2014-3
Lukas Bach
PhD Thesis

Routing and Scheduling Problems - Optimization using Exact and Heuristic Methods

DEPARTMENT OF ECONOMICS AND BUSINESS
AARHUS UNIVERSITY • DENMARK

Routing and Scheduling Problems -Optimization using Exact and Heuristic Methods

by

Lukas Bach

A PhD dissertation submitted to the
School of Business and Social Sciences, Aarhus University,
in partial fulfilment of the PhD degree in Economics and Business

December 2013

Supervisor: Sanne Wøhlk

"If you fail to plan, you are planning to fail"

- Benjamin Franklin

Preface

This PhD dissertation has been prepared at the School of Business and Social Sciences, Aarhus University, in partial fulfilment of the PhD degree in Economics and Business. The work of this PhD has been carried out while affiliated with the (former) department of Business Studies and the department of Economics and Business.

This thesis not only concludes three years of PhD study, but also an eight year journey at the Aarhus School of Business and now Aarhus University as a bachelor, master, and PhD student. Looking back at the last eight years it has been an amazing time, both academically and socially. I would like to thank everyone who has been a part of this journey.

As Sir Winston Churchill said: *"This report, by its very length, defends itself against the risk of being read."*, he might be correct, but that should not deter you.

Acknowledgments

I would first and foremost like to thank my supervisor Sanne Wøhlk, who has been a constant, loyal, dedicated, and at times ferocious support throughout my PhD studies. I do not know many people who are as dedicated to their work as she. I owe her

gratitude for her many contributions towards my PhD studies. I am very fortunate to have had her as my supervisor.

I have had the pleasure to work with many interesting co-authors during my PhD studies. My first introduction to international academic work was in collaboration with Geir Hasle from SINTEF in Oslo. I would like to thank him for his significant contribution to my first published paper, which is also the first chapter of this thesis. I very much look forward to joining him at SINTEF in the coming year and hope for a good ongoing collaboration.

During my PhD study I received a travel scholarship from Aage and Ylva Nimbs foundation, for that I am very thankful. It facilitated some very interesting research visits abroad which would otherwise have been difficult.

For my first stay abroad, I would like to thank Michel Gendreau and CIRRELT for inviting me to a 6 month research visit in Montreal. The visit certainly opened my eyes to the academic world and gave me the opportunity to work with and meet many interesting and skilled people. I would also like to thank Michel Gendreau for taking the time to provide many good insights and for pointing my research in the right direction.

I have also had the pleasure to visit Dennis Huisman and Twan Dollevoet at the Econometric Institute at Erasmus University of Rotterdam. My thanks go to both of them for their collaboration and hospitality. Also, I would like to thank my friends at the Erasmus University for many good times after hours.

I would like to thank all the members of the CORAL group for many interesting discussions, courses, and seminars. It has been a pleasure knowing all of you. A special thanks goes to Christian Larsen for supervising my bachelor thesis and piquing my interest in Operations Research. I would also like to say special thanks to my co-author Jens Lysgaard for his good collaboration on the second chapter of this thesis.

I also owe a great thanks to the staff supporting me: Ingrid Lautrup, Karin Vinding,

Charlotte Sparrevohn, Christel Mortensen, Susanne Christensen, and Betina Sørensen. You have all been great colleagues in each of your capacities.

Thanks are also due to my family and friends who have put up with a lot and been an endless support during my PhD studies.

Last but not least it has been a real pleasure working among my great PhD colleagues. It might be a cliché to say that I would not have been able to do it without them, but it would certainly have been a very different journey. I would like to thank all my present and former colleagues who I shared both the C100 and C200 hallways for participating in creating a great social environment. I would like to thank my office mates Jeanne Andersen, Peter Bodnar, and Maria Elbek. For many interesting discussions, cakes, Friday beers, and endless trips to the coffee machine and mail room I would also like to thank: Tue Christensen, Sune Lauth Gadegaard, David Sloth Pedersen, Camilla Pisani, Lene Gilje Justesen, and Marie Herly. You have all meant much more to me than you know.

Lukas Bach

Århus, December 2013

Contents

Introduction

Structure of the Dissertation

This dissertation consists of four self-contained chapters each of which is presented as a journal paper. The chapters are thematically linked together two and two by their applications. Methodologically, a column generation approach is applied in three of the chapters. The first two chapters are applications of routing problems which are an element also present in the final two chapters.

