the problem, but rather a result of the used algorithm to make this decision. At time $t_i + r_i^{\text{tso}}$ the time slot offer is presented to the customer.

After a selection time of s_i, the customer selects a time slot from $\mathcal{T}_i$ at time $t_i + r_i^{\text{tso}} + s_i$. To make this selection, we assume a customer has an ordered set of preferred time slots $\mathcal{T}_i^{\text{p}} \subseteq \mathcal{T}$, and it selects the first time slot in $\mathcal{T}_i^{\text{p}}$ that is also in $\mathcal{T}_i$. If $\mathcal{T}_i^{\text{p}} \bigcap \mathcal{T}_i = \varnothing$, the customer leaves without placing an order.

If the customer does not leave, it has to be decided after time $t_i + r_i^{\text{tso}} + s_i$ whether the delivery request is accepted or rejected. Let d_i^{acc} be the decision time of this accept/reject check and r_i^{acc} the resulting response time. At time $t_i + r_i^{\text{tso}} + s_i + r_i^{\text{acc}}$ the customer is informed whether the delivery request is accepted.

Next, we define a delivery schedule, which is made after the cut-off time T to serve all accepted customers $\mathcal{C}^{\text{acc}}$. Consider the complete directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where the nodes $\mathcal{V} = \mathcal{C}^{\text{acc}} \bigcup \mathcal{D}$ correspond to the accepted customers $\mathcal{C}^{\text{acc}}$ and the depots $\mathcal{D}$. The set $\mathcal{D}$ could consist of multiple depots and each customer $i \in \mathcal{C}^{\text{acc}}$ can receive its delivery from any of the depots in $\mathcal{D}$. We denote by $\tau_{ij}(t)$ the travel time function, which provides the time to traverse arc $(i, j) \in \mathcal{A}$ when departing at time t. As is common, see for instance Ichoua et al. (2003) and Gendreau et al. (2015), we assume this function is piecewise linear, continuous, and that the arrival time function $\alpha_{ij}(t) = t + \tau_{ij}(t)$ is strictly increasing (*First-in-first-out* property). At each depot $d \in \mathcal{D}$, there are K_d vehicles available for making deliveries, each with a capacity Q. A delivery is made when a vehicle visits a customer along a route. We define a route as a pair (ρ, t^ρ), where ρ is a simple cycle in $\mathcal{G}$ that starts and ends at the same depot, and t^ρ is a vector containing the arrival time at each customers on ρ. The total demands q_i, for all customers i visited on the route, may not exceed the vehicle capacity Q, and, to model labour agreements, each route ρ is constrained

in total duration by D_ρ. Each depot $d \in \mathcal{D}$ has a time window $[a_d, b_d]$ between which routes from that depot must start and finish. Furthermore, the start of service time at a customer is required to be in the selected time slot. Vehicles arriving early must wait at the customer's location. A feasible delivery schedule is a set of at most $K = \sum_{d \in \mathcal{D}} K_d$ routes that visit all customers while satisfying the capacity, time window and route duration constraints.

The problem is 1) at each customer arrival during the ordering process construct a time slot offer, 2) at each customer selection decide to accept or reject a customer, and 3) construct a feasible delivery schedule after the cut-off time. The objective is to maximize the expected number of accepted customers.

We wish to point out a special case of this problem, referred to as *non-simultaneous* DTSM. In this case, it does not occur that customers interact simultaneously with the system. The non-simultaneous DTSM closely resembles most DTSM in the current literature. Instances of non-simultaneous DTSM are for example those in which $r_i^{\text{tso}} = s_i = r_i^{\text{acc}} = 0$ for all $i \in \mathcal{C}$. Also instances in which the difference in arrival time $t_j - t_i$ of two successive customers i and j is always larger than $r_i^{\text{tso}} + s_i + r_i^{\text{acc}}$, are non-simultaneous.

3.4 Dynamic Time Slot Management Framework

In this section, we propose a solution framework for DTSM consisting of two mandatory and one optional procedure. First, when a customer arrives, a time slot offer procedure is applied to construct a time slot offer. Secondly, after a customer has made a selection, an accept/reject procedure is applied to see if a feasible corresponding delivery schedule can be found. These two procedures are mandatory and are potentially aided by storing a current schedule of accepted customers in memory during the entire ordering process. Finally, we propose an improvement procedure which can optionally be used in the background to continually improve this current schedule.

The design of the procedures which we present here is mostly based on the state-of-the-art in DTSM. Note that we therefore only allow valid time slot offers, and customers are only accepted if a feasibility guarantee is found. Furthermore, note that existing procedures are not directly applicable to our new model, but we modify them for this purpose. In particular, we use (parts of) state-of-the-art DTSM procedures as time slot offer, accept/reject or improvement procedures. The strategy employed by these procedures could be characterized as myopically offering as much

time slots as possible to every new customer. This increases the likelihood of an individual customer to place an order. Clearly, this strategy serves as a heuristic for the DTSM problem. Of course, to maximize the expected number of customers, it might actually sometimes be better to not offer all possible time slots to a customer. Such considerations require the anticipation of future customers, which we consider beyond the scope of this chapter.

Because new delivery requests can be made while another is in progress, we need to define the order in which the procedures of different requests are executed. In Section 3.4.1, we elaborate on the issues that arise when multiple customers are in the system simultaneously, and provide our sequence of procedures of different requests. In Sections 3.4.2 through 3.4.4 we provide time slot offer procedures, accept/reject procedures and improvement procedures, respectively. Finally, in Section 3.4.5 we discuss which combinations of procedures are used in our computational experiments.

3.4.1 The Sequence of Procedures

We distinguish between the time at which a procedure can be started at the earliest, referred to as enqueue time, and the time a procedure is actually started, referred to as start time. They need not be the same since a procedure might have to wait for another to finish. If the enqueue time differs from the starting time, we say that a procedure is *blocked*. The arrival time of each delivery request is the enqueue time of a time slot offer procedure, and each moment at which a customer selects a time slot is the enqueue time of an accept/reject procedure. The response time of a time slot offer or accept/reject procedure therefore consists of not only the computation time but also a waiting time between the enqueue time and the starting time. When using the optional improvement procedure, we run it whenever an opportunity presents itself, that is, when no other procedures prevent this. Therefore, we do not specify an enqueue time for the improvement procedure.

In the process of DTSM, it is important to distinguish between two types of operations which the procedures perform on the schedule in memory: *read operations* are used to access the schedule in memory, while *write operations* are used to replace or update the schedule in memory. In particular, the time slot offer procedure performs a read operation and does not perform a write operation. Making a time slot offer can be achieved without modifying any schedule in memory, although of course a copy can be made for private use. The accept/reject procedure performs a read operation, required to check whether the selected time slot can still be offered. If the selection is accepted, a write operation is performed to include the newly placed

order in the delivery schedule. Similarly, the improvement procedure performs a read operation, and potentially a write operation to replace the delivery schedule. Note that in the special case in which some procedures do *not* make use of a delivery schedule in memory, read- and write operations still apply in the same way to the access and updating of the set of accepted customers.

Some of the procedures could be run in parallel, if enough processors are available. However, care needs to be taken of the fact that a write operation might invalidate a prior read operation. For instance, a schedule that was accessed earlier might not contain a newly added order.

Given that enough processors are available for parallel computing, new time slot offer and accept/reject procedures can start while the improvement procedure is still running. The same holds while a time slot offer procedure is still running. However, while an accept/reject procedure is still running, this is different. When we accept a request, we want a feasibility guarantee for this new order. Therefore, while the accept/reject procedure is running, we forbid any write operations because they might invalidate the output. We accomplish this by firstly putting new accept/reject procedures in a queue, if another accept/reject procedure is still running. After an accept/reject procedure terminates, we start the accept/reject procedure from the queue, if any remain, which has the earliest enqueue time.

Secondly, we postpone a write operation of the improvement procedure while an accept/reject write procedure is performed, as this could result in a conflict. If the accept/reject procedure does not perform a write operation, in case of rejection, the improvement procedure is allowed to perform a write operation afterward. Otherwise, if the accept/reject procedure does a write operation, in case of acceptance, any improvement that is potentially found is wasted. Note that it might sometimes be possible to *repair* the results of an improvement run, e.g., to update the resulting schedule with the newly accepted customers. However, we consider such repair schemes beyond the scope of this chapter.

Finally, while an accept/reject procedure is running, a time slot offer procedure could be allowed to start since it never performs a write operation. There is a trade-off, however. When starting a time slot procedure while an accept/reject procedure is running, it could be that the ensuing time slot offer is invalidated almost instantly if a new customer is accepted. For that reason, one might wait for the current accept/reject procedure, or even all accept/reject methods in the queue, to terminate. This gives a lower likelihood of the time slot offer to be invalidated at the cost of increased response time incurred by the customer. In this chapter, we let the time

slot offer procedure wait for the single current accept/reject procedure to finish, but we do not let it wait for the rest of the queue.

Conflicting write operations can alternatively be avoided by interrupting the improvement procedure when an accept/reject write operation is performed. However, this is not a trivial task to achieve. The procedure should constantly check whether a write operation is performed or not. Here lies a challenge. If this is checked very often, this comes at the cost of additional computation time and therefore decreases the effectiveness of the improvement procedure. When multiple improvement procedures may be ran in parallel, this becomes less of an issue. The accept/reject procedure might simply start any new thread running an improvement algorithm.

In this chapter, we assume only one thread is available for improvement procedures. Therefore, we use the end of a write operation as the start time of the improvement procedure. Furthermore, we assume the improvement procedure cannot be interrupted once running, and can only perform a write operation when no other write operations have been performed during the run. If a write operation has preceded in the meantime, the current improvement result is lost. We consider this as a baseline for the different possible interruption schemes. To help minimize possible lost runs, a new improvement procedure is only started once the queue of customers for accept/reject is empty. Here we find yet another trade-off, as for example an improvement procedure could also be started after a single accept/reject procedure which gives more time for improvement although the improvement procedures might in this case become invalidated more often.

In summary, to obtain a feasibility guarantee the sequence of execution of some procedures has to be fixed in our framework. In particular, the accept/reject procedures need to be queued and run in sequence, and an improvement run must be wasted when a write operation occurs, to avoid conflicts in the set of accepted customers and the schedule in memory. The sequence of the other procedures involves more arbitrary choices based on the trade-off between validity and response time of time slot offers, the trade-off between the invalidation and computation time of the improvement runs, and of course also based on the number of threads available in practice.

In the remainder of this chapter, we assume the case in which there are an unlimited number of threads available for time slot offers, and there is only one thread available for improvement. In practice, a company typically needs to have multiple shifts running in parallel already, so it is not unrealistic to assume only one improvement thread is available which runs continuously for each shift. Compared to

Table 3.1: Start times of a time slot offer, accept/reject and improvement procedure based on already running procedures used in this chapter. Bold is used to indicate the choice is necessary for guaranteeing the existence of a feasible schedule.

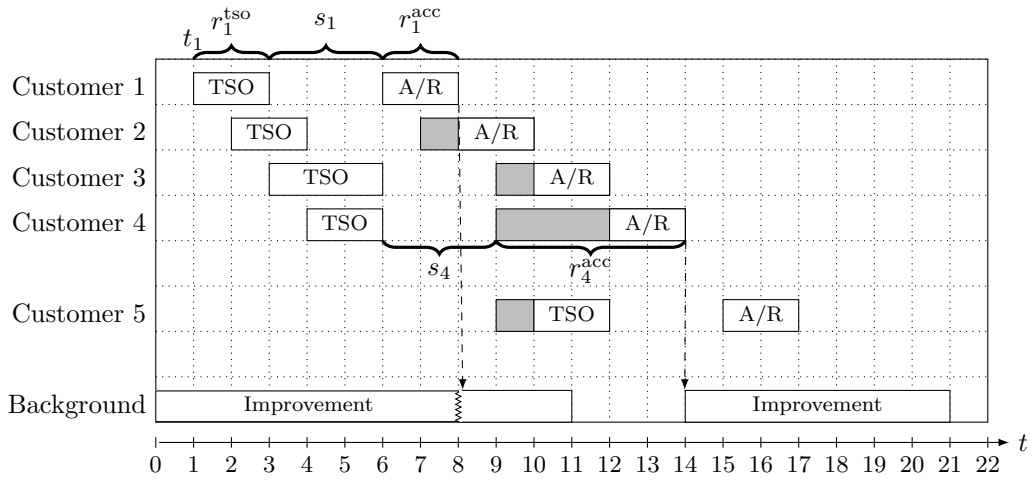
	Already running		
When starting a	Time Slot Offer	Accept/Reject	Improvement
Time Slot Offer	Instant (unlim. threads available)	Blocked: for one A/R	Instant
Accept/Reject	Instant	**Blocked: put in A/R queue**	Instant **(inval. Improv. upon write)**
Improvement	Instant	Wait for empty A/R queue	N/A (one thread available)

improvement procedures the time slot offer procedures are typically quick, so we do not think it unlikely that they could all be run on a sufficient number of dedicated threads and therefore start (almost) immediately. The sequence of procedures in our setting is summarized in Table 3.1. In the next section we provide an example to illustrate this sequence of procedures.

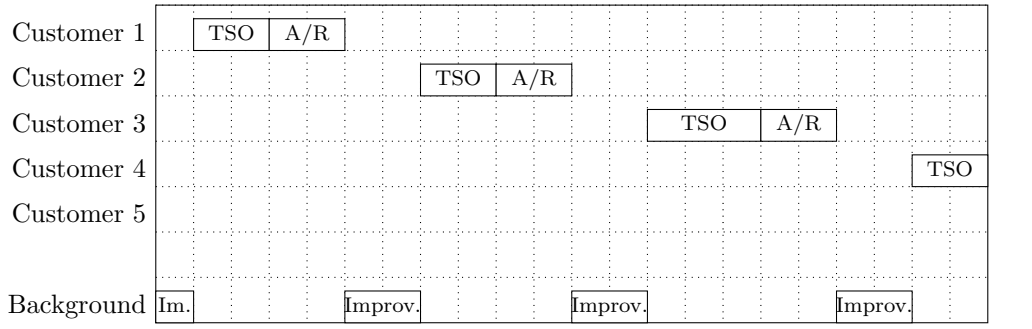
3.4.1.1 Small example

We provide a small example of five arriving customer requests to illustrate the ordering of the procedures, shown in Figure 3.1. The five customers arrive at times $(t_1, t_2, t_3, t_4, t_5) = (1, 2, 3, 4, 9)$, and each customer has a selection time of 3 units. Figure 3.1(a) shows a real-time simulation of our model which includes both decision times and selection times. For comparison, Figure 3.1(b) shows the same simulation for a non-simultaneous instance. In that case, both the decision times and selection times do not play a role, and customers arrive sequentially, as in the current state-of-the-art DTSM procedures. Notice that the arrival times do not influence the simulation in the non-simultaneous case. For illustrative purposes, we assume that there are some accepted customers already at the start of the shown timeline, therefore in both simulations the improvement procedure is started at time zero.

In the real-time simulation with simultaneous interaction, Figure 3.1(a), customer 1 arrives at $t_1 = 1$ and a time slot offer is constructed (TSO). The customer receives the time slot offer after response time r_1^{tso}. After $s_1 = 3$ units of selection time, customer 1 picks a time slot, at which point the accept/reject procedure (A/R) is run. After r_1^{acc} time, customer 1 is accepted. This triggers a write operation, and since the improvement procedure running in the background did not yet finish, its result will be



(a) Simultaneous interaction.



(b) Non-simultaneous interaction.

Figure 3.1: Example of a real-time simulation of DTSM.

lost. In our simulation, the improvement procedure still completes (uninterrupted). The improvement method stops after 11 time units without replacing the schedule in memory.

Customer 2 arrives at $t_2 = 2$, which is during the time slot offer procedure for customer 1. We assume that there are an unlimited amount of threads available for time slot offers, so this procedure can start immediately. After the selection time of customer 2, the accept/reject procedure must be run. However, the accept/reject procedure for customer 1 is still running, so execution is *blocked* until the previous accept/reject finishes. The response time is the actual decision time of the procedure and the waiting time while being *blocked*. In case of customer 2, response time

$r_2^{\text{acc}} = 3$ time units. As can be seen by the response times of customers 3 and 4, the waiting can increase a lot when many accept/reject procedures are blocked: response time r_4^{acc} for customer 4 is already 5 time units. Customer 5 arrives during the accept/reject procedure of customer 3, and the time slot offer procedure is therefore blocked.

After the first improvement procedure, a new improvement run is only started once the queue of accept/reject procedures is empty, which happens at time 14. During this improvement run, customer 5 selects a time slot at time 15 and the accept/reject procedure is run but the time slot is not valid anymore, invalidated for example by the acceptance of customer 4, and customer 5 is rejected. The improvement procedure in the background finishes at time 21, and since no write operations are performed, the schedule is replaced by an improved schedule.

Figure 3.1(b) illustrates the simulation with non-simultaneous interaction. Note that the times do not necessarily compare to those used for Figure 3.1(a). After the first improvement finishes, the first customer arrives and a time slot offer is constructed. The customer then immediately selects a time slot, after which an accept/reject procedure is run. An improvement run is started after the accept/reject procedure finishes, after which a new customer arrives. Notice that all time slot offers will remain valid, and thus no customer will ever be rejected. Also, the improvement procedures are never lost due to preceding write operations of accept/reject procedures. This sequential pattern repeats for all five arriving customers.

3.4.2 Time Slot Offer Procedures

The time slot offer procedure constructs a valid time slot offer. Of course, if another request is accepted during the time between the time slot offer is made and a time slot is selected, the time slot offer might be invalidated. We suggest two different approaches for this. We use 1) a first insert search which is similar to the cheapest insert search of Campbell and Savelsbergh (2006), Yang et al. (2016) and Köhler et al. (2019), and 2) a greedy construction heuristic like Campbell and Savelsbergh (2005), Ehmke and Campbell (2014) and used in Cleophas and Ehmke (2014).

First, we explain the first insert search. In this search, we consider a feasible delivery schedule to be available in memory. This delivery schedules does not contain a visit to the new customer during the considered time slot. For each time slot, we iteratively consider all routes and all positions on these routes in which a visit might be inserted to the new customer during the considered time slot. When a feasible insert position is found, the search immediately terminates and the selected

time slot is included in the time slot offer. Note that for checking feasibility, the forward/backward (F/B) algorithm with static move descriptors is used as described in Chapter 2, which is the fastest known algorithm when both time dependent travel times and route duration constraints are present. Also note that we first go over the empty routes, and then the non-empty routes in the order that they happen to be stored in memory. In the worst case, no feasible insert position can be found, which means that all insert positions need to be considered anyway. So in the sense of worst-case run time, the order of routes is irrelevant. Although, it is of course conceivable that the non-empty routes might be sorted in such a way that a feasible insert position, if it exists, is found on average earlier in the search.

Next, we explain the greedy construction heuristic. The aim is to construct a new schedule from scratch for all currently accepted customers, and then using this schedule to find the time slot offer for the newly arrived customer. This heuristic makes use of a cost criterion, which allows us to compare different delivery schedules. We choose to use the average travel time on an arc as its cost, and define the cost of a delivery schedule as the total costs of the used arcs. This way, we expect a schedule with lower costs to allow for the inclusion of more additional customers than a schedule with higher costs. We initialize with an empty route for every vehicle. Next, iteratively, for every customer that is not yet inserted on a route, a feasible insert position in the schedule which yields the smallest cost increase is found by considering all possible insert positions. The cheapest insert among all customers is performed, i.e., the customer is inserted at that position. This procedure is repeated until all customers are scheduled, or, as might happen, the schedule is found to be infeasible. Then, first search as explained above is used on this constructed schedule to find the time slot offer for the new customer. Also in this case, we use the cheapest insertion construction heuristic with the forward/backward algorithm and static move descriptors as described in Chapter 2 (see Algorithm 2.1).

Note that the greedy construction heuristic does not modify a delivery schedule which is stored in memory, but rather creates a completely new delivery schedule. As a result, the decision time is expected to be greater than that of the first insert search. However, the greedy construction heuristic might be more successful in finding a feasible delivery schedule if the quality of the schedule in memory is low. Also note that this approach resembles the approach used by Campbell and Savelsbergh (2005) without their use of randomness with GRASP. It differs slightly from the approach used by Ehmke and Campbell (2014) and Cleophas and Ehmke (2014), which, for each possible time slot, constructs a schedule including the newly arriving customer.

It will be obvious from our numerical experiments that running multiple greedy construction heuristics in this fashion does not seem a realistic option in terms of response time, at least not when run on a single thread.

3.4.3 Accept/Reject Procedures

The accept/reject procedure takes as input a time slot selection from a customer and checks whether a corresponding feasible delivery schedule can be found. If so, the customer is accepted and the schedule in memory is updated, otherwise the customer is rejected. Also for this procedure, we suggest to use two different approaches. They are 1) a cheapest insert search like Campbell and Savelsbergh (2006), Yang et al. (2016) and Köhler et al. (2019), and 2) a greedy construction heuristic like is also used as time slot offer procedure.

In the cheapest insert search, we go over the delivery schedule in memory until the cheapest feasible insert position is found, where the costs are defined as before as average travel times per arc. If no feasible insert position is found, the customer is rejected. Otherwise, the cheapest insert is performed and the schedule is updated. Note that updating the schedule includes updating the forward/backward data structures as described in Chapter 2. It is not always necessary to make the customer wait until the cheapest insert search is completed before sending a message that the order is accepted. To limit the response time, this message could already be sent when the first feasible insert position is found. In this way, the response time may be lower than the decision time of the procedure. Therefore, in this chapter when presenting response times we only include the first feasible insert time, although the decision time of course represents the computation time of the entire cheapest insert search.

Secondly, we can also apply the greedy construction heuristic provided in Section 3.4.2 as an accept/reject procedure. In this case, if the construction heuristic is unsuccessful, the customer is rejected. Otherwise, the customer is accepted and the current schedule in memory is replaced by the newly constructed schedule. In this case, a customer does have to wait until the heuristic is completed before an accept message might be sent.

3.4.4 Improvement Procedure

The improvement procedure attempts to find a better delivery schedule than the one in memory. We consider a delivery schedule to be better than another if it has lower

costs, where again the costs are defined as average travel times per arc. Recall that this procedure is optional. We suggest two approaches, namely 1) a GRASP like in Campbell and Savelsbergh (2005), Campbell and Savelsbergh (2006) and Yang et al. (2016), and 2) a Neighborhood Search procedure.

The GRASP works almost identical as the greedy construction heuristic explained in Section 3.4.2. However, instead of identifying the single cheapest feasible insert position in each iteration, the l cheapest feasible insert positions are found. Next, one of those l options is randomly selected and performed, each with equal probability. Observe that for $l = 1$ this is equivalent to the greedy construction heuristic. The GRASP is designed to provide different delivery schedules at every run. Therefore, by continually running GRASP as an improvement procedure, a diverse range of delivery schedules is explored. If the cost of the new delivery schedule is lower than the cost of the schedule in memory, the schedule is replaced. Note that, following the discussion in Section 3.4.1, this replacement is not performed as long as an accept/reject procedure is also running. In that case, if a new customer is accepted, the delivery schedule found by the improvement procedure is wasted, while otherwise it is stored until the accept/reject procedure finishes at which time the replacement takes place. This approach strongly resembles the GRASP approaches used in Campbell and Savelsbergh (2005) and Campbell and Savelsbergh (2006), where GRASP is run a fixed number of times between each customer placement. Clearly, a fixed number of runs is not a suitable stopping criterion in our new model. The GRASP of Yang et al. (2016) selects customers at random rather than selecting from a list of best insertions.

Secondly, we propose a neighborhood search approach which uses a lexicographic k-exchange (Kindervater and Savelsbergh, 1997). Each improvement run, one neighborhood search iteration is conducted using the current schedule in memory. An iteration consists of evaluating all schedules that can be obtained by performing a move on the current schedule. The best feasible move, based on a cost criterion, is executed. In a k-exchange neighborhood, a move consist of the exchange of a segments of customers of length up to k in one route with a segment of customers of length up to k in another route. We use the average travel time as arc costs. We use the lexicographic k-exchange with ready-time function tree and forward/backward data structures (TREE+F/B) and static move descriptors as described in Chapter 2 (see Algorithm 2.2).

Table 3.2: DTSM configurations investigated in this chapter. *INSCON uses greedy construction after cheapest insertion in the accept/reject procedure.

	Time Slot Offer	Accept/Reject	Improvement
CON	Greedy Construction	Greedy Construction	No
INS	First Insertion	Cheapest Insertion	No
INSCON	First Insertion	Greedy Construction*	No
INS+GR	First Insertion	Cheapest Insertion	GRASP
INS+NS	First Insertion	Cheapest Insertion	Neighborhood Search

3.4.5 Combinations of Procedures

Observe that there are two options for both the time slot and accept/reject procedures, and no, one or two improvement procedures can be used. We focus on the following five combinations for the remainder of this chapter. Table 3.2 provides a summary of these configurations. The interested reader is referred to Appendix 3.A which contains an overview of the computational complexities of the configurations.

- **CON**
  This configuration does not use a stored schedule in memory, but uses greedy construction for both time slot offer and accept/reject procedures. No improvement is done. In the non-simultaneous case, this configuration resembles the methods used in Campbell and Savelsbergh (2005) without GRASP, Ehmke and Campbell (2014) and Cleophas and Ehmke (2014) without tabu search.

- **INS**
  This configuration uses the faster first insert search for time slot offer, and cheapest insertion for accept/reject to update the schedule in memory. This schedule in memory is not improved. In the non-simultaneous case, this configuration resembles the method used in Köhler et al. (2019) without so-called *flexible* time slot offers.

- **INSCON**
  This configuration is similar to INS, but additionally uses one greedy construction after the cheapest insertion update during the accept/reject. Note that this increases the quality of the schedule in memory but also the decision time of the accept/reject procedure. In the non-simultaneous case, this configuration resembles the methods used by Campbell and Savelsbergh (2006) and Yang et al. (2016), in case their GRASP is limited to one greedy construction ($l = 1$) run.

- **INS+GR**
  This configuration is again similar to INS, but additionally uses the GRASP, with random l possible best insertions, as improvement procedure to find better schedules in the background. Notice that the difference with INSCON is that improving schedules are found by the improvement procedure in background, rather than during the accept/reject procedure. In the non-simultaneous case this also resembles the methods used by Campbell and Savelsbergh (2006) and Yang et al. (2016).

- **INS+NS**
  This configuration is similar to INS+GR, but uses the k-exchange neighborhood search as improvement procedure instead of GRASP. To the best of our knowledge, a k-exchange neighborhood search has not before been used in the context of DTSM.

3.5 Computational Experiments

In this section, we present the results of numerical experiments in which we use the presented DTSM configurations on several randomly generated instances. We use a discrete event simulator to simulate the ordering process and the DTSM configurations, which were both implemented in C++11 and compiled using GCC version 6.3. All simulation runs are executed as a single thread on an Intel® Xeon® E5-2650 v2 with 2.6 GHz (Turbo Boost up to 3.6 GHz) and 64 GB of RAM running Debian Linux version 9. All CPU times were measured using `std::chrono::high_resolution_clock`, a high resolution wall-clock timer, and were used with microsecond ($\mu s = 10^{-6}\,s$) precision inside the real-time simulations. Each individual procedure within the configurations needed at least one microsecond of CPU time on our machine. Although the DTSM configurations are essentially multi-threaded, our custom discrete event simulator allows us to simulate a configuration using only one thread. This way, we avoid the CPU time measurements to include possible overhead that is specific to a multi-threaded/parallel implementation and the used CPU architecture. Only this one thread was run at any time on the CPU.

3.5.1 Instance Generation and Parameters

Our aim is to study the effects of customer selection time and the time between customer arrivals on the performance of the DTSM configurations. We generated 10

Table 3.3: Number of vehicles available and total capacity in number of customers.

	# Vehicles			
n	Total	Fulfill.	Hub	Total cap. (# cust.)
500	14	5	3	462
1000	25	10	5	825
2000	50	20	10	1650
4000	100	40	20	3300
8000	200	80	40	6600

instance sets with $n = 500$, 1000, 2000, 4000, and 8000 arriving customers, which gives a total of 50 instance sets. Each instance set consist of multiple instances in which only the selection time and the interarrival time between the customers are varied. That is, the customer locations, their time slot preference, and order of arrival are fixed for all instances in one instance set.

The characteristics of the instances are inspired by our collaboration with our industry partner AH Online. We choose the delivery area to coincide with the major urban area of The Netherlands. Our region contains four major Dutch cities (Amsterdam, Rotterdam, The Hague and Utrecht), various surrounding urban satellite cities and towns, and rural areas. The region is shown in the inset of Figure 3.2. Data from 2017 is used, in which our selected region contains roughly 3.3 million registered households, which is about 43% of the total number of registered households in the Netherlands (van Leeuwen et al., 2017). We consider four depots, each located with easy highway access near a major city, and each with a fleet of identical vehicles. We assume identical demand for each customer that places an order and vehicles have a capacity limit of 33 customers. Furthermore, each route has a maximum duration of 6 hours due to labour regulations. We identify one depot as a *fulfilment center*, the main warehouse facility where orders are picked, while the three other depots are *hub* locations, where produced orders are only transferred to the delivery vehicles. This difference is reflected in our choice of the available vehicles, as shown in Table 3.3. The fulfilment center has twice the number of vehicles available compared to a single hub location. The total number of vehicles increases linearly with the number of arriving customers, with the exception of $n = 500$ due to rounding, and therefore also the number of customers that can be accepted based on capacity alone. However, in our instances the resulting vehicle routes are mostly constrained by time, i.e., time slots and route duration, rather than by capacity.

We model a morning execution shift with a depot time window of $[06{:}00, 15{:}00]$.

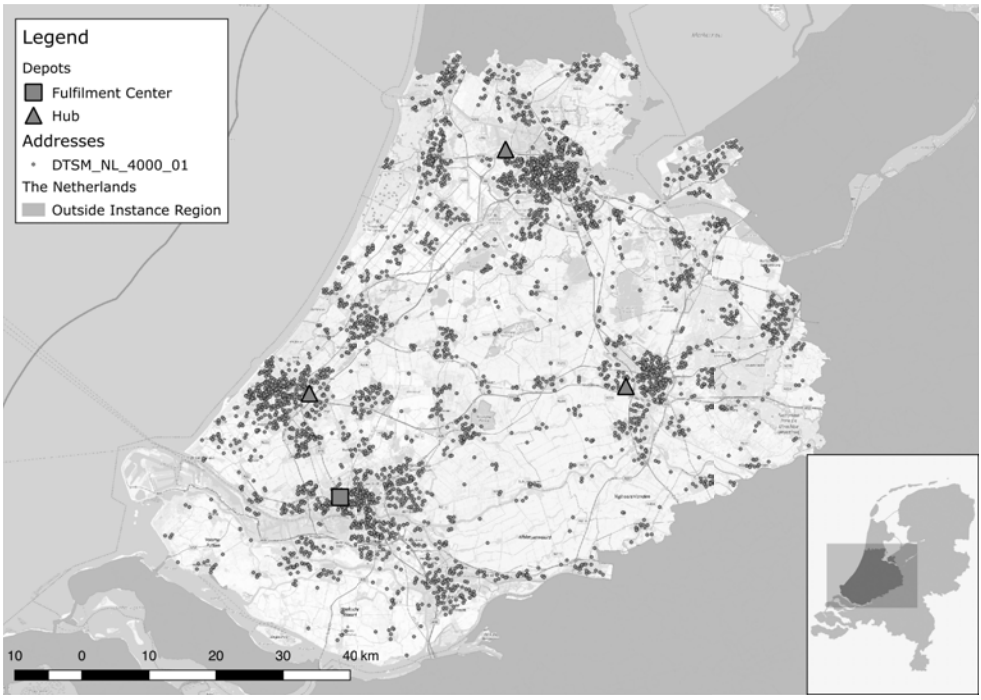


Figure 3.2: Instance region with depot locations and address locations for instance DTSM_NL_4000_01 with $n = 4000$ arriving customers. Inset shows the instance region within The Netherlands. Background layer © OpenStreetMap contributors.

The set of time slots $\mathcal{T}$ is given by:

$$\mathcal{T} = \{\, [07{:}00, 08{:}00]\,, [08{:}00, 14{:}00]\,, [08{:}00, 10{:}00]\,, [09{:}00, 11{:}00]\,, [10{:}00, 12{:}00]\,, \\ [11{:}00, 13{:}00]\,, [12{:}00, 14{:}00]\,\}.$$

Notice that these time slots are overlapping, and mostly 2 hours except for one shorter 1 hour and one longer 6 hour time slot. All time slots are available throughout the whole region.

For each of the 50 instance sets we separately generate customer locations by selecting at random from a list of real address locations. Because our dataset does not include which of the real address locations are households, we in fact first select a zip code (PC4 level) weighted on the number of households (van Leeuwen et al., 2017), and then uniformly select a real address location at random within that zip code. This way we limit over-representation of industrial addresses in our selection.

Table 3.4: Speed Profile

	Zones		
p	06:00–07:00	07:00–10:00	10:00–15:00
6	1.00	0.50	1.00

Figure 3.2 shows the instance region with the depots, and illustrates the customer addresses for a particular instance set of $n = 4000$ arriving customers. The service time of each customer is fixed and equal to 5 minutes.

Each customer has an ordered set of time slot preferences $\mathcal{T}_i^{\mathrm{p}}$ containing $|\mathcal{T}_i^{\mathrm{p}}| = 2$ time slots, which are drawn uniformly from the set of time slots $\mathcal{T}$. This means that each time slot is equally popular, but, because of the different widths among the time slots, this popularity is not evenly spread across the planning horizon $[06{:}00, 15{:}00]$.

Although all presented methods can be used with travel time functions for each arc varying heavy in shape, e.g., in congestion periods and their intensities, we use only one speed profile with nominal travel times for each arc. These nominal (free-flow) travel times between all locations are calculated using *OSRM*, an open-source routing service for *openstreetmap*, and rounded to minutes. Table 3.4 shows the speed-profile used to obtain the time-dependent travel time functions, which models a congestion period in between free flow. The interested reader is referred to, among others, Ichoua et al. (2003) who describe the generation of time-dependent travel time functions from nominal travel times and speed profiles.

In our experiments, one of the aims is to demonstrate the effect that the duration of selection has on performance. For this reason, we consider two cases of customer selection times: 1) each customer has an equal time slot selection time $s = 30$ seconds, and 2) each customer has an equal selection time $s = 0$, which means the customer immediately selects a time slot when offered or leaves.

Furthermore, within each instance set, we consider 7 different cases of the arrival times. All cases are based on a constant interarrival time between the arriving customers, i.e., the arrival time t_i of the ith customer is $t_i = t_{i-1} + \Delta$, with $i \in \{2, \ldots, n\}$, $t_1 = 0$ and interarrival time Δ. We consider the following cases of constant interarrival times: $\Delta = 1\,\mu\mathrm{s}$, $1\,\mathrm{ms}$, $10\,\mathrm{ms}$, $100\,\mathrm{ms}$, $1\,\mathrm{s}$, and $10\,\mathrm{s}$. Again, we choose for fixed interarrival times to facilitate the demonstration of the effect that the interarrival times have on performance.

Note that we do not specify a cut-off time. In our experiments we are interested in the effects that occur when the capacity limit of the system is reached. Therefore, one might think of the cut-off time as being sufficiently large, such that all customers

have been processed before this time.

Moreover, for each instance set we consider an non-simultaneous case, representing the case in which selection times, customer arrival times, and the computation times of the procedures do not play a role. Customers arrive sequentially and all procedures are run in sequence, i.e., time slot offer, then accept/reject and then improvement. Each customer arrives after the previous customer has selected a time slot or leaves and potential improvement methods have finished. Note that in this case, the computation times of all procedures do not affect the event times in the simulation itself.

We use the following parameters for the improvement configurations. The GRASP used within INS+GR selects from $l = 3$ possible moves. Each time a new customer is accepted, the first new GRASP improvement run uses $l = 1$, rather than $l = 3$, and therefore resembles a (deterministic) greedy construction procedure. The INS+NS searches an M-exchange neighborhood, with M the maximum number of customers possible in a route. All possible segments from 1 up to M customers are tried for exchange. We use the ready-time function trees and forward/backward data structures (TREE+F/B) and static move descriptors, as described in Chapter 2, to speed-up the search. Each improvement procedure runs 1 search iteration. At the start of each improvement procedure, the data structures are copied to allow for reversion when a write operation supersedes an improvement. For the non-simultaneous case, the INS+GR improvement is limited to 10 improvements, while the INS+NS improvement is limited to 100 iterations between customer arrivals. In our experiments, a maximum of 100 iterations for INS+NS is not limiting since a local optimum is typically reached in less iterations. As we will see, the performance of INS+NS in the non-simultaneous case is very similar to the INS+NS in the simultaneous case but with long interarrival times. Such a comparison is more difficult for INS+GR, as 10 improvements requires little computation time in the early phase of the simulation with still low number of accepted customers, while it requires a lot during the late phase of the simulation with a high number of accepted customers.

3.5.2 Small Instances

The configurations CON, INS, INSCON, INS+GR and INS+NS (Section 3.4.5) are tested on the small instance sets with $n \in \{500, 1000, 2000\}$ arriving customers. In the following, we first compare the number of accepted and rejected customers, then compare the performance of the time slot offer and accept/reject procedures, and finally compare the performance of the improvement procedures.

In Table 3.5, we show the number of accepted customers, leaving customers and rejected customers, averaged over the 10 instance sets, for a selection time of 30 seconds and the different interarrival time cases. We also show the results of the non-simultaneous case ('Non-sim.'). Recall that the number of accepted customers is the primary objective of the DTSM model, and since each configuration guarantees that a feasible schedule of accepted customers exists, the number of accepted customers is directly comparable.

Both the CON and INS configurations result in a relatively low number of accepted customers, for all interarrival time cases. The CON method uses greedy construction each time a new customer arrives but does not store a schedule in memory. The latter seems essential. While the INS configuration does store a schedule in memory, accepted customers are inserted at the cheapest position in the current schedule. Subsequently, the order and allocated routes of the accepted customers does not change anymore in this configuration. This results in schedules of relatively low quality, i.e., high travel times and low number of customers per route, which limits the number of customers that can be accepted.

Configuration INSCON improves the schedule in memory using a greedy construction within the accept/reject procedure, which results in a relatively high number of accepted customers for any interarrival time. Both INS+GR and INS+NS improve the schedule in the background, and therefore rely on the interarrival time to run improvement procedures. For long interarrival times, both the INS+GR and INS+NS configuration can accept much more customers than the INS configuration. In particular, INS+GR seems slightly better than the INSCON for the interarrival time of 10 seconds, while the INS+NS clearly outperforms both INSCON and INS+GR. However, when the interarrival time shortens, the performance of INS+GR and INS+NS reduces. For very short interarrival times of 1 µs (microsecond), the INS+GR and INS+NS configurations do not get enough time to improve the schedule, and are essentially the same as the INS configuration. Note that slight variations in response times can cause a customers to receive their time slot offer in a different order than in which they arrive, which typically results in a different number of accepted customers. The configuration INS+NS seems to be more affected to the degradation in the number of accepted customers due to the shortening interarrival time than INS+GR, as can be seen in Table 3.5 for instance for $n = 1000$ at interarrival time of 1 ms and for $n = 2000$ at interarrival time of 10 ms.

Table 3.5: Results for the small instance sets with selection time of 30 seconds.

			# Accepted Customers					# Leaving Customers					# Rejected Customers				
n	Sel.	Interarr.	CON	INS	INSCON	INS+GR	INS+NS	CON	INS	INSCON	INS+GR	INS+NS	CON	INS	INSCON	INS+GR	INS+NS
500	30 s	1 µs	133.0	170.8	188.5	171.6	170.4	0.0	0.0	0.0	0.0	0.0	367.0	329.2	311.5	328.4	329.6
		1 ms	133.4	172.2	187.7	183.5	189.0	0.0	0.0	0.0	0.0	0.0	366.6	327.8	312.3	316.5	311.0
		10 ms	133.4	172.2	187.7	186.0	227.6	0.0	0.0	0.0	0.0	0.0	366.6	327.8	312.3	314.0	272.4
		100 ms	133.4	172.5	187.8	191.8	230.9	65.8	23.4	12.8	10.2	1.8	300.8	304.1	299.4	298.0	267.3
		1 s	132.9	173.2	189.1	196.5	230.6	336.3	285.5	272.2	267.0	226.3	30.8	41.3	38.7	36.5	43.1
		10 s	134.7	173.7	192.8	203.2	228.2	362.2	322.0	303.5	292.4	266.0	3.1	4.3	3.7	4.4	5.8
		Non-sim.	132.5	172.8	188.6	190.4	229.6	367.5	327.2	311.4	309.6	270.4	0.0	0.0	0.0	0.0	0.0
1000	30 s	1 µs	302.3	314.7	389.7	315.7	312.3	0.0	0.0	0.0	0.0	0.0	697.7	685.3	610.3	684.3	687.7
		1 ms	302.4	313.6	387.4	366.7	327.1	0.0	0.0	0.0	0.0	0.0	697.6	686.4	612.6	633.3	672.9
		10 ms	302.4	313.6	387.4	393.7	434.6	0.0	0.0	0.0	0.0	0.0	697.6	686.4	612.6	606.3	565.4
		100 ms	302.4	315.2	388.4	393.9	487.3	396.3	369.4	284.4	276.7	177.8	301.3	315.4	327.2	329.4	334.9
		1 s	302.4	316.7	394.1	394.3	483.7	666.3	644.9	562.3	566.0	471.8	31.3	38.4	43.6	39.7	44.5
		10 s	302.8	313.9	396.4	399.5	486.4	693.6	681.9	598.8	595.7	507.9	3.6	4.2	4.8	4.8	5.7
		Non-sim.	301.4	313.7	398.3	397.6	490.5	698.6	686.3	601.7	602.4	509.5	0.0	0.0	0.0	0.0	0.0
2000	30 s	1 µs	741.6	676.6	936.7	681.3	674.2	0.0	0.0	0.0	0.0	0.0	1258.4	1323.4	1063.3	1318.7	1325.8
		1 ms	742.4	689.5	933.1	689.2	696.5	0.0	0.0	0.0	0.0	0.0	1257.6	1310.5	1066.9	1310.8	1303.5
		10 ms	742.4	689.5	933.1	905.2	800.8	0.0	0.0	0.0	0.0	0.0	1257.6	1310.5	1066.9	1094.8	1199.2
		100 ms	739.6	692.5	933.8	944.1	1085.3	203.4	964.6	0.5	684.0	428.5	1057.0	342.9	1065.7	371.9	486.2
		1 s	742.4	688.6	931.8	925.9	1180.0	1226.9	1269.2	1031.2	1033.7	768.9	30.7	42.2	37.0	40.4	51.1
		10 s	742.0	688.8	931.1	932.1	1173.3	1254.5	1306.4	1064.5	1063.7	820.7	3.5	4.8	4.4	4.2	6.0
		Non-sim.	742.5	689.4	930.4	934.4	1176.8	1257.5	1310.6	1069.6	1065.6	823.2	0.0	0.0	0.0	0.0	0.0

Of interest is also the number of rejected customers, i.e., customers that selected a time slot from their offer that is unavailable at the time of processing accept/reject, versus the number of leaving customers, i.e., customers that did not receive at least one of their preferred time slots in their offer, which includes customers that received an empty time slot offer. A rejection occurs when a selected time slot is invalidated during the selection time of a customer, and possibly some blocking accept/reject response time. This can happen due to write operations in the meantime, both from accept/reject procedures which change the schedule and set of accepted customers, and from improvement procedures which only change the schedule.

Table 3.5 shows that the number of rejected customers is of the same order as the number of customers arriving/selecting a time slot within the selection time of a single customer. For example, for $n = 2000$ the average number of rejected customers is 3 – 6 for the interarrival time of 10 seconds, while 30 – 51 customers are rejected on average for the interarrival time of 1 second. The average number of rejections is slightly higher for configurations in which the schedule itself might change more frequently, like INS+GR and INS+NS, compared to CON, which does not use a schedule in memory and therefore seems to give more stable time slot offers.

For short interarrival times, 1 µs, 1 ms, and 10 ms, notice that all n customers have already arrived for a time slot offer *before* the first customer (first to receive an offer) has selected a time slot. Therefore, all customers will receive a full time slot offer ($\mathcal{T}_i = \mathcal{T}$) and thus no customers will decide to leave, since they can always pick their preferred time slot.

In the non-simultaneous case, the computation times of all procedures do not influence the simulation, therefore all procedures finish as if done immediately. In this case all time slot offers remain valid and thus no customers are rejected.

Table 3.6: Results for the small instance sets with selection time of 0 seconds.

			# Accepted Customers					# Leaving Customers					# Rejected Customers				
n	Sel.	Interarr.	CON	INS	INSCON	INS+GR	INS+NS	CON	INS	INSCON	INS+GR	INS+NS	CON	INS	INSCON	INS+GR	INS+NS
500	0 s	1 µs	135.5	168.2	188.3	166.7	170.7	0.0	0.0	0.0	0.0	0.0	364.5	331.8	311.7	333.3	329.3
		1 ms	136.4	172.8	190.1	182.9	195.2	11.7	327.2	0.0	317.1	304.8	351.9	0.0	309.9	0.0	0.0
		10 ms	133.1	172.8	192.2	182.8	226.6	358.9	327.2	270.8	317.2	273.4	8.0	0.0	37.0	0.0	0.0
		100 ms	132.5	172.8	188.6	187.9	229.6	367.5	327.2	311.4	312.1	270.4	0.0	0.0	0.0	0.0	0.0
		1 s	132.5	172.8	188.6	194.4	229.6	367.5	327.2	311.4	305.6	270.4	0.0	0.0	0.0	0.0	0.0
		10 s	132.5	172.8	188.6	201.0	229.6	367.5	327.2	311.4	299.0	270.4	0.0	0.0	0.0	0.0	0.0
1000	0 s	1 µs	287.6	316.2	388.4	317.3	308.6	0.0	0.0	0.0	0.0	0.0	712.4	683.8	611.6	682.7	691.4
		1 ms	296.5	313.7	389.5	383.5	340.0	0.0	686.3	0.0	616.5	660.0	703.5	0.0	610.5	0.0	0.0
		10 ms	297.0	313.7	392.4	398.6	445.9	180.5	686.3	0.6	601.4	554.1	522.5	0.0	607.0	0.0	0.0
		100 ms	301.2	313.7	398.3	394.1	492.3	697.8	686.3	601.0	605.9	507.7	1.0	0.0	0.7	0.0	0.0
		1 s	301.4	313.7	398.3	397.6	490.5	698.6	686.3	601.7	602.4	509.5	0.0	0.0	0.0	0.0	0.0
		10 s	301.4	313.7	398.3	396.4	490.5	698.6	686.3	601.7	603.6	509.5	0.0	0.0	0.0	0.0	0.0
2000	0 s	1 µs	739.5	684.2	925.9	693.3	686.7	0.0	0.0	0.0	0.0	0.0	1260.5	1315.8	1074.1	1306.7	1313.3
		1 ms	742.7	689.4	927.4	684.3	691.4	0.0	1310.6	0.0	1315.7	1308.6	1257.3	0.0	1072.6	0.0	0.0
		10 ms	740.9	689.4	926.1	902.8	825.8	0.0	1310.6	0.0	1097.2	1174.2	1259.1	0.0	1073.9	0.0	0.0
		100 ms	742.3	689.4	925.5	951.8	1117.7	459.6	1310.6	3.7	1048.2	882.3	798.1	0.0	1070.8	0.0	0.0
		1 s	742.5	689.4	930.4	926.3	1174.5	1257.1	1310.6	1069.6	1073.7	825.5	0.4	0.0	0.0	0.0	0.0
		10 s	742.5	689.4	930.4	930.3	1176.8	1257.5	1310.6	1069.6	1069.7	823.2	0.0	0.0	0.0	0.0	0.0

The blocking of accept/reject procedures also has an effect on the number of rejected customers. This can be seen more easily when the selection time is set to 0 seconds, i.e., customers select the time slot immediately. The results for the selection time of 0 seconds are shown in Table 3.6. Although customers select their time slot immediately, these selected time slots can still become invalidated at the moment of the accept/reject check due to blocking. Recall that upon selecting a time slot, a customer is put into a queue to be processed by the accept/reject procedure. As the number of accepted customers grows during the ordering process, more time is needed for the accept/reject procedures. If the time between the customer selections, which is related to the interarrival time, becomes shorter than the time needed to execute an accept/reject procedure, the queue grows, and some time slot offers might become invalid.

For an interarrival time of 1 µs, blocking occurs for all configurations, and this can been seen by the high number of rejected customers in Table 3.6. However, CON and INSCON retain this high number of rejections for longer interarrival times, up to 100 ms for $n = 2000$. Since both CON and INSCON use greedy construction in the accept/reject procedure, these need significantly more time to complete and therefore blocking already occurs for longer interarrival times. This effect on the number of rejected customers also occurs for the selection time of 30 seconds in Table 3.5, as can been seen by comparing to the case of 0 seconds: for $n = 2000$ customers and interarrival time of 100 ms the number of rejected customers of CON and INSCON are much higher than for the other configurations.

Besides the primary objective of accepted customers, the responsiveness of the different configurations is important. Table 3.7 shows the maximum decision time ('max. CPU'), maximum response time and the number of times the procedure was blocked, for both the Time Slot Offer and the accept/reject procedure, averaged over the 10 instance sets. The configurations CON, INS and INSCON are shown in the table. We use 'INS*' to indicate that the results of the configuration INS, which are shown, are very similar to the results of the INS+GR and of the INS+NS configurations not shown here.

Both for CON and INSCON, the decision times for accept/reject are much higher than for INS. For short interarrival times, i.e., in the same order of magnitude as the decision times, this causes blocking which is reflected in both the number of times that the accept/reject procedure was blocked, and the increasing maximum response times. For $n = 2000$ and an interarrival time of 100 ms, the maximum response time for CON is on average 537 seconds (almost 9 minutes), for INSCON

it is 177 seconds (almost 3 minutes), while the INS* configurations only require 0.2 milliseconds. Upon selecting a time slot, some customers have to wait these response times before acceptance is confirmed. While INSCON does improve the number of accepted customers compared to INS, as shown in Table 3.5, the resulting response times become very high when customers arrive quite frequently. For the INS* configurations, blocking only occurs for the shortest interarrival time of 1 µs. For $n = 2000$, customers may have to wait up to on average 182 milliseconds. Recall that the Time Slot Offer procedures wait for at most one already started accept/reject procedure. This can be seen by the number of blocked Time Slot Offer procedures and increased response times. For short interarrival times, 1 µs, 1 ms and 10 ms, the time slot offers take little time since no customers are accepted yet.

Table 3.7: Time slot offer and accept/reject results for the small instance sets with selection time of 30 seconds.

			Time Slot Offer									Accept/Reject								
			max. CPU (ms)			max. Response (ms)			# Blocked			max. CPU (ms)			max. Response (ms)			# Blocked		
n	Sel.	Interarr.	CON	INS*	INSCON	CON	INS*	INSCON	CON	INS*	INSCON	CON	INS*	INSCON	CON	INS*	INSCON	CON	INS*	INSCON
500	30 s	1 µs	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	23.3	0.2	33.7	5974.7	17.0	1978.8	499.0	499.0	499.0
		1 ms	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	29.8	0.2	33.3	5581.6	0.1	1600.9	472.3	0.0	476.0
		10 ms	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	25.6	0.2	34.0	1763.2	0.1	319.7	360.1	0.0	155.7
		100 ms	17.2	0.2	0.2	17.2	0.2	0.2	0.0	0.0	0.0	21.1	0.2	34.3	21.1	0.1	0.1	0.0	0.0	0.0
		1 s	24.4	0.3	0.2	24.4	0.3	0.2	0.0	0.0	0.0	14.9	0.2	31.4	14.9	0.1	0.0	0.0	0.0	0.0
		10 s	28.1	0.3	0.2	28.1	0.3	0.2	0.0	0.0	0.0	14.9	0.2	35.1	14.9	0.1	0.0	0.0	0.0	0.0
		Non-sim.	17.3	0.3	0.2	17.3	0.3	0.2	0.0	0.0	0.0	14.1	0.2	32.5	14.1	0.1	0.0	0.0	0.0	0.0
1000	30 s	1 µs	0.1	0.0	0.1	0.1	0.0	0.1	0.0	0.0	0.0	114.3	0.3	168.8	62698.3	55.3	18641.8	999.0	999.0	999.0
		1 ms	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.0	114.9	0.4	170.4	61534.5	0.2	17368.8	973.9	0.0	978.1
		10 ms	0.1	0.1	0.0	0.1	0.1	0.0	0.0	0.0	0.0	114.4	0.4	174.3	53264.8	0.2	13004.1	895.8	0.0	907.9
		100 ms	113.1	0.5	0.5	155.6	0.5	104.0	150.0	0.0	66.6	110.1	0.4	173.6	110.1	0.2	125.1	5.1	0.0	67.2
		1 s	112.8	0.6	0.3	112.8	0.6	0.3	0.0	0.0	0.0	90.7	0.3	175.2	90.7	0.1	0.1	0.0	0.0	0.0
		10 s	78.9	0.4	0.3	78.9	0.4	0.3	0.0	0.0	0.0	71.1	0.2	158.5	71.1	0.1	0.1	0.0	0.0	0.0
		Non-sim.	110.2	0.5	0.3	110.2	0.5	0.3	0.0	0.0	0.0	76.7	0.3	175.8	76.7	0.1	0.1	0.0	0.0	0.0
2000	30 s	1 µs	0.2	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	612.5	0.5	1162.1	799572.6	181.7	283202.8	1999.0	1999.0	1999.0
		1 ms	0.2	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	622.9	0.5	1144.4	799382.8	0.3	277178.1	1972.1	0.0	1976.6
		10 ms	0.2	0.0	0.1	0.2	0.0	0.1	0.0	0.0	0.0	617.3	0.5	1158.5	782296.1	0.2	262187.3	1897.8	0.0	1905.5
		100 ms	586.4	0.8	0.2	1091.1	0.8	992.7	1374.2	0.0	1402.9	612.9	0.5	1147.4	536632.6	0.2	177381.6	1469.2	0.0	1702.4
		1 s	618.2	0.9	0.8	716.0	0.9	258.1	33.8	0.0	5.3	569.3	0.5	1152.8	569.3	0.2	257.8	0.0	0.0	5.3
		10 s	611.6	0.7	0.8	611.6	0.7	0.8	0.0	0.0	0.0	553.3	0.4	1170.6	553.3	0.2	0.2	0.0	0.0	0.0
		Non-sim.	602.8	1.0	0.7	602.8	1.0	0.7	0.0	0.0	0.0	533.9	0.5	1165.3	533.9	0.2	0.1	0.0	0.0	0.0

Table 3.8: Improvement results for small instance sets with selection time of 30 seconds.

			# Started		# Finished		# Sol. Replaced		max. CPU (ms)		Run. Frac. (%)	
n	Sel.	Interarr.	INS+GR	INS+NS	INS+GR	INS+NS	INS+GR	INS+NS	INS+GR	INS+NS	INS+GR	INS+NS
500	30 s	1 µs	1.0	1.0	1.0	1.0	0.3	1.0	32.1	42.2	64.2	65.8
		1 ms	142.0	101.3	82.0	61.5	2.0	15.2	41.3	29.9	100.0	93.0
		10 ms	1636.4	313.2	1474.3	251.3	14.0	99.3	61.2	43.7	100.0	39.7
		100 ms	17162.3	352.4	16971.7	352.2	25.7	121.5	64.9	21.6	100.0	2.6
		1 s	178077.2	352.9	177881.8	352.9	40.6	122.3	92.2	19.9	100.0	0.3
		10 s	1792659.0	350.4	1792457.2	350.4	58.3	122.2	114.7	18.9	100.0	0.0
1000	30 s	1 µs	1.0	1.0	1.0	1.0	0.9	1.0	92.8	106.3	60.6	65.0
		1 ms	149.0	91.0	84.0	51.0	3.3	8.9	142.1	109.5	100.0	96.8
		10 ms	1249.4	377.1	1051.6	258.1	13.1	108.2	258.7	187.4	100.0	87.4
		100 ms	13035.0	829.3	12662.7	792.3	45.8	342.5	290.5	132.0	100.0	12.3
		1 s	134982.5	815.0	134589.3	815.0	62.7	331.3	350.8	82.3	100.0	1.0
		10 s	1450496.3	830.7	1450097.9	830.7	80.7	344.3	298.6	84.6	100.0	0.1
2000	30 s	1 µs	1.0	1.0	1.0	1.0	1.0	1.0	464.1	556.1	71.8	74.7
		1 ms	116.3	95.9	55.3	58.6	3.4	11.6	473.3	612.5	100.0	98.7
		10 ms	1052.8	369.8	849.6	244.7	12.3	95.2	1380.6	698.7	100.0	93.7
		100 ms	9911.3	1207.6	9355.0	958.9	44.9	501.0	1934.6	2095.2	100.0	79.9
		1 s	98007.7	2192.2	97116.4	2177.8	110.8	1012.2	1891.5	403.1	100.0	5.9
		10 s	962569.7	2195.8	961638.7	2195.8	132.6	1022.5	1950.3	342.4	100.0	0.6

The improvement of the schedule during the ordering process helps increase the quality of the solution which results in an increased number of customers that can be accepted. While the INSCON configuration improves the solution in the accept/reject procedure, both INS+GR and INS+NS continuously run improvements in the background. Table 3.8 compares the number of improvement runs started ('# Started'), number of improvement runs finished without write operations in the meantime ('# Finished'), the number of actual schedules improved ('# Sol. Replaced'), the maximum CPU time for a single run in milliseconds, and the fraction of the whole order process which was spent in total by the improvement procedures ('Run. Frac.'), averaged over the 10 instance sets. This running fraction is given by $\frac{\sum_t d_t^{\text{imp}}}{t^{\text{last}} - s}$, i.e., the sum of improvement CPU times d_t^{imp} divided by the last event time t^{last} minus the selection time s of the first arriving customer.

Except for 1 µs interarrival time, the INS+GR uses in almost all cases the complete ordering process to run improvements, while the running fraction decreases for INS+NS with longer interarrival times. This is because the GRASP method used by the INS+GR can run continuously resulting in slightly different schedules, while the neighborhood search method used by the INS+NS can become stuck in a local optimum, in which case no further runs are executed until a new customer is accepted. This difference is also reflected in the number of started and finished improvement runs.

The number of schedules actually improved is higher for the INS+NS than for the INS+GR configurations. However, the improvements found by the INS+NS are

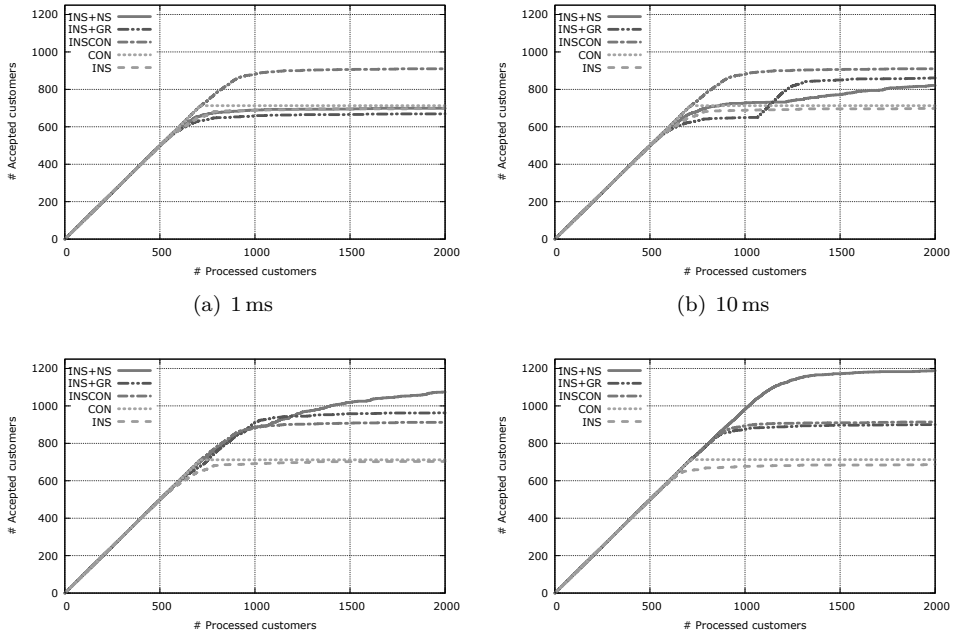
local, changing only two routes at once, while the INS+GR can replace the whole schedule at once. For long interarrival times, the INS+NS outperforms INS+GR in terms of accepted customers while only a fraction of the ordering process is actually used for improvement. As the interarrival time increases, the number of accepted customers increases and also the maximum CPU time of a single improvement run. When the interarrival time, and thereby the time between selections, becomes in the order of the maximum CPU time of a single run, the improvement procedure is unable to finish a single run before new write operations and cannot contribute to improving the schedule.

Interestingly, the maximum CPU time for a single run also increases for INS+NS when interarrival time shortens, for instance at 100 ms for $n = 2000$. This is mainly caused by the static move descriptors, which are used to speed up the neighborhood search by storing the best move for each route pair, becoming much less effective when new customers are accepted too frequently. This phenomenon is explored further in the next section, where we investigate the performance of the configurations during the ordering process.

3.5.3 During the Ordering Process

To illustrate the performance of the configurations during the ordering process, we present the results of one single instance set *DTSM_NL_2000_01* with $n = 2000$ arriving customers. The other instances sets with $n = 2000$ arriving customers show very similar characteristics.

Figure 3.3 shows the number of accepted customers during the simulation for different interarrival times. Here, we compare the number of processed customers, which indicates the combined number of customers that have been processed by the accept/reject procedure and the number of customers that decided to leave after their selection time. All configurations start by accepting all arriving customers, which accounts for the linear increase initially for all configurations and interarrival times, as can be observed in Figure 3.3. At some point this increase flattens, as some customers leave or are rejected. For the CON configuration, the number of accepted customers flattens very abruptly, irrespective of the interarrival time. While the flattening in the curves of the number of accepted customer for the INS and the INSCON configurations is more smooth, both are, as with CON, (mostly) unaffected by the interarrival times. However, both INS+GR and INS+NS are affected by the interarrival times. While in Figure 3.3(a) for interarrival time of 1 ms, they both behave similarly to INS, in Figure 3.3(b) with interarrival time of 10 ms they



(a) 1 ms (b) 10 ms

(c) 100 ms (d) 1 s

Figure 3.3: Number of accepted customers during simulation of instance DTSM_NL_2000_01 with $n = 2000$ arriving customers using the DTSM configurations for different customer interarrival times: (a) 1 ms, (b) 10 ms, (c) 100 ms, and (d) 1 s.

can find some improvements, but only after a number of customers start leaving or are rejected, which happens roughly after the 1000th processed customer. Notice INS+GR improves the schedule once to allow for much more accepted customers, while INS+NS conducts more gradual improvements of the schedule. In Figure 3.3(b) for interarrival time of 1 s, the time between customer placements is large enough for both improvement procedures to conduct numerous successful runs.

Figure 3.4 shows the number of rejected customers compared to the number of processed customers. For very short interarrival times, no customer will leave and therefore a high number of customers is rejected. This can be observed by the linear increase in rejected customers for all configurations and interarrival time of 10 ms in Figure 3.4(a), and for configurations CON and INSCON for interarrival time 100 ms in Figure 3.4(b). In the other case, in particular for longer interarrival times 1 s and 10 s, Figure 3.4(c) and Figure 3.4(d) respectively, some differences in

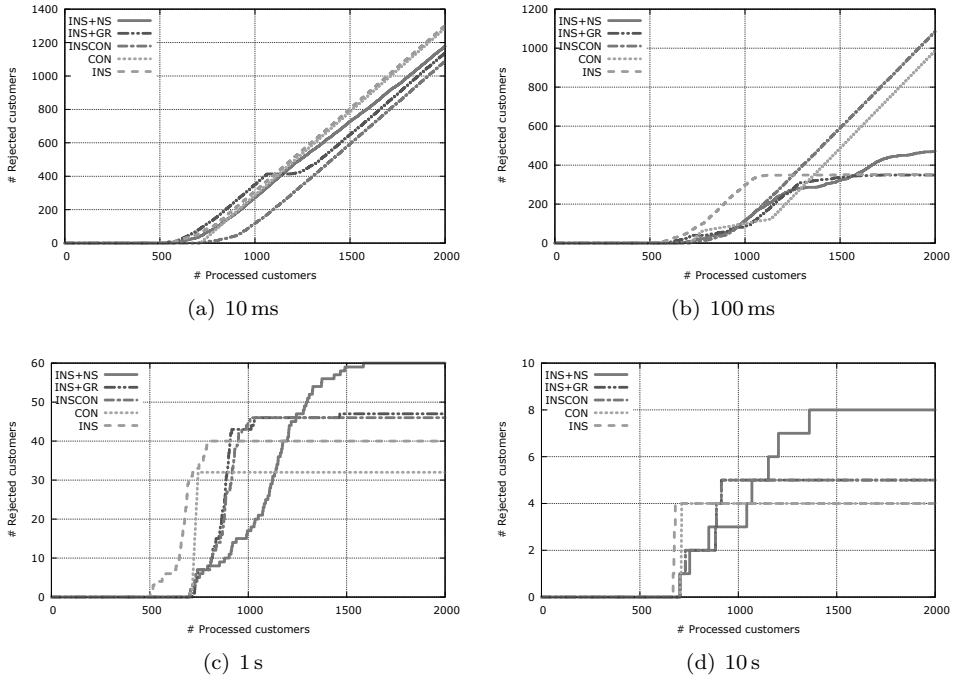
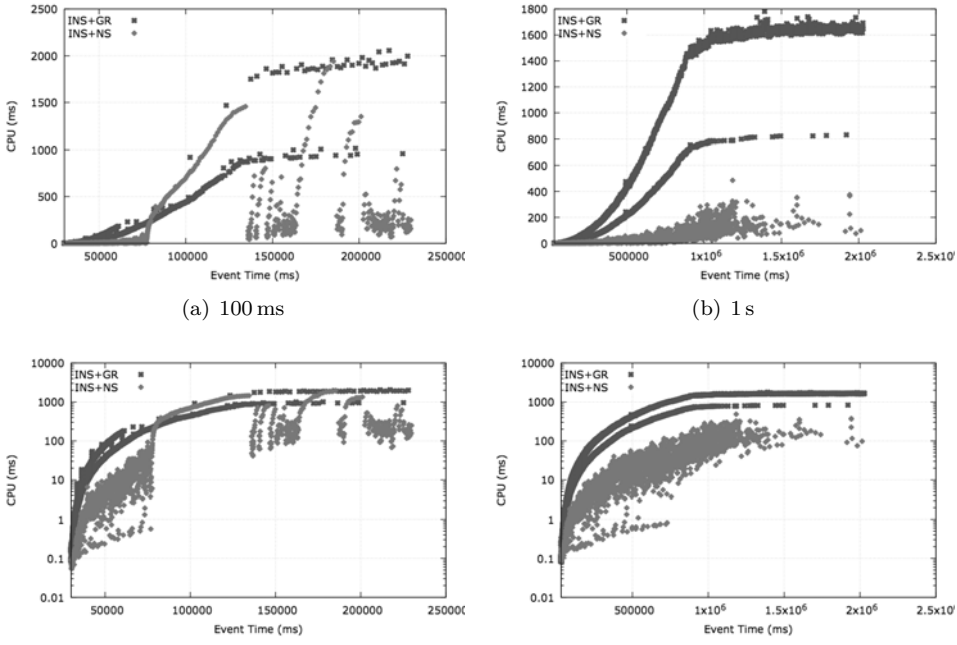


Figure 3.4: Number of rejected customers during simulation of instance DTSM_NL_2000_01 with $n = 2000$ arriving customers using the DTSM configurations for different customer interarrival times: (a) 10 ms, (b) 100 ms, (c) 1 s, and (d) 10 s.

rejected customers between the configurations can be observed. CON rejects the least customers and only during a short period of the order process. INS+NS rejects the most customers, and still does so during the late stages of the ordering process. Besides the higher number of customers which are accepted, INS+NS also changes the schedule more frequently, which might explain the higher number of rejections.

Figure 3.5 shows the CPU times for each improvement run of INS+GR and INS+NS. Here, the actual event times during the ordering process are shown. We show the results for the interarrival times of 100 ms and 1 s, and use both a regular and a logarithmic scale for the CPU times. Here, the CPU times of the runs finishing without intermediate writes as well as with (unable to finish) are plotted. The INS+GR configuration shows two 'polynomials' which flattens after some time. These two shapes are directly related to the number of accepted customers. The lower points are the first runs after each new customer acceptance with $l = 1$ random



(a) 100 ms (b) 1 s

(c) 100 ms; Logscale (d) 1 s; Logscale

Figure 3.5: Improvement run CPU times during simulation of instance DTSM_NL_2000_01 with $n = 2000$ arriving customers using the DTSM configurations for different customer interarrival times: (a) 100 ms, (b) 1 s; and, using a logarithmic scale for the CPU times: (c) 100 ms, and (d) 1 s.

possible insertions, while the upper points are the other runs with $l = 3$ random possible insertions, which require more computation time.

The number of INS+GR runs is lower for 100 ms than for 1 s. For INS+NS, an additional effect plays a role. As the CPU time for a single run approaches the interarrival time, which is roughly the time between selections, not only is the run unable to finish in time, also the static move descriptors structures are not updated. In the subsequent runs, not only all moves involving a changed route of the last executed move must be evaluated, but also all moves involving routes changed by new accepted customers. If again new customers are accepted during the new run, this problem remains and might become worse. Only when the rate at which customers are accepted starts to decrease (due to some customers leaving or being rejected), the run can finish in time and update the structures. In Figure 3.5(c) this can been seen for instance before event time 150000 ms. As the successful improvements result

in more customers to be accepted, the issue repeats again. In Figure 3.5(d), the interarrival time of 1 second is high enough such that every INS+NS improvement run can finish in time, and therefore this effect is not present here.

3.5.4 Large Instances

Both for the CON and INSCON configuration the response times increase to more than 4 minutes for $n = 2000$ customer instances. This seems too high for many practical applications, and the response time will increase further for larger instances. Therefore, for the large instance sets with $n \in \{4000, 8000\}$ arriving customers, we test only the INS, INS+GR and INS+NS configurations.

Table 3.9 shows the results for the selection time of 30 seconds. As before, improvement of the schedule in the background by INS+GR and INS+NS helps to accept more customers while keeping the response times low. INS+NS outperforms INS+GR and INS on the number of accepted customers for long interarrival times, but suffers more from short interarrival times than INS+GR. This might be due to the static move descriptors being less effective as the time of single runs become higher than the interarrival times. This can also been seen by the higher maximum CPU time for a single run with an interarrival time of 1 s. Furthermore, the number of rejected customers is also higher for the INS+NS. For all three configurations, the response times are low, and blocking only occurs for both $n = 4000$ and $n = 8000$ arriving customers when the interarrival time is 1 µs. For long interarrival times, the INS+NS is not running continuously as it gets stuck in local optima. However, for the more difficult $n = 8000$ instances, more iterations are needed before a local optimum is reached. Therefore, more improvement runs are executed and also the running fraction is higher in absolute sense, compared to the smaller instances.

Table 3.9: Results for the large instance sets.

			# Accepted Customers			# Leaving Customers			# Rejected Customers			max. Response (ms)						Improvement			
												Time Slot Offer			Accept/Reject			max CPU (ms)		Run. Frac. (%)	
n	Sel.	Interarr.	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS	INS+GR	INS+NS	INS+GR	INS+NS
4000	30 s	1 µs	1498.9	1513.4	1497.0	0.0	0.0	0.0	2501.1	2486.6	2503.0	0.1	0.1	0.1	764.4	793.7	753.7	2925.2	3198.0	78.6	80.8
		1 ms	1503.0	1512.4	1532.2	0.0	0.0	0.0	2497.0	2487.6	2467.8	0.1	0.1	0.1	0.5	0.8	0.7	2891.8	3470.2	100.0	99.6
		10 ms	1503.0	1648.8	1564.5	0.0	0.0	0.0	2497.0	2351.2	2435.5	0.0	0.1	0.1	0.5	0.7	0.9	4682.3	3624.7	100.0	97.1
		100 ms	1505.6	2177.1	2134.7	2111.0	959.8	1150.2	383.4	863.1	715.1	2.3	1.7	2.1	0.4	0.6	0.7	13078.9	10014.1	100.0	90.1
		1 s	1506.8	2235.1	2705.9	2449.0	1671.8	1160.0	44.2	93.1	134.1	2.3	3.0	2.8	0.4	0.6	0.6	16701.2	14074.4	100.0	51.7
		10 s	1505.3	2168.6	2736.3	2490.1	1826.2	1257.7	4.6	5.2	6.0	2.1	3.7	1.7	0.3	0.5	0.3	15899.6	1160.3	100.0	2.7
8000	30 s	1 µs	3387.8	3375.9	3370.5	0.0	0.0	0.0	4612.2	4624.1	4629.5	0.1	0.1	0.1	3046.9	3065.9	3150.7	21938.2	18906.0	87.7	85.7
		1 ms	3386.0	3412.4	3390.7	0.0	0.0	0.0	4614.0	4587.6	4609.3	0.1	0.1	0.1	2.7	1.4	1.5	22594.2	19270.0	100.0	99.9
		10 ms	3390.0	3379.1	3445.0	1345.5	1360.2	1293.4	3264.5	3260.7	3261.6	3.8	3.1	3.0	1.2	1.5	1.2	22337.2	20354.2	100.0	98.9
		100 ms	3398.2	4225.1	3948.9	4203.5	3164.9	3436.8	398.3	610.0	614.3	4.0	5.3	5.2	1.2	1.1	1.2	71924.3	27089.5	100.0	95.1
		1 s	3403.8	5177.2	5091.3	4554.9	2671.3	2674.5	41.3	151.5	234.2	4.1	5.4	6.6	0.8	1.1	1.6	251786.9	44650.3	100.0	80.7
		10 s	3401.3	5135.6	6083.3	4595.0	2854.1	1905.6	3.7	10.3	11.1	4.0	6.0	3.0	0.7	1.3	0.8	293546.1	3296.2	100.0	10.5

3.6 Discussion and Future Research

In this section, we comment on some fundamental choices we have made in the design of our DTSM model and procedures, and provide directions for future research.

3.6.1 Rejection

In this chapter, to cope with selection time we propose to include the possibility to reject a customer if a time slot gets invalidated. Of course, selection time could be dealt with without ever having to reject a customer. For instance, customers can be forced to wait for a time slot offer, until all previous customers have made a selection. Now a time slot offer is never invalidated, so no rejection occurs, but the number of customers that can be processed this way is severely limited.

A more fruitful approach might be to keep track of current time slot offers and considering these when making a new time slot offer. This way a time slot offer can be made that remains valid for any selection by the previously arrived customers. Here we are faced with a combinatorial explosion of the number of contingencies to account for. Furthermore, accounting for all these contingencies has the drawback that new time slot offers might be overly conservative, leading to less orders being placed. This issue could be remedied by deliberately limiting the number of time slots offered to customers, which effectively limits the number of contingencies to account for. We believe such an approach merits further investigation.

3.6.2 Feasibility Guarantee

We have deliberately chosen to only accept customers when a guarantee is found that a feasible schedule exists. Finding this guarantee affects the complexity of the accept/reject procedure, at the expense of computation time. As a result, in the design and selection of an accept/reject method, there is a trade-off between the response time and the number of unnecessary rejections. Furthermore, depending on the used method, that a guarantee cannot be found does not necessarily mean a feasible schedule does not exist. After the cut-off time, more time might be available which could be used to find a feasible schedule. In the future, alternative procedures could be designed in which a customer can be accepted without having an explicit guarantee that a feasible schedule exists. The trade-off here is between the number of customers that can be accepted in a limited response time, and the likelihood of infeasibility.

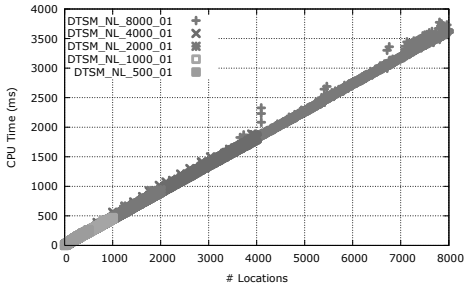


Figure 3.6: Calculation times of a nominal travel time matrix update ($2 \times n$ travel times) using *OSRM*.

3.6.3 Travel Time Computation

In our experiments, we have implicitly assumed that the travel time functions for any pair of customers are already precomputed. This is not the case in all applications which necessitates the computation of travel times when a customer arrives in the system. Even more so, in our experience with industry partners, we have seen that the time to compute these travel times is not negligible. To illustrate this, in Figure 3.6 we have plotted the computation time in milliseconds of updating a nominal travel time matrix for up to 8000 already accepted customers, using the software package *OSRM*. Observe that updating a travel time matrix requires the computation of the travel time from a new customer to all currently accepted customers and the reverse. For instance for 1000 customers, around 500 milliseconds are needed, and for 6000 customers around 2750 milliseconds are needed. This severely impacts the response times of a DTSM procedure. Further research is required to limit this impact. It might for instance not always be necessary to fully update the travel time matrix.

3.6.4 Implementation

We highlight some of the choices in our implementation of the DTSM procedures, that could be tweaked for particular applications. Firstly, as a cost criterion in our heuristics, we used average travel time. We can imagine that other cost criteria may lead to better results, like inserting customers at the position that instead maximizes some slack or regret measure. Furthermore, we have chosen to store only one current schedule in memory. Alternatively, multiple schedules might be stored, which could improve the effectiveness of insertion procedures at the expense of increased computation efforts.

In our experiments, we have chosen for a specific set-up of parallel processes. We assume an unlimited amount of threads are available for making time slot offers, one for accept/reject procedures and one for improvement procedures. In practice, given a limited amount of threads, the procedures should be allocated to threads, potentially dynamically. Further investigation is required to find good allocation schemes.

3.7 Conclusion

In this chapter, we investigate the real-time performance of Dynamic Time Slot Management for attended home delivery. We address two key complications that stand in the way of application in practice, which arise from simultaneous interaction of customers with a DTSM system in real-time: i) additional waiting time, ii) invalidation of time slot offers.

We formulate a DTSM model incorporating these simultaneous interactions and propose a framework which consists of time slot offer, accept/reject and improvement procedures, which can be run in parallel. To guarantee the existence of a feasible delivery schedule of the set of accepted customers, the time slot selected by the customer is re-evaluated in a queue. Effects such as *blocking* procedures and *invalidated* time slots arise, which increase the response time of the system, as well as possibly rejecting customers *after* selecting a time slot.

We modify state-of-the-art DTSM procedures proposed in the literature to fit this framework. An extensive computational study provides valuable insights in the effects of customer selection time on the number of customers that are rejected, as well as insights in the maximum decision and response times. Also, we show the trade-off between number of accepted customers and responsiveness of the procedures, which emerges as customers arrival rates increase, especially when the time between customer placements becomes in the order of the decision times.

Our investigation opens many possible directions for further research. Our framework includes an improvement procedure, which utilizes the time between customer placements to improve the schedule in memory. We use a GRASP construction heuristic and an k-exchange neighborhood search procedure, but more elaborate improvement schemes might be more effective. For instance, a variable neighborhood search can use small neighborhoods when the time between customer placements is short, while larger neighborhoods can be used when the time between placements is long. Also, to avoid getting stuck in local optima, meta-heuristic techniques proposed

Table 3.10: Complexities of the procedures within the investigated DTSM configurations.