The first two chapters consider the Mixed Capacitated General Routing Problem. The first lower bound tailored to the problem is presented in the first chapter, and in the second we further add to the literature by presenting an exact Branch-and-Cut-and-Price algorithm.

The third and fourth chapters both consider problems related to timetabling at a real world freight railway operator. The first of the two designs a timetable which is feasible with respect to fulfilling all demands, accessing infrastructure, and the usage of engine resources. The final chapter partially integrates the crew planning with the timetabling in order to adjust the timetable when planning the crew. This proves to give significant cost reductions.

The Mixed Capacitated General Routing Problem

The first part of the thesis considers the Mixed Capacitated General Routing Problem (MCGRP), also referred to as the Node, Edge, and Arc Routing Problem (NEARP). In the first chapter we refer to the problem as NEARP, and in the second chapter it is referred to as MCGRP. They are used interchangeably in this thesis. At the time of publishing the first chapter, the naming of the problem was ambiguous in the literature, but it now seems to have evolved towards MCGRP. The reference to NEARP is kept in the first chapter to keep consistency between the thesis and the published version of the chapter. The work on this part of the thesis has been carried out in collaboration with Geir Hasle and Sanne Wøhlk on the first chapter, and with Jens Lysgaard and Sanne Wøhlk on the second chapter.

The Mixed Capacitated General Routing Problem generalizes the classical Capacitated Vehicle Routing Problem (CVRP) and the Capacitated Arc Routing Problem (CARP). It captures important aspects of many real-life routing problems that are not adequately modeled by neither the CVRP nor the CARP. Applications of arc routing problems are street sweeping, gritting, and snow clearing. However, the arc routing model has been advocated in the literature for problems where the demand is located in nodes, for instance distribution of subscription newspapers to households and municipal pick-up of waste. In real life cases, there are often thousands of points to be serviced along a subset of all road links in the area. Such cases are often formulated as CARPs, typically with a large reduction of problem size. However, the reduction from points to arcs does not always provide an adequate model.

The MCGRP/NEARP is defined on a connected multi-graph with nodes, directed arcs, and undirected edges, all of which can be demanded. There are non-negative costs for traversing an edge/arc with or without servicing it. The demands must be serviced by a fleet of identical vehicles each with limited capacity. The vehicles are initially located in a special depot node. The fleet of vehicles can be bounded. The goal

is to identify a number of tours for the vehicles such that all demands are satisfied, while respecting the limited capacity and minimizing the cost of the tours.

The literature on the MCGRP was relatively scarce when the first paper was published. Hence new benchmark instances are introduced to complement the ones presented by Prins and Bouchenoua (2004). These test instances are used in both chapters on the MCGRP. In addition, new instances presented by Bosco et al. (2013) are treated in the second chapter.

Chapter 1: A Lower Bound for the Node, Edge, and Arc Routing Problem

To merge the arc/node routing problems and enable modeling of real-life situations makes good sense as we get a better representation of reality. But in spite of its importance only few studies of the NEARP are found in the literature after its introduction. The studies discuss NEARP solutions based on heuristics, providing upper bounds, but no lower bound is known. This inspires us to add to the NEARP literature by providing the first lower bound tailored to the NEARP in this chapter.

Lower bounds have been developed for many Vehicle Routing Problem (VRP) variants. Many of these are based on cutting planes. Also for the General Routing Problem, which is uncapacitated, there is a tradition of obtaining lower bounds through algorithms involving cutting planes. In contrast to VRP the tradition for CARP is to develop combinatorial lower bounds. The majority of these are based on the construction of one or several matchings. One of the best such lower bounds is the Multiple Cuts Node Duplication Lower Bound (MCNDLB), Wøhlk (2006). The algorithm we provide for the NEARP is a further development of the MCNDLB for the CARP.

When we started our work, only one set of benchmark instances existed for the NEARP: the CBMix instances by Prins and Bouchenoua (2004). All of these instances are based on graphs with a grid structure. To ensure more variation of the test platform for future algorithm developments and for testing the lower bound algorithm

described above, this work also contributes by developing two new sets of benchmark instances. The first set is called the BHW benchmark and is based on 20 classical CARP instances. Some edges have been transformed to directed arcs and node demands have been added. The second set, the DI-NEARP benchmark, contains 24 instances. It is based on six real life NEARP cases from the design of carrier routes for home delivery of subscription newspapers and other products in the Nordic countries.

For the existing CBMix instances, we have compared the best known upper bounds with our lower bound. Here the tests result in gaps between the LB and UB from 3.0 % to 39.9 %. For the BHW and DI-NEARP instances, the first upper bounds are provided by Hasle et al. (2012) and compared to the lower bound developed in this chapter.

Chapter 2: A Branch-and-Cut-and-Price Algorithm for the Mixed Capacitated General Routing Problem

In this chapter, we propose a Branch-and-Cut-and-Price (BCP) algorithm for obtaining optimal solutions for the Mixed Capacitated General Routing Problem and present computational results based on a number of standard benchmark instances.

New test instances are provided by Bosco et al. (2013), and they also provide the first integer programming formulation for the MCGRP. They extend valid inequalities for the CARP to the MCGRP, and embed them into a Branch-and-Cut algorithm that is tested on 12 sets of instances constructed from CARP benchmarks. The proposed method can solve only small-size instances and is very sensitive to the number of vehicles. Optimal solutions were also provided for two of the CBMix instances.

Due to the limitations with regard to the number of vehicles in the approach of Bosco et al. (2013), we introduce a BCP algorithm. This algorithm is better suited to handle larger numbers of vehicles available as we decompose the problem and are thus able to remove the vehicle dimension from the 3-index model proposed by Bosco et al. (2013).

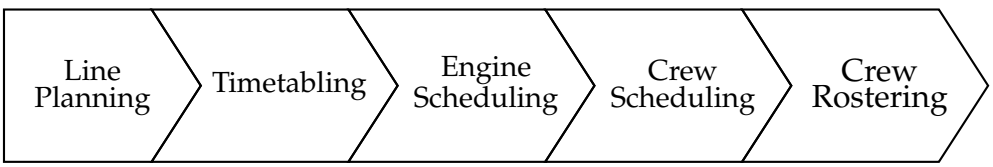
The BCP algorithm is tested and compared to the results reported by Bosco et al. (2013) and with best known upper and lower bounds for all instances. We provide new optimal solutions, tighten gaps, and generally decrease run times for the *mggdb* class of instances. Furthermore, we provide new upper bounds to a few CBMix instances and provide new best lower bounds in many cases for the CBMix and BHW instances as well.

Timetabling at a Freight Railway Operator

In the two final chapters we investigate to ich degree a Freight Railway Operator can benefit from integrating engine and crew planning when designing the timetable. We investigate the integration of a Train Timetabling Problem / Engine Scheduling Problem and a Crew Scheduling Problem (CSP), based on a case at a freight railway operator, DB Schenker Rail Scandinavia (DBSRS).

We consider the timetable design by selecting the time of service for demands among a set of discrete points of service within a time-window. The timetable is periodic at a weekly level, i.e., the same timetable is used every week, and it is used for a year.

The planning process at railway operators is described in various levels of detail by, among others, Huisman et al. (2005), Caprara et al. (2007), and Lusby et al. (2011). In general, the planning process can be described in five phases as follows;



In the Timetabling Problem we design a timetable for the desired lines from the Line Planning Problem and set the departure and arrival times. Also, access to the

infrastructure (railway network) is allocated to ensure a feasible timetable. This part is completed before we assign engines in the Engine Scheduling Problem to the lines in accordance with the timetable. We ensure that this allocation produces a feasible plan. The objective is often to minimize cost. In these chapters we aim to integrate engine routing and crew planning, respectively, in the timetabling phase.

The timetabling at a freight railway operator starts about a year before the timetable is to become effective. Much freight railway traffic is performed on a regular basis for industrial customers receiving or shipping goods on a regular basis. Hence most contracts are settled a long time in advance and are for weekly recurring services. At this tactical level of planning a lot of time is available to optimize the timetable. In general we consider run times of up to 24 hours as being acceptable. The models developed in these two chapters should also be considered as strategic tools for decision support. This could include decisions regarding the number of engines that should be available in the long run as well as where to station crew and/or decide on the closing of crew bases.

Chapter 3: Freight Railway Operator Timetabling and Engine Scheduling

This chapter has been written in a collaboration with Michel Gendreau and Sanne Wøhlk. We solve a timetabling problem at a freight railway operator. The objective of the model is to minimize cost of track usage and deadheading while servicing all demands within their time-windows, maintaining a feasible infrastructure usage, having full demand coverage, and adhering to the engine availability. The model is solved by a Branch-and-Price algorithm where feasible engine schedules are designed in a pricing problem by a labeling algorithm. The cost of using the infrastructure is time-dependent. The model provided in this chapter fixes service times in the timetable and produces a set of schedules for the engines available.