	Time Slot Offer	Accept/Reject	Improvement
CON	$\mathcal{O}\big(M^2n^2p + \lvert\mathcal{T}\rvert\, Mnp\big)$	$\mathcal{O}\big(M^2n^2p\big)$	–
INS	$\boldsymbol{\mathcal{O}(\lvert\mathcal{T}\rvert\, Mnp)}$	$\boldsymbol{\mathcal{O}(Mnp)}$	–
INSCON	$\boldsymbol{\mathcal{O}(\lvert\mathcal{T}\rvert\, Mnp)}$	$\mathcal{O}\big(M^2n^2p\big)$	–
INS+GR	$\boldsymbol{\mathcal{O}(\lvert\mathcal{T}\rvert\, Mnp)}$	$\boldsymbol{\mathcal{O}(Mnp)}$	$\mathcal{O}\big(M^2n^2p + Mn^2 \log l\big)$
INS+NS	$\boldsymbol{\mathcal{O}(\lvert\mathcal{T}\rvert\, Mnp)}$	$\boldsymbol{\mathcal{O}(Mnp)}$	$\mathcal{O}\big(M^2nk^2p\big)$

for traditional vehicle routing problems, such as destroy and recreate operators in adaptive large neighborhood search, can be used.

Appendix

3.A Complexities of the DTSM Procedures

Table 3.10 shows the complexities of the DTSM procedures within the configurations presented in Section 3.4.5. Bold is used to indicate the lowest complexity among the configurations. Here, n is the number of accepted customers, $|\mathcal{T}|$ is the number of time slots, p is the highest number of breakpoints among the time-dependent travel time functions, l is the number of best moves used in GRASP and k is the maximum exchange segment length in the neighborhood search. We assume that the number of customers in a route is limited to M. This can be due to capacity restrictions, or due to time restrictions. Note that $M \sim \mathcal{O}(n)$ in case the routes are not restricted in number of customers. Most complexity results are discussed in Chapter 3. A greedy construction requires $\mathcal{O}\big(M^2n^2p\big)$ operations and a single k-exchange neighborhood search iteration requires $\mathcal{O}\big(M^2nk^2p\big)$ operations. We note that the accept/reject procedure for INS and related configurations requires $\mathcal{O}(Mnp)$ operations, since in worse-case all n possible insertion positions have to be checked, each requiring $\mathcal{O}(Mp)$ operations (see Chapter 3). A time slot offer for INS and related configurations requires this number of operations for each time slot, resulting in $\mathcal{O}(|\mathcal{T}|\, Mnp)$ operations. Finally, a single run of the INS+GR improvement procedure requires $\mathcal{O}\big(M^2n^2p + Mn^2 \log l\big)$ operations. The first term is the same as for a greedy construction, while the second term corresponds to the work needed to keep the list of l best insertions sorted during each iteration of the construction.

Chapter 4

Strategic Time Slot Management: A Priori Routing for Online Grocery Retailing

Thomas R. Visser and Martin W.P. Savelsbergh

4.1 Introduction

This chapter studies a novel variant of Time Slot Management inspired by its application in online grocery retailing. Online retailing continue to grow, and consumers, but also small businesses, purchase more and more products online. Many of these products require *attended* delivery, i.e., delivery can only take place when the consumer is present. Examples of such products include, furniture, white goods, and groceries. Delivery failures, for instance when the consumer is not at home, are costly, because products have to be returned and stored, and deliveries have to be re-scheduled. In case of groceries, the cost of a delivery failure may be even higher, as most grocery products are perishable and re-delivery of the (same) product is not possible. To minimize the chance of delivery failures, online retailers typically allow their customers to choose a delivery time slot. These time slots can range from 1 to 6 hours, and can have different delivery fees. The retailer agrees to deliver any ordered products in the time slot chosen by the customer. While offering delivery

This chapter is based on Visser and Savelsbergh (2019).

time slots improves customer service and reduces the chance of delivery failures, its implementation is challenging as time slot management needs to balance the benefit of accepting orders and the cost of delivering accepted orders.

In this chapter, we focus on time slot management for online grocery retailers. During the ordering process, customers log-in on a grocery retailer's website, fill their order basket, and place their order by selecting an available (delivery) time slot. Customers can select from among a set of available time slots in the few days (or even weeks). At the cut-off time for deliveries on a particular day, usually 12 to 18 hours before the departure of the delivery vehicles, vehicle routes and schedules for delivery of the placed orders are generated, and, afterwards, picking of the orders can commence in the fulfillment center. Then, if no issues arise during the execution of the delivery routes, customers receive their placed orders during their selected time slot. Online grocery retailers face varying daily demand (i.e., the number and the location of customers placing orders on any given day), but, because of the recurring nature of grocery purchases, also observe recurring patterns. Customers often have a favorite time slots and delivery days, and often order with some regularity, e.g., every week or other week.

Time Slot Management (TSM), as the name suggests, refers to the methods employed to manage the availability of time slots during the ordering process. TSM methods can be divided in two classes: static and dynamic (Agatz et al., 2013). Static Time Slot Management, sometimes called static slotting, partitions the set of customer locations into geographical regions and limits the set of available time slots in a particular region (Agatz et al., 2011; Hernandez et al., 2017); sometimes pricing of time slots is also considered (Klein et al., 2019). Dynamic Time Slot Management, sometimes called dynamic slotting, changes the availability of time slots in real time based on previously accepted orders. While most methods proposed in the literature are based on vehicle routing, some methods use static time slot order limits for each geographical region (Bruck et al., 2018). These limits on the number of customer orders that can be placed in a region are determined upfront, and are based expected demand and expected delivery capacity, like in Static TSM. With static time slot order limits, the management of time slots during the ordering process is straightforward, but determining the regions and the limit for each region, given unknown varying demand is challenging. Moreover, the number of vehicles required to deliver orders can vary from day to day. After the cut-off time, vehicle routes and schedules are generated, and only upon completion of this process is the number of vehicles required known, which may cause operational challenges. Furthermore,

picking of customer orders at the fulfillment center can only start once the vehicle routes and schedules are generated, because orders delivered in the same route are typically picked together. Dynamic Time Slot Management methods based on vehicle routing typically maintain partial delivery routes and schedules during the ordering process and use these to guide decisions on the availability of time slots (Campbell and Savelsbergh, 2005; Ehmke and Campbell, 2014; Köhler et al., 2019) – sometimes pricing of deliveries is also considered (Campbell and Savelsbergh, 2006; Cleophas and Ehmke, 2014; Yang et al., 2016). By maintaining partial vehicle routes and schedules it becomes easier to adjust to variations in demand and it allows control of the number of vehicles required to deliver orders. After the cut-off time, the vehicle routes and schedules are re-optimized. Again, picking of customer orders at the fulfillment center can only start once the vehicle routes and schedules have been finalized. Clearly, the management of time slots during the ordering process is much more involved than with static time slot order limits, especially if detailed and accurate vehicle routes and schedules are maintained, e.g., accounting for time-dependent travel times and driver breaks (Ehmke and Campbell, 2014). Therefore, it is not surprising, that online grocery retailers are continuing to look for business models that simplify time slot management without reducing customer service and increasing delivery costs.

In this chapter, we investigate a novel variant of TSM which we call *Strategic Time Slot Management* (STSM). It is motivated by the current practice of online grocery retailer Picnic in The Netherlands (`https://picnic.app/nl/`). Picnic raised a record €100 million in venture funding after just 1.5 years of operations (Sterling, 2018). Picnic operates in all major city centers of the Netherlands and delivers customer orders using small electric delivery vehicles with limited range and capacity. Picnic does not charge a delivery fee to their customers. Picnic offers only a single 1-hour time slot each day, but it varies over the days of the week (e.g., the same single 1-hour time slot is available to a customer every Monday, but a different time slots is available every Tuesday). This suggests that Picnic designs, for each day of the week, *a priori* routes, covering all customer locations, and assigns time slots to the customer locations visited on these a priori routes *before* the order placement process. Then, during the ordering process, the availability of time slots is managed using these a priori routes. Customers that place an order in a time slot available to them are inserted in a delivery route associated with the a priori route. A time slot for a customer location in an a priori route is available as long as the location can be feasibly visited given the orders that have already been placed; otherwise, the time

slot is no longer offered. Managing time slots in this way is easy if delivery routes "follow" the a priori routes, skipping locations that can no longer be visited during the assigned time slot.

Employing STSM implies that fewer time slots are available to customers, but it allows for more cost-effective operations and offering free delivery to customers. Picnic's success indicates that is a winning business proposition. Because a priori routes are designed to be operated for a period of time, it is easier to incorporate knowledge of the order patterns of customers, which can have advantages especially in urban areas with large number of densely distributed customers. Maybe more importantly, no (re)optimization of vehicle routes and schedules is required after the cut-off time, which means that a later cut-off time can be offered to customers. And not only that, picking at the fulfillment center can usually start *before* the cut-off time, as soon as (the initial part of) the delivery routes are known because of orders that have already been placed. This not only improves efficiency of fulfillment center operations, it, again, allows the cut-off time to be pushed later. Finally, the use of a priori routes induces delivery route consistency, which implies that drivers familiarize themselves with their delivery routes, which is helpful in densely populated city centers and improves customer service (Kovacs et al., 2014).

Clearly, the design of the a priori routes and time slot assignments is critical to the success of STSM. This design problem shares some characteristics with the stochastic vehicle routing problems studied in the literature (see Gendreau et al. (2014); Oyola et al. (2017, 2018) for recent surveys on stochastic VRPs). However, much of the research on stochastic VRPs has focused on uncertainty regarding the *size* of demand. In the context of STSM, it is not the size of demand that matters most, because vehicle capacity is rarely constraining, but rather the uncertainty regarding the placement of orders (i.e., stochastic customer presence), because it is the time available to make deliveries that is constraining. The concept of, or use of the term, a priori routes is also quite common (see Campbell and Thomas (2008a) for a survey on a priori routing). In all the considered settings, there is a design phase, in which a priori routes have to be determined, and an execution phase, in which the uncertain quantity is revealed and the a priori routes have to be executed, given a set of *recourse actions* or *penalities* to handle situations in which certain constraints are violated. Typically, the a priori routes are constructed using probabilistic information about the uncertain quantity and the set of recourse actions or penalties. A common approach is to formulate the (design) problem as a two-stage stochastic program.

For stochastic customer presence, most research has focused on the Probabilistic

Traveling Salesman Problem (PTSP) and its variants (see for instance Jaillet, 1988; Campbell and Thomas, 2008b; Voccia et al., 2013; Angelelli et al., 2017). The basic problem seeks an a priori route with minimum expected cost. Campbell and Thomas (2008b) consider a variant in which customers have deadlines and Voccia et al. (2013) consider a variant in which customers have time windows. Erera et al. (2009) study a related a priori routing problem in which customers with time windows are assigned to both a primary and a backup route, and once customer presence is revealed, recourse actions can move the customers from the primary to a backup route to improve costs or to recover feasibility. The setting we consider is somewhat different, in the sense that there are no recourse actions or penalties. Time slot management ensures the feasibility of the delivery routes at the end of the ordering processing, i.e., customers are revealed sequentially and are only shown time slots for which delivery is still feasible. To the best of our knowledge, such a setting, i.e., an ordering process with time slot management, has not yet been considered in the a priori routing literature.

The planning problem at the heart of STSM seeks not only a set of a priori routes, but also the assignment of a time slot to each of the locations visited on the a priori routes. The assignment of time slots to vehicle routes has been investigated in other contexts. In the Time Window Assignment Vehicle Routing Problem (TWAVRP) (Spliet and Gabor, 2015; Spliet and Desaulniers, 2015) time slots have to be assigned to customers before their demand is known, and vehicle routes are generated only when demand is revealed. Spliet and Gabor (2015) formulate the problem as a two-stage stochastic program and develop a branch-and-cut-and-price approach. The Consistent Vehicle Routing Problem (Groër et al., 2009), and its singe-vehicle variants (Subramanyam and Gounaris (2016), Subramanyam and Gounaris (2018)), seek vehicle routes that are to be executed on multiple days, with known, but varying customer presence on each of the days, and that minimize the difference in the time of service of a customer on the different days. In these settings all customer demand needs to be served, whereas in our setting time slot management during the ordering process may result in some realized customer demand not being served. Furthermore, in our setting a priori routing and time slot assignment are integrated into a single planning problem, rather than treating these aspects separately.

The contributions of this chapter are as follows:

- We introduce the concept of strategic TSM, a novel variant of TSM inspired by operations of a Dutch online grocery retailer.
- We derive a number of properties of the single-vehicle variant of the strategic TSM planning problem, and use these observations to develop an efficient dy-

namic programming algorithm for exactly calculating the expected revenue of an a priori route.

- We present a two-stage stochastic programming formulation for the single-vehicle variant of the strategic TSM planning problem, and develop a Monte-Carlo Sample Average Approximation (SAA) Method (Kleywegt et al., 2002). Time slot management during the ordering process, i.e., when orders are sequentially revealed, leads to an "evaluation" recourse, which has not been seen in the context of SAA.

- We propose a number of heuristic methods for the single-vehicle variant of the strategic TSM planning problem, which balance quality and solution time.

- We compare solution approaches on instances with up to 12 customer locations on an a priori route, which implies around $1.3 \cdot 10^9$ possible customer arrival scenarios. Moreover, we investigate the impact of different time slot configurations (time slot width and whether or not time slots overlap) and the relation between the duration of an a priori route and the target duration of the delivery route (which impacts time slot management).

The chapter is structured as follows. In Section 4.2, we introduce the Strategic TSM problem, focusing specifically on the "simpler" single-vehicle case, and we present some observations that help guide the design of solution approaches. In Section 4.3, we discuss algorithms to exactly evaluate the expected revenue of an a priori route, which is a core component of our solution approaches. In Section 4.4, we formulate the problem as a two-stage stochastic program and propose an SAA method for its solution. In Section 4.5, we present heuristic solution approaches seeking to balance quality and solution time. In Section 4.6, we discuss the results of an extensive computational study. Finally, in Section 4.7, we present concluding remarks and suggest future research directions.

4.2 Problem Description

We consider a retailer that offers its online customers a small number of time slots during which a delivery can take place. The retailer has a fleet of identical vehicles to make deliveries. Each vehicle starts and ends its delivery route at the retailer's fulfillment center. For each of its customers, the retailer knows the delivery location, the order size, the revenue, and the order placement probability. Observe that the

only stochastic feature in this setting is whether or not a customer places an order. In practice, a customer's order size and revenue are likely to be stochastic as well.

Customers can place an order up to a cut-off time, some hours before delivery will take place. We assume that the likelihood that a customer places an order is independent of the delivery time slots offered and is not correlated to the order placement of other customers. When placing an order, a customer must select a delivery time slot during which delivery will take place at his delivery location. A vehicle that arrives early at a delivery location must wait.

The retailer seeks to *design* a set of delivery routes, such that each customer, i.e., its delivery location, is visited on at least one of the routes, and associated time slots, one for each location visited, so as to maximize the *expected* revenue.

We assume that the set of possible time slots that can be assigned to a customer location has already been decided. The time slots may overlap, but they all have the same width, and they cover the entire planning horizon. The subset of possible time slots for a delivery location contains those time slots that are feasible for that location, i.e., that overlap with the time period defined by the earliest time a vehicle can reach the location and the latest time a vehicle can depart the location to return to the fulfillment center before the end of the planning horizon.

As will become evident soon, even solving the special case in which the retailer has only a single vehicle with infinite capacity and assigns only a single time slot to each delivery location is surprisingly challenging and gives rise to insightful observations. For the remainder, therefore, we focus on this special case, leaving the general case for future research.

4.2.1 The Single-Vehicle Case

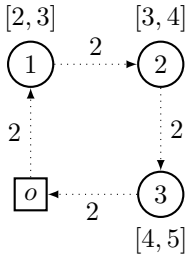
The problem is defined on a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, with $\mathcal{V} = \mathcal{V}_c \cup \{o, d\}$ the set of vertices, where $\mathcal{V}_c = \{1, 2, \ldots, n\}$ is the set of customer delivery locations and where, for convenience, we represent the fulfillment center with a start and an end node, o and d, respectively, to be able to distinguish the departure and return of the vehicle, and with $\mathcal{A}$ the set of (directed) arcs connecting the nodes. We let $t_{ij} \geq 0$ denote the travel time associated with arc $(i, j) \in \mathcal{A}$. We assume that service times are included in the travel times, and that travel times satisfy the triangle inequality. The retailer has a single vehicle with unlimited capacity to make deliveries, which reflects that it is time rather than capacity that restricts the delivery route. When a time slot $s = [a_s, b_s]$ is assigned to a location in the delivery route, the earliest time a delivery can be made at that location is a_s and the latest time a delivery can

be made at that location is b_s. A vehicle arriving early must wait at the location. A set $\mathcal{T}$ of possible time slots to be assigned to delivery locations is given. The time slots in $\mathcal{T}$ may overlap, but we assume their width is equal, and they cover the entire planning horizon $[0, T]$, with T the planning horizon. The set of possible time slots $\mathcal{T}_i \subset \mathcal{T}$ for location i contains the time slots which overlap with the period defined by the earliest time a vehicle reach that location, i.e., $t_{o,i}$, and the latest time a vehicle has to depart from that location (to return to the fulfillment center before the end of the planning horizon), i.e., $T - t_{i,d}$. The fulfillment center has time window $[a_o, b_o] = [a_d, b_d] = [0, T]$. We identify the set of customers $\mathcal{C}$ with their delivery locations, i.e., $\mathcal{V}_c$ (and use these interchangeably from now on). Each customer $c \in \mathcal{C}$ has an order placement probability $p_c \in (0, 1]$, and, when served, results in a revenue r_c for the retailer. We assume that order placement probabilities are iid and independent of the time slot assigned to the delivery location.

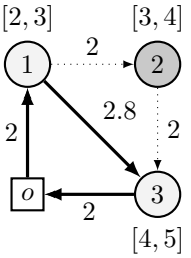
The retailer seeks to design an *a priori* delivery route, visiting all delivery locations, and associated time slots, one for each location, so as to maximize the *expected* revenue.

Let Ω be the set of all possible scenarios of order placements. A single scenario $\omega \in \Omega$ can be described by a sequence of delivery locations, representing which customers have placed an order and in what sequence – the exact times of the order placements are not important. Furthermore, we assume that each permutation of customer placements is equally likely, meaning that there is no dependence between customers and their position in the sequence. Each customer is equally likely appear early in the sequence as to appear late in the sequence. (Note that when the order placement probabilities are equal, i.e., $p_c = p$ for $c \in \mathcal{C}$, all possible scenarios are equally likely – given the iid assumption.)

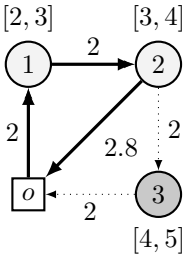
The revenue for a scenario $\omega \in \Omega$ is determined as follows. During the order placement phase, an arriving order is inserted in the *actual* delivery route, i.e., the delivery route to be executed after the cut-off time, based on the delivery location's position in the a priori route. That is, the delivery location is inserted in the actual delivery route after the delivery locations of orders placed earlier and that precede it in the a priori route, and before the delivery locations of orders placed earlier and that succeed it in the a priori route. After the insertion of an order, any delivery location that has become time infeasible, i.e., for which it is no longer possible to make a delivery during its assigned time slot, is removed, and orders for these locations will be skipped from that point on. After all orders in ω have been processed, i.e., have either been inserted or skipped, the revenue of the scenario is simply the sum



(a) An a priori route with assigned time slots.



(b) Delivery route for customer realization $(3, 1, 2)$. After customers 3 and 1 place an order, customer 2 cannot by infeasibility.



(c) Delivery route for customer realization $(2, 1, 3)$. Now customer 3 cannot place an order.

Figure 4.1: Small example of Strategic Time Slot Management.

of the revenues of the orders that have been inserted in the actual delivery route. The expected revenue for an a priori route is the sum of the revenues of all possible scenarios for that a priori route weighted by the probability of occurrence of the scenarios.

Observe that (to keep operations simple) the delivery locations in the actual delivery route are visited in the same order as in the a priori route.

4.2.1.1 Small Example

To illustrate the single-vehicle problem, we now present a small example. Let us consider three customer locations $\mathcal{V}_c = \{1, 2, 3\}$ with coordinates $\{(2, 0), (2, 2), (0, 2)\}$, respectively, and a fulfillment center located at coordinates $(0, 0)$ with a planning horizon $[0, T] = [0, 7]$. Each customer location has equal order probability $p = \frac{1}{2}$ and equal revenue $r = 1$. The set of possible time slots is given by $\mathcal{T} = \{[0, 1], [1, 2], \ldots, [6, 7]\}$. The travel times are given by the Euclidean distances.

A solution to the single-vehicle problem consist of an a priori route and a time slot assignment. Let us consider the a priori route $\rho = (o, 1, 2, 3, d)$ and time slot assignment $\{[2, 3], [3, 4], [4, 5]\}$ for locations $\{1, 2, 3\}$, respectively. This solution is shown in Figure 4.1(a).

Suppose now, with this design in place, that customer 3, then customer 1, and then customer 2, (each in $\mathcal{C}$) seek to place an order. This corresponds to scenario $\omega = (3, 1, 2) \in \Omega$. After placement of customer 3, the delivery route will be $(o, 3, d)$,

and after placement of customer 1, the delivery route will be $(o, 1, 3, d)$, since each customer is inserted at their corresponding position in the a priori route and both can be inserted without violating time slots or the fulfillment center time window. This is shown in Figure 4.1(b). Notice that now, insertion of customer 2 at its position in the a priori route, resulting in delivery route $(o, 1, 2, 3, d)$, will violate the fulfillment center time window. This latter route requires a total of 8 time units while the fulfillment center time window is $[0, 7]$. Therefore, the time slot for customer 2 will be removed, and this customer cannot place an order. The resulting delivery route has revenue 2. In Figure 4.1(c), we now consider scenario $\omega = (2, 1, 3) \in \Omega$. Now customer 3 cannot place an order, resulting in a different delivery route $(o, 1, 2, d)$ with revenue 2. Notice that not only which customers arrive, but also the sequence in which they arrive determines the eventual delivery route. We have to take this into account when designing the a priori route and time slot assignment.

4.2.1.2 Observations

We highlight some interesting (and possibly unexpected) observations regarding this single-vehicle design problem.

Observation 4.1. *When the planning horizon T is greater than or equal to T^{TSP}, the minimum duration tour visiting all customer locations, the single-vehicle problem is trivial. An optimal a priori route is a minimum duration tour and an optimal time slot assignment is one in which the time slot assigned to a customer location contains the arrival time of the tour at that location. The expected revenue is $\sum_{i \in \mathcal{V}_c} p_i r_i$.*

It is natural to think that by increasing the planning horizon T, i.e., the time available for the delivery route, the expected revenue will increase (or at least not decrease). However, it turns out that this is not always true.

Observation 4.2. *Increasing the planning horizon T, i.e., the delivery route time limit, for a given a prior route may decrease the expected revenue.*

Figure 4.2 shows a simple example of this with three customers. Customers have probability $p = \frac{1}{2}$ and coordinates $\{(0, 2), (0, 1), (0, -1)\}$, respectively, and the travel times are equal to the Euclidean distances. Customer 1 has revenue 10, and customers 2 and 3 each have revenue 1. Suppose all customers are assigned time slot $[0, 6]$. When increasing the planning horizon from $6 - \epsilon$ to 6, the expected revenue (associated with the optimal solution) decreases from 4.646 to 4.625. The reason is that when the planning horizon is $6 - \epsilon$, the revenue for scenario $\omega = (2, 3, 1)$ is 11

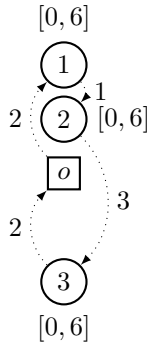


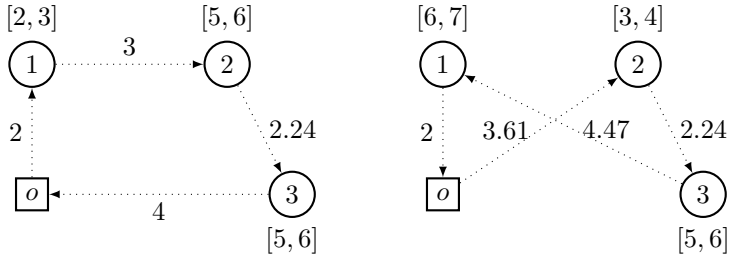
Figure 4.2: A design for a small example with three customers for which increasing the planning horizon from $6 - \epsilon$ to 6 results in a decrease in expected revenue.

(customer 3 cannot place order), but when the planning horizon is 6, the revenue reduces to 2 (customer 3 can place order, but customer 1 not). This decrease in revenue in this scenario is more than the increase in revenue in the other scenarios, and therefore the expected revenue decreases. We observe this effect also in larger instances, even when customers have equal revenue $r = 1$, equal probability $p = \frac{1}{2}$, and the minimum duration tour is taken as the a priori route.

It is natural to think that the a priori route has to be a minimum duration tour visiting all customer locations. However, it turns out that this is not always true.

Observation 4.3. *The optimal a priori route is not always a minimum duration tour.*

We can show this with an example, depicted in Figure 4.3, with three customers, each with probability $p = \frac{1}{2}$ and revenue $r = 1$, planning horizon $T = 9.85$, and possible time slots $\mathcal{T} = \{[0, 1], [1, 2], \ldots\}$. Customers have coordinates $\{(2, 0), (2, 3), (0, 4)\}$, respectively, and the travel times are equal to the Euclidean distances. Figure 4.3(a) shows the optimal design when the a priori route is forced to be a minimum duration tour (the design decision involves the direction in which the minimum duration tour is traversed in the time slot assignment). The expected revenue is 1.083. In this design, customers $\{1, 2\}$ can be served together, but customers $\{2, 3\}$ cannot be served together due to their assigned time slots, and customers $\{1, 3\}$ cannot be served together due to the available time in planning horizon. Figure 4.3(b) shows the optimal design when the a priori route is not forced to the minimum duration tour. The expected revenue is 1.25. In this design, customers $\{1, 2\}$ and customers $\{2, 3\}$ can be served together, while only customers $\{1, 3\}$ can-



(a) An optimal solution with expected revenue 1.083 when a priori route is fixed to the TSP optimal tour.

(b) An optimal solution with expected revenue 1.25, without fixing the a priori route.

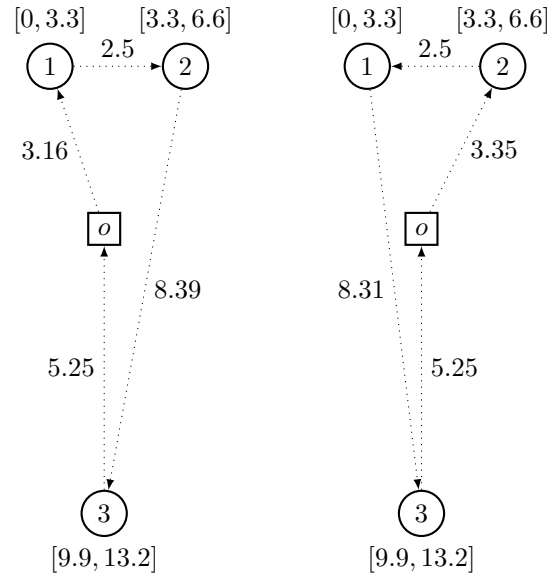
Figure 4.3: Two designs for a small example with three customers and planning horizon $T = 9.85$.

not be served together.

It is natural to think that the time slots along the a priori route should be *ascending* in both start- and end times, i.e., the start- and end times of the time slots increase along the a priori route. However, it turns out that this is not always true.

Observation 4.4. *In an optimal design, the time slots assigned to the customer locations are not always ascending along the a priori route.*

We can show this with an example, depicted in Figure 4.4, of three customers with equal probability $p = \frac{1}{2}$ and revenue of 1 for customers 1 and 2 and revenue of 5 for customer 3. Customers have coordinates $\{(-1, 3), (1.5, 3), (0, -5.25)\}$, respectively, and the travel times are, again, equal to the Euclidean distances. The set of possible time slots is $\mathcal{T} = \{[0, 3.3], [3.3, 6.6], \ldots\}$ and the planning horizon is $T = 19.29$. Figure 4.4(a) shows the optimal design when time slots are forced to be ascending along the a priori route, which results in an expected revenue of 3.208. Figure 4.4(b) shows the optimal design when there are no restrictions on the time slot assignments, which results in an expected revenue of 3.250. Note that time slot of customer 2 is $[3.3, 6.6]$ and that the time slot of the next customer on the a prior route, i.e., customer 1, is $[0, 3.3]$. The increase in expected revenue is due to scenarios $\omega = \{1, 2, 3\}$ and $\omega = \{2, 1, 3\}$. When the time slots are forced to be ascending, the resulting design allows customers 1 and 2 to place their orders, which prevents customer 3 the higher revenue to place its order. When the time slot assignment is unrestricted, this situation is averted. We observe this effect also in larger instances, even when customers have equal revenue $r = 1$ and probability $p = \frac{1}{2}$.



(a) Time slots restricted to be ascending; expected Revenue: 3.208.

(b) Time slots not restricted; expected Revenue: 3.250.

Figure 4.4: Two designs for a small example with 3 customers and planning horizon $T = 19.29$.

4.3 Expected Revenue Calculation

In this section, we present two algorithms for calculating the expected revenue of a given a priori route and time slot assignment. Calculating the expected revenue of a given solution efficiently is of critical importance to our solution methods. While sampling can be used to approximate the expected revenue, it turns out that exact calculation of the expected revenue, over all possible scenarios, can be done quite efficiently.

Obtaining the revenue of a single scenario ω, i.e., a sequence of arriving customers, can be done by checking whether an arriving customer can feasibly be inserted in the (partial) delivery route, and, if so, inserting it into the route. This requires $\mathcal{O}(k^2)$ operations for a scenario with k arriving customers. However, the number of scenarios $|\Omega|$ gets extremely large quickly, as the number of scenarios for an instance with n customer locations is $\mathcal{O}(n!)$, making brute-force enumeration computationally prohibitive. Fortunately, the observations presented earlier allow us to reduce the number of operations required substantially.

In the following, we present a naive enumeration algorithm and a customized dynamic programming algorithm for calculating the expected revenue of an a priori route. Both algorithms exploit the observations presented in Section 4.2.1.2.

4.3.1 Enumeration Algorithm

The expected revenue $\bar{r}$ of a given a priori route ρ and time slot assignment y can be calculated as follows:

$$\bar{r} = \sum_{\omega \in \Omega} \bar{r}_\omega = \sum_{S \subseteq \mathcal{V}_c} \sum_{\omega \in \text{Perm}\,(S)} \bar{r}_\omega = \sum_{S \subseteq \mathcal{V}_c} \sum_{\omega \in \text{Perm}\,(S)} p_\omega r_\omega, \tag{4.1}$$

with Perm (S) the set of all permutations of set of customers S, and $\bar{r}_\omega$ the probability-weighted revenue collected in scenario ω. Using our problem assumptions on the probability distribution of $\omega \in \Omega$, the probability p_ω of a single scenario ω, which is an ordered sequence of customer arrivals, can be determined as follows:

$$p_\omega = \prod_{i \in \omega} p_i \prod_{j \in \mathcal{V}_c \setminus \omega} (1 - p_j) \frac{1}{|\omega|!}, \tag{4.2}$$

with $|\omega|$ the number of customers wanting to place an order in scenario ω. The first part of the expression on the right-hand side represents the probability that the customers in ω want to place an order and the second part of the expression on the right-hand side represents the probability that they do so in the sequence specified by ω. The revenue collected in scenario ω is given by

$$r_\omega = \sum_{i \in \omega} r_i z_i^\omega, \tag{4.3}$$

with z_i^ω for customer i in ω an indicator variable specifying whether or not customer i can place an order given the orders that have already been accepted from customers arriving before i in the sequence.

While the expected revenue can be calculated by summing the probability-weighted revenues $\bar{r}_\omega$, for every scenario $\omega \in \Omega$, the enumeration algorithm exploits the following observation to reduce the number of terms in the summation. For every subset $S \subseteq \mathcal{V}_c$ of customers, if all can be feasibly inserted together in the a priori route with the time slot assignment, then all customer arrival sequences which are a permutation of subset S will result in the same delivery route and thus give the same revenue. In particular, the contribution $\bar{r}_S$ to the expected revenue of all permutations of subset

S, which is the sum $\bar{r}_S = \sum_{\omega \in \text{Perm}\,(S)} \bar{r}_\omega$, in that case reduces to

$$\begin{aligned} \bar{r}_S &= \prod_{i \in S} p_i \prod_{j \in \mathcal{V}_c \setminus S} (1 - p_j) \sum_{\omega \in \text{Perm}\,(S)} \frac{1}{|S|!} \sum_{i \in S} r_i \\ &= \prod_{i \in S} p_i \prod_{j \in \mathcal{V}_c \setminus S} (1 - p_j) \sum_{i \in S} r_i, \end{aligned} \tag{4.4}$$

where we use the fact that all permutations ω of S are equally likely.

The enumeration algorithm evaluates every subset $S \subseteq \mathcal{V}_c$, and starts by checking only one particular permutation ω of S. If all customer locations in ω can be inserted together in the given a priori route and time slot assignment, then the running expected revenue, the partial sum of $\bar{r}$, is increased by $\bar{r}_S$ given by (4.4) and the algorithm continues evalutating another subset S. However, if one or more customers in ω cannot be feasibly inserted together in the given a priori route and time slot assignment, the enumeration algorithm needs to calculate the contribution $\bar{r}_\omega = p_\omega r_\omega$ to the running expected revenue of each permutation ω of the subset S separately using (4.2) and (4.3).

For n customer locations, there are 2^n possible unordered subsets of customers and $|\Omega| = \sum_{k=0}^{n} \frac{n!}{k!}$ possible ordered subsequences of customer arrivals (see Sequence A000522 in Sloane, 2010). All unordered subsequences are checked by the enumeration algorithm, and typically not all ordered subsequences. However, in worst-case all ordered subsequences need to be checked. The naive enumeration algorithm therefore has a worst-case complexity of $\mathcal{O}\big(\sum_{k=0}^{n} \frac{n!}{k!} \cdot n^2\big) = \mathcal{O}\big(e \cdot n! \cdot n^2\big) = \mathcal{O}((n+2)!)$. We notice that the run-time of this algorithm suffers from the many redundant calculations that are done. These redundant calculations can be avoided by using a dynamic programming label extension algorithm.

4.3.2 DP Label Extension Algorithm

The Dynamic Programming algorithm exploits the following observations to reduce the number of operations needed for the revenue calculation. For any two scenarios ω_1 and ω_2, sequences of arriving customers (willing to place an order), which have equal sets of arriving customers (but in a different order) and equal sets of placed customers, i.e., their resulting delivery routes are equal, will also have the same contribution to the expected revenue, i.e., $\bar{r}_{\omega_1} = \bar{r}_{\omega_2}$. Moreover, in this case, the scenario $\omega_1' = \omega_1 \circ \{i\}$, in which customer i arrives after the customers of scenario ω_1, will have the same placed customer set, resulting delivery route and contribution to

the expected revenue as scenario $\omega_2' = \omega_2 \circ \{i\}$, for each customer $i \in \mathcal{V}_c \setminus \omega_1 = \mathcal{V}_c \setminus \omega_2$.

These observations lead to the following label definition. A label $L = \left(\mathcal{V}^{\text{arr}}, \mathcal{V}^{\text{placed}}, m\right)$ is characterized by: (1) a subset $\mathcal{V}^{\text{arr}}(L) \subseteq \mathcal{V}_c$ of arrived customers (wanting to place an order), (2) a subset $\mathcal{V}^{\text{placed}}(L) \subseteq \mathcal{V}^{\text{arr}}(L)$ of placed customers (arrived and able to place their order), and (3) a multiplier $m(L)$ denoting the total number of scenarios which share the same subsets of arrived and placed customers. Notice that the contribution $\bar{r}(L)$ of a single label L, which represents all scenarios that share the subsets of arrived and placed customers, to the expected revenue is:

$$\bar{r}(L) = \prod_{i \in \mathcal{V}^{\text{arr}}(L)} p_i \prod_{j \in \mathcal{V}_c \setminus \mathcal{V}^{\text{arr}}(L)} (1 - p_j) \frac{m(L)}{|\mathcal{V}^{\text{arr}}(L)|!} \sum_{i \in \mathcal{V}^{\text{placed}}(L)} r_i. \tag{4.5}$$

The expected revenue $\bar{r}$ for the given a priori route and time slot assignment is the sum of all label contributions $\bar{r}(L)$:

$$\bar{r} = \sum_{k=0}^{n} \sum_{L \in \mathcal{L}_k} \bar{r}(L), \tag{4.6}$$

with $\mathcal{L}_k$ the set of all possible labels with exactly $|\mathcal{V}^{\text{arr}}(L)| = k$ arriving customers.

It is convenient to also record the probability $p(L)$ of the scenarios and the sum of revenues $r(L)$ of the placed customers in the label:

$$p(L) = \prod_{i \in \mathcal{V}^{\text{arr}}(L)} p_i \prod_{j \in \mathcal{V}_c \setminus \mathcal{V}^{\text{arr}}(L)} (1 - p_j), \tag{4.7}$$

$$r(L) = \sum_{i \in \mathcal{V}^{\text{placed}}(L)} r_i. \tag{4.8}$$

The contribution $\bar{r}(L)$ of label L to the expected revenue then becomes

$$\bar{r}(L) = p(L) \frac{m(L)}{|\mathcal{V}^{\text{arr}}(L)|!} r(L). \tag{4.9}$$

Algorithm 4.1 shows the Dynamic Programming algorithm for calculating the expected revenue $\bar{r}$ over all scenarios exactly for a given a priori route ρ and time slot assignment y by extending labels. Here, the label sets $\mathcal{L}_k$, for $k \in \{0, 1, \ldots, n\}$, includes all labels L with $|\mathcal{V}^{\text{arr}}(L)| = k$ number of arrived customers. The contribution of each label L in $\mathcal{L}_k$ is added to the running expected revenue $\bar{r}$, and then each label L is extended by adding a customer i not yet arrived to that label. Then, it is checked if customer i can be inserted in the a priori route ρ with time slot assignment

y given the already placed customers in label L. The set of placed customers and the sum of placed customers are updated accordingly. Then, it is checked if a label $\tilde{L}$ with the same subset of arrived customers and the same subset of placed customers already exists in the label set $\mathcal{L}_{k+1}$. If so, the multiplier of the current label is added to that of the existing label. If not, a new label is added to the label set $\mathcal{L}_{k+1}$.

Algorithm 4.1 Dynamic Programming Algorithm

Input: A priori route ρ and time slot assignment y.
Output: Exact expected revenue $\bar{r}$.

$\bar{r} \leftarrow 0$
$\mathcal{V}^{\text{arr}} \leftarrow \varnothing$, $\mathcal{V}^{\text{placed}} \leftarrow \varnothing$, $p \leftarrow \prod_{i \in \mathcal{V}_c} (1 - p_i)$, $m \leftarrow 1$, $r \leftarrow 0$
$\mathcal{L}_0 \leftarrow \{(\mathcal{V}^{\text{arr}}, \mathcal{V}^{\text{placed}}, p, m, r)\}$
for $k = 0, 1, \ldots, n$ **do**
  for each $L = (\mathcal{V}^{\text{arr}}, \mathcal{V}^{\text{placed}}, m, p, r) \in \mathcal{L}_k$ **do**
    $\bar{r} \leftarrow \bar{r} + p \cdot \frac{m}{|\mathcal{V}^{\text{arr}}|!} \cdot r$
    for each $i \in \mathcal{V}_c \setminus \mathcal{V}^{\text{arr}}$ **do**
      $\tilde{\mathcal{V}}^{\text{arr}} \leftarrow \mathcal{V}^{\text{arr}} \cup \{i\}$
      if IsFeasibleInsertion(i, L, ρ, y) **then**
        $\tilde{\mathcal{V}}^{\text{placed}} \leftarrow \mathcal{V}^{\text{placed}} \cup \{i\}$
        $r' \leftarrow r + r_i$
      else
        $\tilde{\mathcal{V}}^{\text{place}} \leftarrow \mathcal{V}^{\text{placed}}$
        $r' \leftarrow r$
      if there is an $\tilde{L} \in \mathcal{L}_{k+1}$ with $\mathcal{V}^{\text{arr}}(\tilde{L}) = \tilde{\mathcal{V}}^{\text{arr}}$ and $\mathcal{V}^{\text{placed}}(\tilde{L}) = \tilde{\mathcal{V}}^{\text{placed}}$ **then**
        $m(\tilde{L}) \leftarrow m(\tilde{L}) + m$
      else
        $p' \leftarrow p \cdot \frac{p_i}{1 - p_i}$
        $\mathcal{L}_{k+1} \leftarrow \mathcal{L}_{k+1} \cup \{(\tilde{\mathcal{V}}^{\text{arr}}, \tilde{\mathcal{V}}^{\text{placed}}, m, p', r')\}$

The following Theorem characterizes the number of operations required by the algorithm. We are able to give a tight lower bound and an upper bound on the number of operations. It is currently unknown to us if the upper bound is tight.