We show that it is possible to solve the timetabling problem while considering the

engines available and thus we ensure that the fixed timetable is not only feasible, but also optimal with respect to reducing the cost of using the engines.

The model is tested on instances based on a real life case at DBSRS. We are able to solve the problem in reasonable time for real life sized instances and much faster for smaller test instances. This model is used in the final chapter to provide input for the crew scheduling.

Chapter 4: Integrating Timetabling and Crew Scheduling at a Freight Railway Operator

This chapter has been written in collaboration with Dennis Huisman and Twan Dollevoet. As described above, Crew Scheduling is often a phase in a sequential railway planning approach. The timetable used has a great impact on crew scheduling, especially as to which tasks can be combined into a duty. Consequences for the crew scheduling phase are often not considered when designing the timetable. We therefore aim to investigate the extent to which crew scheduling consequences can be handled in the timetable design.

In this chapter we develop a model that allows the timetable to be further adjusted in the crew scheduling phase. We do this by allowing the time of service of the demands to be changed, while staying within modified time windows. These modified time windows ensure that the schedules for the engines provided in the previous chapter remain feasible.

We develop a Branch-and-Price algorithm for the partially Integrated Crew Scheduling Problem which is solved heuristically to get solutions within a reasonable time. We show that it is indeed possible to integrate crew scheduling with the earlier phases of the planning process. We also show that significant cost reductions can be achieved. This is shown on real life sized instances also used in the previous chapter.

Chapter 1

A Lower Bound for the Node, Edge, and Arc Routing Problem

History: *This chapter has been prepared in collaboration with Geir Hasle and Sanne Wøhlk. It has been published in Computers & Operations Research Volume 40(4), 2013. The chapter has been presented at: ODYSSEUS, 5th International Workshop on Freight Transportation and Logistics, May 21-25, 2012, Mykonos, Greece, and WARP 1, 1st Workshop on Arc Routing Problems, 22-24 May, 2013, Copenhagen, Denmark.*

A Lower Bound for the Node, Edge, and Arc Routing Problem

Lukas Bach[†], Geir Hasle[‡] and Sanne Wøhlk[†]

[†]Cluster for Operations Research And Logistics,
Department of Economics and Business,
Aarhus University, Denmark

[‡]Department of Applied Mathematics,
SINTEF ICT, Oslo, Norway

Abstract

The Node, Edge, and Arc Routing Problem (NEARP) was defined by Prins and Bouchenoua in 2004, although similar problems have been studied before. This problem, also called the Mixed Capacitated General Routing Problem (MCGRP), generalizes the classical Capacitated Vehicle Routing Problem (CVRP), the Capacitated Arc Routing Problem (CARP), and the General Routing Problem. It captures important aspects of real-life routing problems that were not adequately modeled in previous Vehicle Routing Problem (VRP) variants. The authors also proposed a memetic algorithm procedure and defined a set of test instances called the CBMix benchmark. The NEARP definition and investigation contribute to the development of rich VRPs. In this chapter we present the first lower bound procedure for the NEARP. It is a further development of lower bounds for the CARP. We also define two novel sets of test instances to complement the CBMix benchmark. The first is based on well-known CARP instances; the second consists of real life cases of newspaper delivery routing. We provide numerical results in the form of lower and best known upper bounds for all instances of all three benchmarks. For three

of the instances, the gap between the upper and lower bound is closed. The average gap is 25.1%. As the lower bound procedure is based on a high quality lower bound procedure for the CARP, and there has been limited work on approximate solution methods for the NEARP, we suspect that a main reason for the rather large gaps is the quality of the upper bound. This fact, and the high industrial relevance of the NEARP, should motivate more research on approximate and exact methods for this important problem.

Keywords: Vehicle Routing; Node Routing; Arc Routing; General Routing; VRP; CARP; NEARP; MCGRP; Bound; Benchmark; Experiment

1.1 Introduction

The Vehicle Routing Problem (VRP) captures the essence of allocation and routing of vehicles at minimal cost, given transportation demand. Hence, it is central to effective and efficient transportation management. VRP research is regarded as one of the great successes of Operations Research, partly due to the emergence of a solution tool industry. Results have been disseminated and exploited in industry. The VRP, construed in a wide sense, is a family of problems. Since the first definition of the classical, Capacitated VRP (CVRP) in Dantzig and Ramser (1959), many generalizations have been studied in a systematic fashion. Typically, exact and approximate solution methods have been proposed and investigated for each new VRP variant that has been defined. For an introduction and a survey of the VRP literature, we refer to Toth and Vigo (2001); Golden et al. (2010).