Theorem 4.5. *The number of operations required by the Dynamic Programming Algorithm for an a priori route ρ and time slot assignment y with $|\mathcal{V}_c| = n$ customer locations is at least $\mathcal{O}(2^n n^2)$, but not more than $\mathcal{O}(3^n n^2)$.*

Proof. Each label is uniquely characterized by its set of arrived customers and set of placed customers. This means that each label set $\mathcal{L}_k$ contains at most one label with a unique combination of both sets. Furthermore, the label set $\mathcal{L}_k$, for $k \in \{0, 1, \ldots, n\}$, contains only labels L with a number of k arrived customers: $|\mathcal{V}^{\text{arr}}(L)| = k$. There

are at most $\binom{n}{k}$ possible different sets of arriving customers of size k. Also, for each unique set of arriving customers $\mathcal{V}^{\text{arr}}(L)$, there are at most 2^k possible set of placed customers $\mathcal{V}^{\text{placed}}(L)$, since $\mathcal{V}^{\text{placed}}(L) \subseteq \mathcal{V}^{\text{arr}}(L)$ holds for any label L. Therefore, the total number of labels in $\mathcal{L}_k$ for $k \in \{0, 1, \ldots, n\}$ does not exceed $\binom{n}{k}2^k$. We note that this number is not necessary tight for every k. Each label $L \in \mathcal{L}_k$ has $|\mathcal{V}_c \setminus \mathcal{V}^{\text{arr}}(L)| = n - k$ possible extensions, which are all checked in the algorithm. Checking if a single customer can be feasibly inserted at a given position in a delivery route (containing the placed customers) can be done in $\mathcal{O}(k)$ operations. Alternatively, earliest and latest start of service times of the placed customers in the delivery route of a label can be kept in memory, reducing the number of operations required by the feasibility check to $\mathcal{O}(1)$. However, if the insertion is feasible, these earliest and latest start of service times need to be updated, which still requires $\mathcal{O}(k)$ operations and thus only lowers the absolute number of operations. Then, the algorithm checks if a label $\tilde{L}$ already exists with the same set of arrived customers and the same set of placed customers. We assume the labels in the label sets $\mathcal{L}_k$ are stored in a data structure which allows search and insertion to be done in logarithmic time complexity, for instance a binary tree data structure. This allows a number of $|\mathcal{L}_{k+1}| \leq \binom{n}{k+1}2^{k+1}$ stored labels to be searched in $\mathcal{O}\Big(\log\big(\binom{n}{k+1}2^{k+1}\big)\Big) = \mathcal{O}\big(2\log 2^{k+1}\big) = \mathcal{O}(k)$ operations, and a new label can be added in the same number of operations complexity. Alternatively, two nested binary tree data structures, one for the possible sets $\mathcal{V}^{\text{arr}}$ and one for the possible sets $\mathcal{V}^{\text{placed}}$, can be used. This results in the same number of operations: $\mathcal{O}\Big(\log\big(\binom{n}{k+1}\big) + \log 2^{k+1}\Big) = \mathcal{O}(k)$. Each stage $k \in \{0, 1, \ldots, n-1\}$ of the algorithm therefore requires no more than $\mathcal{O}\big(\binom{n}{k}2^k(n-k)k\big)$ operations, and in total the algorithms requires no more than $\mathcal{O}\big(\sum_{k=0}^{n}\binom{n}{k}2^k(n-k)k\big) = \mathcal{O}\big(3^n n^2\big)$ operations. It is currently unknown by the authors if this number of operations is tight. However, a tight lower bound can be given by considering the following special case. Suppose that all arriving customers can be feasibly placed together in the a priori route, i.e., $\mathcal{V}^{\text{arr}}(L) = \mathcal{V}^{\text{placed}}(L)$ for any label L. In that case the number of possible labels $|\mathcal{L}_k| \leq \binom{n}{k}$ for $k \in \{0, 1, \ldots, n\}$, and therefore the number of operations needed by the algorithm is then at most $\mathcal{O}\big(\sum_{k=0}^{n}\binom{n}{k}(n-k)k\big) = \mathcal{O}\big(2^n n^2\big)$. □

Theorem 4.5 shows that the worst case number of operations needed by the Dynamic Programming algorithm, $\mathcal{O}\big(3^n n^2\big)$, is much lower than the worst case number of operations needed by the Enumeration algorithm, $\mathcal{O}\big(n! \cdot n^2\big)$. The Dynamic Programming algorithm does need more memory then the enumeration algorithm: at each stage $k \in \{0, 1, \ldots, n-1\}$ both label sets $\mathcal{L}_k$ and $\mathcal{L}_{k+1}$ need to be kept in memory, while the enumeration algorithm requires almost no memory at all. Com-

putational experiments show that the reduction in the number of operations allows us to calculate the expected revenue of a priori routes and time slot assignments with $n = 12$ customer locations, over $|\Omega| \approx 1.3 \cdot 10^9$ scenarios, exactly in under 10 seconds of computation time, while the enumeration algorithm takes over 60 minutes.

4.4 Solution Method

We propose an Sample Average Approximation (SAA) Monte-Carlo approach to solve the Single-Vehicle problem of Section 4.2.1. This method uses sampling of scenarios to solve large stochastic programs. In the following, we present a two-stage stochastic program which solves the single-vehicle problem of Section 4.2.1 exactly, and then give an overview of the SAA approach.

4.4.1 Stochastic Programming Formulation

The single-vehicle variant can be formulated as a two-stage mixed-integer stochastic program. The a priori route ρ and time slot assignment y are the first stage decisions, and the evaluation of the revenue of placed customer orders is the second stage "decision". In the second stage, there are no recourse options, so it is purely an evaluation stage. The objective of the stochastic program is the revenue obtained by the placed customer orders averaged over all possible scenarios. We now formulate the first stage, then the second stage and afterwards provide the objective of the stochastic program.

We need the following decision variables for the first stage. Let binary variables x_{ij} for arc $(i, j) \in \mathcal{A}$ be one if the arc is used in the a priori route, and zero otherwise. Let binary variables y_{is} for customer location $i \in \mathcal{V}_c$ and time slot $s \in \mathcal{T}_i$ be one if this time slot is assigned to this location, and zero otherwise. These are the main decision variables of the first stage. For subtour elimination constraints we introduce single commodity flow variables f_{ij}, which represent the position of location $i \in \mathcal{V}$ in the a priori route if arc $(i, j) \in \mathcal{A}$ is used in the a priori route, and are zero otherwise. Note that variables f_{ij} are fixed when binary variables x_{ij} are fixed. The first stage

of the stochastic program can be formulated as follows:

$$\sum_{j \in \mathcal{V} \setminus \{o,i\}} x_{ij} = 1 \qquad \forall i \in \mathcal{V}_\mathrm{c}, \tag{4.10}$$

$$\sum_{i \in \mathcal{V} \setminus \{j,d\}} x_{ij} = 1 \qquad \forall j \in \mathcal{V}_\mathrm{c}, \tag{4.11}$$

$$\sum_{j \in \mathcal{V}_\mathrm{c}} x_{oj} = 1, \ \sum_{i \in \mathcal{V}_\mathrm{c}} x_{id} = 1, \tag{4.12}$$

$$\sum_{j \in \mathcal{V} \setminus \{o\}} f_{ij} - \sum_{j \in \mathcal{V} \setminus \{d\}} f_{ji} = 1 \qquad \forall i \in \mathcal{V}_\mathrm{c}, \tag{4.13}$$

$$f_{oj} = 0, \qquad \forall j \in \mathcal{V}_\mathrm{c}, \tag{4.14}$$

$$x_{ij} \leq f_{ij} \leq n x_{ij} \qquad \forall (i,j) \in \mathcal{A},\ i \neq o, \tag{4.15}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \tag{4.16}$$

$$\sum_{s \in \mathcal{T}_i} y_{is} = 1 \qquad \forall i \in \mathcal{V}_\mathrm{c}, \tag{4.17}$$

$$y_{is} \in \{0,1\} \qquad \forall s \in \mathcal{T}_i,\ i \in \mathcal{V}_\mathrm{c}. \tag{4.18}$$

Essentially, the first stage consists of a TSP single-commodity flow formulation for the a priori route and the time slot assignment part. Constraints (4.10)–(4.12) ensure that the a priori route starts and ends at the fulfillment center and that every customer location is visited. Subtours in the a priori route are eliminated by Constraints (4.13)–(4.15). These are known as single-commodity flow constraints. Constraints (4.17) ensure that every customer location i is assigned exactly one time slot from the set of possible time slots $\mathcal{T}_i$.

The second stage of the stochastic program ensures that the expected revenue of an a priori route and time slot assignment is obtained. We introduce some additional notation. Let variable u_i denote the position of customer location $i \in \mathcal{V}_\mathrm{c}$ in the a priori route. Let the second stage binary decision variable z_i^ω for customer location $i \in \omega$ arriving in scenario ω denote if this customer can place an order in scenario ω using the first stage a priori route and time slot assignment solution. Furthermore, let the set of vertices $\mathcal{V}_\mathrm{c}^{\omega h} \subseteq \mathcal{V}_\mathrm{c}$ contain the first h customer locations that arrived in scenario ω, for $h \in \{1, \ldots, |\omega|\}$. Also, let $\mathcal{V}^{\omega h} = \mathcal{V}_\mathrm{c}^{\omega h} \cup \{o, d\}$. Similarly, let the arc set $\mathcal{A}_\mathrm{c}^{\omega h}$ contain all arcs in $\mathcal{A}$ that are between vertices in $\mathcal{V}_\mathrm{c}^{\omega h}$ and the arc set $\mathcal{A}^{\omega h}$ contain all arcs in $\mathcal{A}$ that are between vertices in $\mathcal{V}^{\omega h}$. In the second stage, we essentially have for each scenario ω and for each number of arrived customers $h \in \{1, \ldots, |\omega|\}$ in that scenario a partial graph which is used to find the current delivery (execution)

route containing the currently placed customer locations. Binary decision variables $x_{ij}^{\omega h}$ are one if arc (i,j) is used in the delivery route of placed customers in scenario ω with the first h customers arrived. Continuous decision variables $t_i^{\omega h}$ denote the time at which service starts at location $i \in \mathcal{V}^{\omega h}$, or, in case of fulfillment center start vertex o and in case of fulfillment center end vertex d, the time of departure or arrival at the fulfillment center, respectively. In case a customer cannot place an order, the corresponding start of service time decision variable is not used.

The second stage of the stochastic program can be formulated as follows:

$$u_i = \sum_{j \in \mathcal{V} \setminus \{o,i\}} f_{ij} \qquad \forall i \in \mathcal{V}_\mathrm{c}, \tag{4.19}$$

$$\sum_{j \in \mathcal{V}^{\omega h} \setminus \{o,i\}} x_{ij}^{\omega h} = z_i^\omega \qquad \forall i \in \mathcal{V}_\mathrm{c}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.20}$$

$$\sum_{i \in \mathcal{V}^{\omega h} \setminus \{j,d\}} x_{ij}^{\omega h} = z_j^\omega \qquad \forall i \in \mathcal{V}_\mathrm{c}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.21}$$

$$\sum_{j \in \mathcal{V}^{\omega h} \setminus \{o\}} x_{oj}^{\omega h} = 1, \quad \sum_{i \in \mathcal{V}^{\omega h} \setminus \{d\}} x_{id}^{\omega h} = 1 \qquad \forall h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.22}$$

$$u_i + 1 \le u_j + n\big(1 - x_{ij}^{\omega h}\big) \qquad \forall (i,j) \in \mathcal{A}_\mathrm{c}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.23}$$

$$t_i^{\omega h} + t_{ij} \le t_j^{\omega h} + T\big(1 - x_{ij}^{\omega h}\big) \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.24}$$

$$\sum_{s \in \mathcal{T}_i} a_s y_{is} \le t_i^{\omega h} \le \sum_{s \in \mathcal{T}_i} b_s y_{is} \qquad \forall i \in \mathcal{V}_\mathrm{c}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.25}$$

$$u_i \ge 1 \qquad \forall i \in \mathcal{V}_c, \tag{4.26}$$

$$z_i^\omega \in \{0,1\} \qquad \forall i \in \omega,\ \omega \in \Omega, \tag{4.27}$$

$$x_{ij}^{\omega h} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega, \tag{4.28}$$

$$0 \le t_i^{\omega h} \le T \qquad \forall i \in \mathcal{V}^{\omega h},\ h \in \{1,\dots,|\omega|\},\ \omega \in \Omega. \tag{4.29}$$

The position variables u_i are set by constraints (4.19) using the first stage commodity flow variables f_{ij}. Constraints (4.20)–(4.22) ensure that the delivery route in scenario ω after the first h customer arrivals contains only locations of the placed customer, and start and end locations of the fulfillment center. Notice that the z_j^ω variables couple the different h stages: If a customer j arriving as hth customer in a scenario ω may place the order ($z_j^\omega = 1$), then the customer location must be included in the current delivery route, stage h, as well as all subsequent delivery routes of stages,

$h+1,\ldots,|\omega|$. Otherwise, if the customer j may not place the order ($z_j^\omega = 0$), then this location should not be visited in the current delivery route and also all subsequent delivery routes, i.e., stages $h, h+1,\ldots,|\omega|$. Constraints (4.23) make sure that the placed customer locations in the delivery route follow their position in the a priori route. Furthermore, these constraints ensure that the delivery routes do not contain subtours. Start of service times along the (partial) delivery routes are set by constraints (4.24), and constraints (4.25) ensures that the start of service times of the placed customer locations are feasible with respect to their assigned time slot. Both sets of constraints also ensure that the delivery route respects the fulfillment center time window $[0,T]$ (planning horizon). As a result, arriving customers (willing to place their order) cannot place their order if including them in the current delivery route will violate the time slots- or the fulfillment center time window restrictions.

The objective of the stochastic program is given by

$$\max \sum_{\omega\in\Omega} p_\omega \sum_{i\in\omega} r_i z_i^\omega. \tag{4.30}$$

This is the expected revenue, which is obtained by summation of the revenues obtained in each scenario $\omega\in\Omega$ weighted by the probability of each scenario p_ω. These probabilities are given by (4.2).

The two stages, i.e., constraints (4.10)–(4.29), together with objective (4.30) form a stochastic program which can be solved as a MIP. Although this formulation can be solved for small instances, it has a "flaw": the MIP has full information of the complete customer sequence ω and can therefore *anticipate* not yet arrived customers. That is, even when a customer $i=\omega(h)$ arrives as the hth customer in ω and can be feasibly inserted in the current delivery route, the MIP may decide to prohibit the customer from placing an order, i.e., set $z_i^\omega = 0$. This can be beneficial as it may allow placement of high revenue customers arriving later in scenario ω. However, in our setting this look-ahead is not possible: If an arriving customer can feasibly be served, the customer is allowed to place an order. Forcing *non-anticipation* can be incorporated in the model, but requires the introduction of (many) variables and constraints. Essentially, earliest and latest arrival time decision variables, $e_i^{\omega h}$ and $l_i^{\omega h}$, respectively, allow determination of whether the insertion of an arriving customer $i'=\omega(h)$ will violate $e_{i'}^{\omega h} \le l_{i'}^{\omega h}$ given earliest and latest arrival times of the other locations in the current delivery route. This customer must place an order if the insertion is feasible, and a customer must be prohibited from placing an order if the insertion is not feasible. The non-anticipatory constraints are 'big-

M' type constraints, as they are linearizations of constraints with $\max\{\cdot,\cdot\}$ and $\min\{\cdot,\cdot\}$ operations. The non-anticipatory constraints can be found in Appendix 4.A. The stochastic program for our problem, i.e., with non-anticipation constraints, is therefore given by constraints (4.10)–(4.29) and constraints (4.37)–(4.60) together with the objective (4.30).

4.4.2 Sample Average Approximation

When the number of customer locations n gets large, the above stochastic programming formulation is intractable as the number of scenarios becomes prohibitive (it grows exponentially – $\mathcal{O}(n!)$). A Sample Average Approximation (SAA) approach alleviates this issue by repeatedly solving the stochastic program with only a sampled subset of scenarios. We refer the interested reader to Kleywegt et al. (2002), who investigate the mathematical details of the method. For convenience, but also since our evaluation of the true solution objective differs slightly from the method described by Kleywegt et al. (2002), we summarize here the most important results.

Let Ω_N be a single sample consisting of N randomly sampled scenarios. Then the *sample problem* for our stochastic programming formulation becomes

$$R(\Omega_N) = \max \frac{1}{N} \sum_{\omega \in \Omega_N} r_i z_i^\omega,$$

subject to constraints (4.10)–(4.29) and (4.37)–(4.60), in which the set of all scenarios Ω is replaced by the sample Ω_N.

In the SAA approach, we generate M samples Ω_N^1, Ω_N^2, …, Ω_N^M, each containing N randomly sampled scenarios. For each sample Ω_N^m with $m = 1, \ldots, M$, the sample problem is solved to obtain an optimal solution (ρ^m, y^m) with objective $R(\Omega_N^m)$. Among these M solutions (ρ^m, y^m), we pick the best solution (ρ^*, y^*) using exact evaluation of the expected revenue:

$$(\rho^*, y^*) \in \underset{m=1,\ldots,M}{\arg\max} \left\{ r(\rho^m, y^m) \right\},$$

with $r(\rho^m, y^m)$ the exact expected revenue of solution (ρ^m, y^m). This can be calculated efficiently by our customized Dynamic Programming algorithm (see Section 4.3.2). We obtain an estimate of the optimality gap by using the average objec-

tive $\bar{R}$ over the M sample problems given by

$$\bar{R} = \frac{1}{M} \sum_{m=1}^{M} R(\Omega_N^m).$$

An estimation of the optimality gap is given by

$$\Delta = \bar{R} - r(\rho^*, y^*). \tag{4.31}$$

The variance $\sigma^2_{\bar{R}-r(\rho^*,y^*)}$ of this optimality gap estimator is estimated by

$$\sigma^2_{\bar{R}-r(\rho^*,y^*)} = \sigma^2_{\bar{R}} = \frac{1}{(M-1)M} \sum_{m=1}^{M} (R(\Omega_N^m) - \bar{R})^2.$$

Notice that this does not include any variance associated with the expected revenue of the solution, since we can determine this expected revenue exactly, while typically in an SAA approach the expected revenue is obtained by estimation. The variance of the optimality gap estimator can be used to construct a one-sided $(1-\alpha)$ confidence interval for the estimated optimality gap:

$$\bar{R} - r(\rho^*, y^*) + \Phi^{-1}(1-\alpha)\frac{\sigma_{\bar{R}}}{\sqrt{M}}, \tag{4.32}$$

with Φ^{-1} the inverse cumulative standard normal distribution function. This one-sided confidence interval is useful in assessing the quality of the current best solution (ρ^*, y^*) and providing statistical bounds on the objective value of the optimal solution.

4.5 Heuristic Methods

In this section, we present four heuristic methods for solving the strategic time slot management problem for a single vehicle. The first three simplify the stochastic programming formulation to make it more tractable, the last one is a simple pragmatic heuristic that ignores the uncertainty. The three simplifications of the stochastic programming formulation can be solved using the SAA approached presented in Section 4.4.2.

4.5.1 SAA with Fixed A Priori Route

In the first simplification, an a priori tour is generated independently and the SAA approach is used to assign a time slot to each customer location. Although we have shown that an *optimal* TSP tour, one that minimizes travel times, is not always the best a priori tour (Observation 4.3), it is likely to be a good a priori tour. Furthermore, obtaining an optimal TSP tour is not difficult for the instance sizes we are interested in ($n \leq 12$). Note that when the travel times are symmetric, the optimal TSP tour has to be made directed by choosing an orientation.

Once an a priori tour has been generated, the time slot assignment can be obtained using the stochastic program of Section 4.4.1 and fixing the decision variables related to the a priori tour. That is, all decision variables related to the a priori tour, x_{ij}, f_{ij}, u_i, are fixed, the first stage constraints reduce to only (4.17) and (4.18), and the second stage reduces to (4.20)–(4.29) and (4.37)–(4.60). The SAA approach presented in Section 4.4.2 can be used to solve this stochastic program, which is likely to be more efficient than solving the full stochastic program of Section 4.4.1.

It seems natural to use an *optimal* TSP tour, one that minimize the travel time, as a priori tour, but this not required.

4.5.2 SAA with Ascending Time Slots

In the second simplification, we enforce that the time slots are ascending along the a priori route, i.e., if the a priori route is $(o, 1, 2, \ldots, n, d)$, then $a_1 \leq a_2 \leq \ldots \leq a_n$. Although we have shown that ascending time slots along the a priori route is not always optimal (Observation 4.4), in most cases an optimal solution in which the time slots are ascending exists. Therefore, it is natural to consider a variant of our SAA approach in which the assigned time slots are restricted to being ascending along the a priori tour. We simply add the following constraints to the stochastic program:

$$\sum_{s \in \mathcal{T}} a_s y_{is} \leq \sum_{s \in \mathcal{T}} a_s y_{js} + T\,(1 - x_{ij}) \qquad \forall i, j \in \mathcal{V}_c, i \neq j, \tag{4.33}$$

$$\sum_{s \in \mathcal{T}} b_s y_{is} \leq \sum_{s \in \mathcal{T}} b_s y_{js} + T\,(1 - x_{ij}) \qquad \forall i, j \in \mathcal{V}_c, i \neq j, \tag{4.34}$$

Adding these constraints decreases the solution space, which is likely to result in a faster solution times.

4.5.3 SAA without Non-Anticipation

In the third simplification, we drop the non-anticipatory constraints (4.37)–(4.60). Most of these are 'big-M' constraints, which typically result in weak LP relaxation bounds and large search trees. Therefore, it might be computationally advantageous to use a variant of the stochastic program in which the non-anticipation constraints are omitted (i.e., only use (4.10)–(4.29)). Note that the bounds obtained in the SAA method now correspond to a relaxation of the problem and therefore do not provide insights in the optimal objective value of the original problem. However, the MIP might be easier to solve than the full stochastic program, and therefore might speed-up the SAA method.

4.5.4 A Linear Scaling Heuristic

The Linear Scaling Heuristic (LSH) is a two-phase heuristic (similar to SAA with Fixed A Priori Route) in which an a priori tour is generated first and then, given this a priori tour, time slot are assigned to each customer location using a simple linear scaling rule. The time slot assignment rule is based on the idea of linearly scaling the visit times of the customer locations in the a priori route to "fit" the planning horizon.

Before describing the time slot assignment rule, it is useful to introduce some additional notation. Let e_i and l_i for each customer location $i \in \mathcal{V}_c$ be the earliest and latest times that delivery can start at the customer location when directly visited after- and before the fulfillment center, respectively, i.e.,

$$e_i = a_o + t_{oi} = t_{oi},$$
$$l_i = b_d - t_{id} = T - t_{id},$$

for each $i \in \mathcal{V}_c$. Note that these quantities are independent of the a priori tour. We assume that $e_i \leq l_i$ holds for each customer location $i \in \mathcal{V}_c$, i.e., it is possible to service each customer location. The time slot to be assigned to customer location i needs to have overlap with $[e_i, l_i]$, or else the customer location cannot be served in tour and will always be rejected. By definition, the set of possible time slots $\mathcal{T}_i$ of customer location i contains only time slots which have overlap with $[e_i, l_i]$. Let the a priori route be $\rho^{\text{TSP}} = (o, 1, 2, \ldots, n, d)$. Note that, typically, the a priori route is not feasible itself, i.e., the length of the a priori route is more than T. As we have seen in Observation 4.1, in case the a priori route is feasible, then it is optimal to

assign each customer location i a time slot which contains t_i^{TSP}, the time the a priori route visits customer location i.

LSH linearly scales down times t_i^{TSP} to ensure that these times lie within the planning horizon $[0, T]$. The resulting scaled times $\tilde{t}_i$ are then used to assign the time slot $s_i^* \in \mathcal{T}_i$ to customer location i. More specifically, we obtain the scaled time $\tilde{t}_i$ for customer location $i \in \mathcal{V}_{\text{c}}$ as

$$\tilde{t}_i = t_i^{\text{TSP}} \cdot \frac{\min\{T, t_d^{\text{TSP}}\}}{t_d^{\text{TSP}}}.$$

Should the scaled time $\tilde{t}_i$ lie outside the interval $[e_i, l_i]$, then we adjust it to the boundary of the interval, i.e.,

$$\tilde{t}_i = \begin{cases} l_i & \text{if } \tilde{t}_i > l_i, \\ \tilde{t}_i & \text{if } \tilde{t}_i \in [e_i, l_i], \\ e_i & \text{if } \tilde{t}_i < e_i. \end{cases}$$

To select a time slot for customer i using the scaled time $\tilde{t}_i$, we calculate a score c_{is} for each time slot $s \in \mathcal{T}_i$, and pick the time slot s^* with the highest score: $s^* = \arg\max_{s \in \mathcal{T}_i} c_{is}$. Should there be a tie between multiple time slots, then the time slot starting the earliest is chosen. We consider the following score function:

$$c_{is} = (\tilde{t}_i - a_s) \cdot (b_s - \tilde{t}_i).$$

This score function can be calculated quickly. It has the property that positives scores are given for scaled times inside the time slot and have a peak in the center of the time slot. Remember that we assume that the set of possible time slots $\mathcal{T}$ has time slots with equal width. Therefore, should the time slots in $\mathcal{T}$ be non-overlapping, then using these score functions simply results in selecting the time slot which has overlap with the scaled time. However, when the time slots in $\mathcal{T}$ overlap, then using these score functions results in selecting the time slot with its center closest to the scaled time.

4.6 Computational Experiments

In this section, we present the results of numerical experiments in which we use the presented methods on randomly generated instances. The methods have been

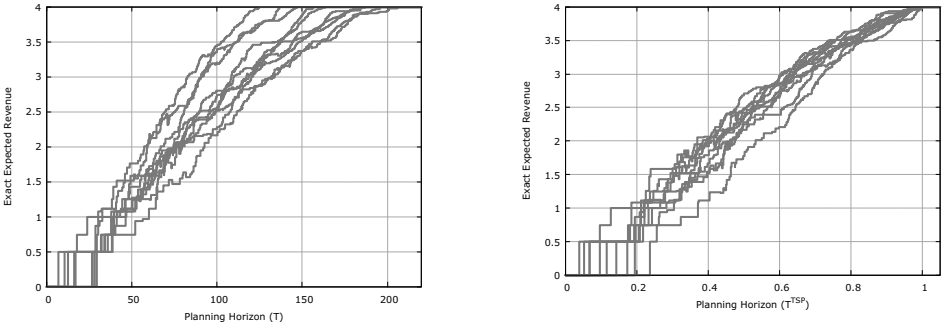
implemented in Python version 2.7.11, and the MIPs are solved using Gurobi version 8.0.1 with the default settings. The runs are executed as a single thread on an Intel® Xeon® E5-2650 v2 with 2.6 GHz (Turbo Boost up to 3.6 GHz) and 32 GB of RAM running Debian Linux version 9. All CPU times reported have been obtained using the wall-clock timer `timeit.default_timer()`. Only a single Python thread was active at any time on the CPU, while the MIP solver Gurobi had access to all available CPU cores.

4.6.1 Instance Generation and Parameters

We generated sets of 10 instances for $n = 4$, 8, and 12 customer locations. These locations have integer coordinates which are randomly uniform in a $[0, 60] \times [0, 60]$ square. The fulfillment center is located in the center of the region (coordinates $(30, 30)$). The travel times between two locations is taken to be the Euclidean distance rounded up to two decimals and the service time at a location is taken to be zero. The customers have equal order placement probability $p = 0.5$ and result in equal revenue $r = 1$.

For each instance, it is also necessary to specify a horizon T and set of time slots $\mathcal{T}$. To account for the fact that not all customers will place an order, the company may design an a priori route with a duration that exceeds the operating horizon T. Depending on the company's risk tolerance (and service level targets), the company may choose an a priori route with a duration that results in an expected delivery route length of T or less (where the expected delivery route length is a function of the order placement probabilities of the customers). Indeed, by Observation 4.1, a horizon T equal to the minimum duration tour visiting all customer locations T^{TSP} implies that all customers can place an order (regardless of the sequence in which they place an order) and the company has no risk (while the actual delivery routes may vary in duration). Therefore, to reflect these choices, we model the company's risk tolerance, e.g., low, medium, and, high, by setting the horizon T as a function of T^{TSP}, specifically $T = 0.9T^{\text{TSP}}$, $T = 0.75T^{\text{TSP}}$, and $T = 0.6T^{\text{TSP}}$. (That is, to simplify instance generation, rather than designing instances for which the a priori route has duration of $\frac{1}{0.9}T$, $\frac{1}{0.75}T$, or $\frac{1}{0.6}T$, we do the reverse.) Thus, we generate three "regimes" of instances, characterized by the associated risk tolerance. Note that within a regime (set of instances), the duration T^{TSP} can vary substantially. Instances with locations that are relatively close together typically have a shorter a priori tour than instances with locations that are relatively far apart. The benefit of setting the horizon as a fraction of the minimum duration tour rather than as an

absolute value is highlighted in Figure 4.5. The expected revenue is more consistent



(a) Absolute.

(b) Factor of T^{TSP}.

Figure 4.5: Exact expected revenues for the c60 instances with 8 customers using the minimum duration tour as a priori route and no time slots, $\mathcal{T} = \{[0, T]\}$, plotted using (a) an absolute horizon and (b) plotted using a horizon relative to the duration of the optimal tour (T^{TSP}).

when the horizon is set as a fraction of the minimum duration tour rather than as an absolute value.

We consider two time slot sets $\mathcal{T}$: non-overlapping and overlapping. Both sets have time slots with a fixed width w. The non-overlapping time slot set is given by

$$\mathcal{T} = \{[0, w], [w, 2w], [2w, 3w], \ldots\}, \tag{4.35}$$

and the overlapping time slot set is given by

$$\mathcal{T} = \left\{[0, w], \left[\frac{1}{2}w, \frac{3}{2}w\right], [w, 2w], \left[\frac{3}{2}w, \frac{5}{2}w\right], \ldots\right\}. \tag{4.36}$$

The latter set has two overlapping time slots for each time $t \in [0, T]$. Similar to the horizon, we choose the time slot width as fraction of the duration of the a priori route, i.e., $w = 0.25T^{\mathrm{TSP}}$ and $0.125T^{\mathrm{TSP}}$. As a benchmark, we also consider the case of "no slots", in which essentially each customer locations gets a time slot equal to the full horizon, i.e., a possible time slot set of $\mathcal{T} = \{[0, T]\}$.

The instances generated are recorded in VRP-REP XML format (Mendoza et al. (2014), see `http://vrp-rep.org`). We also include an optimal directed TSP tour and its duration. This directed TSP tour is used by the methods that require a fixed a priori route. The TSP tour is obtained by solving the formulation (4.10)–(4.16)

with the objective of minimizing tour duration. For convenience, the horizons and time slot sets are also included in each instance file.

All SAA methods are run with sample sizes varying from $N = 2$ up to $N = 64$ (depending on instance size). Each SAA run uses a fixed sample size and $M = 20$ repetitions. After solving each MIP, the exact expected revenue is calculated using the DP algorithm presented in Section 4.3.2. During the run, the best solution found so far is kept in memory and used as starting solution for the MIP solver.

Not surprisingly, the full SAA method described in Section 4.4 is very demanding computationally. Therefore, we decided to run the SAA with Fixed A Priori Route method one time ($M = 1$) using the same sample and use the solution obtained as starting solution for the first MIP of the full SAA method run. This computation time is included when reporting solution time for the full SAA method. In our experiments, we solve all MIPs to optimality, i.e., no time limit was set. (Imposing a time limit increased the optimality gap estimates and decreased the solution quality significantly.)

4.6.2 Expected Revenue Calculation

We use exact calculation of the expected revenue in all our presented methods. While the Linear Scaling Heuristic of Section 4.5.4 requires only one evaluation at the end to obtain the expected revenue, the SAA methods require M evaluations. Thus, for the SAA methods, it is critical that the expected revenue calculation are done efficiently, in a reasonable amount of time.

In Table 4.1, we compare the computing times of the enumeration algorithm (ENUM) presented in Section 4.3.1 with the customized dynamic programming algorithm (DP) presented in Section 4.3.2 on instances of different sizes, where the Linear Scaling Heuristic (LSH) was used to obtain the solutions. Both algorithms were given a time limit of 1 hour for evaluation of the expected revenue of each instance. The time slot width w is set to $0.25T^{\text{TSP}}$ and the horizon T is set to $0.90T^{\text{TSP}}$, $0.75T^{\text{TSP}}$ and $0.60T^{\text{TSP}}$. We also report the total number of scenarios $|\Omega|$ over which the expected revenue is calculated. The computing times reported are averages over the 10 instances in a set and given in seconds. For convenience, we also report the speed-up of the DP algorithm over the ENUM algorithm.

While the ENUM algorithm is slightly faster on small instances with 4 customer locations, the DP algorithm outperforms the ENUM algorithm on mid and large size instances with 8 and 12 customer locations. Moreover, we see that the ENUM algorithm reaches the time limit of 1 hour for the evaluation of the expected cost of

Table 4.1: Comparison of the computing times required to evaluate the exact expected revenue using the enumeration algorithm and the dynamic programming algorithm.

					CPU time (s)		Speed up
	n	$\lvert\Omega\rvert$	w	T	ENUM	DP	DP
c60	4	65	0.25	0.90	0.0012	0.0031	0.38
			0.25	0.75	0.0011	0.0034	0.33
			0.25	0.60	0.0015	0.0036	0.42
c60	8	109601	0.25	0.90	4.42	0.12	37.87
			0.25	0.75	4.16	0.16	25.35
			0.25	0.60	3.85	0.17	22.31
c60	12	$1.3 \cdot 10^9$	0.25	0.90	> 3600	2.58	> 1398
			0.25	0.75	> 3600	4.80	> 750
			0.25	0.60	> 3600	6.07	> 592

solutions for instance with 12 customer instances, while the DP algorithm takes less than 10 seconds for these instances. We observe too that the computing time depends on the horizon T. Whereas the ENUM algorithm requires less computing time for instances with a shorter horizon T, the DP algorithm requires more computing time for instances with a shorter horizon T. This is probably due to the fact that the required number operations, which we investigated in Section 4.3.2, tends towards the lower bound $\mathcal{O}\big(2^n n^2\big)$ in case of a long horizon, while it tends more to the upper bound $\mathcal{O}\big(3^n n^2\big)$ in case of a short horizon.

4.6.3 Non-overlapping Time Slots

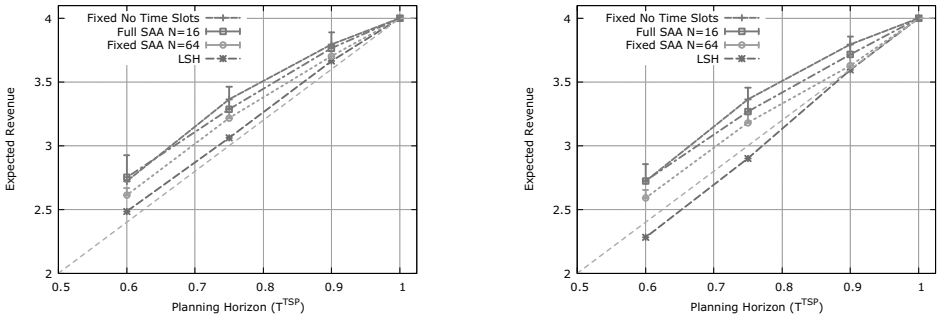
LSH (Section 4.5.4), SAA with Fixed A Priori Route (Section 4.5.1) and Full SAA (Section 4.4) are tested on the instance sets with non-overlapping time slot sets (4.35). The expected revenue of the best found solution and the computation time in seconds, both averaged over each set of 10 instances, are presented in Table 4.2. We also report the expected revenue when the a priori tour is chosen to be the minimum duration tour and each location can be visited at any time during the horizon ("No Slots"). This expected revenue serves as an upper bound on the expected revenue, although it must be noted that it is not an exact upper bound. The time slot width w is set to $0.25T^{\text{TSP}}$ and $0.125T^{\text{TSP}}$, and the horizon T is set to $0.90T^{\text{TSP}}$, $0.75T^{\text{TSP}}$ and $0.60T^{\text{TSP}}$. For $n = 4$ customers, sample size $N = 64$ is used for both SAA with Fixed A Priori Route and Full SAA. For these small instances, however, the complete stochastic programs of Section 4.4.1 containing all scenarios ($|\Omega| = 65$) can be solved for both variants, i.e., without sampling. While both SAA methods and

the complete stochastic programs produce solutions with the same expected revenue, the computation times differ since the SAA methods rely on solving $M = 20$ MIPs, while the the complete stochastic programs are only solved once. In Table 4.2, the marked computation times (*) where obtained by solving the complete stochastic program once, rather than using the SAA method. For $n = 8$ customers, sample size $N = 64$ is used for SAA with Fixed A Priori Route and $N = 16$ for Full SAA. For $n = 12$ customers, sample size $N = 32$ is used for SAA with Fixed A Priori Route. Unfortunately, Full SAA could not solve these instances in a reasonable amount of time.

Table 4.2: Results for the instances with non-overlapping time slots.

				Exact Exp. Rev.				CPU time (s)		
					SAA	SAA			SAA	SAA
	n	w	T	LSH	Fixed	Full	No Slots	LSH	Fixed	Full
c60	4	0.25	0.90	1.73	1.75	1.79	1.79	0.00	1.49*	25.57*
		0.25	0.75	1.51	1.54	1.56	1.59	0.00	1.43*	28.36*
		0.25	0.60	1.26	1.28	1.31	1.31	0.00	0.82*	6.46*
c60	4	0.125	0.90	1.65	1.70	1.73	1.79	0.00	4.84*	70.78*
		0.125	0.75	1.39	1.53	1.55	1.59	0.00	2.69*	31.58*
		0.125	0.60	1.14	1.27	1.30	1.31	0.00	1.02*	9.61*
c60	8	0.25	0.90	3.66	3.70	3.77	3.79	0.12	166.66	3018.92
		0.25	0.75	3.06	3.22	3.29	3.36	0.16	268.51	6248.68
		0.25	0.60	2.49	2.61	2.75	2.72	0.17	59.94	1429.13
c60	8	0.125	0.90	3.60	3.63	3.72	3.79	0.12	724.26	4752.46
		0.125	0.75	2.90	3.18	3.27	3.36	0.16	735.78	5332.23
		0.125	0.60	2.28	2.59	2.72	2.72	0.17	246.61	1077.33
c60	12	0.25	0.90	5.56	5.69		5.81	2.58	303.39	
		0.25	0.75	4.79	4.96		5.17	4.80	649.04	
		0.25	0.60	3.90	4.05		4.23	6.07	337.70	
c60	12	0.125	0.90	5.44	5.63		5.81	2.82	1298.79	
		0.125	0.75	4.59	4.93		5.17	4.78	2204.55	
		0.125	0.60	3.67	4.02		4.23	5.73	994.01	

As expected, we see that the expected revenue depends on the horizon T, and that it increases for longer horizons. This is also illustrated in Figure 4.6 for the $n = 8$ customer instances. Not surprisingly, the best found solutions are obtained by Full SAA. Interestingly, the expected revenues obtained by Full SAA approach the No Slots upper bound. While SAA with Fixed A Priori Route does not always find the best solutions, the revenues obtained are on average only 3% worse and the computation times are much smaller. This indicates that the optimal TSP tour is good a priori tour. LSH obtains solutions that are on average 7% lower than the best found solutions, but the method requires relatively little computation time. Notice that the computation time required by LSH essentially consists of the single



(a) Time slot width $w = 0.25T^{\text{TSP}}$.

(b) Time slot width $w = 0.125T^{\text{TSP}}$.

Figure 4.6: Results for the instances with $n = 8$ customers and non-overlapping time slot sets. The error bars of the SAA methods indicate the average estimated optimality gap with the confidence interval. The grey dashed line highlights an (imaginary) linear relation between planning horizon and expected revenue.

expected revenue calculation. LSH performs especially well in case of a long horizon $T = 0.90T^{\text{TSP}}$. When decreasing the time slot width w, from $0.25T^{\text{TSP}}$ to $0.125T^{\text{TSP}}$, the expected revenue of the best found solutions (using Full SAA) decreases by only 1 – 2%. Also, the solutions obtained by SAA with Fixed A Priori Route have 1 – 2% lower expected revenue when lowering the time slot width, while the solutions obtained by LSH are much worse. The time slot set $\mathcal{T}$ contains twice as many time slots compared to width $w = 0.25T^{\text{TSP}}$. It seems that the simple decision rule used by LSH is less effective when time slots are smaller. Moreover, the computation time needed by both SAA methods increases, also probably because the number of possible time slots increases.

The SAA methods rely on repeated solving MIPs containing only a sampled subset of N scenarios. This sample size N not only affects the quality of the obtained solution, but also the computation time needed, and the quality of the obtained estimate of the optimality gap. Table 4.3 compares the quality of the solution, the estimated optimality gap (Equation (4.31)) and the confidence interval of the optimality gap for sample sizes of $N = 2$ up to 64 obtained by Full SAA. The reported confidence interval is the rightmost term in Equation (4.32) with $\alpha = 0.05$. We observe from the table that the estimated optimality gaps and the confidence intervals decrease as the sample size increases. We observe too that for the highest sample sizes, the estimated gap and the confidence interval are both relatively small and of the same order of magnitude, making it likely that a (near-)optimal solution is found.

The expected revenue does not increase much when increasing the sample size beyond a certain point, while computation times do increase noticeably. Similarly, Table 4.4 compares the quality of the solution, the estimated optimality gap (Equation (4.31)) and the confidence interval of the optimality gap for sample sizes of $N = 2$ up to 64 of SAA With Fixed A Priori Route. The estimated optimality gaps and confidence intervals decrease as the sample size increases. Compared to the Full SAA method, the computation times and the estimated optimality gaps are smaller, which indicates that this restricted problem is easier to solve. The interested reader is referred to Appendix 4.B, which contains more detailed results of our experiments.

Figure 4.7 shows the relation between solution quality and computation time for the $n = 8$ customer instances with time slot width $w = 0.125T^{\text{TSP}}$, averaged over all instances in the set and all horizons $T \in \{0.60, 0.75, 0.90\}\, T^{\text{TSP}}$. The expected revenue is shown as a fraction of the expected revenue of the best found solutions. The numbers above the points indicate the sample size N of the SAA methods. We see that the revenues associated with LSH solutions are on average around 90% of the revenues of the best found solutions, while SAA Fixed A Priori Route converges to around 97% of the revenues of the best found solutions. Sample sizes of $N = 16$ and higher contribute little to improve the solution quality compared to $N = 8$, while requiring more computation time. The best solutions are found by Full SAA. However, Full SAA performs worse than SAA Fixed A Priori Route for sample size $N = 2$. Samples with only two scenarios are likely too small to properly capture the stochastic information.

Table 4.3: Results of Full SAA with $M = 20$ samples. Each row shows averages over 10 instances with n customers and non-overlapping time slot sets.

				Exact Expected Revenue						Estimated Gap						Gap Confidence Interval ($\alpha = 0.05$)						CPU time (s)					
	n	w	T	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
c60	4	0.25	0.90	1.71	1.78	1.79	1.79	1.79	1.79	0.15	0.04	-0.04	-0.02	-0.01	-0.00	0.22	0.15	0.11	0.07	0.06	0.04	3.59	1.57	3.13	7.91	26.14	86.85
		0.25	0.75	1.54	1.56	1.56	1.56	1.56	1.56	0.07	0.05	-0.00	0.03	0.03	0.01	0.19	0.14	0.09	0.07	0.04	0.03	7.90	1.22	2.53	6.12	18.46	58.67
		0.25	0.60	1.30	1.30	1.31	1.31	1.31	1.31	-0.01	-0.03	0.00	0.01	0.01	-0.00	0.16	0.10	0.07	0.06	0.04	0.03	0.65	0.86	1.52	3.30	7.90	19.17
c60	4	0.125	0.90	1.71	1.72	1.73	1.73	1.73	1.73	0.07	0.09	0.05	0.02	0.03	0.02	0.22	0.15	0.10	0.07	0.05	0.04	13.80	1.99	3.74	11.40	49.02	234.12
		0.125	0.75	1.52	1.54	1.55	1.55	1.55	1.55	0.13	0.08	0.05	-0.01	0.01	0.02	0.19	0.12	0.09	0.07	0.05	0.03	0.66	0.91	2.20	5.70	16.63	92.60
		0.125	0.60	1.26	1.30	1.30	1.30	1.30	1.30	0.06	-0.00	0.01	0.00	-0.00	0.00	0.15	0.12	0.08	0.05	0.04	0.02	0.60	0.71	1.30	2.72	6.43	22.26
c60	8	0.25	0.90	3.50	3.66	3.75	3.77			0.38	0.16	0.05	0.01			0.36	0.23	0.16	0.11			48.63	126.27	1065.00	3018.92		
		0.25	0.75	3.10	3.20	3.26	3.29			0.38	0.22	0.13	0.08			0.32	0.21	0.14	0.09			62.74	182.23	809.21	6248.68		
		0.25	0.60	2.54	2.68	2.74	2.75			0.34	0.21	0.10	0.08			0.26	0.17	0.14	0.09			45.07	74.39	243.42	1429.13		
c60	8	0.125	0.90	3.43	3.66	3.70	3.72			0.43	0.16	0.18	0.02			0.33	0.24	0.16	0.12			41.27	130.25	740.59	4752.46		
		0.125	0.75	2.97	3.19	3.25	3.27			0.52	0.28	0.18	0.09			0.28	0.20	0.15	0.09			29.01	127.16	584.37	5332.23		
		0.125	0.60	2.57	2.63	2.71	2.72			0.36	0.24	0.11	0.04			0.25	0.19	0.11	0.09			25.49	58.10	155.56	1077.33		

Table 4.4: Results of SAA with Fixed A Priori Route with $M = 20$ samples. Each row shows averages over 10 instances with n customers and non-overlapping time slot sets.

				Exact Expected Revenue						Estimated Gap						Gap Confidence Interval ($\alpha = 0.05$)						CPU time (s)					
	n	w	T	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
c60	4	0.25	0.90	1.74	1.75	1.75	1.75	1.75	1.75	0.10	0.05	-0.02	-0.00	0.00	0.00	0.22	0.15	0.11	0.07	0.06	0.04	0.75	0.77	1.30	2.39	4.68	9.58
		0.25	0.75	1.54	1.54	1.54	1.54	1.54	1.54	0.07	0.05	-0.00	0.02	0.03	0.00	0.19	0.14	0.09	0.06	0.04	0.03	0.88	0.74	1.24	2.40	4.63	9.30
		0.25	0.60	1.28	1.28	1.28	1.28	1.28	1.28	0.01	-0.01	0.02	0.02	0.02	0.00	0.15	0.10	0.07	0.05	0.04	0.02	0.49	0.65	1.18	2.21	4.26	8.55
c60	4	0.125	0.90	1.70	1.70	1.70	1.70	1.70	1.70	0.07	0.08	0.04	0.02	0.03	0.01	0.22	0.14	0.10	0.07	0.05	0.04	1.74	0.89	1.60	2.97	6.99	21.96
		0.125	0.75	1.51	1.53	1.53	1.53	1.53	1.53	0.13	0.06	0.03	-0.03	0.00	0.00	0.19	0.12	0.09	0.07	0.05	0.03	0.51	0.81	1.44	2.65	5.20	17.47
		0.125	0.60	1.27	1.27	1.27	1.27	1.27	1.27	0.05	0.02	0.02	0.01	0.01	0.01	0.15	0.11	0.08	0.05	0.04	0.02	0.50	0.71	1.25	2.29	4.56	9.89
c60	8	0.25	0.90	3.57	3.70	3.70	3.70	3.70	3.70	0.27	0.07	0.03	0.01	0.01	0.02	0.35	0.23	0.16	0.11	0.08	0.05	4.24	5.45	8.58	16.74	40.77	166.66
		0.25	0.75	3.19	3.20	3.22	3.22	3.22	3.22	0.21	0.12	0.06	0.04	0.04	0.02	0.30	0.19	0.14	0.09	0.06	0.05	4.65	6.39	10.45	22.18	59.21	268.51
		0.25	0.60	2.57	2.60	2.61	2.61	2.61	2.61	0.19	0.15	0.08	0.06	0.03	0.01	0.25	0.16	0.13	0.08	0.06	0.04	4.95	6.11	8.89	14.75	27.39	59.94
c60	8	0.125	0.90	3.57	3.62	3.63	3.63	3.63	3.63	0.23	0.12	0.16	0.03	0.03	0.03	0.32	0.23	0.15	0.12	0.07	0.05	4.36	6.01	11.24	23.23	81.86	724.26
		0.125	0.75	3.06	3.18	3.18	3.18	3.18	3.18	0.33	0.17	0.12	0.06	0.04	0.03	0.27	0.19	0.15	0.09	0.07	0.05	4.81	6.72	11.78	26.16	77.38	735.78
		0.125	0.60	2.53	2.56	2.59	2.59	2.59	2.59	0.29	0.20	0.09	0.03	0.04	0.02	0.23	0.17	0.10	0.08	0.06	0.04	5.19	6.73	10.34	18.84	45.54	246.61
c60	12	0.25	0.90	5.50	5.68	5.69	5.69	5.69		0.24	0.12	0.02	0.02	0.03		0.45	0.30	0.21	0.16	0.09		63.80	57.72	65.10	106.05	303.39	
		0.25	0.75	4.80	4.94	4.96	4.95	4.96		0.46	0.19	0.15	0.08	0.04		0.37	0.26	0.17	0.13	0.09		86.74	88.76	112.23	202.94	649.04	
		0.25	0.60	3.96	4.02	4.04	4.05	4.05		0.51	0.27	0.17	0.12	0.08		0.32	0.22	0.15	0.11	0.08		103.15	108.98	120.03	161.55	337.70	
c60	12	0.125	0.90	5.32	5.62	5.63	5.63	5.63		0.56	0.28	0.11	0.07	0.02		0.42	0.30	0.18	0.13	0.10		67.38	61.96	81.77	217.91	1298.79	
		0.125	0.75	4.68	4.84	4.92	4.93	4.93		0.53	0.33	0.17	0.06	0.05		0.37	0.26	0.19	0.13	0.08		86.49	89.03	119.52	251.69	2204.55	
		0.125	0.60	3.87	3.98	4.00	4.02	4.02		0.57	0.33	0.22	0.03	0.07		0.31	0.22	0.16	0.11	0.09		100.80	104.58	127.63	204.72	994.01	

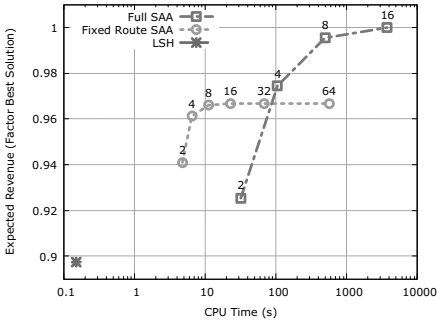


Figure 4.7: Solution quality versus computation time needed by the solution methods for solving the $n = 8$ customer instances with non-overlapping time slot of width $w = 0.125T^{\text{TSP}}$.

4.6.4 Overlapping Time Slots

To assess the benefit of having overlapping time slots, we also solve the instance sets with the time slot set (4.36) using LSH, SAA With Fixed A Priori Route and Full SAA. Table 4.5 shows the expected revenue and computation time in seconds for the methods, using the same sample sizes as in Table 4.2. We also report the expected revenue and computation time as a fraction of the corresponding values for the non-overlapping time slot set. The methods are tested on the instances with $n = 8$ and 12 customers, with time slot width $w = 0.25T^{\text{TSP}}$ and $0.125T^{\text{TSP}}$, and with horizons $T \in \{0.90, 0.75, 0.60\}\, T^{\text{TSP}}$. The column "Ov." indicates how many time slots in $\mathcal{T}$ are overlapping for any single point in time. We observe that the revenues increase by around 3 – 5 % when using LSH, by around 2 – 3 % when using SAA with Fixed A Priori Route and 1% – 2% when using Full SAA, while the best solutions are again obtained by Full SAA for the $n = 8$ customer instances. However, the computation times for the SAA methods increase dramatically: computation times increased 2 to 6 times when solving the instances with the overlapping time slot set compared to solving the instances with the non-overlapping time slot set. Thus, the use of overlapping time slots leads improves the quality of the a priori route and time slot assignment, but, in case of the SAA methods, also to higher computation times.

4.6.5 Non-Anticipation and Ascending Time Slots

So far, we have investigated the performance of LSH, SAA with Fixed A Priori Route and Full SAA. In Section 4.5, we presented two additional heuristics based on sim-

Table 4.5: Results for the instances with overlapping time slots.

					Exact Exp. Rev.			Exact Exp. Rev. (Fact. Non-overlap)			CPU time (s)			CPU time (Fact. Non-overlap)		
	n	w	Ov.	T	LSH	SAA Fixed	SAA Full	LSH	SAA Fixed	SAA Full	LSH	SAA Fixed	SAA Full	LSH	SAA Fixed	SAA Full
c60	8	0.25	2	0.90	3.79	3.79	3.81	1.036	1.025	1.011	0.11	354.53	11033.42	0.970	2.127	3.655
		0.25	2	0.75	3.26	3.30	3.35	1.064	1.027	1.019	0.16	838.62	32696.81	0.973	3.123	5.233
		0.25	2	0.60	2.58	2.67	2.78	1.039	1.021	1.010	0.18	171.17	7666.13	1.044	2.855	5.364
c60	8	0.125	2	0.90	3.72	3.73	3.76	1.035	1.028	1.013	0.12	806.40	10819.73	0.967	1.113	2.277
		0.125	2	0.75	3.02	3.24	3.29	1.040	1.017	1.008	0.16	1449.00	31261.28	0.988	1.969	5.863
		0.125	2	0.60	2.35	2.63	2.74	1.033	1.015	1.007	0.17	533.17	5578.24	1.008	2.162	5.178
c60	12	0.25	2	0.90	5.81	5.81		1.046	1.021		2.20	871.08		0.856	2.871	
		0.25	2	0.75	5.06	5.13		1.056	1.035		4.80	4389.65		1.000	6.763	
		0.25	2	0.60	4.05	4.18		1.038	1.033		6.26	1878.74		1.031	5.563	
c60	12	0.125	2	0.90	5.72	5.76		1.050	1.022		2.55	1828.14		0.905	1.408	
		0.125	2	0.75	4.72	5.01		1.030	1.016		4.81	15955.04		1.007	7.237	
		0.125	2	0.60	3.76	4.06		1.026	1.011		5.54	6772.20		0.967	6.813	

plifications of the stochastic program: SAA without Non-Anticipation and SAA with Ascending Time Slots. In SAA without Non-Anticipation, the non-anticipatory constraints are relaxed, while in SAA with Ascending Time Slots, constraints requiring the time slots to be ascending along the a priori route are added. Next, we investigate the benefits of these two simplifications when they are incorporated in SAA with Fixed A Priori Route.

Table 4.6 shows the expected revenue and computation time of SAA without Non-Anticipation and SAA with Ascending Time Slots as a fraction of the corresponding values when using SAA with Fixed A Priori Route, for non-overlapping time slot set (4.35) and time slot width $w = 0.125T^{\text{TSP}}$, which are the more difficult instance sets. We see that SAA without Non-Anticipation for high sample sizes does not affect the solution quality, but that it requires less computation time, for the $n = 8$ customer instance set on average only 21% of the computation time required with customer anticipation. For the $n = 12$ customer instance set, the computation time benefits are mixed. When the horizon is long ($T = 0.9T^{\text{TSP}}$), computation times are significantly smaller, but when the horizon is short ($T = 0.6T^{\text{TSP}}$), the computation time significantly increases. We observe more consistent computational benefits for SAA with Ascending Time Slots, especially for large sample sizes ($N \geq 16$). Interestingly, for small sample sizes ($N \leq 4$) and long horizons ($T = 0.75T^{\text{TSP}}$ and $T = 0.90T^{\text{TSP}}$) enforcing ascending time slots results in an increase in expected revenue. More generally, when the horizon is long, restricting time slots to be ascending along the (fixed TSP) a priori route does not appear to be limiting (in fact, it may offer a benefit). This is likely due to the fact that with a long horizon, most customer arrival sequences lead in their entirety to feasible delivery routes, which, by construction, visit customer locations in the same order as the a priori route.

Table 4.6: Results of SAA without Non-Anticipation and SAA with Ascending Time Slots.

				Exact Expected Revenue (Factor of Fixed Route SAA)												CPU time (Factor of Fixed Route SAA)											
				Fixed Route SAA without Non-Anticip.						Fixed Route SAA with Asc. Time Slots						Fixed Route SAA without Non-Anticip.						Fixed Route SAA with Asc. Time Slots					
	n	w	T	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
c60	8	0.125	0.90	1.009	1.003	1.000	1.000	1.000	1.000	1.013	1.002	1.000	1.000	1.000	1.000	0.609	0.514	0.437	0.374	0.296	0.138	0.810	0.820	0.782	0.743	0.646	0.578
		0.125	0.75	1.023	0.993	1.001	0.999	0.999	0.999	1.036	1.002	1.001	1.000	1.000	1.000	0.644	0.559	0.535	0.544	0.429	0.203	0.797	0.797	0.733	0.637	0.640	0.523
		0.125	0.60	1.018	1.007	0.997	0.999	0.999	1.000	1.002	0.995	0.983	0.982	0.982	0.982	0.668	0.616	0.584	0.561	0.505	0.296	0.810	0.779	0.740	0.665	0.577	0.435
c60	12	0.125	0.90	0.995	1.001	1.000	1.000	1.000		1.050	1.001	1.000	1.000	1.000		1.259	1.203	0.959	0.480	0.195		0.890	0.983	0.967	0.833	0.752	
		0.125	0.75	1.019	1.007	1.001	1.000	1.000		1.047	1.014	1.001	1.000	0.999		1.271	1.220	1.096	1.877	0.742		1.004	1.004	0.926	0.704	0.549	
		0.125	0.60	1.005	0.999	0.999	0.999	1.000		1.016	0.990	0.988	0.986	0.986		1.251	1.248	1.290	2.028	2.154		1.076	1.042	0.950	0.719	0.422	

4.7 Discussion and Future Research Directions

In this chapter, we introduce Strategic Time Slot Management, a novel variant of Time Slot Management that simplifies the management of time slots during the ordering process, allows smoothing of fulfillment center operations, and creates delivery consistency. We investigate the design problem for the single-vehicle (or single-route) case. We model the design problem, which involves finding an a priori route and a time slot assignment, as a two-stage stochastic program (where the second stage is an "evaluation" stage modeling the the customer ordering process). We develop a Sample Average Approximation approach for its solution (which uses a highly efficient Dynamic Programming algorithm to evaluate the expected revenue over all possible order placement sequences). An extensive computational study shows the efficacy of the solution approach and provides valuable insights in the benefits of Strategic Time Slot Management.

We see many opportunities for further research into this exciting innovative practice in the online grocery retail sector. We discuss some of these in this final section. Specifically, we consider the extension to the multi-vehicle setting, the aggregation of customer locations, the dynamic management of a priory routes, and the use of Benders Decomposition to improve the efficiency of the Sample Average Approximation approach.

In practice, online grocery retailers employ multiple vehicles to serve their customers. Our two-stage stochastic program can be modified to allow for multiple a priori routes, but it is highly likely that its solution will be more challenging, because the a priori routes need to be balanced across the customer locations. However, a few natural phases solution approaches are worth exploring. Once it has been decided which groups of customer locations to serve using a single a priori route, the problem decomposes into single-vehicle problems for each group of customer locations, and either one of the solution approaches developed in this chapter can be applied. From a business perspective, the multi-vehicle case opens up other interesting opportunities. By allowing customer locations to be included in multiple a priori routes, their service can be increased.

In our definition of the Strategic Time Slot Management design problem, we have assumed that the locations that have to be visited represent a delivery address. In practice, it is more likely that the locations that have to be visited represent areas (i.e., groups or clusters of delivery addresses). In such an environment, new design decisions arise, e.g., how to define the areas, and the proposed methodology may have to be revisited. It is possible to adjust the probabilities p_i (and revenues r_i) to

reflect knowledge about the customers in an area, but it may be better to consider a different approach, e.g., a discrete probability distribution for the number of order placements in an area. Furthermore, it may no longer be reasonable to assume that the service time at a "location" is constant.

In practice, the set of customers may vary over time (Picnic's client base is growing rapidly) and the customer characteristics may vary over time (e.g., order placement probability and revenue). This suggest dynamic management of the a priori routes and time slot assignments. However, customer value consistency and may not like to see their assigned time slots change (too) frequently.

Our presented two-stage stochastic program contains many big-M constraints, especially in the second stage, which results in weak LP relaxations and, consequently, large search trees and long solve times. However, as we have seen, given an a priori route and time slot assignment, the second stage is "simply" an evaluation of the expected revenue. Benders decomposition may allow us to solve the integer programs more efficient by exploiting this structure, i.e., incorporate knowledge about the expected costs of a design in the form of optimality cuts. Only further research can tell whether this can lead to computationally viable approaches.

Appendix

4.A Non-anticipatory Constraints

In this appendix, we present the non-anticipatory constraints of the stochastic programming formulation in Section 4.4.1. These constraints ensure that the hth arriving customer $\omega(h)$ in scenario ω places an order if and only if it can be feasibly inserted in the current delivery route (containing only placed customers up to $h-1$). We introduce the following additional decision variables: $e_i^{\omega h}$ and $l_i^{\omega h}$ are the earliest arrival time and latest arrival time at customer i, respectively, given the current delivery route after h customer arrivals in scenario ω. In particular, $e_{\omega(h)}^{\omega h}$ ($l_{\omega(h)}^{\omega h}$) are the earliest (latest) arrival times given that current arriving customer $\omega(h)$ is inserted (even if infeasable) in the delivery route. Furthermore, $\tilde{x}_i^{\omega hF}$ and $\tilde{x}_j^{\omega hB}$ denote if forward arc $(i, \omega(h))$ and backward arc $(\omega(h), j)$, respectively, are used when inserting customer $\omega(h)$ in the current delivery route. Note that we cannot use the delivery route variables $x_{ij}^{\omega h}$ since insertion of customer $\omega(h)$ might be infeasable, and thus this customer will not be inserted into the delivery route. Finally, decision variables $w_i^{\omega hF}$ and $w_i^{\omega hB}$ and indicate if there is waiting time at customer i due to

the time slot, in case of earliest arrival time (F) and in case of latest arrival time (B), respectively. These are essentially used for booking-keeping. Let $t^{\max}$ be equal to the maximum travel time and let ϵ be the travel time precision.

The non-anticipatory constraints are given by:

$$e_j^{\omega h} \geq \sum_{s \in \mathcal{T}_j} a_s y_{js} \qquad \forall j \in \mathcal{V}_{\mathrm{c}}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.37}$$

$$e_j^{\omega h} \geq e_i^{\omega h} + t_{ij} - T(1 - x_{ij}^{\omega h}) \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ i, j \neq \omega(h),\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.38}$$

$$e_{\omega(h)}^{\omega h} \geq e_i^{\omega h} + t_{i,\omega(h)} - (T + t^{\max})(1 - \tilde{x}_i^{\omega h\mathrm{F}}) \qquad \forall (i,\omega(h)) \in \mathcal{A}^{\omega h}, h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.39}$$

$$e_j^{\omega h} \leq \sum_{s \in \mathcal{T}_j} a_s y_{js} + (T + t^{\max})(1 - w_j^{\omega h\mathrm{F}}) \qquad \forall j \in \mathcal{V}_{\mathrm{c}}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.40}$$

$$e_j^{\omega h} \leq e_i^{\omega h} + t_{ij} + T(1 + w_j^{\omega h\mathrm{F}} - x_{ij}^{\omega h}) \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ i, j \neq \omega(h),\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.41}$$

$$e_{\omega(h)}^{\omega h} \leq e_i^{\omega h} + t_{i,\omega(h)} + (T + t^{\max})(1 + w_{\omega(h)}^{\omega h\mathrm{F}} - \tilde{x}_i^{\omega h\mathrm{F}}) \qquad \forall (i,\omega(h)) \in \mathcal{A}^{\omega h}, h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.42}$$

$$l_i^{\omega h} \leq \sum_{s \in \mathcal{T}_i} b_s y_{is} \qquad \forall i \in \mathcal{V}_{\mathrm{c}}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.43}$$

$$l_i^{\omega h} \leq l_j^{\omega h} - t_{ij} + T(1 - x_{ij}^{\omega h}) \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ i, j \neq \omega(h),\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.44}$$

$$l_{\omega(h)}^{\omega h} \leq l_j^{\omega h} - t_{\omega(h),j} + (T + t^{\max})(1 - \tilde{x}_j^{\omega h\mathrm{B}}) \qquad \forall (\omega(h),j) \in \mathcal{A}^{\omega h}, h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.45}$$

$$l_i^{\omega h} \geq \sum_{s \in \mathcal{T}_j} b_s y_{is} - (T + t^{\max})(1 - w_i^{\omega h\mathrm{B}}) \qquad \forall i \in \mathcal{V}_{\mathrm{c}}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.46}$$

$$l_i^{\omega h} \geq l_j^{\omega h} - t_{ij} - T(1 + w_i^{\omega h\mathrm{B}} - x_{ij}^{\omega h}) \qquad \forall (i,j) \in \mathcal{A}^{\omega h},\ i, j \neq \omega(h),\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.47}$$

$$l_{\omega(h)}^{\omega h} \geq l_j^{\omega h} - t_{\omega(h),j} - (T + t^{\max})(1 + w_{\omega(h)}^{\omega h\mathrm{B}} - \tilde{x}_j^{\omega h\mathrm{B}}) \qquad \forall (\omega(h),j) \in \mathcal{A}^{\omega h}, h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.48}$$

$$e_o^{\omega h} = a_o = 0,\ l_d^{\omega h} = b_d = T \qquad \forall h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.49}$$

$$\sum_{i \in \mathcal{V}^{\omega h} \setminus \{\omega(h), d\}} \tilde{x}_i^{\omega h\mathrm{F}} = 1 \qquad \forall h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.50}$$

$$\sum_{j \in \mathcal{V}^{\omega h} \setminus \{o, \omega(h)\}} \tilde{x}_j^{\omega h\mathrm{B}} = 1 \qquad \forall h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.51}$$

$$\tilde{x}_i^{\omega h\mathrm{F}} + \tilde{x}_j^{\omega h\mathrm{B}} \leq 1 + x_{ij}^{\omega, h-1} \qquad \forall (i,j) \in \mathcal{A}^{\omega, h-1},\ (i,j) \neq (o,d),\ h \in \{2, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.52}$$

$$\tilde{x}_o^{\omega h\mathrm{F}} + \tilde{x}_d^{\omega h\mathrm{B}} \leq 2 - \sum_{j \in \mathcal{V}^{\omega, h-1} \setminus \{d\}} x_{oj}^{\omega, h-1} \qquad \forall h \in \{2, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.53}$$

$$u_i + 1 \leq u_{\omega(h)} + n(1 - \tilde{x}_i^{\omega h\mathrm{F}}) \qquad \forall (i,\omega(h)) \in \mathcal{A}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.54}$$

$$u_{\omega(h)} + 1 \leq u_j + n(1 - \tilde{x}_j^{\omega h\mathrm{B}}) \qquad \forall (\omega(h),j) \in \mathcal{A}^{\omega h},\ h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.55}$$

$$e_{\omega(h)}^{\omega h} \leq l_{\omega(h)}^{\omega h} + 2(T + t^{\max})(1 - z_{\omega(h)}^{\omega}) \qquad \forall h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.56}$$

$$e_{\omega(h)}^{\omega h} \geq l_{\omega(h)}^{\omega h} + \epsilon - 2(T + t^{\max}) z_{\omega(h)}^{\omega} \qquad \forall h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.57}$$

$$w_i^{\omega h\mathrm{F}} \in \{0,1\},\ w_j^{\omega h\mathrm{B}} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}^{\omega h}, h \in \{1, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.58}$$

$$\tilde{x}_i^{\omega h\mathrm{F}} \in \{0,1\},\ \tilde{x}_j^{\omega h\mathrm{B}} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}^{\omega, h-1}, h \in \{2, \ldots, |\omega|\},\ \omega \in \Omega, \tag{4.59}$$

$$\tilde{x}_o^{\omega 1\mathrm{F}} = 1,\ \tilde{x}_d^{\omega 1\mathrm{B}} = 1 \qquad \forall \omega \in \Omega. \tag{4.60}$$

Essentially, the first constraints ensure the earliest arrival times and latest arrival times are set exactly. These are obtained by linearizing:
$e_j^{\omega h} = \max\left\{\sum_{s\in\mathcal{T}_j} a_s y_{js},\, e_i^{\omega h} + t_{ij}\right\}$ in case $x_{ij}^{\omega h} = 1$ and
$e_{\omega(h)}^{\omega h} = \max\left\{\sum_{s\in\mathcal{T}_{\omega(h)}} a_s y_{\omega(h)s},\, e_i^{\omega h} + t_{i,\omega(h)}\right\}$ in case $\tilde{x}_i^{\omega hF} = 1$, and
$l_i^{\omega h} = \min\left\{\sum_{s\in\mathcal{T}_i} b_s y_{is},\, l_j^{\omega h} - t_{ij}\right\}$ in case $x_{ij}^{\omega h} = 1$ and
$l_{\omega(h)}^{\omega h} = \min\left\{\sum_{s\in\mathcal{T}_{\omega(h)}} b_s y_{\omega(h)s},\, l_j^{\omega h} - t_{\omega(h),j}\right\}$ in case $\tilde{x}_j^{\omega hB} = 1$. Next, the $\tilde{x}_i^{\omega hF}$ and $\tilde{x}_j^{\omega hB}$ decision variables are set correctly. Finally, the feasibility check involves checking if $e_{\omega(h)}^{\omega h} \leq l_{\omega(h)}^{\omega h}$. The revenue $z_{\omega(h)}^{\omega}$ of the hth arriving customer in scenario ω is only obtained if and only if $e_{\omega(h)}^{\omega h} \leq l_{\omega(h)}^{\omega h}$. The time precision ϵ is used to capture the infeasible case: $e_{\omega(h)}^{\omega h} \geq l_{\omega(h)}^{\omega h} + \epsilon$.

4.B Detailed Results

In this section, we provide some detailed results of our computational study. In the following tables, the left column contains the instance names, which consist of "STSM_c60_", followed by the number of customer locations $n \in \{4, 8, 12\}$, followed by the instance number (1–10), "_Wf" with the time slot width factor ("90", "75", "60"), followed by "o2" in case of the overlapping time slot set, and followed by "_Tf" with the planning horizon factor ("25", "12-5").

Table 4.7: Detailed results for the Linear Scaling Heuristic, the SAA method with Fixed A Priori Route and the full SAA method, over the $n = 4$ customer instances with non-overlapping set of possible time slots. Bold expected revenues indicate the best known solution.

	Exact Expected Revenue												CPU time (s)											
		Fixed Route SAA						Full SAA						Fixed Route SAA						Full SAA				
	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
STSM_c60_4_1_Wf25_Tf90	1.6615	1.6615	1.6615	1.6615	1.6615	1.6615	1.6615	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	0.0024	2.8154	0.8644	1.4682	2.5595	5.2293	11.8274	25.2291	1.6237	3.6011	9.8755	31.1044	123.1838
STSM_c60_4_2_Wf25_Tf90	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0031	0.4909	0.6796	1.1927	2.3375	4.2163	9.2481	0.5308	0.6891	2.0809	3.8852	8.3945	41.7038
STSM_c60_4_3_Wf25_Tf90	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0023	0.4273	0.7656	1.2991	2.5274	4.6988	9.7983	0.8247	1.2134	1.8576	7.6939	17.1046	95.0539
STSM_c60_4_4_Wf25_Tf90	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0024	0.4956	0.6977	1.3499	2.3511	4.7108	9.4819	0.5260	0.8053	1.5980	4.6505	13.8634	31.7876
STSM_c60_4_5_Wf25_Tf90	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0021	0.6368	0.7851	1.4786	2.3171	4.5692	8.7096	2.0146	1.5829	5.0816	9.1930	31.4700	91.3528
STSM_c60_4_6_Wf25_Tf90	1.8021	1.8021	1.8021	1.8021	1.8021	1.8021	1.8021	1.8021	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0032	0.5775	0.8206	1.2226	2.1618	5.0714	9.4624	1.5233	1.9320	2.5396	4.2746	29.2446	83.3330
STSM_c60_4_7_Wf25_Tf90	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0022	0.4816	0.7497	1.3151	2.4907	4.6567	9.0390	1.0653	3.4697	4.9168	14.9386	48.4599	132.4651
STSM_c60_4_8_Wf25_Tf90	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	1.7500	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0032	0.5695	0.6900	1.2878	2.3719	4.9947	9.4658	2.0288	1.5289	4.9127	11.1669	41.0009	108.4211
STSM_c60_4_9_Wf25_Tf90	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	0.0026	0.5359	0.7925	1.2765	2.4725	4.2511	9.3801	0.8706	1.1230	2.0323	4.7806	10.0183	50.0919
STSM_c60_4_10_Wf25_Tf90	1.6615	1.7500	1.8021	1.8021	1.8021	1.8021	1.8021	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0022	0.5155	0.8425	1.1304	2.2799	4.3995	9.4262	1.3343	1.7247	2.7198	8.6871	30.7061	111.1003
STSM_c60_4_1_Wf25_Tf75	1.4792	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0041	0.5348	0.8195	1.2399	2.7573	4.7782	10.0782	0.7855	0.9505	2.7119	10.3745	22.4577	86.4923
STSM_c60_4_2_Wf25_Tf75	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0024	0.5078	0.7021	1.1473	2.1543	3.9260	7.9171	0.6082	0.6699	1.1235	2.1235	4.9994	10.0728
STSM_c60_4_3_Wf25_Tf75	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	1.3958	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	0.0031	0.4380	0.7377	1.1834	2.1342	4.2448	9.1078	0.7640	1.4940	2.7186	6.4130	14.8199	63.7753
STSM_c60_4_4_Wf25_Tf75	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0040	0.4202	0.6292	1.1739	2.3418	4.1000	8.3974	0.4971	0.6523	1.5417	2.7402	6.0544	19.2312
STSM_c60_4_5_Wf25_Tf75	1.5625	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0042	3.1403	0.8872	1.2740	2.5439	5.0595	10.4118	64.5953	3.0614	5.2664	11.1210	36.1713	127.5366
STSM_c60_4_6_Wf25_Tf75	1.4792	1.4688	1.4792	1.4792	1.4792	1.4792	1.4792	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0024	0.4787	0.7631	1.2345	2.4812	4.8103	9.5262	0.5882	0.7045	1.4973	3.9339	14.6264	37.1571
STSM_c60_4_7_Wf25_Tf75	1.4688	1.5833	1.5833	1.5833	1.5833	1.5833	1.5833	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	**1.6615**	0.0029	0.5408	0.6867	1.1922	2.4978	4.8892	10.0838	1.2724	1.1611	3.1828	8.4496	28.5784	96.9226
STSM_c60_4_8_Wf25_Tf75	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0027	0.5889	0.7355	1.3146	2.2828	4.8438	9.3381	0.9659	1.3857	3.4231	6.4650	23.8824	64.5649
STSM_c60_4_9_Wf25_Tf75	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0024	1.5800	0.6631	1.2384	2.3909	4.5175	8.7567	8.1236	0.6615	1.2043	2.3138	6.7692	13.2432
STSM_c60_4_10_Wf25_Tf75	1.5625	1.5625	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0033	0.5903	0.7516	1.3969	2.3756	5.1798	9.3854	0.7811	1.4944	2.6549	7.2672	26.2311	67.7513
STSM_c60_4_1_Wf25_Tf60	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	0.0033	0.5032	0.7752	1.0546	2.2310	4.5086	9.0551	1.1184	1.9572	1.5741	4.8055	13.5819	27.3861
STSM_c60_4_2_Wf25_Tf60	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	0.0024	0.4658	0.6333	0.9596	1.8514	3.5886	7.6681	0.5865	0.6402	1.0163	1.8270	3.5581	7.7978
STSM_c60_4_3_Wf25_Tf60	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	0.0025	0.4593	0.5674	1.1517	2.1471	3.9949	8.2498	0.5151	0.5274	1.1016	2.0517	3.8511	8.2685
STSM_c60_4_4_Wf25_Tf60	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0025	0.5309	0.5719	1.1075	2.3156	4.1835	7.8556	0.5915	0.5800	1.0915	2.3628	4.3407	8.9472
STSM_c60_4_5_Wf25_Tf60	1.1146	1.1146	1.1146	1.1146	1.1146	1.1146	1.1146	**1.2552**	**1.2552**	**1.2552**	**1.2552**	**1.2552**	**1.2552**	0.0028	0.4225	0.6150	1.3255	2.2888	4.5850	8.8924	0.5125	0.6036	1.1124	2.2844	4.5782	10.8052
STSM_c60_4_6_Wf25_Tf60	1.2552	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	1.2552	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	0.0025	0.4774	0.6615	1.1647	2.3665	4.0090	8.6591	0.4548	1.0170	2.3503	5.1282	10.6993	24.0093
STSM_c60_4_7_Wf25_Tf60	1.3750	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	**1.4792**	1.3958	**1.4792**	**1.4792**	**1.4792**	**1.4792**	0.0027	0.6103	0.6621	1.2510	2.1507	4.7050	9.4230	0.8658	0.8959	2.4721	4.6770	14.9027	44.5551
STSM_c60_4_8_Wf25_Tf60	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	0.0027	0.5264	0.6921	1.2027	2.2351	4.3561	8.7167	0.9045	0.9145	1.5610	3.7322	9.9562	28.0523
STSM_c60_4_9_Wf25_Tf60	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0023	0.4485	0.6318	1.2761	2.0957	4.1100	8.4794	0.4328	0.6282	1.3166	2.4432	4.6381	12.5873
STSM_c60_4_10_Wf25_Tf60	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	0.0026	0.4909	0.6441	1.3433	2.4315	4.5552	8.5428	0.5558	0.8455	1.6513	3.6496	8.9288	19.3212
STSM_c60_4_1_Wf12-5_Tf90	1.6615	1.6615	1.6615	1.6615	1.6615	1.6615	1.6615	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	0.0024	7.5700	0.9299	1.5761	3.3393	10.9902	38.4730	86.2555	1.6130	3.6359	16.1260	55.6250	247.7796
STSM_c60_4_2_Wf12-5_Tf90	1.3750	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0032	0.4582	0.8574	1.4453	2.5554	4.9989	10.8290	0.4920	1.0970	1.3952	4.0245	11.6819	43.5793
STSM_c60_4_3_Wf12-5_Tf90	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	0.0023	0.5048	0.9131	1.6266	2.9285	5.8763	17.4688	0.6895	1.8726	4.2310	11.9103	53.4223	280.7118
STSM_c60_4_4_Wf12-5_Tf90	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0036	0.5659	0.8316	1.6485	2.7910	5.8778	11.9474	0.5474	1.1867	2.0411	4.1659	15.0265	46.7993
STSM_c60_4_5_Wf12-5_Tf90	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	0.0027	0.5804	0.8738	1.7252	3.0778	8.6712	22.3905	1.2922	2.1237	5.0661	18.3024	81.5265	421.8939
STSM_c60_4_6_Wf12-5_Tf90	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	**1.8021**	1.7500	**1.8021**	**1.8021**	**1.8021**	**1.8021**	0.0024	5.6047	0.8421	1.3051	3.0585	7.1254	25.6569	44.5425	1.1427	2.3842	10.8330	57.2006	238.5430
STSM_c60_4_7_Wf12-5_Tf90	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	**1.8750**	**1.8750**	**1.8750**	**1.8750**	**1.8750**	0.0023	0.4848	0.9489	1.6744	3.0593	7.0180	23.2387	0.4630	4.4672	4.1142	11.6297	54.0899	186.4464
STSM_c60_4_8_Wf12-5_Tf90	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	**1.7500**	0.0022	0.5025	0.9682	1.7258	3.1715	6.4608	26.3827	2.1051	4.3455	7.5056	15.0714	66.0987	444.9649
STSM_c60_4_9_Wf12-5_Tf90	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0025	0.5311	0.8264	1.6370	2.7554	5.8854	20.4292	0.5380	0.9501	2.1475	7.6439	25.5300	119.6919
STSM_c60_4_10_Wf12-5_Tf90	1.5000	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	1.7500	**1.8021**	**1.8021**	**1.8021**	**1.8021**	0.0024	0.6351	0.9103	1.5991	2.9580	6.9573	22.7811	1.1028	1.1064	4.8624	14.3005	70.4046	310.7989
STSM_c60_4_1_Wf12-5_Tf75	1.3958	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0040	0.4688	0.9131	1.5397	2.8105	5.9529	26.0239	0.5063	1.0347	2.8046	7.1306	25.3897	132.7182
STSM_c60_4_2_Wf12-5_Tf75	1.2552	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0025	0.4389	0.7586	1.2524	2.2051	4.6475	9.7199	0.5019	0.6765	1.4352	2.0027	5.7087	20.3300
STSM_c60_4_3_Wf12-5_Tf75	1.3750	**1.3958**	1.3750	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	0.0025	0.4947	0.7691	1.4590	2.4011	4.6586	10.8821	0.6158	0.7406	1.7315	3.8623	13.3538	61.0304
STSM_c60_4_4_Wf12-5_Tf75	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0039	0.4942	0.8202	1.5609	2.4940	4.8821	10.0908	0.6489	0.8866	2.0406	3.0190	7.9366	35.4175
STSM_c60_4_5_Wf12-5_Tf75	1.3958	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0031	0.5205	0.9002	1.4825	2.8260	5.7130	28.5125	0.6399	1.1545	2.5329	8.3855	26.9210	189.2329
STSM_c60_4_6_Wf12-5_Tf75	1.3594	1.4688	1.4688	1.4688	1.4688	1.4688	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0025	0.5187	0.7520	1.2792	2.7095	5.0596	10.4021	0.5222	0.7772	1.3208	3.1917	10.6988	43.7553
STSM_c60_4_7_Wf12-5_Tf75	1.2552	1.5833	1.5833	1.5833	1.5833	1.5833	1.5833	1.5833	1.5833	**1.6615**	**1.6615**	**1.6615**	**1.6615**	0.0027	0.5966	0.8252	1.5485	2.9704	5.4037	21.6392	1.1146	0.9836	3.1228	10.7572	24.2798	143.0879
STSM_c60_4_8_Wf12-5_Tf75	1.3750	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	1.4688	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0026	0.5335	0.7825	1.5225	3.0910	5.5869	23.9201	0.7562	1.2579	3.4717	11.1544	29.1031	143.6727
STSM_c60_4_9_Wf12-5_Tf75	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0024	0.5331	0.8147	1.3303	2.4131	5.0020	10.8423	0.6541	0.7813	1.4196	2.3552	5.3715	22.3695
STSM_c60_4_10_Wf12-5_Tf75	1.3958	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	**1.5833**	0.0030	0.5449	0.7710	1.4305	2.5543	5.0622	22.6646	0.6690	0.7951	2.0864	5.0258	17.4979	134.3897
STSM_c60_4_1_Wf12-5_Tf60	0.9375	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	0.0028	0.5139	0.7990	1.4122	2.4842	4.8346	11.3654	0.6427	0.8571	1.4942	3.1194	9.0814	44.7025
STSM_c60_4_2_Wf12-5_Tf60	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	**0.8750**	0.0023	0.4078	0.5742	1.1574	1.9290	4.0797	8.0723	0.4940	0.5631	1.0847	1.8122	3.8803	7.6118
STSM_c60_4_3_Wf12-5_Tf60	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	**1.1146**	0.0026	0.5521	0.7569	1.1221	2.2552	4.5229	9.4198	0.6480	0.6971	1.0485	2.1206	4.2972	8.7764
STSM_c60_4_4_Wf12-5_Tf60	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0033	0.4945	0.6194	1.1077	2.0981	4.2030	8.3652	0.5886	0.5950	1.1033	2.0456	4.3137	8.9354
STSM_c60_4_5_Wf12-5_Tf60	0.9375	1.1146	1.1146	1.1146	1.1146	1.1146	1.1146	**1.2552**	**1.2552**	**1.2552**	**1.2552**	**1.2552**	**1.2552**	0.0028	0.4771	0.7513	1.2902	2.2940	4.8291	11.0497	0.5610	0.6626	1.2262	2.6916	5.6144	16.2488
STSM_c60_4_6_Wf12-5_Tf60	0.9375	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	1.2552	**1.3594**	**1.3594**	**1.3594**	**1.3594**	**1.3594**	0.0028	0.4840	0.7366	1.1281	2.1817	4.2876	9.1154	0.6192	0.7783	1.2534	2.5460	6.2173	17.2593
STSM_c60_4_7_Wf12-5_Tf60	1.3750	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	**1.3958**	0.0026	0.5612	0.7611	1.3799	2.4012	5.2520	9.9638	0.6873	0.8685	1.8031	4.4869	13.0605	43.7891
STSM_c60_4_8_Wf12-5_Tf60	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	1.2552	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	0.0025	0.4950	0.7018	1.3191	2.5396	5.0070	11.2721	0.6222	0.7712	1.5253	3.2125	7.6117	30.7989
STSM_c60_4_9_Wf12-5_Tf60	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	**1.4688**	0.0024	0.4412	0.6848	1.1355	2.0758	4.1465	8.8745	0.5738	0.6717	1.1172	2.2641	4.6802	10.8011
STSM_c60_4_10_Wf12-5_Tf60	0.9375	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	1.2552	**1.3750**	**1.3750**	**1.3750**	**1.3750**	**1.3750**	0.0042	0.5429	0.6667	1.4579	2.6234	4.4167	11.3971	0.6061	0.6779	1.3233	2.8905	5.5714	33.6590