The VRP is a computationally very hard discrete optimization problem. For industrial cases of reasonable size, one normally has to resort to approximate methods.

Efficient procedures for generating proven lower bounds for the optimal value are important both to practice and theory. First, they may speed up exact methods. Second, they provide a benchmark for approximate methods that provide feasible solutions and hence upper bounds on the optimal value. Obviously, a zero gap between an upper and a lower bound for a given instance proves that the value is optimal. A large gap may be due to a poor quality lower bound, a feasible solution of bad quality, or both.

There has been a tremendous increase in the ability to produce exact and approximate solutions to VRP variants over the past 50 years. A few years ago, the best exact methods could consistently solve instances of the CVRP with up to some 70 customers to optimality in reasonable time. Today, the number is above 100, see for instance Baldacci et al. (2010). Approximate methods such as metaheuristics, matheuristics, and heuristic column generation seem to provide high quality solutions in realistic times even for large-size instances of complex VRP variants. For a categorized bibliography of metaheuristics for the VRP, we refer to Gendreau et al. (2008). Doerner and Schmid (2010) give a survey of matheuristics for VRPs. Feillet (2010) gives a tutorial on column generation for the VRP.

As problems are regarded as being solved for practical purposes, researchers turn to new extensions and larger-size instances. This trend is enhanced by market pull from the tool industry and their end users. The somewhat imprecise term "rich VRP" has recently been introduced to denote variants that are close to capturing all the essential aspects of some subset of real-life routing problems. Generalizations of models in the literature are defined, exact and approximate methods are proposed and investigated, and lower bounds are developed.

In contrast to the CVRP where demand for service is located in the nodes of the network, arc routing problems have been proposed to model the situation where demand is located on edges or arcs in a transportation network, see Dror (2000). Of partic-

ular industrial relevance is the Capacitated Arc Routing Problem (CARP) defined by Golden and Wong (1981) and its generalizations, as the CARP model contains multiple vehicles with capacity.

There has been a tendency in the literature to dichotomize routing problems into arc routing problems and node routing problems. Some cases are naturally modeled as arc routing because the demand is fundamentally defined on arcs or edges in a transportation network. Prime examples are street sweeping, gritting, and snow clearing. However, the arc routing model has been advocated in the literature for problems where the demand is located in nodes, for instance distribution of subscription newspapers to households and municipal pickup of waste, particularly in urban areas. In real-life cases, there are often thousands or tens of thousands of points to be serviced along a subset of all road links in the area. Such cases are often formulated as CARPs, typically with a drastic reduction of problem size.

Prins and Bouchenoua (2004) motivate and define the Node, Edge, and Arc Routing Problem (NEARP). They state that: *"Despite the success of metaheuristics for the VRP and the CARP, it is clear that these two problems cannot formalize the requirements of many real-world scenarios."* Their example is urban waste collection, where most demand may adequately be modeled on street segments, but there may also be demand located in points, for instance at supermarkets. Hence, they motivate a generalization of both the classical CVRP and the CARP. To this end, they define the NEARP as a combination of the CVRP and the CARP, which can also be viewed as a capacitated extension of the General Routing Problem (Orloff, 1974). They propose a memetic algorithm for the NEARP and investigate it empirically on standard CVRP and CARP instances from the literature. The authors also create a NEARP benchmark consisting of 23 grid-based test cases, the so-called CBMix-instances, and provide experimental results for their proposed algorithm.