Table 4.8: Detailed results for the Linear Scaling Heuristic, the SAA method with Fixed A Priori Route and the full SAA method, over the $n = 8$ customer instances with non-overlapping set of possible time slots. Bold expected revenues indicate the best known solution.

	Exact Expected Revenue												CPU time (s)											
		Fixed Route SAA						Full SAA					Fixed Route SAA						Full SAA					
	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$			
STSM_c60_8_1_Wf25_Tf90	**3.9031**	**3.9031**	**3.9031**	**3.9031**	**3.9031**	**3.9031**	**3.9031**	3.6094	**3.9031**	**3.9031**	**3.9031**	0.0950	3.4415	4.4383	6.6333	11.4799	21.9048	43.9930	16.5561	40.0104	195.6657	1776.0971		
STSM_c60_8_2_Wf25_Tf90	3.5232	3.3281	**3.7271**	**3.7271**	**3.7271**	**3.7271**	**3.7271**	3.3739	3.6094	**3.7271**	**3.7271**	0.1154	3.9636	4.8255	8.5346	15.1091	27.2425	88.3367	40.3693	121.2001	738.6884	2133.1758		
STSM_c60_8_3_Wf25_Tf90	3.7822	3.8496	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8309	**3.8752**	**3.8752**	0.1033	3.8968	4.8266	6.9839	14.2025	26.8991	56.1909	41.9264	38.4854	260.9196	2500.5682		
STSM_c60_8_4_Wf25_Tf90	3.7526	3.7526	3.7526	3.7526	3.7526	3.7526	3.7526	3.7526	3.7652	3.7897	**3.8685**	0.1144	4.0004	5.2342	9.0758	16.5579	38.5290	96.4141	35.7914	239.3991	961.4871	3188.7477		
STSM_c60_8_5_Wf25_Tf90	3.5997	3.6094	3.6094	3.6094	3.6094	3.6094	3.6094	3.6623	3.7718	**3.8032**	3.7941	0.1232	4.1491	5.5844	9.5241	16.8424	60.5236	287.1675	62.9295	102.9088	5549.5045	4453.8667		
STSM_c60_8_6_Wf25_Tf90	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.4688	3.6723	3.7790	**3.7986**	0.1101	4.4451	5.9422	9.3176	16.6980	35.5423	213.7624	83.6241	159.1652	663.2798	3206.9923		
STSM_c60_8_7_Wf25_Tf90	3.7114	3.7114	3.7114	3.7114	3.7114	3.7114	3.7114	3.6466	3.5670	3.7114	**3.7443**	0.1164	4.6133	5.6425	9.9454	15.0967	36.7025	177.9607	49.2442	131.1813	1117.8503	2862.7628		
STSM_c60_8_8_Wf25_Tf90	3.6004	3.1945	3.5892	3.6535	3.6535	3.6535	3.6535	3.1605	3.5901	**3.7283**	**3.7283**	0.1178	4.7082	5.7314	8.3310	16.1865	53.6368	223.8823	74.1009	115.0879	387.7500	2036.1914		
STSM_c60_8_9_Wf25_Tf90	3.4514	3.3544	3.4847	3.4847	3.4847	3.4847	3.4847	3.0694	3.4052	3.5549	**3.5738**	0.1350	4.6519	6.6524	9.0949	29.6796	64.3446	280.8740	46.8127	242.0769	405.7612	5471.5734		
STSM_c60_8_10_Wf25_Tf90	3.6417	3.3404	3.6417	3.6417	3.6417	3.6417	3.6417	3.3404	3.4688	**3.6417**	**3.6417**	0.1354	4.5454	5.5988	8.3403	15.5081	42.3359	198.0217	34.9472	73.2097	369.0437	2559.2699		
STSM_c60_8_1_Wf25_Tf75	3.1775	3.2958	3.2044	3.2958	3.2958	3.2958	3.2958	3.1589	3.3785	3.3911	**3.4141**	0.1559	4.5643	6.4295	8.9811	19.9606	61.6603	278.5116	48.5438	167.0430	569.2882	4027.0389		
STSM_c60_8_2_Wf25_Tf75	2.8570	3.3652	3.3875	3.3875	3.3875	3.3875	3.3875	3.3652	**3.4040**	3.3467	**3.4040**	0.1779	4.2026	5.5323	8.1894	14.8325	29.0533	69.6919	28.3048	100.2021	233.0675	1489.8695		
STSM_c60_8_3_Wf25_Tf75	3.0509	3.2067	3.2559	3.2559	3.2559	3.2559	3.2559	3.1995	3.3404	**3.3661**	**3.3661**	0.1624	4.6289	5.7173	10.1604	21.5367	45.3234	310.1643	124.1529	404.1429	555.8855	9126.9930		
STSM_c60_8_4_Wf25_Tf75	3.3846	3.3945	3.3846	3.3958	3.3958	3.3958	3.3958	3.2168	3.2733	**3.4013**	3.3958	0.1438	4.6547	5.8701	11.6729	20.6831	49.0279	225.9224	93.1554	239.1346	2976.6316	9096.4595		
STSM_c60_8_5_Wf25_Tf75	3.2940	3.3700	3.3700	3.3700	3.3700	3.3700	3.3700	3.3297	3.2392	3.3752	**3.3940**	0.1389	4.5104	6.5466	9.9706	20.6422	60.1048	234.4747	37.0950	227.4143	709.7707	5723.8579		
STSM_c60_8_6_Wf25_Tf75	3.1957	3.1957	3.2209	3.2209	3.2209	3.2209	3.2209	3.1946	3.1247	3.2684	**3.3062**	0.1598	4.7442	6.4841	10.4842	24.7503	84.8021	293.6710	64.6147	101.4237	609.6594	6224.1714		
STSM_c60_8_7_Wf25_Tf75	3.0415	3.0624	3.1768	**3.1985**	**3.1985**	**3.1985**	**3.1985**	2.9704	3.1768	**3.1985**	**3.1985**	0.1639	4.4341	6.2788	9.5096	22.7311	50.1757	259.0523	25.3499	89.5528	240.9273	5098.2877		
STSM_c60_8_8_Wf25_Tf75	2.8794	2.9951	2.9951	2.9951	2.9951	2.9951	2.9951	2.8679	3.1023	3.0417	**3.1036**	0.1699	4.8698	6.9787	10.6423	24.6122	67.0397	338.4555	71.1735	209.9303	651.8929	6062.6008		
STSM_c60_8_9_Wf25_Tf75	3.0821	3.0365	3.0365	3.0821	3.0821	3.0821	3.0821	2.9832	3.0972	3.1752	**3.2005**	0.1574	4.6310	6.7447	12.2971	23.2291	64.0581	376.0803	48.5877	120.3312	598.5351	4319.1691		
STSM_c60_8_10_Wf25_Tf75	2.6537	2.9700	2.9700	2.9700	2.9700	2.9700	2.9700	2.7427	2.8546	**3.0808**	**3.0808**	0.2104	5.2452	7.3652	12.5547	28.8211	80.9024	299.0979	86.4154	163.1040	946.4163	11318.3486		
STSM_c60_8_1_Wf25_Tf60	2.8176	2.8480	2.8480	2.8480	2.8480	2.8480	2.8480	2.9334	2.8792	2.9868	**2.9875**	0.1651	4.8108	6.3189	8.3898	15.6796	29.7538	64.2492	33.1143	103.8569	261.8309	1634.7490		
STSM_c60_8_2_Wf25_Tf60	2.2488	2.8836	2.8836	2.8836	2.8836	2.8836	2.8836	2.8836	2.8836	**2.9725**	**2.9725**	0.1892	4.8524	5.5318	8.4785	13.6876	25.8520	52.5835	17.6806	32.3701	147.7220	665.8355		
STSM_c60_8_3_Wf25_Tf60	2.6838	2.6838	2.7082	2.7082	2.7082	2.7082	2.7082	2.4965	2.7576	2.8226	**2.8566**	0.1604	4.8657	5.8194	9.2240	14.8513	27.1714	59.5624	29.5206	87.6572	325.1373	2143.9151		
STSM_c60_8_4_Wf25_Tf60	2.6894	2.7835	2.7835	2.8140	2.8140	2.8140	2.8140	2.7537	2.8933	**2.9227**	**2.9227**	0.1662	4.6648	5.9374	8.7172	15.1675	25.8052	55.7624	111.7284	96.4924	359.6765	3835.2491		
STSM_c60_8_5_Wf25_Tf60	2.8006	2.8006	2.7843	2.7843	2.8006	2.8006	2.8006	2.7422	2.8791	**2.9738**	2.9677	0.1440	4.6440	5.9317	9.3655	13.8207	26.5754	68.1286	34.9557	104.3251	465.2014	1379.4820		
STSM_c60_8_6_Wf25_Tf60	2.5238	2.6614	2.7231	2.7231	2.7231	2.7231	2.7231	2.5813	2.7251	2.8292	**2.8437**	0.1647	5.1073	6.3265	8.4686	14.3497	25.2397	54.8248	102.2259	89.8386	153.5815	968.9124		
STSM_c60_8_7_Wf25_Tf60	2.4110	2.4853	2.4491	2.4853	2.4853	2.4853	2.4853	2.4298	2.5578	2.6669	**2.6914**	0.1869	5.3267	6.5883	8.8033	16.1161	37.6671	90.3331	32.7989	68.4255	194.7554	1194.7916		
STSM_c60_8_8_Wf25_Tf60	2.3610	2.4214	2.5111	2.5111	2.5111	2.5111	2.5111	2.3847	2.5787	**2.6293**	**2.6293**	0.1842	4.9680	6.0706	8.6449	15.0399	25.7868	56.5473	49.6963	68.3227	175.6209	1054.3839		
STSM_c60_8_9_Wf25_Tf60	2.1964	2.0017	2.1855	2.1964	2.1964	2.1964	2.1964	2.2078	2.3875	**2.3901**	2.3875	0.1691	4.9952	6.2922	9.4278	13.9326	24.8817	49.5870	21.8400	44.7094	160.8514	654.4509		
STSM_c60_8_10_Wf25_Tf60	2.1178	2.1366	2.1318	2.1527	2.1527	2.1527	2.1527	2.0281	2.2081	2.2502	**2.2523**	0.1963	5.3146	6.3208	9.3621	14.8717	25.2167	47.8552	17.1185	47.9426	189.8527	759.5215		
STSM_c60_8_1_Wf12-5_Tf90	**3.7615**	3.7500	3.7500	**3.7615**	**3.7615**	**3.7615**	**3.7615**	3.3281	3.7500	3.6952	**3.7615**	0.1040	3.9563	4.9225	8.7993	18.5688	54.8438	538.7934	40.1630	26.5288	415.2341	4592.2336		
STSM_c60_8_2_Wf12-5_Tf90	3.3856	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	**3.7271**	3.7081	**3.7271**	**3.7271**	0.1318	4.2028	6.4548	11.0658	28.1497	110.0208	1037.2753	28.8422	150.1300	791.0801	4334.6335		
STSM_c60_8_3_Wf12-5_Tf90	3.7822	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.4688	3.8750	3.8750	**3.8752**	0.0971	3.5378	5.0760	8.0878	17.2512	36.2496	128.7391	23.0065	87.9150	185.2899	1141.4483		
STSM_c60_8_4_Wf12-5_Tf90	3.7526	3.7500	3.7526	3.7526	3.7526	3.7526	3.7526	3.7500	3.6737	3.7442	**3.8138**	0.1152	3.9937	5.9574	9.9091	19.1638	50.2655	336.4036	26.1002	290.5343	1025.1359	3275.3905		
STSM_c60_8_5_Wf12-5_Tf90	3.5061	3.5390	3.6094	3.6094	3.6094	3.6094	3.6094	3.5390	3.7500	3.7500	**3.7567**	0.1249	4.4999	5.8132	11.9400	21.4197	72.4673	769.1475	54.6559	67.8557	1417.8360	5317.1676		
STSM_c60_8_6_Wf12-5_Tf90	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.4421	3.6723	**3.7318**	**3.7318**	0.1062	4.3968	5.2746	12.2487	23.8709	74.8885	729.5053	25.6565	61.3926	817.3733	5875.7588		
STSM_c60_8_7_Wf12-5_Tf90	3.6265	3.4583	3.6309	3.6265	3.6309	3.6309	3.6309	3.1773	3.6106	**3.6999**	**3.6999**	0.1314	4.4428	6.1455	11.6838	20.8904	68.3698	656.3330	17.6250	107.0085	409.3292	2688.8927		
STSM_c60_8_8_Wf12-5_Tf90	3.5862	3.5044	3.5436	3.6016	3.6016	3.6016	3.6016	3.3120	3.5862	**3.6330**	**3.6330**	0.1228	4.3163	6.6641	10.3772	23.3863	98.0254	956.7291	14.7194	250.3772	346.3732	7843.6362		
STSM_c60_8_9_Wf12-5_Tf90	3.3914	3.3287	3.3557	3.3914	3.3914	3.3914	3.3914	3.2917	3.4784	3.5370	**3.5563**	0.1527	5.4617	7.4493	14.8303	37.4131	145.0833	1129.6980	152.9094	174.8461	1345.9369	10790.7560		
STSM_c60_8_10_Wf12-5_Tf90	3.4860	3.3531	3.5177	3.5177	3.5177	3.5177	3.5177	3.3101	3.4736	3.5773	**3.6038**	0.1349	4.8348	6.3029	13.4438	22.1509	108.3703	959.9330	28.9909	85.8991	652.3539	1664.6751		
STSM_c60_8_1_Wf12-5_Tf75	2.9789	3.0521	3.2658	3.2658	3.2658	3.2658	3.2658	2.7122	3.3087	**3.4141**	**3.4141**	0.1688	4.7476	7.0400	11.7453	23.3689	71.2499	581.9934	20.0721	182.5327	517.8883	4804.9619		
STSM_c60_8_2_Wf12-5_Tf75	2.7283	3.3281	3.3875	3.3875	3.3875	3.3875	3.3875	3.3805	3.3875	**3.4222**	3.4146	0.1461	4.3266	6.0447	10.4924	19.1942	36.9817	428.6065	37.2190	63.2814	319.2527	1893.4117		
STSM_c60_8_3_Wf12-5_Tf75	3.0278	2.9816	3.2552	3.2552	3.2552	3.2552	3.2552	3.1048	3.1381	**3.3661**	**3.3661**	0.1480	4.9671	6.3671	10.6689	24.4786	71.6143	752.2795	38.8600	101.2073	588.4222	7958.0495		
STSM_c60_8_4_Wf12-5_Tf75	3.1502	3.3945	**3.3958**	**3.3958**	**3.3958**	**3.3958**	**3.3958**	3.3574	3.3653	**3.3958**	**3.3958**	0.1442	4.7602	7.1971	11.0376	19.2292	55.1553	667.6586	42.2215	317.8649	1180.1973	6034.6153		
STSM_c60_8_5_Wf12-5_Tf75	3.2746	3.1849	3.3404	3.3019	3.3404	3.3404	3.3404	3.0621	3.3404	3.3031	**3.3923**	0.1224	4.7135	5.8769	13.2683	27.2301	104.3487	874.4288	20.6508	115.8654	1048.9828	11314.5908		
STSM_c60_8_6_Wf12-5_Tf75	2.7507	3.0917	3.0969	3.1224	3.1273	3.1273	3.1273	2.7701	3.1573	3.2419	**3.2872**	0.1656	5.2662	7.2626	12.1276	25.5297	102.1824	762.5815	52.1618	120.9413	767.1049	4240.4143		
STSM_c60_8_7_Wf12-5_Tf75	2.8231	3.0624	**3.1773**	**3.1773**	**3.1773**	**3.1773**	**3.1773**	2.9664	3.1096	**3.1773**	**3.1773**	0.1702	4.7757	7.0377	10.8474	28.3924	52.9676	618.6154	24.9892	105.7839	329.2869	4675.8158		
STSM_c60_8_8_Wf12-5_Tf75	2.8222	2.6844	2.8682	2.8849	2.8849	2.8849	2.8849	2.7480	2.9210	**3.0812**	**3.0812**	0.1543	4.9005	6.7596	10.8923	32.0799	85.1768	837.5822	20.6083	87.1496	352.8540	4906.6893		
STSM_c60_8_9_Wf12-5_Tf75	2.8966	3.0365	3.0365	3.0406	3.0406	3.0406	3.0406	3.0365	**3.1469**	**3.1469**	**3.1469**	0.1541	4.6107	6.7199	13.5884	28.5326	93.9260	995.7139	19.4317	69.5771	287.7850	2085.8885		
STSM_c60_8_10_Wf12-5_Tf75	2.5620	2.7354	2.9374	2.9374	2.9374	2.9374	2.9374	2.5495	**3.0035**	2.9763	**3.0035**	0.2049	5.0259	6.9042	13.1637	33.5195	100.2246	838.3208	13.9019	107.4354	451.9738	5407.8218		
STSM_c60_8_1_Wf12-5_Tf60	2.7750	2.7509	2.8176	2.8238	2.8238	2.8238	2.8238	2.9218	2.8426	**2.9875**	2.9825	0.1364	4.9115	6.5530	10.1542	20.2583	41.0107	264.6366	34.5398	90.8302	214.4942	1724.5450		
STSM_c60_8_2_Wf12-5_Tf60	1.8749	2.8836	2.8836	2.8836	2.8836	2.8836	2.8836	2.9055	2.8397	**2.9725**	**2.9725**	0.1927	4.9751	6.7795	10.0193	17.8106	44.6785	246.1209	20.2827	38.9297	98.3588	486.6320		
STSM_c60_8_3_Wf12-5_Tf60	2.4849	2.6725	2.7082	2.7082	2.7082	2.7082	2.7082	2.7514	2.8065	2.8226	**2.8566**	0.1633	5.0570	6.0915	10.6632	18.0639	53.4839	224.0327	34.3140	44.6249	244.1074	760.5341		
STSM_c60_8_4_Wf12-5_Tf60	2.5780	2.7600	2.8140	2.8140	2.8140	2.8140	2.8140	2.5822	2.8404	2.9031	**2.9291**	0.1518	5.2500	7.2289	10.4625	19.3675	46.0957	405.9890	28.2953	85.5509	335.2479	2448.3325		
STSM_c60_8_5_Wf12-5_Tf60	2.7073	2.6321	2.7709	2.7709	2.7709	2.7709	2.7709	2.6869	**2.8693**	**2.8693**	**2.8693**	0.1597	5.6231	7.2911	10.9253	20.6142	54.7886	232.4117	67.2963	138.6443	267.0550	1995.8084		
STSM_c60_8_6_Wf12-5_Tf60	2.3956	2.5702	2.6319	2.6385	2.6385	2.6385	2.6385	2.5014	2.7487	2.8118	**2.8292**	0.1818	5.0186	7.3724	9.9299	19.1178	44.2544	306.2187	16.5422	71.7356	81.8062	527.9710		
STSM_c60_8_7_Wf12-5_Tf60	2.0461	2.3785	2.4072	2.4517	2.4517	2.4517	2.4517	2.5965	2.5607	**2.6323**	**2.6323**	0.1914	5.8367	7.2842	12.0329	23.6500	58.8941	382.9372	19.4411	57.2712	94.8675	1716.7317		
STSM_c60_8_8_Wf12-5_Tf60	2.0954	2.3183	2.4788	2.4788	2.4788	2.4788	2.4788	2.3949	2.3194	2.5093	**2.5374**	0.1632	4.8633	6.2641	9.3935	17.5570	46.0856	227.0039	10.4070	29.2446	79.8245	582.9247		
STSM_c60_8_9_Wf12-5_Tf60	2.0771	2.1931	1.9527	2.1931	2.1931	2.1931	2.1931	**2.3875**	2.1798	**2.3875**	**2.3875**	0.1730	4.8156	6.4007	9.5061	15.9667	34.9632	102.2418	11.5050	13.4315	47.5519	197.7013		
STSM_c60_8_10_Wf12-5_Tf60	1.7699	2.1178	2.1021	2.1178	2.1366	2.1366	2.1366	1.9844	**2.2506**	**2.2506**	**2.2506**	0.2167	5.5228	6.0753	10.3477	15.9671	31.1015	74.4960	12.2588	10.7288	92.3193	332.1204		

Table 4.9: Detailed results for the Linear Scaling Heuristic and the SAA method with Fixed A Priori Route, over the $n = 12$ customer instances with non-overlapping set of possible time slots. Bold expected revenues indicate the best known solution.

	Exact Expected Revenue						CPU time (s)					
		Fixed Route SAA						Fixed Route SAA				
	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	LSH	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$
STSM_c60_12_1_Wf25_Tf90	5.5843	5.6822	**5.8249**	**5.8249**	**5.8249**	**5.8249**	2.1853	54.0131	46.0148	53.2191	70.1212	106.7082
STSM_c60_12_2_Wf25_Tf90	5.4224	5.3321	5.5830	**5.6157**	**5.6157**	**5.6157**	2.3778	66.7636	62.5304	69.7368	112.8079	390.8039
STSM_c60_12_3_Wf25_Tf90	5.6583	5.6422	**5.7727**	**5.7727**	**5.7727**	**5.7727**	2.3212	63.3391	54.9321	62.2141	160.2439	196.5460
STSM_c60_12_4_Wf25_Tf90	5.1805	5.4543	**5.6442**	**5.6442**	**5.6442**	**5.6442**	2.7320	68.5092	60.2900	68.2493	113.2490	524.9806
STSM_c60_12_5_Wf25_Tf90	**5.8490**	5.2299	**5.8490**	**5.8490**	**5.8490**	**5.8490**	2.3378	65.0484	51.2255	57.0308	79.7275	146.3618
STSM_c60_12_6_Wf25_Tf90	**5.6852**	5.5991	**5.6852**	**5.6852**	**5.6852**	**5.6852**	2.4116	54.4269	55.4539	61.6628	80.4407	219.9455
STSM_c60_12_7_Wf25_Tf90	5.5325	5.6277	**5.7689**	**5.7689**	**5.7689**	**5.7689**	2.4971	65.8473	57.1857	59.4171	89.4279	237.7508
STSM_c60_12_8_Wf25_Tf90	**5.4736**	**5.4736**	**5.4736**	**5.4736**	**5.4736**	**5.4736**	3.2367	71.4775	75.2617	88.4082	161.0514	540.4247
STSM_c60_12_9_Wf25_Tf90	**5.5121**	5.3137	5.4468	**5.5121**	**5.5121**	**5.5121**	3.3342	67.3734	64.5633	74.7631	122.7309	561.0369
STSM_c60_12_10_Wf25_Tf90	5.6938	5.6179	**5.7926**	**5.7926**	**5.7926**	**5.7926**	2.3168	61.2263	49.7309	56.3057	70.7395	109.3079
STSM_c60_12_1_Wf25_Tf75	5.1949	4.8283	**5.2049**	**5.2049**	**5.2049**	**5.2049**	4.4816	82.9824	85.9124	106.8819	180.5659	471.3842
STSM_c60_12_2_Wf25_Tf75	4.8804	4.8055	**5.1626**	**5.1626**	**5.1626**	**5.1626**	3.6263	93.3897	86.7213	117.3851	220.9816	851.7633
STSM_c60_12_3_Wf25_Tf75	4.9778	5.1488	5.1817	5.2596	**5.2650**	**5.2650**	4.4646	78.3220	62.8467	82.3130	176.1439	452.6085
STSM_c60_12_4_Wf25_Tf75	5.1404	5.0765	**5.3122**	**5.3122**	**5.3122**	**5.3122**	3.5138	72.2123	72.0937	85.8769	127.5098	500.1635
STSM_c60_12_5_Wf25_Tf75	4.5680	4.5558	4.5680	**4.5785**	4.5558	**4.5785**	5.3836	96.8942	107.9190	143.2834	295.0963	953.2172
STSM_c60_12_6_Wf25_Tf75	4.2664	**4.3203**	4.2771	**4.3203**	**4.3203**	**4.3203**	5.8707	96.3667	117.1630	144.9426	229.2006	699.3830
STSM_c60_12_7_Wf25_Tf75	**4.7278**	4.4259	4.6977	**4.7278**	**4.7278**	**4.7278**	5.6674	98.2860	104.3954	131.8186	229.0817	783.5195
STSM_c60_12_8_Wf25_Tf75	4.3223	4.5276	**4.6622**	**4.6622**	**4.6622**	**4.6622**	5.7252	88.2139	100.4725	129.8239	233.9715	800.7313
STSM_c60_12_9_Wf25_Tf75	4.6654	5.1495	**5.1868**	**5.1868**	**5.1868**	**5.1868**	5.4438	80.9890	66.7838	80.8778	152.4848	384.7232
STSM_c60_12_10_Wf25_Tf75	**5.1418**	**5.1418**	**5.1418**	**5.1418**	**5.1418**	**5.1418**	3.8209	79.7024	83.2738	99.1277	184.3195	592.8886
STSM_c60_12_1_Wf25_Tf60	4.6521	**4.7447**	**4.7447**	**4.7447**	4.7394	**4.7447**	3.9741	82.9817	83.8924	87.4160	115.0189	204.7282
STSM_c60_12_2_Wf25_Tf60	3.9421	3.9459	3.9421	**3.9598**	**3.9598**	**3.9598**	7.3086	100.3619	113.4457	124.0396	185.6677	393.9685
STSM_c60_12_3_Wf25_Tf60	4.6486	4.7520	**4.8303**	**4.8303**	**4.8303**	**4.8303**	4.7805	88.5578	86.7858	91.4642	129.4033	373.7768
STSM_c60_12_4_Wf25_Tf60	4.0876	4.3004	**4.3275**	**4.3275**	**4.3275**	**4.3275**	6.6449	100.8419	93.8195	112.8896	171.1242	402.5906
STSM_c60_12_5_Wf25_Tf60	3.5724	3.7863	3.8017	3.8017	3.8017	**3.8039**	6.7885	112.7140	124.2569	132.9490	168.3619	337.0604
STSM_c60_12_6_Wf25_Tf60	3.1733	3.2514	3.2592	3.3049	**3.3548**	**3.3548**	5.5494	106.1913	104.2180	122.8342	136.2410	177.9632
STSM_c60_12_7_Wf25_Tf60	3.6965	3.5156	3.6965	**3.8633**	**3.8633**	**3.8633**	8.3091	124.0270	135.1714	146.9474	192.9509	415.9389
STSM_c60_12_8_Wf25_Tf60	3.5972	3.4636	**3.6759**	**3.6759**	**3.6759**	**3.6759**	5.5628	111.3648	114.9324	130.9928	169.6236	313.4555
STSM_c60_12_9_Wf25_Tf60	3.6877	**3.9640**	**3.9640**	3.9596	**3.9640**	**3.9640**	5.0215	106.3236	123.7089	123.7566	165.0307	343.3478
STSM_c60_12_10_Wf25_Tf60	3.9391	3.8737	3.9704	**3.9812**	**3.9812**	**3.9812**	6.7810	98.1170	109.5882	127.0347	182.0349	414.1833
STSM_c60_12_1_Wf12-5_Tf90	5.5066	5.2448	5.7500	5.7655	**5.7776**	**5.7776**	2.3850	65.8500	44.1981	54.3693	134.9812	385.2594
STSM_c60_12_2_Wf12-5_Tf90	5.2433	5.2524	**5.5833**	**5.5833**	**5.5833**	**5.5833**	2.6582	69.4539	57.8022	101.4496	414.6294	2069.4738
STSM_c60_12_3_Wf12-5_Tf90	5.6390	5.0976	**5.7286**	**5.7286**	**5.7286**	**5.7286**	2.4031	68.5153	55.7336	63.3229	225.7595	872.2101
STSM_c60_12_4_Wf12-5_Tf90	5.1462	5.4619	5.5669	**5.6117**	**5.6117**	**5.6117**	2.8610	57.5870	57.6327	70.9460	216.9346	1904.9969
STSM_c60_12_5_Wf12-5_Tf90	**5.7490**	5.7168	**5.7490**	**5.7490**	**5.7490**	**5.7490**	2.4456	62.5716	60.5600	70.8758	127.3747	871.9879
STSM_c60_12_6_Wf12-5_Tf90	5.5047	**5.5559**	**5.5559**	**5.5559**	**5.5559**	**5.5559**	2.8432	65.0964	60.5588	66.9923	190.8727	867.3575
STSM_c60_12_7_Wf12-5_Tf90	5.5279	5.4533	**5.7061**	**5.7061**	**5.7061**	**5.7061**	2.4781	72.9719	60.3661	83.5022	179.9716	1566.4406
STSM_c60_12_8_Wf12-5_Tf90	**5.4736**	4.9246	**5.4736**	**5.4736**	**5.4736**	**5.4736**	3.3094	78.3622	88.0674	124.4634	327.7584	1679.8830
STSM_c60_12_9_Wf12-5_Tf90	5.0745	5.2109	5.3197	5.3543	**5.3596**	**5.3596**	4.5551	74.9225	82.1746	110.3169	247.6428	2204.1870
STSM_c60_12_10_Wf12-5_Tf90	5.5781	5.2649	**5.7604**	**5.7604**	**5.7604**	**5.7604**	2.2301	58.4285	52.5357	71.4133	113.1881	566.0928
STSM_c60_12_1_Wf12-5_Tf75	4.9443	4.8513	**5.2279**	**5.2279**	**5.2279**	**5.2279**	4.2709	77.2775	78.0905	99.8830	165.3338	1685.1748
STSM_c60_12_2_Wf12-5_Tf75	4.7780	4.7968	5.0964	**5.1197**	**5.1197**	**5.1197**	3.5259	81.2134	91.3775	129.2035	324.5923	3708.0222
STSM_c60_12_3_Wf12-5_Tf75	4.9258	5.0872	**5.2435**	**5.2435**	**5.2435**	**5.2435**	4.0064	78.8456	73.1175	91.9901	182.2727	1008.1529
STSM_c60_12_4_Wf12-5_Tf75	4.9006	5.1293	4.9375	**5.1416**	5.1293	**5.1416**	4.1193	80.7601	86.6242	111.1064	257.6471	2221.3376
STSM_c60_12_5_Wf12-5_Tf75	4.3132	4.1362	4.3226	4.4760	**4.5618**	**4.5618**	5.8712	87.9523	104.4498	154.9304	382.9253	3427.8149
STSM_c60_12_6_Wf12-5_Tf75	4.2771	4.0287	**4.3481**	**4.3481**	**4.3481**	**4.3481**	5.1084	95.3078	105.7635	136.5887	230.4836	1297.6855
STSM_c60_12_7_Wf12-5_Tf75	4.6342	4.5628	4.5810	**4.7403**	**4.7403**	**4.7403**	5.7335	105.9455	112.5144	142.3870	398.4774	3768.3559
STSM_c60_12_8_Wf12-5_Tf75	4.0351	4.5276	4.5579	**4.6790**	**4.6790**	**4.6790**	6.2053	92.3490	91.8528	133.5928	254.6292	1934.4967
STSM_c60_12_9_Wf12-5_Tf75	4.3364	4.9625	**5.1997**	5.1876	**5.1997**	**5.1997**	4.9446	84.4399	62.1639	90.6096	129.2090	814.3490
STSM_c60_12_10_Wf12-5_Tf75	4.7296	4.7046	4.9183	**5.0401**	**5.0401**	**5.0401**	3.9727	80.7794	84.3630	104.8995	191.2982	2180.1435
STSM_c60_12_1_Wf12-5_Tf60	4.0790	**4.6866**	4.6136	**4.6866**	**4.6866**	**4.6866**	4.9431	81.7832	74.1336	90.1975	120.4882	475.6193
STSM_c60_12_2_Wf12-5_Tf60	3.7493	3.8748	3.9192	3.9192	**3.9325**	**3.9325**	6.3762	95.5789	103.9744	144.2867	248.6884	959.7797
STSM_c60_12_3_Wf12-5_Tf60	4.3761	**4.7754**	4.7607	**4.7754**	**4.7754**	**4.7754**	4.0898	83.4859	87.3460	101.1399	196.4607	2605.6792
STSM_c60_12_4_Wf12-5_Tf60	3.9297	4.0539	**4.2919**	**4.2919**	**4.2919**	**4.2919**	6.1308	94.6704	100.5323	137.3664	252.3394	1212.5255
STSM_c60_12_5_Wf12-5_Tf60	3.3277	3.5935	3.5458	**3.7208**	**3.7208**	**3.7208**	6.0105	106.0885	116.7447	133.8756	227.0442	1029.3909
STSM_c60_12_6_Wf12-5_Tf60	2.8374	3.0901	3.3548	3.3548	**3.3653**	**3.3653**	5.6934	98.6656	98.4402	112.0250	140.7067	316.6168
STSM_c60_12_7_Wf12-5_Tf60	3.4996	3.3783	**3.8011**	3.7021	**3.8011**	**3.8011**	7.4473	125.8111	132.1124	145.0633	226.8241	946.1685
STSM_c60_12_8_Wf12-5_Tf60	3.5021	3.4548	3.6872	3.6872	**3.6945**	**3.6945**	5.1737	106.0972	107.5046	119.5732	187.0179	686.9838
STSM_c60_12_9_Wf12-5_Tf60	3.5624	3.8829	**3.9481**	**3.9481**	**3.9481**	**3.9481**	5.4866	106.0334	115.9209	149.3930	237.4099	657.4959
STSM_c60_12_10_Wf12-5_Tf60	3.8185	3.9296	3.8856	**3.9495**	3.9393	3.9375	5.9062	109.8266	109.0889	143.4043	210.2703	1049.8073

Table 4.10: Detailed results for the instances with overlapping sets of possible time slots. Bold expected revenues indicate the best known solution.

	Exact Exp. Rev.			CPU time (s)		
	LSH	SAA Fixed	SAA Full	LSH	SAA Fixed	SAA Full
STSM_c60_8_1_Wf12-5o2_Tf90	3.8091	3.8091	**3.8482**	0.1122	496.7008	4803.4290
STSM_c60_8_2_Wf12-5o2_Tf90	3.5833	3.5922	**3.7271**	0.1041	1138.4635	7770.2280
STSM_c60_8_3_Wf12-5o2_Tf90	3.8752	3.8752	**3.8933**	0.0996	176.8945	1928.3496
STSM_c60_8_4_Wf12-5o2_Tf90	3.7526	3.8027	**3.8368**	0.1240	446.3710	6349.6148
STSM_c60_8_5_Wf12-5o2_Tf90	3.7394	3.7394	**3.7757**	0.1065	611.0241	8574.7235
STSM_c60_8_6_Wf12-5o2_Tf90	3.7723	3.7723	**3.7785**	0.1182	1026.2595	10678.3739
STSM_c60_8_7_Wf12-5o2_Tf90	3.7714	**3.7775**	**3.7775**	0.1125	791.9994	2793.7330
STSM_c60_8_8_Wf12-5o2_Tf90	3.7278	**3.7351**	**3.7351**	0.1239	730.1920	19331.7780
STSM_c60_8_9_Wf12-5o2_Tf90	3.5243	3.5243	**3.5984**	0.1432	1705.1862	41888.1546
STSM_c60_8_10_Wf12-5o2_Tf90	**3.6705**	**3.6705**	**3.6705**	0.1364	940.9034	4078.8893
STSM_c60_8_1_Wf12-5o2_Tf75	3.3292	3.3292	**3.4141**	0.1411	1830.8676	30803.0189
STSM_c60_8_2_Wf12-5o2_Tf75	2.9868	3.3875	**3.4222**	0.1364	772.5951	4302.2054
STSM_c60_8_3_Wf12-5o2_Tf75	3.1212	3.2828	**3.3570**	0.1581	1319.1536	82933.8089
STSM_c60_8_4_Wf12-5o2_Tf75	3.1619	**3.3958**	3.3812	0.1524	1126.0975	39800.3692
STSM_c60_8_5_Wf12-5o2_Tf75	3.2183	3.3855	**3.4278**	0.1347	1234.1769	20971.7485
STSM_c60_8_6_Wf12-5o2_Tf75	3.0200	3.2366	**3.3497**	0.1668	908.8826	23894.2788
STSM_c60_8_7_Wf12-5o2_Tf75	2.9472	**3.1960**	**3.1960**	0.1640	1827.0931	25937.2847
STSM_c60_8_8_Wf12-5o2_Tf75	2.7794	2.9490	**3.0812**	0.1763	2390.5861	54197.6124
STSM_c60_8_9_Wf12-5o2_Tf75	2.6080	3.1074	**3.2148**	0.1522	1822.7014	10039.0893
STSM_c60_8_10_Wf12-5o2_Tf75	2.9915	3.0929	**3.1054**	0.1781	1257.8017	19733.4258
STSM_c60_8_1_Wf12-5o2_Tf60	2.7330	2.8238	**2.9825**	0.1633	541.1096	9784.6687
STSM_c60_8_2_Wf12-5o2_Tf60	2.1463	2.8836	**2.9725**	0.1561	369.8185	2268.8561
STSM_c60_8_3_Wf12-5o2_Tf60	2.4849	2.8153	**2.8566**	0.1768	489.8252	4643.4262
STSM_c60_8_4_Wf12-5o2_Tf60	2.5780	2.8140	**2.9355**	0.1569	795.0578	17883.8778
STSM_c60_8_5_Wf12-5o2_Tf60	2.7772	2.8611	**2.8769**	0.1621	1011.1436	7311.9611
STSM_c60_8_6_Wf12-5o2_Tf60	2.3956	2.7231	**2.8437**	0.1881	478.5715	2665.4961
STSM_c60_8_7_Wf12-5o2_Tf60	2.1989	2.4517	**2.7145**	0.1673	926.8967	6718.1597
STSM_c60_8_8_Wf12-5o2_Tf60	2.0286	2.5787	**2.6270**	0.1786	414.6327	2377.4768
STSM_c60_8_9_Wf12-5o2_Tf60	2.1560	2.1931	**2.3875**	0.1704	169.9912	1202.7767
STSM_c60_8_10_Wf12-5o2_Tf60	2.0510	2.1366	**2.2506**	0.2248	134.6988	925.7496
STSM_c60_12_1_Wf12-5o2_Tf90	5.7974	**5.8249**		2.3256	378.4126	
STSM_c60_12_2_Wf12-5o2_Tf90	**5.7550**	**5.7550**		2.5474	1962.8252	
STSM_c60_12_3_Wf12-5o2_Tf90	**5.8939**	**5.8939**		2.0300	287.9451	
STSM_c60_12_4_Wf12-5o2_Tf90	5.7387	**5.7933**		2.3546	973.0745	
STSM_c60_12_5_Wf12-5o2_Tf90	5.8454	**5.8663**		2.3382	475.6658	
STSM_c60_12_6_Wf12-5o2_Tf90	**5.6214**	**5.6214**		2.5086	2396.2897	
STSM_c60_12_7_Wf12-5o2_Tf90	5.8165	**5.8278**		2.2851	1021.1384	
STSM_c60_12_8_Wf12-5o2_Tf90	5.3339	**5.4822**		4.0867	7250.0155	
STSM_c60_12_9_Wf12-5o2_Tf90	5.5364	**5.6307**		2.9939	3186.1611	
STSM_c60_12_10_Wf12-5o2_Tf90	5.8258	**5.8744**		2.0144	349.8826	
STSM_c60_12_1_Wf12-5o2_Tf75	5.2630	**5.3346**		4.1298	5402.3894	
STSM_c60_12_2_Wf12-5o2_Tf75	5.0978	**5.1974**		3.3740	14447.6435	
STSM_c60_12_3_Wf12-5o2_Tf75	5.0251	**5.2953**		3.4756	5928.6411	
STSM_c60_12_4_Wf12-5o2_Tf75	4.9754	**5.2042**		3.4820	7656.0720	
STSM_c60_12_5_Wf12-5o2_Tf75	4.4311	**4.6868**		7.4584	63771.7534	
STSM_c60_12_6_Wf12-5o2_Tf75	4.1540	**4.3882**		5.3126	9362.9319	
STSM_c60_12_7_Wf12-5o2_Tf75	4.7184	**4.9418**		5.8420	30246.9255	
STSM_c60_12_8_Wf12-5o2_Tf75	4.1531	**4.7199**		6.0980	10533.0046	
STSM_c60_12_9_Wf12-5o2_Tf75	4.4392	**5.2466**		5.2953	2497.1227	
STSM_c60_12_10_Wf12-5o2_Tf75	4.9871	**5.0715**		3.6438	9703.8714	
STSM_c60_12_1_Wf12-5o2_Tf60	4.3852	**4.7251**		4.0970	2209.5098	
STSM_c60_12_2_Wf12-5o2_Tf60	**3.9438**	3.9354		5.6232	10980.0181	
STSM_c60_12_3_Wf12-5o2_Tf60	4.0510	**4.8840**		5.4026	7141.3359	
STSM_c60_12_4_Wf12-5o2_Tf60	4.0184	**4.3244**		6.3066	12546.3243	
STSM_c60_12_5_Wf12-5o2_Tf60	3.5936	**3.8084**		5.3802	6986.7271	
STSM_c60_12_6_Wf12-5o2_Tf60	3.1447	**3.3653**		4.9358	603.6254	
STSM_c60_12_7_Wf12-5o2_Tf60	3.3955	**3.8703**		7.2837	7756.2249	
STSM_c60_12_8_Wf12-5o2_Tf60	3.5997	**3.7108**		5.0177	4120.8814	
STSM_c60_12_9_Wf12-5o2_Tf60	3.6352	**4.0073**		4.7563	6936.4447	
STSM_c60_12_10_Wf12-5o2_Tf60	3.8829	**3.9784**		6.5914	8440.9017	

Table 4.11: Detailed results of the fixed a priori route SAA method without non-anticipation and ascending time slot constrained models.

	Exact Expected Revenue												CPU time (s)											
	Fixed Route SAA without Non-Anticip.						Fixed Route SAA with Asc. Time Slots						Fixed Route SAA without Non-Anticip.						Fixed Route SAA with Asc. Time Slots					
	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
STSM_c60_8_1_Wf12-5_Tf90	3.7271	3.7615	3.7615	3.7615	3.7615	3.7615	3.7500	3.7615	3.7615	3.7615	3.7615	3.7615	2.4183	2.5801	3.9271	7.4264	15.8959	45.0299	3.3011	3.9169	7.3390	14.6990	31.7612	376.2952
STSM_c60_8_2_Wf12-5_Tf90	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	3.4688	2.6040	3.1202	5.2243	9.0929	24.9866	83.2923	3.4324	5.3698	8.6577	20.5252	63.1485	471.0335
STSM_c60_8_3_Wf12-5_Tf90	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	3.8750	2.2741	2.5159	3.6938	6.2540	14.3819	28.8374	2.9344	4.1879	6.5308	13.3571	29.1872	112.8012
STSM_c60_8_4_Wf12-5_Tf90	3.7500	3.7526	3.7526	3.7526	3.7526	3.7526	3.7500	3.7526	3.7526	3.7526	3.7526	3.7526	2.2837	2.9607	4.4024	7.2868	13.3788	36.1131	3.3101	4.7490	8.2128	14.6286	36.3415	217.0630
STSM_c60_8_5_Wf12-5_Tf90	3.6094	3.6094	3.6094	3.6094	3.6094	3.6094	3.5535	3.6094	3.6094	3.6094	3.6094	3.6094	2.5978	2.9894	4.7340	8.1718	19.9849	64.8780	3.7408	4.7346	9.2928	16.2516	51.1763	403.5600
STSM_c60_8_6_Wf12-5_Tf90	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	3.6723	2.7666	2.7376	4.6795	8.0213	24.8834	98.6641	3.2979	4.1457	9.7408	17.0279	46.8285	464.5423
STSM_c60_8_7_Wf12-5_Tf90	3.4583	3.6309	3.6309	3.6309	3.6309	3.6309	3.6265	3.6309	3.6309	3.6309	3.6309	3.6309	2.9208	3.0603	4.9243	8.2980	19.8742	82.5935	3.4216	5.1894	8.5727	16.0881	52.7549	404.9751
STSM_c60_8_8_Wf12-5_Tf90	3.5436	3.6016	3.6016	3.6016	3.6016	3.6016	3.5436	3.5862	3.6016	3.6016	3.6016	3.6016	2.6729	3.5160	4.6316	10.1374	26.7600	107.8462	3.4643	5.8256	7.9409	18.7856	62.3044	575.8033
STSM_c60_8_9_Wf12-5_Tf90	3.3836	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.3914	3.1187	3.9878	7.0661	13.6640	58.2805	350.1663	4.5906	6.0472	11.4334	25.2074	87.4793	601.4894
STSM_c60_8_10_Wf12-5_Tf90	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	3.5177	2.9366	3.4231	5.8410	8.6144	23.7610	100.1150	3.8700	5.1000	10.1624	16.0353	68.0360	558.9897
STSM_c60_8_1_Wf12-5_Tf75	2.9796	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.2658	3.0558	3.5040	6.4973	11.0704	31.5784	107.8857	3.6149	5.6549	8.9807	16.7690	55.4372	447.9929
STSM_c60_8_2_Wf12-5_Tf75	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	3.3875	2.6311	2.9497	4.5604	8.0277	14.2662	30.0408	3.4786	4.7057	6.9731	11.7955	22.4847	62.3979
STSM_c60_8_3_Wf12-5_Tf75	3.2301	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.2552	3.1525	3.4958	6.0082	12.4979	30.9285	135.6080	3.7003	4.8488	8.7547	17.5858	62.2315	460.8348
STSM_c60_8_4_Wf12-5_Tf75	3.3945	3.3945	3.3945	3.3958	3.3945	3.3945	3.3945	3.3958	3.3958	3.3958	3.3958	3.3958	3.1897	3.9581	5.6595	9.2092	20.3343	75.7201	3.6125	5.5472	8.3418	14.2431	35.5176	297.8309
STSM_c60_8_5_Wf12-5_Tf75	3.2737	3.3157	3.3404	3.3404	3.3404	3.3404	3.2746	3.3404	3.3404	3.3404	3.3404	3.3404	3.0474	3.4143	6.0717	12.7319	31.6774	79.7117	3.6609	4.7409	9.6385	19.1618	64.2913	388.2181
STSM_c60_8_6_Wf12-5_Tf75	3.0917	3.0969	3.1224	3.1273	3.1273	3.1273	3.1224	3.1273	3.1224	3.1273	3.1273	3.1273	3.2461	4.0869	6.5142	13.8971	41.0246	266.6782	4.3119	6.2680	9.4924	18.4323	60.9058	448.0205
STSM_c60_8_7_Wf12-5_Tf75	3.1773	3.0624	3.1773	3.1773	3.1773	3.1773	3.1773	3.1773	3.1773	3.1773	3.1773	3.1773	3.1575	3.9364	5.4151	14.3135	25.4096	78.8207	3.8969	5.8379	7.4699	16.2835	33.7906	340.6008
STSM_c60_8_8_Wf12-5_Tf75	2.8682	2.8682	2.8682	2.8682	2.8682	2.8682	2.8682	2.8849	2.8849	2.8849	2.8849	2.8849	3.1596	3.8662	6.5449	16.3215	31.2751	126.8162	4.0825	5.2869	8.6509	19.8237	50.0239	506.4594
STSM_c60_8_9_Wf12-5_Tf75	3.0365	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	3.0406	2.9899	4.0061	6.4390	15.1562	36.4679	251.5266	4.0041	5.0414	8.4999	14.7995	54.2874	381.5282
STSM_c60_8_10_Wf12-5_Tf75	2.8253	2.8646	2.9374	2.9374	2.9374	2.9374	2.8646	2.9374	2.9374	2.9374	2.9374	2.9374	3.3371	4.3536	9.3516	29.0292	69.2299	341.3582	3.9499	5.6559	9.5786	17.7568	56.0491	510.8178
STSM_c60_8_1_Wf12-5_Tf60	2.7569	2.8193	2.8193	2.8193	2.8238	2.8238	2.7750	2.8238	2.8238	2.8238	2.8238	2.8238	3.2557	3.9000	5.1441	9.0675	18.9008	45.5700	3.7897	4.9560	7.6189	13.9170	27.5799	171.9146
STSM_c60_8_2_Wf12-5_Tf60	2.8836	2.8836	2.8836	2.8836	2.8836	2.8836	2.7885	2.7885	2.7885	2.7885	2.7885	2.7885	3.2223	3.8018	5.5926	9.6311	18.3875	43.3384	4.0331	5.1990	7.3350	11.4290	22.2420	46.7502
STSM_c60_8_3_Wf12-5_Tf60	2.6838	2.6838	2.6838	2.6838	2.7082	2.7082	2.6499	2.6499	2.6499	2.6499	2.6499	2.6499	3.2902	3.6742	5.7314	8.9747	23.4747	54.1036	4.1055	4.8997	7.7262	13.2604	35.4110	212.8174
STSM_c60_8_4_Wf12-5_Tf60	2.8140	2.8140	2.8140	2.8140	2.8140	2.8140	2.6835	2.7111	2.7111	2.7111	2.7111	2.7111	3.4378	4.4670	6.1328	10.5467	22.0836	58.0159	4.2688	5.6767	8.4842	13.7978	30.6732	117.3729
STSM_c60_8_5_Wf12-5_Tf60	2.7709	2.7709	2.7351	2.7709	2.7351	2.7709	2.7543	2.7709	2.7709	2.7709	2.7709	2.7709	3.8753	4.5008	7.1720	11.7259	28.5874	88.0497	4.2550	5.1070	7.1264	11.0616	21.5481	45.4145
STSM_c60_8_6_Wf12-5_Tf60	2.6385	2.6385	2.6385	2.6385	2.6385	2.6385	2.5743	2.5798	2.5798	2.5798	2.5798	2.5798	3.5764	4.3349	5.8300	10.9631	23.5014	97.9710	4.3001	5.7772	7.3826	12.0223	26.1502	150.4211
STSM_c60_8_7_Wf12-5_Tf60	2.4517	2.3650	2.4517	2.4517	2.4517	2.4517	2.3785	2.3785	2.3785	2.3785	2.3785	2.3785	3.4163	5.3658	8.3058	16.5302	34.7903	123.0234	4.5202	5.7700	8.3511	13.9507	29.0170	111.6655
STSM_c60_8_8_Wf12-5_Tf60	2.4467	2.4788	2.4788	2.4788	2.4788	2.4788	2.3983	2.3983	2.3983	2.3983	2.3983	2.3983	3.1733	3.7920	5.5009	11.9789	28.6332	146.0873	4.0354	4.9971	7.3054	12.8555	28.1082	123.4183
STSM_c60_8_9_Wf12-5_Tf60	2.1931	2.1750	2.1931	2.1931	2.1931	2.1931	2.1931	2.1931	2.1931	2.1931	2.1931	2.1931	3.4964	3.6102	4.9818	8.1620	16.4449	36.8356	4.1642	5.0193	7.1409	11.0895	21.2128	46.7235
STSM_c60_8_10_Wf12-5_Tf60	2.0822	2.1178	2.1178	2.1366	2.1366	2.1366	2.1366	2.1366	2.1366	2.1366	2.1366	2.1366	3.9116	4.0517	6.0174	8.1891	15.3016	36.3302	4.5699	5.0887	8.0286	11.8658	20.9211	45.3334
STSM_c60_12_1_Wf12-5_Tf90	5.2464	5.7500	5.7482	5.7655	5.7776		5.5363	5.7776	5.7500	5.7776	5.7776		83.7416	50.8089	61.1498	79.3689	109.8826		52.2678	44.2673	53.1667	126.0778	333.7008	
STSM_c60_12_2_Wf12-5_Tf90	5.2433	5.5756	5.5833	5.5833	5.5833		5.5756	5.5833	5.5736	5.5833	5.5833		87.1618	70.4414	74.6887	137.7360	481.0526		59.0494	60.7431	87.4320	339.3201	1374.2260	
STSM_c60_12_3_Wf12-5_Tf90	5.0976	5.7286	5.7286	5.7286	5.7286		5.6623	5.7286	5.7286	5.7286	5.7286		89.9179	64.8665	70.2441	114.4821	186.8967		54.6424	53.5981	59.7212	160.6991	808.7656	
STSM_c60_12_4_Wf12-5_Tf90	5.4619	5.6117	5.6117	5.6117	5.6117		5.5240	5.6117	5.6117	5.6117	5.6037		75.0044	78.8100	73.2261	98.7951	217.0527		57.8245	60.8042	71.1250	160.4782	1294.6037	
STSM_c60_12_5_Wf12-5_Tf90	5.6067	5.7490	5.7490	5.7490	5.7490		5.7490	5.7490	5.7490	5.7490	5.7490		85.0054	73.5252	72.6361	87.3116	234.9051		59.3452	61.5641	70.1823	112.6171	622.5531	
STSM_c60_12_6_Wf12-5_Tf90	5.5559	5.5559	5.5559	5.5559	5.5559		5.5559	5.5559	5.5559	5.5559	5.5559		81.3550	72.1905	74.7952	90.9266	154.3548		62.0900	59.1844	66.7256	163.5777	724.7494	
STSM_c60_12_7_Wf12-5_Tf90	5.4532	5.7061	5.7061	5.7061	5.7061		5.7061	5.6414	5.7061	5.7061	5.7061		91.6315	74.4922	78.8027	93.9101	270.6743		65.2318	54.7727	95.0020	148.2436	1159.7824	
STSM_c60_12_8_Wf12-5_Tf90	4.7170	5.4736	5.4736	5.4736	5.4736		5.4736	5.4736	5.4736	5.4736	5.4736		96.3188	94.7179	105.2276	122.9318	436.6353		71.4073	83.8464	95.4203	261.2101	1199.8132	
STSM_c60_12_9_Wf12-5_Tf90	5.2454	5.3596	5.3596	5.3596	5.3596		5.3121	5.3543	5.3596	5.3596	5.3596		76.7321	97.2040	101.7171	138.6826	270.1679		63.8044	77.2254	120.1937	217.1184	1719.5768	
STSM_c60_12_10_Wf12-5_Tf90	5.2732	5.7604	5.7604	5.7604	5.7604		5.7504	5.7604	5.7604	5.7604	5.7604		81.6896	68.2787	72.0228	81.5619	170.6479		53.7201	53.3864	71.4326	126.3429	526.0402	
STSM_c60_12_1_Wf12-5_Tf75	4.8513	5.2279	5.2279	5.2279	5.2279		5.2279	5.2279	5.2279	5.2279	5.2279		98.3162	98.8826	109.2436	134.7372	365.6516		78.0342	83.8173	100.5930	133.0182	1041.1875	
STSM_c60_12_2_Wf12-5_Tf75	4.9806	4.9763	5.1197	5.1197	5.1197		5.0272	5.1197	5.1197	5.1197	5.1197		112.8603	109.1091	131.4278	853.9698	1710.3347		80.6986	85.4832	105.6722	182.2824	1419.9356	
STSM_c60_12_3_Wf12-5_Tf75	5.0365	5.1533	5.2435	5.2435	5.2435		5.0386	5.1418	5.2122	5.2112	5.2122		102.8032	88.1420	82.5830	113.3189	192.1517		79.3797	72.9448	91.8609	145.6492	846.5322	
STSM_c60_12_4_Wf12-5_Tf75	5.0490	4.9538	5.1416	5.1416	5.1416		5.1416	5.1416	5.1416	5.1416	5.1416		103.6698	99.9329	115.8350	319.3993	1296.6096		82.1390	84.3757	105.2749	197.2661	1063.4402	
STSM_c60_12_5_Wf12-5_Tf75	4.2956	4.5317	4.5618	4.5618	4.5618		4.5618	4.4836	4.5618	4.5618	4.5618		124.1180	121.9281	171.3355	735.2106	3637.0891		95.6363	114.5503	145.2865	218.8141	1717.4212	
STSM_c60_12_6_Wf12-5_Tf75	4.2632	4.3481	4.3481	4.3481	4.3481		4.3481	4.3481	4.3481	4.3481	4.3481		119.0631	127.8721	166.9888	450.8146	1770.3602		97.8766	102.5467	125.3987	195.9718	901.4580	
STSM_c60_12_7_Wf12-5_Tf75	4.6020	4.6710	4.7155	4.7403	4.7403		4.7048	4.7403	4.7403	4.7403	4.7403		149.2846	146.1491	186.3191	1411.9680	4833.8691		105.8977	106.3960	126.0700	222.7280	2071.7740	
STSM_c60_12_8_Wf12-5_Tf75	4.4945	4.6790	4.6790	4.6790	4.6790		4.6790	4.6790	4.6790	4.6790	4.6790		105.6408	114.5681	147.5254	434.4093	1342.2964		94.6567	97.4407	121.5170	186.8270	1113.3455	
STSM_c60_12_9_Wf12-5_Tf75	5.1997	5.1997	5.1997	5.1997	5.1997		5.1997	5.1997	5.1997	5.1997	5.1997		81.5818	75.1289	82.4635	97.3169	187.5717		71.2577	61.0995	85.0182	124.2602	701.4577	
STSM_c60_12_10_Wf12-5_Tf75	4.8898	5.0401	5.0401	5.0401	5.0401		5.0355	5.0401	5.0401	5.0401	5.0401		102.0338	104.7939	115.7785	173.3158	1027.9595		83.1579	84.9354	100.3256	164.4683	1228.4116	
STSM_c60_12_1_Wf12-5_Tf60	4.6866	4.6866	4.6866	4.6866	4.6866		4.6866	4.6866	4.6866	4.6866	4.6866		109.4825	92.9320	100.4600	121.2256	206.6856		78.0694	74.0172	84.3795	95.5767	328.2925	
STSM_c60_12_2_Wf12-5_Tf60	3.8828	3.8962	3.9192	3.9192	3.9271		3.8610	3.8928	3.8928	3.8928	3.8928		116.9711	135.3956	180.1247	431.2156	2385.3898		116.6058	122.7827	134.8512	158.9862	388.5506	
STSM_c60_12_3_Wf12-5_Tf60	4.7754	4.7754	4.7754	4.7754	4.7754		4.6140	4.5733	4.6140	4.6140	4.6140		101.6079	104.6101	118.9192	212.8830	1097.4690		94.8699	99.7918	107.5646	144.0709	818.6896	
STSM_c60_12_4_Wf12-5_Tf60	4.0332	4.2919	4.2919	4.2919	4.2919		4.1793	4.2919	4.2919	4.2919	4.2919		117.0939	114.5617	150.8606	427.9423	1535.0184		91.6307	89.6395	113.2007	151.3897	437.6976	
STSM_c60_12_5_Wf12-5_Tf60	3.6824	3.7208	3.7208	3.7208	3.7208		3.6824	3.5831	3.6824	3.6824	3.6824		129.5768	134.5546	188.1788	539.3760	6775.7358		113.6634	116.5301	124.3152	139.7335	374.2221	
STSM_c60_12_6_Wf12-5_Tf60	3.0322	3.2697	3.3548	3.3548	3.3548		3.2697	3.2697	3.2727	3.2727	3.2727		134.0134	128.2238	137.4901	172.6271	353.4880		106.5911	112.0920	124.2948	142.5646	239.6341	
STSM_c60_12_7_Wf12-5_Tf60	3.5672	3.6668	3.6600	3.8011	3.8011		3.6712	3.6595	3.6595	3.6712	3.6712		135.8225	147.0092	183.0306	701.9748	4389.4471		127.3216	130.3706	142.0604	177.5961	402.6723	
STSM_c60_12_8_Wf12-5_Tf60	3.4862	3.5914	3.6872	3.6872	3.6872		3.5756	3.5803	3.5803	3.5756	3.5803		137.0103	141.1547	183.9404	448.9265	1488.3729		109.5195	109.5975	114.5667	147.2923	367.7725	
STSM_c60_12_9_Wf12-5_Tf60	3.9366	3.9380	3.9481	3.9481	3.9481		3.8829	3.9380	3.9481	3.9481	3.9481		141.5846	156.1215	214.8520	714.2293	1314.4637		131.4733	116.5352	136.7639	157.3700	329.9344	
STSM_c60_12_10_Wf12-5_Tf60	3.8139	3.9176	3.9495	3.9495	3.9495		3.9296	3.9296	3.9393	3.9393	3.9393		137.4779	151.0314	188.0568	382.2591	1864.6716		114.4817	117.8070	130.0936	156.9745	511.5410	

Chapter 5

Summary and conclusions

In this dissertation, we investigate the use of vehicle routing and time slot management in online retailing. By offering time slots, online retailers can decrease the chances of delivery failures and provide their customers with a high level of service. However, the management of these time slots can be challenging, as the customer demand can vary heavily, and the amount of available delivery vehicles and drivers may be limited. Dynamic Time Slot Management (DTSM) is a class of methods which dynamically construct time slot offers based on previously placed customer orders. Our focus is the use of vehicle routing heuristics within DTSM to help retailers manage the availability of time slots in real time. In the following, we summarize the main findings of each chapter.

Chapter 2 investigates the efficient evaluation of moves in neighborhood search heuristics for a class of time-dependent vehicle routing problems. While time-dependent travel times and route duration constraints are useful in modeling road congestion and driver labour agreements, their combination increases the computational complexity of move evaluations. We investigate the use of piecewise linear functions, and observe that these can be evaluated in various orders when computing the route duration. This is used to develop a new tree based data structure to improve the complexity of computations and memory usage. The presented methods have the best known computational complexity, while they do not require a lexicographic order of search.

Our numerical experiments illustrate the trade-off between computation time and memory use. On 1000 customer instances, our methods are able to speed up a construction heuristic by up to 8.89 times and an exchange neighborhood improvement heuristic by up to 3.94 times, without requiring excessive amounts of memory.

These algorithms are used in Chapter 3, where we investigate the performance of Dynamic Time Slot Management in a real-time setting. We present a DTSM model to overcome a crucial challenge which hinders the widespread adoption of DTSM in practice. In our model, we incorporate the time it takes a customer to select a time slot from an offer, as well as the decision time needed to construct such an offer. Because customers typically do not choose instantaneously, a time slot offer might be invalidated by other customers who place an order in the meantime. To guarantee that a feasible delivery schedule exists, the time slot selected by the customer needs to be re-evaluated in a queue, which introduces additional waiting time. We show that state-of-the-art DTSM procedures are not well equipped to deal with these issues, resulting in poor performance or long response times, i.e., waiting times experienced by customers. We present several procedures to deal with this.

In our experiments, we simulate a real-time order process with up to 8000 customers arriving in a time span of as much as 80000 seconds or as little as 8 milliseconds. Our computational study shows the effect of customer arrival rates on incurred waiting time and the effects of invalided time slot offers. Furthermore, we see the benefit of quick time slot offer and check procedures, in combination with background improvement procedures, which utilize the time between customer placements for improvement of the delivery schedule in memory.

To further decrease the time slot offer decision times, it might be beneficial to move some of the work to a strategic phase, i.e., before the ordering process. In Chapter 4, we present Strategic Time Slot Management, a novel variant of Dynamic Time Slot Management, which utilizes a priori delivery routes and time slot assignment. In online grocery retailing, many customers tend to order with some regularity, and have favorite time slots and delivery days. This information can be exploited in a strategic phase. Inspired by current practice, we propose the use of a priori routes and time slot assignment, which simplifies the management of time slot during the ordering processes considerably, while still guaranteeing a feasible schedule of placed customer orders. Moreover, it allows smoothing of the fulfillment center operations and creates delivery consistency. We model the design problem of finding a priori routes and time slot assignment, and investigate the single vehicle (or single route) case. We propose a 2-stage stochastic programming formulation for the design problem and develop a sample average approximation solution approach. The expected revenue of an a priori route is evaluated by an efficient dynamic program. Furthermore, we present a simple heuristic based on a fixed route and a time slot assignment decision rule.

An extensive computational study on random instances with up to 12 customers shows the efficacy of the proposed solution methods, and illustrates the relation between the expected number of accepted customers and the available time for the route. Moreover, while the optimal a priori route is not always a minimum duration (TSP) tour, it provides a solution which is close, 97% on average, and lowers computation times drastically. Our computational study also provides insights in the design of the set of possible time slots. In particular, when the time slot width is halved, the expected number of accepted customers decreases by 2%. On average 2% more customers can be gained by extending the set of possible time slots with overlapping ones.

References

N. Agatz, A. M. Campbell, M. Fleischmann, and M. W. P. Savelsbergh. Time slot management in attended home delivery. *Transportation Science*, 45(3):435–449, 2011.

N. Agatz, A. M. Campbell, M. Fleischmann, J. van Nunen, and M. W. P. Savelsbergh. Revenue management opportunities for internet retailers. *Journal of Revenue & Pricing Management*, 12(2):128–138, 2013.

Akamai. Akamai Online Retail Performance Report: Milliseconds Are Critical, Apr. 2017. Retrieved from `https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp`.

E. Angelelli, C. Archetti, C. Filippi, and M. Vindigni. The probabilistic orienteering problem. *Computers & Operations Research*, 81:269 – 281, 2017.

S. Balseiro, I. Loiseau, and J. Ramonet. An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. *Computers & Operations Research*, 38(6):954 – 966, 2011.

B. P. Bruck, J.-F. Cordeau, and M. Iori. A practical time slot management and routing problem for attended home services. *Omega*, 81:208 – 219, 2018.

A. M. Campbell and M. W. P. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

A. M. Campbell and M. W. P. Savelsbergh. Decision support for consumer direct grocery initiatives. *Transportation Science*, 39(3):313–327, 2005.

A. M. Campbell and M. W. P. Savelsbergh. Incentive schemes for attended home delivery services. *Transportation Science*, 40(3):327–341, 2006.

A. M. Campbell and B. W. Thomas. Challenges and advances in a priori routing. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces, pages 123–142. Springer US, Boston, MA, 2008a.

A. M. Campbell and B. W. Thomas. Probabilistic traveling salesman problem with deadlines. *Transportation Science*, 42(1):1–21, 2008b.

C. Cleophas and J. F. Ehmke. When are deliveries profitable? *Business & Information Systems Engineering*, 6(3):153–163, 2014.

S. Dabia, S. Ropke, T. van Woensel, and T. De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3): 380–396, 2013.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6 (1):80–91, 1959.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Chapter 5: Orthogonal range searching. In *Computational Geometry: Algorithms and Applications*, pages 95–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

G. Desaulniers, O. B. Madsen, and S. Ropke. Chapter 5: The Vehicle Routing Problem with Time Windows. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, volume 18 of *MOS-SIAM Series on Optimization*, chapter 5, pages 119–159. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, second edition, 2014.

A. V. Donati, R. Montemanni, N. Casagrande, A. E. Rizzoli, and L. M. Gambardella. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, 185(3):1174 – 1191, 2008.

J. F. Ehmke and A. M. Campbell. Customer acceptance mechanisms for home deliveries in metropolitan areas. *European Journal of Operational Research*, 233(1):193 – 207, 2014.

A. L. Erera, M. Savelsbergh, and E. Uyar. Fixed routes with backup vehicles for stochastic vehicle routing problems with time constraints. *Networks*, 54(4):270–283, 2009.

M. A. Figliozzi. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, 48(3):616 – 636, 2012.

A. Garcia, P. Vansteenwegen, O. Arbelaitz, W. Souffriau, and M. T. Linaza. Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research*, 40(3):758 – 774, 2013.

D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research*, 62:36 – 50, 2015.

H. Gehring and J. Homberger. A parallel two-phase metaheuristic for routing problems with time windows. *Asia - Pacific Journal of Operational Research*, 18(1):35–47, 2001.

M. Gendreau, O. Jabali, and W. Rei. Chapter 8: Stochastic vehicle routing problems. In P. Toth and D. Vigo, editors, *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*, chapter 8, pages 213–239. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, second edition, 2014.

M. Gendreau, G. Ghiani, and E. Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189 – 197, 2015.

G. Ghiani and E. Guerriero. A Note on the Ichoua, Gendreau, and Potvin (2003) Travel Time Model. *Transportation Science*, 48(3):458–462, 2014.

C. Groër, B. Golden, and E. Wasil. The consistent vehicle routing problem. *Manufacturing & Service Operations Management*, 11(4):630–643, 2009.

A. Gunawan, H. C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332, 2016.

H. Hashimoto, M. Yagiura, and T. Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, 5(2):434 – 456, 2008.

F. Hernandez, M. Gendreau, and J.-Y. Potvin. Heuristics for tactical time slot management: a periodic vehicle routing problem view. *International Transactions in Operational Research*, 24(6):1233–1252, 2017.

S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

S. Irnich. A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. *INFORMS Journal on Computing*, 20(2):270–287, 2008.

S. Irnich, P. Toth, and D. Vigo. Chapter 1: The Family of Vehicle Routing Problems. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, volume 18 of *MOS-SIAM Series on Optimization*, chapter 1, pages 1–33. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, second edition, 2014.

P. Jaillet. A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36(6):929–936, 1988.

G. A. P. Kindervater and M. W. P. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

R. Klein, M. Neugebauer, D. Ratkovitch, and C. Steinhardt. Differentiated time slot pricing under routing considerations in attended home delivery. *Transportation Science*, 53(1): 236–255, 2019.

A. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

C. Köhler, J. F. Ehmke, and A. M. Campbell. Flexible time window management for attended home deliveries. *Omega*, 2019. doi: 10.1016/j.omega.2019.01.001. URL `http://www.sciencedirect.com/science/article/pii/S030504831830803X`.

A. L. Kok, E. W. Hans, J. M. J. Schutten, and W. H. M. Zijm. A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible Services and Manufacturing Journal*, 22(1):83–108, 2010.

A. L. Kok, E. W. Hans, and J. M. J. Schutten. Optimizing departure times in vehicle routes. *European Journal of Operational Research*, 210(3):579 – 587, 2011.

A. A. Kovacs, B. L. Golden, R. F. Hartl, and S. N. Parragh. Vehicle routing problems in which consistency considerations are important: A survey. *Networks*, 64(3):192–213, 2014.

N. Labadie, C. Prins, and C. Prodhon. Metaheuristics generating a sequence of solutions. In *Metaheuristics for Vehicle Routing Problems*, pages 39–75. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1st edition, 2016.

J. Mendoza, C. Guéret, M. Hoskins, H. Lobit, V. Pillac, T. Vidal, and D. Vigo. VRP-REP: a vehicle routing community repository, 06 2014. URL `http://vrp-rep.org`.

J. Oyola, H. Arntzen, and D. L. Woodruff. The stochastic vehicle routing problem, a literature review, Part II: solution methods. *EURO Journal on Transportation and Logistics*, 6(4):349–388, Dec 2017.

J. Oyola, H. Arntzen, and D. L. Woodruff. The stochastic vehicle routing problem, a literature review, Part I: models. *EURO Journal on Transportation and Logistics*, 7(3): 193–221, Sep 2018.

J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331 – 340, 1993. ISSN 0377-2217.

G. Reinelt. TSPLIB–A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

J. Russell and R. Liao. Singles' Day: China's $25 billion shopping festival explained. *TechCrunch*, Nov. 2018. Retrieved from `https://techcrunch.com/2018/11/09/alibaba-singles-day-11-festival/`.

M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

N. J. A. Sloane. Sequence A000522. *The On-Line Encyclopedia of Integer Sequences*, 2010. URL `https://oeis.org/A000522`.

R. Spliet and G. Desaulniers. The discrete time window assignment vehicle routing problem. *European Journal of Operational Research*, 244(2):379 – 391, 2015.

R. Spliet and A. F. Gabor. The Time Window Assignment Vehicle Routing Problem. *Transportation Science*, 49(4):721–731, 2015.

R. Spliet, S. Dabia, and T. van Woensel. The time window assignment vehicle routing problem with time-dependent travel times. *Transportation Science*, 52(2):261–276, 2018.

T. Sterling. Startup Picnic runs grocery delivery bus in Dutch online shopping boom. *Reuters*, September 2018. Retrieved from `https://www.reuters.com/article/us-netherlands-grocery-internet/startup-picnic-runs-grocery-delivery-bus-in-dutch-online-shopping-boom-idUSKCN1LZ244`.

A. Subramanyam and C. E. Gounaris. A branch-and-cut framework for the consistent traveling salesman problem. *European Journal of Operational Research*, 248(2):384 – 395, 2016.

A. Subramanyam and C. E. Gounaris. A decomposition algorithm for the consistent traveling salesman problem with vehicle idling. *Transportation Science*, 52(2):386–401, 2018.

N. van Leeuwen, T. Guldemond, and F. Faqiri. Statistische gegevens per vierkant en postcode 2017. *Statistics Netherlands (CBS)*, Nov. 2017. Retrieved from `https://www.cbs.nl/nl-nl/dossier/nederland-regionaal/geografische-data/gegevens-per-postcode`.

C. Verbeeck, K. Sörensen, E.-H. Aghezzaf, and P. Vansteenwegen. A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research*, 236(2):419 – 432, 2014.

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1 – 21, 2013.

T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, 65(2):102–128, 2015.

T. R. Visser and M. W. P. Savelsbergh. Strategic Time Slot Management: A Priori Routing for Online Grocery Retailing. Technical Report EI2019-04, Econometric Institute Research Papers, Jan. 2019. URL `http://hdl.handle.net/1765/114947`.

T. R. Visser and R. Spliet. Efficient Move Evaluations for Time-Dependent Vehicle Routing Problems. Technical Report EI2017-23, Econometric Institute Research Papers, Aug. 2017. URL `http://hdl.handle.net/1765/100852`.

S. A. Voccia, A. M. Campbell, and B. W. Thomas. The probabilistic traveling salesman problem with time windows. *EURO Journal on Transportation and Logistics*, 2(1):89–107, May 2013.

X. Yang, A. K. Strauss, C. S. M. Currie, and R. Eglese. Choice-based demand management and vehicle routing in e-fulfillment. *Transportation Science*, 50(2):473–488, 2016.

E. E. Zachariadis and C. T. Kiranoudis. A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research*, 37(12):2089 – 2105, 2010.

Abstract

Online retailing continues to grow, and consumers, but also small businesses, purchase more and more products online. Many of these products require attended delivery for which the customer needs to stay at home to receive their purchases. To decrease the chances of costly delivery failures and to provide customers with a high level of service, many online retailers offer their customers a menu of delivery time slots. The management of these time slots can be challenging, as the customer demand can vary heavily, and the amount of available delivery vehicles and drivers may be limited. Dynamic Time Slot Management (DTSM) is a class of methods which dynamically construct time slot offers based on previously placed customer orders. Our focus is the use of vehicle routing heuristics within DTSM to help retailers manage the availability of time slots in real time.

In this dissertation, we explore several challenges that hinder the widespread adoption of DTSM in practice. As vehicle routing is used in real time, the computation time plays a crucial role. We study pre-calculation techniques for a particular class of vehicle routing problems, and illustrate the trade-off between computation time and memory use. Furthermore, as multiple customers arrive and interact with the DTSM system simultaneously, several previously unstudied issues arise. We model such simultaneous interactions and study their impact on the real-time performance of the system in terms of response times and number of accepted customers. Finally, we explore a novel variant of DTSM in which routes and time slots are assigned a priori in a strategical phase to simplify their real-time management. Although this reduces the number of different time slots that can be offered to customers, advantages include smoothing of fulfillment center operations and delivery consistency.

Samenvatting (Summary in Dutch)

Online retailing groeit nog steeds en consumenten, maar ook kleine bedrijven, bestellen steeds meer producten online. Veel van deze producten kunnen enkel afgeleverd worden als de klant thuis is om ze in ontvangst te nemen. Om de kans op kostbare gemiste bezorgingen te verkleinen en de mate van service te verhogen, laten veel online bezorgbedrijven hun klanten kiezen uit een aantal bezorgblokken. De sturing van de beschikbaarheid van deze bezorgblokken is uitdagend, gezien de klantvraag zeer kan varieëren en het aantal voertuigen en chauffeurs gelimiteerd kan zijn. *Dynamic Time Slot Management* (DTSM) is een klasse van methoden waarbij een bezorgblokaanbod aan de klant dynamisch wordt geconstrueerd op basis van de reeds geplaatste klantorders.

In dit proefschrift onderzoeken we een aantal uitdagingen die een praktische implementatie van DTSM in de weg staan. Zo zijn bij het gebruik van real-time voertuigroutering de berekentijden cruciaal. We onderzoeken voorberekeningstechnieken voor een klasse van voertuigrouteringsproblemen en illustreren de balans tussen rekentijd en geheugengebruik. Verder, gezien meerdere klanten simultaan arriveren en interacteren met het bezorgbloksysteem, ontstaan er problemen die nog niet eerder zijn bestudeerd. We modelleren deze simultane interacties en onderzoeken hun impact op de real-time prestaties van het systeem op het gebied van responstijden en aantal geaccepteerde klanten. Tot slot onderzoeken we een nieuwe variant van DTSM waarbij routes en bezorgblokken vooraf geconstrueerd en toegewezen worden in een strategische planfase. Dit ter versimpeling van de dynamische sturing van de bezorgblokbeschikbaarheid. Ondanks het verminderde aanbod van verschillende bezorgblokken, zijn de voordelen onder meer de versoepeling van de magazijnwerkzaamheden en de consistentie in de bezorgingen.

About the cover

The cover of this dissertation consists of three layers. The upper layer illustrates a typical time slot page shown to a customer. Some time slots are offered (blue and green, with delivery fees), while others are unavailable (grey, with 'FULL'). The middle layer illustrates a geographical map of in total 7646 different customer locations. These customer locations were chosen to represent the image of the time slot page. Green (red) color represents the (non-)availability of a new customer from that location. The lower layer illustrates a vehicle routing schedule of 7646 placed customers, using in total 240 vehicles from 4 depots. This schedule was the result of simulating the customers as described in Chapter 3, using the INS+NS method.

About the author

Thomas R. Visser was born on the 11th of March 1990 in Utrecht, The Netherlands. In 2012, he obtained two bachelor degrees, both in Mathematical Sciences and in Physics at Utrecht University. During this period he also participated in the Honors Program of Experimental Physics. In 2012, he continued his studies at Utrecht University with a master in Mathematical Sciences, with specializations in operations research, optimization and stochastics. During this period, he also worked as a student-assistant at the consultancy and planning software firm ORTEC. In 2015 Thomas started a PhD project at Erasmus University Rotterdam, under the supervision of dr. Remy Spliet and prof.dr. Albert Wagelmans. This project was in collaboration with online grocery retailer Albert Heijn Online and ORTEC. Thomas looked into designing sustainable last-mile delivery services, with a strong focus on practical applicability. He presented his work not only on several academic conferences like TRISTAN, TSL, IFORS and VeRoLog but also at professional events like the Emerce eFulfilment & Logistics conference, where he was one of the invited speakers. Currently, Thomas works at ORTEC, where he works on bringing his research into practice.

Portfolio

Working papers and Reports

Thomas R. Visser and Remy Spliet (2017). Efficient Move Evaluations for Time-Dependent Vehicle Routing Problems, *Econometric Institute Research Papers*, EI2017-23. *Accepted for publication in Transportation Science.*

Thomas R. Visser and Martin W.P. Savelsbergh (2018). Strategic Time Slot Management: A Priori Routing for Online Grocery Retailing, *Econometric Institute Research Papers*, EI2019-04.

Thomas R. Visser, Niels Agatz and Remy Spliet (2019). When microseconds add up: On the real-time performance of Dynamic Time Slot Management. *In preparation for submission.*

Research Visits

Multiple visits to the H. Milton Stewart School of Industrial and Systems Engineering (Georgia Institute of Technology) in Atlanta, USA. Collaboration with Prof. dr. Martin Savelsbergh on a new study.

Teaching and Supervising activities

Teaching Assistant and Case Design for *Introductory Seminar Case Operations Research* (Bachelor course), 2016-2018.

Teaching Assistant for *Linear Programming* (Bachelor course), 2015-2016.

Supervision of bachelor theses, Erasmus School of Economics (ESE).

Second reader of master theses, Erasmus School of Economics (ESE).

PhD Courses and Certificates

Algorithms and Complexity
Interior Point Methods
Markov Decision Processes
Integer Programming Methods
Noncooperative Games
Robust Optimization
Stochastic Programming
Networks and Polyhedra
Convex Analysis for Optimization
Reading Group Constraint Programming
Scientific Integrity
Publishing Strategy
Cambridge English: Proficiency

Conference Presentations

TRISTAN 2016, Oranjestad, Aruba.
TSL Workshop 2016, Atlanta, USA.
Emerce eFulfillment 2016, Utrecht, The Netherlands.
LNMB Conference 2017, Lunteren, The Netherlands.
Last-Mile Delivery Workshop 2017, Mannheim, Germany.
VeRoLog 2017, Amsterdam, The Netherlands.
IFORS 2017, Québec, Canada.
NWO Logistics, Midterm Conference 2017, Utrecht, The Netherlands.
Last-Mile Delivery Workshop 2018, Rotterdam, The Netherlands.
Last-Mile Delivery Workshop 2019, Magdeburg, Germany.

Other Presentations

Constraint Programming seminar, 2016, Rotterdam, The Netherlands.
ORTEC seminar, 2017, Zoetermeer, The Netherlands.
Guest lecture, Supply Chain Management Honours Programme, 2017, Rotterdam School of Management, The Netherlands.
Econometric Institute seminar, 2017, Rotterdam, The Netherlands.
Logistikmanagement seminar, 2017, Mainz, Germany.
OPAC seminar, 2017, Eindhoven, The Netherlands.
ORTEC seminar, 2018, Zoetermeer, The Netherlands.

Reviewing activities for Peer-reviewed Journals

Networks

Transportation Science

The ERIM PhD Series

The ERIM PhD Series contains PhD dissertations in the field of Research in Management defended at Erasmus University Rotterdam and supervised by senior researchers affiliated to the Erasmus Research Institute of Management (ERIM). All dissertations in the ERIM PhD Series are available in full text through the ERIM Electronic Series Portal: repub.eur.nl/pub. ERIM is the joint research institute of the Rotterdam School of Management (RSM) and the Erasmus School of Economics (ESE) at the Erasmus University Rotterdam (EUR).

Dissertations in the last four years

Ahmadi, S., *A motivational perspective to decision-making and behavior in organizations*, Promotors: Prof. J.J.P. Jansen & Prof. T.J.M. Mom, EPS-2019-477-S&E, repub.eur.nl/pub/116727

Akemu, O., *Corporate Responses to Social Issues: Essays in Social Entrepreneurship and Corporate Social Responsibility*, Promotors: Prof. G.M. Whiteman & Dr. S.P. Kennedy, EPS-2017-392-ORG, repub.eur.nl/pub/95768

Albuquerque de Sousa, J.A., *International stock markets: Essays on the determinants and consequences of financial market development*, Promotors: Prof. M.A. van Dijk & Prof. P.A.G. van Bergeijk, EPS-2019-465-F&A, repub.eur.nl/pub/115988

Alexiou, A., *Management of Emerging Technologies and the Learning Organization: Lessons from the Cloud and Serious Games Technology*, Promotors: Prof. S.J. Magala, Prof. M.C. Schippers & Dr. I. Oshri, EPS-2016-404-ORG, repub.eur.nl/pub/93818

Alserda, G.A.G., *Choices in Pension Management*, Promotors: Prof. S.G. van der Lecq & Dr. O.W. Steenbeek, EPS-2017-432-F&A, repub.eur.nl/pub/103496

Arampatzi, E., *Subjective Well-Being in Times of Crises: Evidence on the Wider Impact of Economic Crises and Turmoil on Subjective Well-Being*, Promotors: Prof. H.R. Commandeur, Prof. F. van Oort & Dr. M.J. Burger, EPS-2018-459-S&E, repub.eur.nl/pub/111830

Avci, E., *Surveillance of Complex Auction Markets: a Market Policy Analytics Approach*, Promotors: Prof. W. Ketter, Prof. H.W.G.M. van Heck & Prof. D.W. Bunn, EPS-2018-426-LIS, repub.eur.nl/pub/106286

Balen, T.H. van, *Challenges of Early Stage Entrepreneurs : the Roles of Vision Communication and Team Membership Change*, Promotors: Prof. J.C.M van den Ende & Dr. M. Tarakci, EPS-2018-468-LIS, repub.eur.nl/pub/115654

Benschop, N, *Biases in Project Escalation: Names, frames & construal levels*, Promotors: Prof. K.I.M. Rhode, Prof. H.R. Commandeur, Prof. M. Keil & Dr. A.L.P. Nuijten, EPS-2015-375-S&E, repub.eur.nl/pub/79408

Bernoster, I., *Essays at the Intersection of Psychology, Biology, and Entrepreneurship*, Promotors: Prof. A.R. Thurik, Prof. I.H.A. Franken & Prof. P.J.F Groenen, EPS-2018-463-S&E, repub.eur.nl/pub/113907

Beusichem, H.C. van, *Firms and Financial Markets: Empirical Studies on the Informational Value of Dividends, Governance and Financial Reporting*, Promotors: Prof. A. de Jong & Dr. G. Westerhuis, EPS-2016-378-F&A, repub.eur.nl/pub/93079

Bouman, P., *Passengers, Crowding and Complexity: Models for Passenger Oriented Public Transport*, Promotors: Prof. L.G. Kroon, Prof. A. Schöbel & Prof. P.H.M. Vervest, EPS-2017-420-LIS, repub.eur.nl/pub/100767

Bunderen, L. van, *Tug-of-War: Why and when teams get embroiled in power struggles*, Promotors: Prof. D.L. van Knippenberg & Dr. L. Greer, EPS-2018-446-ORG, repub.eur.nl/pub/105346

Burg, G.J.J. van den, *Algorithms for Multiclass Classification and Regularized Regression*, Promotors: Prof. P.J.F. Groenen & Dr. A. Alfons, EPS-2018-442-MKT, repub.eur.nl/pub/103929

Chammas, G., *Portfolio concentration*, Promotor: Prof. J. Spronk, EPS-2017-410-F&E, repub.eur.nl/pub/94975

Consiglio, I., *Others: Essays on Interpersonal and Consumer Behavior*, Promotor: Prof. S.M.J. van Osselaer, EPS-2016-366-MKT, repub.eur.nl/pub/79820

Cranenburgh, K.C. van, *Money or Ethics: Multinational corporations and religious organisations operating in an era of corporate responsibility*, Promotors: Prof. L.C.P.M. Meijs, Prof. R.J.M. van Tulder & Dr. D. Arenas, EPS-2016-385-ORG, repub.eur.nl/pub/93104

Darnihamedani, P., *Individual Characteristics, Contextual Factors and Entrepreneurial Behavior*, Promotors: Prof. A.R. Thurik & S.J.A. Hessels, EPS-2016-360-S&E, repub.eur.nl/pub/93280

Dennerlein, T., *Empowering Leadership and Employees' Achievement Motivations: the Role of Self-Efficacy and Goal Orientations in the Empowering Leadership Process*, Promotors: Prof. D.L. van Knippenberg & Dr. J. Dietz, EPS-2017-414-ORG, repub.eur.nl/pub/98438

Depecik, B.E., *Revitalizing brands and brand: Essays on Brand and Brand Portfolio Management Strategies*, Promotors: Prof. G.H. van Bruggen, Dr. Y.M. van Everdingen and Dr. M.B. Ataman, EPS-2016-406-MKT, repub.eur.nl/pub/93507

Duijzer, L.E., *Mathematical Optimization in Vaccine Allocation*, Promotors: Prof. R. Dekker & Dr. W.L. van Jaarsveld, EPS-2017-430-LIS, repub.eur.nl/pub/101487

Duyvesteyn, J.G., *Empirical Studies on Sovereign Fixed Income Markets*, Promotors: Prof. P. Verwijmeren & Prof. M.P.E. Martens, EPS-2015-361-F&A, repub.eur.nl/pub/79033

El Nayal, O.S.A.N., *Firms and the State: An Examination of Corporate Political Activity and the Business-Government Interface*, Promotor: Prof. J. van Oosterhout & Dr. M. van Essen, EPS-2018-469-S&E, repub.eur.nl/pub/114683

Elemes, A., *Studies on Determinants and Consequences of Financial Reporting Quality*, Promotor: Prof. E. Peek, EPS-2015-354-F&A, repub.eur.nl/pub/79037

Erlemann, C., *Gender and Leadership Aspiration: The Impact of the Organizational Environment*, Promotor: Prof. D.L. van Knippenberg, EPS-2016-376-ORG, repub.eur.nl/pub/79409

Faber, N., *Structuring Warehouse Management*, Promotors: Prof. M.B.M. de Koster & Prof. A. Smidts, EPS-2015-336-LIS, repub.eur.nl/pub/78603

Feng, Y., *The Effectiveness of Corporate Governance Mechanisms and Leadership Structure: Impacts on strategic change and firm performance*, Promotors: Prof. F.A.J. van den Bosch, Prof. H.W. Volberda & Dr. J.S. Sidhu, EPS-2017-389-S&E, repub.eur.nl/pub/98470

Fernald, K., *The Waves of Biotechnological Innovation in Medicine: Interfirm Cooperation Effects and a Venture Capital Perspective*, Promotors: Prof. E. Claassen, Prof. H.P.G. Pennings & Prof. H.R. Commandeur, EPS-2015-371-S&E, repub.eur.nl/pub/79120

Fisch, C.O., *Patents and trademarks: Motivations, antecedents, and value in industrialized and emerging markets*, Promotors: Prof. J.H. Block, Prof. H.P.G. Pennings & Prof. A.R. Thurik, EPS-2016-397-S&E, repub.eur.nl/pub/94036

Fliers, P.T., *Essays on Financing and Performance: The role of firms, banks and board*, Promotors: Prof. A. de Jong & Prof. P.G.J. Roosenboom, EPS-2016-388-F&A, repub.eur.nl/pub/93019

Frick, T.W., *The Implications of Advertising Personalization for Firms, Consumer, and Ad Platfroms*, Promotors: Prof. T. Li & Prof. H.W.G.M. van Heck, EPS-2018-452-LIS, repub.eur.nl/pub/110314

Fytraki, A.T., *Behavioral Effects in Consumer Evaluations of Recommendation Systems*, Promotors: Prof. B.G.C. Dellaert & Prof. T. Li, EPS-2018-427-MKT, repub.eur.nl/pub/110457

Gaast, J.P. van der, *Stochastic Models for Order Picking Systems*, Promotors: Prof. M.B.M de Koster & Prof. I.J.B.F. Adan, EPS-2016-398-LIS, repub.eur.nl/pub/93222

Ghazizadeh, P., *Empirical Studies on the Role of Financial Information in Asset and Capital Markets*, Promotors: Prof. A. de Jong & Prof. E. Peek. EPS-2019-470-F&A repub.eur.nl/pub/114023

Giurge, L., *A Test of Time; A temporal and dynamic approach to power and ethics*, Promotors: Prof. M.H. van Dijke & Prof. D. De Cremer, EPS-2017-412-ORG, repub.eur.nl/pub/98451

Gobena, L., *Towards Integrating Antecedents of Voluntary Tax Compliance*, Promotors: Prof. M.H. van Dijke & Dr. P. Verboon, EPS-2017-436-ORG, repub.eur.nl/pub/103276

Groot, W.A., *Assessing Asset Pricing Anomalies*, Promotors: Prof. M.J.C.M. Verbeek & Prof. J.H. van Binsbergen, EPS-2017-437-F&A, repub.eur.nl/pub/103490

Hanselaar, R.M., *Raising Capital: On pricing, liquidity and incentives*, Promotors: Prof. M.A. van Dijk & Prof. P.G.J. Roosenboom, EPS-2018-429-F&A-9789058925404, repub.eur.nl/pub/113274

Harms, J. A., *Essays on the Behavioral Economics of Social Preferences and Bounded Rationality*, Promotors: Prof. H.R. Commandeur & Dr. K.E.H. Maas, EPS-2018-457-S&E, repub.eur.nl/pub/108831

Hekimoglu, M., *Spare Parts Management of Aging Capital Products*, Promotor: Prof. R. Dekker, EPS-2015-368-LIS, repub.eur.nl/pub/79092

Hendriks, G., *Multinational Enterprises and Limits to International Growth: Links between Domestic and Foreign Activities in a Firm's Portfolio*, Promotors: Prof. P.P.M.A.R. Heugens & Dr. A.H.L Slangen, EPS-2019-464-S&E, repub.eur.nl/pub/114981

Hengelaar, G.A., *The Proactive Incumbent: Holy grail or hidden gem? Investigating whether the Dutch electricity sector can overcome the incumbent's curse and lead the sustainability transition*, Promotors: Prof. R.J. M. van Tulder & Dr. K. Dittrich, EPS-2018-438-ORG, repub.eur.nl/pub/102953

Hogenboom, A.C., *Sentiment Analysis of Text Guided by Semantics and Structure*, Promotors: Prof. U. Kaymak & Prof. F.M.G. de Jong, EPS-2015-369-LIS, repub.eur.nl/pub/79034

Hollen, R.M.A., *Exploratory Studies into Strategies to Enhance Innovation-Driven International Competitiveness in a Port Context: Toward Ambidextrous Ports*, Promotors: Prof. F.A.J. Van Den Bosch & Prof. H.W. Volberda, EPS-2015-372-S&E, repub.eur.nl/pub/78881

Jacobs, B.J.D., *Marketing Analytics for High-Dimensional Assortments*, Promotors: Prof. A.C.D. Donkers & Prof. D. Fok, EPS-2017-445-MKT, repub.eur.nl/pub/103497

Jia, F., *The Value of Happiness in Entrepreneurship*, Promotors: Prof. D.L. van Knippenberg & Dr. Y. Zhang, EPS-2019-479-ORG, repub.eur.nl/pub/115990

Kahlen, M. T., *Virtual Power Plants of Electric Vehicles in Sustainable Smart Electricity Markets*, Promotors: Prof. W. Ketter & Prof. A. Gupta, EPS-2017-431-LIS, repub.eur.nl/pub/100844

Kampen, S. van, *The Cross-sectional and Time-series Dynamics of Corporate Finance: Empirical evidence from financially constrained firms*, Promotors: Prof. L. Norden & Prof. P.G.J. Roosenboom, EPS-2018-440-F&A, repub.eur.nl/pub/105245

Karali, E., *Investigating Routines and Dynamic Capabilities for Change and Innovation*, Promotors: Prof. H.W. Volberda, Prof. H.R. Commandeur & Dr. J.S. Sidhu, EPS-2018-454-S&E, repub.eur.nl/pub/106274

Keko, E., *Essays on Innovation Generation in Incumbent Firms*, Promotors: Prof. S. Stremersch & Dr. N.M.A. Camacho, EPS-2017-419-MKT, repub.eur.nl/pub/100841

Kerkkamp, R.B.O., *Optimisation Models for Supply Chain Coordination under Information Asymmetry*, Promotors: Prof. A.P.M. Wagelmans & Dr. W. van den Heuvel, EPS-2018-462-LIS, repub.eur.nl/pub/109770

Khattab, J., *Make Minorities Great Again: a contribution to workplace equity by identifying and addressing constraints and privileges*, Promotors: Prof. D.L. van Knippenberg & Dr. A. Nederveen Pieterse, EPS-2017-421-ORG, repub.eur.nl/pub/99311

Kim, T. Y., *Data-driven Warehouse Management in Global Supply Chains*, Promotors: Prof. R. Dekker & Dr. C. Heij, EPS-2018-449-LIS, repub.eur.nl/pub/109103

Klitsie, E.J., *Strategic Renewal in Institutional Contexts: The paradox of embedded agency*, Promotors: Prof. H.W. Volberda & Dr. S. Ansari, EPS-2018-444-S&E, repub.eur.nl/pub/106275

Kong, L., *Essays on Financial Coordination*, Promotors: Prof. M.J.C.M. Verbeek, Dr. D.G.J. Bongaerts & Dr. M.A. van Achter. EPS-2019-433-F&A, repub.eur.nl/pub/114516

Koolen, D., *Market Risks and Strategies in Power Systems Integrating Renewable Energy*, Promotors: Prof. W. Ketter & Dr. R. Huisman, EPS-2019-467-LIS, repub.eur.nl/pub/115655

Krämer, R., *A license to mine? Community organizing against multinational corporations*, Promotors: Prof. R.J.M. van Tulder & Prof. G.M. Whiteman, EPS-2016-383-ORG, repub.eur.nl/pub/94072

Kyosev, G.S., *Essays on Factor Investing*, Promotors: Prof. M.J.C.M. Verbeek & Dr. J.J. Huij, EPS-2019-474-F&A, repub.eur.nl/pub/116463

Lamballais, T., *Optimizing the Performance of Robotic Mobile Fulfillment Systems*, Promotors: Prof. M.B.M de Koster & Prof. R. Dekker & Dr. D. Roy, EPS-2019-411-LIS, repub.eur.nl/pub/116477

Lee, C.I.S.G., *Big Data in Management Research: Exploring New Avenues*, Promotors: Prof. S.J. Magala & Dr. W.A. Felps, EPS-2016-365-ORG, repub.eur.nl/pub/79818

Legault-Tremblay, P.O., *Corporate Governance During Market Transition: Heterogeneous responses to Institution Tensions in China*, Promotor: Prof. B. Krug, EPS-2015-362-ORG, repub.eur.nl/pub/78649

Lenoir, A.S., *Are You Talking to Me? Addressing Consumers in a Globalised World*, Promotors: Prof. S. Puntoni & Prof. S.M.J. van Osselaer, EPS-2015-363-MKT, repub.eur.nl/pub/79036

Leung, W.L., *How Technology Shapes Consumption: Implications for Identity and Judgement*, Promotors: Prof. S. Puntoni & Dr. G. Paolacci, EPS-2019-485-MKT, repub.eur.nl/pub/117432

Li, D., *Supply Chain Contracting for After-sales Service and Product Support*, Promotor: Prof. M.B.M. de Koster, EPS-2015-347-LIS, repub.eur.nl/pub/78526

Li, X., *Dynamic Decision Making under Supply Chain Competition*, Promotors: Prof. M.B.M de Koster, Prof. R. Dekker & Prof. R. Zuidwijk. EPS-2018-466-LIS, repub.eur.nl/pub/114028

Liu, N., *Behavioral Biases in Interpersonal Contexts*, Promotors: Prof. A. Baillon & Prof. H. Bleichrodt, EPS-2017-408-MKT, repub.eur.nl/pub/95487

Ma, Y., *The Use of Advanced Transportation Monitoring Data for Official Statistics*, Promotors: Prof. L.G. Kroon & Dr. J. van Dalen, EPS-2016-391-LIS, repub.eur.nl/pub/80174

Maas, A.J.J., *Organizations and their external context: Impressions across time and space*, Promotors: Prof. P.P.M.A.R. Heugens & Prof. T.H. Reus, EPS-2019-478-S&E, repub.eur.nl/pub/116480

Maira, E., *Consumers and Producers*, Promotors: Prof. S. Puntoni & Prof. C. Fuchs, EPS-2018-439-MKT, repub.eur.nl/pub/104387

Mell, J.N., *Connecting Minds: On The Role of Metaknowledge in Knowledge Coordination*, Promotor: Prof. D.L. van Knippenberg, EPS-2015-359-ORG, repub.eur.nl/pub/78951

Meulen, D. van der, *The Distance Dilemma: the effect of flexible working practices on performance in the digital workplace*, Promotors: Prof. H.W.G.M. van Heck & Prof. P.J. van Baalen, EPS-2016-403-LIS, repub.eur.nl/pub/94033

Moniz, A, *Textual Analysis of Intangible Information*, Promotors: Prof. C.B.M. van Riel, Prof. F.M.G de Jong & Dr. G.A.J.M. Berens, EPS-2016-393-ORG, repub.eur.nl/pub/93001

Mulder, J., *Network design and robust scheduling in liner shipping*, Promotors: Prof. R. Dekker & Dr. W.L. van Jaarsveld, EPS-2016-384-LIS, repub.eur.nl/pub/80258

Neerijnen, P., *The Adaptive Organization: the socio-cognitive antecedents of ambidexterity and individual exploration*, Promotors: Prof. J.J.P. Jansen, P.P.M.A.R. Heugens & Dr. T.J.M. Mom, EPS-2016-358-S&E, repub.eur.nl/pub/93274

Okbay, A., *Essays on Genetics and the Social Sciences*, Promotors: Prof. A.R. Thurik, Prof. Ph.D. Koellinger & Prof. P.J.F. Groenen, EPS-2017-413-S&E, repub.eur.nl/pub/95489

Oord, J.A. van, *Essays on Momentum Strategies in Finance*, Promotor: Prof. H.K. van Dijk, EPS-2016-380-F&A, repub.eur.nl/pub/80036

Peng, X., *Innovation, Member Sorting, and Evaluation of Agricultural Cooperatives*, Promotor: Prof. G.W.J. Hendriks, EPS-2017-409-ORG, repub.eur.nl/pub/94976

Pennings, C.L.P., *Advancements in Demand Forecasting: Methods and Behavior*, Promotors: Prof. L.G. Kroon, Prof. H.W.G.M. van Heck & Dr. J. van Dalen, EPS-2016-400-LIS, repub.eur.nl/pub/94039

Petruchenya, A., *Essays on Cooperatives: Emergence, Retained Earnings, and Market Shares*, Promotors: Prof. G.W.J. Hendriks & Dr. Y. Zhang, EPS-2018-447-ORG, repub.eur.nl/pub/105243

Plessis, C. du, *Influencers: The Role of Social Influence in Marketing*, Promotors: Prof. S. Puntoni & Prof. S.T.L.R. Sweldens, EPS-2017-425-MKT, repub.eur.nl/pub/103265

Pocock, M., *Status Inequalities in Business Exchange Relations in Luxury Markets*, Promotors: Prof. C.B.M. van Riel & Dr. G.A.J.M. Berens, EPS-2017-346-ORG, repub.eur.nl/pub/98647

Pozharliev, R., *Social Neuromarketing: The role of social context in measuring advertising effectiveness*, Promotors: Prof. W.J.M.I. Verbeke & Prof. J.W. van Strien, EPS-2017-402-MKT, repub.eur.nl/pub/95528

Protzner, S., *Mind the gap between demand and supply: A behavioral perspective on demand forecasting*, Promotors: Prof. S.L. van de Velde & Dr. L. Rook, EPS-2015-364-LIS, repub.eur.nl/pub/79355

Reh, S.G., *A Temporal Perspective on Social Comparisons in Organizations*, Promotors: Prof. S.R. Giessner, Prof. N. van Quaquebeke & Dr. C. Troster, EPS-2018-471-ORG, repub.eur.nl/pub/114522

Riessen, B. van, *Optimal Transportation Plans and Portfolios for Synchromodal Container Networks*, Promotors: Prof. R. Dekker & Prof. R.R. Negenborn, EPS-2018-448-LIS, repub.eur.nl/pub/105248

Rietdijk, W.J.R., *The Use of Cognitive Factors for Explaining Entrepreneurship*, Promotors: Prof. A.R. Thurik & Prof. I.H.A. Franken, EPS-2015-356-S&E, repub.eur.nl/pub/79817

Roza, L., *Employee Engagement in Corporate Social Responsibility: A collection of essays*, Promotor: Prof. L.C.P.M. Meijs, EPS-2016-396-ORG, repub.eur.nl/pub/93254

Rösch, D., *Market Efficiency and Liquidity*, Promotor: Prof. M.A. van Dijk, EPS-2015-353-F&A, repub.eur.nl/pub/79121

Schie, R. J. G. van, *Planning for Retirement: Save More or Retire Later?*, Promotors: Prof. B. G. C. Dellaert & Prof. A.C.D. Donkers, EOS-2017-415-MKT, repub.eur.nl/pub/100846

Schoonees, P., *Methods for Modelling Response Styles*, Promotor: Prof. P.J.F. Groenen, EPS-2015-348-MKT, repub.eur.nl/pub/79327

Schouten, K.I.M., *Semantics-driven Aspect-based Sentiment Analysis*, Promotors: Prof. F.M.G. de Jong, Prof. R. Dekker & Dr. F. Frasincar, EPS-2018-453-LIS, repub.eur.nl/pub/112161

Schouten, M.E., *The Ups and Downs of Hierarchy: the causes and consequences of hierarchy struggles and positional loss*, Promotors; Prof. D.L. van Knippenberg & Dr. L.L. Greer, EPS-2016-386-ORG, repub.eur.nl/pub/80059

Sihag, V., *The Effectiveness of Organizational Controls: A meta-analytic review and an investigation in NPD outsourcing*, Promotors: Prof. J.C.M van den Ende & Dr. S.A. Rijsdijk, EPS-2019-476-LIS, repub.eur.nl/pub/115931

Smit, J., *Unlocking Business Model Innovation: A look through the keyhole at the inner workings of Business Model Innovation*, Promotor: Prof. H.G. Barkema, EPS-2016-399-S&E, repub.eur.nl/pub/93211

Straeter, L.M., *Interpersonal Consumer Decision Making*, Promotors: Prof. S.M.J. van Osselaer & Dr. I.E. de Hooge, EPS-2017-423-MKT, repub.eur.nl/pub/100819

Stuppy, A., *Essays on Product Quality*, Promotors: Prof. S.M.J. van Osselaer & Dr. N.L. Mead. EPS-2018-461-MKT, repub.eur.nl/pub/111375

Subaşi, B., *Demographic Dissimilarity, Information Access and Individual Performance*, Promotors: Prof. D.L. van Knippenberg & Dr. W.P. van Ginkel, EPS-2017-422-ORG, repub.eur.nl/pub/103495

Suurmond, R., *In Pursuit of Supplier Knowledge: Leveraging capabilities and dividing responsibilities in product and service contexts*, Promotors: Prof. J.Y.F Wynstra & Prof. J. Dul. EPS-2018-475-LIS, repub.eur.nl/pub/115138

Szatmari, B., *We are (all) the champions: The effect of status in the implementation of innovations*, Promotors: Prof. J.C.M van den Ende & Dr. D. Deichmann, EPS-2016-401-LIS, repub.eur.nl/pub/94633

Toxopeus, H.S., *Financing sustainable innovation: From a principal-agent to a collective action perspective*, Promotors: Prof. H.R. Commandeur & Dr. K.E.H. Maas. EPS-2019-458-S&E, repub.eur.nl/pub/114018

Turturea, R., *Overcoming Resource Constraints: The Role of Creative Resourcing and Equity Crowdfunding in Financing Entrepreneurial Ventures*, Promotors: Prof. P.P.M.A.R Heugens, Prof. J.J.P. Jansen & Dr. I. Verheuil, EPS-2019-472-S&E, repub.eur.nl/pub/112859

Valogianni, K., *Sustainable Electric Vehicle Management using Coordinated Machine Learning*, Promotors: Prof. H.W.G.M. van Heck & Prof. W. Ketter, EPS-2016-387-LIS, repub.eur.nl/pub/93018

Vandic, D., *Intelligent Information Systems for Web Product Search*, Promotors: Prof. U. Kaymak & Dr. Frasincar, EPS-2017-405-LIS, repub.eur.nl/pub/95490

Verbeek, R.W.M., *Essays on Empirical Asset Pricing*, Promotors: Prof. M.A. van Dijk & Dr. M. Szymanowska, EPS-2017-441-F&A, repub.eur.nl/pub/102977

Vermeer, W., *Propagation in Networks:The impact of information processing at the actor level on system-wide propagation dynamics*, Promotor: Prof. P.H.M.Vervest, EPS-2015-373-LIS, repub.eur.nl/pub/79325

Versluis, I., *Prevention of the Portion Size Effect*, Promotors: Prof. Ph.H.B.F. Franses & Dr. E.K. Papies, EPS-2016-382-MKT, repub.eur.nl/pub/79880

Vishwanathan, P., *Governing for Stakeholders: How Organizations May Create or Destroy Value for their Stakeholders*, Promotors: Prof. J. van Oosterhout & Prof. L.C.P.M. Meijs, EPS-2016-377-ORG, repub.eur.nl/pub/93016

Vlaming, R. de, *Linear Mixed Models in Statistical Genetics*, Promotors: Prof. A.R. Thurik, Prof. P.J.F. Groenen & Prof. Ph.D. Koellinger, EPS-2017-416-S&E, repub.eur.nl/pub/100428

Vries, H. de, *Evidence-Based Optimization in Humanitarian Logistics*, Promotors: Prof. A.P.M. Wagelmans & Prof. J.J. van de Klundert, EPS-2017-435-LIS, repub.eur.nl/pub/102771

Vries, J. de, *Behavioral Operations in Logistics*, Promotors: Prof. M.B.M de Koster & Prof. D.A. Stam, EPS-2015-374-LIS, repub.eur.nl/pub/79705

Wagenaar, J.C., *Practice Oriented Algorithmic Disruption Management in Passenger Railways*, Promotors: Prof. L.G. Kroon & Prof. A.P.M. Wagelmans, EPS-2016-390-LIS, repub.eur.nl/pub/93177

Wang, P., *Innovations, status, and networks*, Promotors: Prof. J.J.P. Jansen & Dr. V.J.A. van de Vrande, EPS-2016-381-S&E, repub.eur.nl/pub/93176

Wang, R., *Corporate Environmentalism in China*, Promotors: Prof. P.P.M.A.R Heugens & Dr. F. Wijen, EPS-2017-417-S&E, repub.eur.nl/pub/99987

Wasesa, M., *Agent-based inter-organizational systems in advanced logistics operations*, Promotors: Prof. H.W.G.M van Heck, Prof. R.A. Zuidwijk & Dr. A. W. Stam, EPS-2017-LIS-424, repub.eur.nl/pub/100527

Wessels, C., *Flexible Working Practices: How Employees Can Reap the Benefits for Engagement and Performance*, Promotors: Prof. H.W.G.M. van Heck, Prof. P.J. van Baalen & Prof. M.C. Schippers, EPS-2017-418-LIS, repub.eur.nl/pub/99312

Wiegmann, P.M., *Setting the Stage for Innovation: Balancing Diverse Interests through Standardisation*, Promotors: Prof. H.J. de Vries & Dr. K. Blind, EPS-2019-473-LIS, repub.eur.nl/pub/114519

Wijaya, H.R., *Praise the Lord! : Infusing Values and Emotions into Neo-Institutional Theory*, Promotors: Prof. P.P.M.A.R. Heugens & Prof. J.P. Cornelissen, EPS-2019-450-S&E, repub.eur.nl/pub/115973

Williams, A.N., *Make Our Planet Great Again: A Systems Perspective of Corporate Sustainability*, Promotors: Prof. G.M. Whiteman & Dr. S. Kennedy, EPS-2018-456-ORG, repub.eur.nl/pub/111032

Witte, C.T., *Bloody Business: Multinational investment in an increasingly conflict-afflicted world*, Promotors: Prof. H.P.G. Pennings, Prof. H.R. Commandeur & Dr. M.J. Burger, EPS-2018-443-S&E, repub.eur.nl/pub/104027

Ye, Q.C., *Multi-objective Optimization Methods for Allocation and Prediction*, Promotors: Prof. R. Dekker & Dr. Y. Zhang, EPS-2019-460-LIS, repub.eur.nl/pub/116462

Ypsilantis, P., *The Design, Planning and Execution of Sustainable Intermodal Port-hinterland Transport Networks*, Promotors: Prof. R.A. Zuidwijk & Prof. L.G. Kroon, EPS-2016-395-LIS, repub.eur.nl/pub/94375

Yuan, Y., *The Emergence of Team Creativity: a social network perspective*, Promotors: Prof. D. L. van Knippenberg & Dr. D. A. Stam, EPS-2017-434-ORG, repub.eur.nl/pub/100847

Yuferova, D., *Price Discovery, Liquidity Provision, and Low-Latency Trading*, Promotors: Prof. M.A. van Dijk & Dr. D.G.J. Bongaerts, EPS-2016-379-F&A, repub.eur.nl/pub/93017

Zhang, Q., *Financing and Regulatory Frictions in Mergers and Acquisitions*, Promotors: Prof. P.G.J. Roosenboom & Prof. A. de Jong, EPS-2018-428-F&A, repub.eur.nl/pub/103871

Zuber, F.B., *Looking at the Others: Studies on (un)ethical behavior and social relationships in organizations*, Promotor: Prof. S.P. Kaptein, EPS-2016-394-ORG, repub.eur.nl/pub/94388

Online retailing continues to grow, and consumers, but also small businesses, purchase more and more products online. Many of these products require attended delivery for which the customer needs to stay at home to receive their purchases. To decrease the chances of costly delivery failures and to provide customers with a high level of service, many online retailers offer their customers a menu of delivery time slots. The management of these time slots can be challenging, as the customer demand can vary heavily, and the amount of available delivery vehicles and drivers may be limited. Dynamic Time Slot Management (DTSM) is a class of methods which dynamically construct time slot offers based on previously placed customer orders. Our focus is the use of vehicle routing heuristics within DTSM to help retailers manage the availability of time slots in real time.

In this dissertation, we explore several challenges that hinder the widespread adoption of DTSM in practice. As vehicle routing is used in real time, the computation time plays a crucial role. We study pre-calculation techniques for a particular class of vehicle routing problems, and illustrate the trade-off between computation time and memory use. Furthermore, as multiple customers arrive and interact with the DTSM system simultaneously, several previously unstudied issues arise. We model such simultaneous interactions and study their impact on the real-time performance of the system in terms of response times and number of accepted customers. Finally, we explore a novel variant of DTSM in which routes and time slots are assigned a priori in a strategical phase to simplify their real-time management. Although this reduces the number of different time slots that can be offered to customers, advantages include smoothing of fulfillment center operations and delivery consistency.

ERiM

The Erasmus Research Institute of Management (ERIM) is the Research School (Onderzoekschool) in the field of management of the Erasmus University Rotterdam. The founding participants of ERIM are the Rotterdam School of Management (RSM), and the Erasmus School of Economics (ESE). ERIM was founded in 1999 and is officially accredited by the Royal Netherlands Academy of Arts and Sciences (KNAW). The research undertaken by ERIM is focused on the management of the firm in its environment, its intra- and interfirm relations, and its business processes in their interdependent connections.

The objective of ERIM is to carry out first rate research in management, and to offer an advanced doctoral programme in Research in Management. Within ERIM, over three hundred senior researchers and PhD candidates are active in the different research programmes. From a variety of academic backgrounds and expertises, the ERIM community is united in striving for excellence and working at the forefront of creating new business knowledge.

ERiM

ERIM PhD Series
Research in Management

Erasmus University Rotterdam (EUR)
Erasmus Research Institute of Management
Mandeville (T) Building
Burgemeester Oudlaan 50
3062 PA Rotterdam, The Netherlands

P.O. Box 1738
3000 DR Rotterdam, The Netherlands
T +31 10 408 1182
E info@erim.eur.nl
W www.erim.eur.nl