We would like to enhance the motivation for the NEARP and further emphasize

its high importance to practice. The arc routing model for node-based demand cases such as subscription newspaper delivery is based on an underlying idea of abstraction. Some form of abstraction may be necessary to contain the computational complexity resulting from a large number of demand points in industrial routing. The assumption that all point-based demands can be aggregated into edges or arcs may be crude in practice. It may lead to solutions that are unnecessarily costly, as partial traversal of edges is not possible. In industry, a route planning task may cover areas that have a mixture of urban, suburban, and rural parts where many demand points will be far apart and aggregation would impose unnecessary constraints on visit sequences. A more sophisticated type of abstraction is aggregation of demand based on the underlying transportation network topology. Such aggregation procedures must also take capacity, time, and travel restrictions into consideration to avoid aggregation that would lead to impractical or low quality plans. In general, such procedures will produce a NEARP instance with a combination of demands on arcs, edges, and nodes. It is therefore imperative to eliminate the arc/node routing dichotomy and thus enable the modeling of the continuum of node and arc routing problems needed for representational adequacy in real-life situations. The introduction of the NEARP was a significant step towards the goal of rich VRP.

Despite its importance, studies of the NEARP are scarce in the literature. The first we know of is the paper by Pandit and Muralidharan (1995). They address a generalized version of the NEARP, i.e., routing a heterogeneous fixed fleet of vehicles over specified segments and nodes of a street network, and also include a route duration constraint. The problem is denoted the Capacitated General Routing Problem (CGRP). The authors formally define the CGRP and design a heuristic for solving it. They generate random test instances with inspiration from curb-side waste collection in residential areas on a network with 50 nodes and 100 arcs. They also investigate the proposed method on random instances of the Capacitated Chinese Postman Problem for which

they had two lower bound procedures.

In Gutiérrez et al. (2002), the homogeneous fleet specialization of the CGRP studied by Pandit and Muralidharan is investigated. They call the problem the Capacitated General Routing Problem on Mixed Graphs (CGRP-m) and propose a heuristic that compares favorably with the heuristic by Pandit and Muralidharan on the homogeneous fleet case.

Kokubugata et al. (2007) study problems from city logistics, including the VRP with Time Windows and the NEARP. They propose a Simulated Annealing metaheuristic for solving these problems. Computational results for the CBMix instances of Prins and Bouchenoua are presented, with several improvements. Hasle et al. (2012) describe results from experiments on NEARP test instances using their industrial VRP solver Spider (Hasle and Kloster, 2007; SINTEF, 2012), and report new best-known results.

The first integer programming formulation for the NEARP was developed by Bosco et al. (2013). They extended valid inequalities for the CARP to the NEARP, and embedded them into a branch-and-cut algorithm that was tested on 12 sets of instances constructed from CARP benchmarks. The proposed method could solve only small-size instances, involving at most seven vehicles. Optimal solutions were also provided for two of the smallest CBMix instances.

Lower bounds have been developed for many VRP variants. Many of these are based on cutting planes. See Bosco et al. (2013) and Lysgaard et al. (2004) for state-of-the-art lower bounds for the CVRP. Also for the General Routing Problem, there is a tradition of obtaining lower bounds through algorithms involving cutting planes. See Corberán et al. (2001), Corberán et al. (2006), and Corberán et al. (2007) for some of the best lower bound algorithms for this problem.

For the CARP, the academic tradition has been to develop combinatorial lower bounds. Such lower bounds are based on the theory from combinatorial optimization rather than on linear programming. The majority of these bounds are based on the

construction of one or several matchings. The best such lower bound is the Multiple Cuts Node Duplication Lower Bound (MCNDLB), Wøhlk (2006), with the extensions added in Ahr (2004). Good lower bounds based on other strategies are the Hierarchical Relaxations Lower Bound, Amberg and Voss (2002), and LP-based bounds, Belenguer and Benavent (2003); Martinelli et al. (2011). Recent exact algorithms using strong lower bounding procedures are found in Bartolini et al. (2013); Bode and Irnich (2012). See Ahr (2004) for an overview of CARP lower bounds and Wøhlk (2008) for a recent survey on CARP in general.

The main contribution of this chapter is to provide the first (to the best of our knowledge) lower bound procedure for the NEARP. This bound is inspired by the MCNDLB for CARP and its extensions. We also define two new sets of test instances that complement the grid-based CBMix instances of Prins and Bouchenoua. The first set is called the BHW benchmark. It is based on 20 well-known CARP instances from the literature. The second is called the DI-NEARP benchmark and consists of 24 instances defined from real cases of newspaper delivery routing. For all test instances, we provide numerical results in the form of lower and best known upper bound. The remainder of this chapter is organized as follows. In Section 2, we formally state the Node, Edge, and Arc Routing Problem and in Section 3, we describe our lower bound algorithm for the problem and argue its correctness. In Section 4, we present two new benchmarks for the NEARP, and in Section 5 we give computational results. Finally, in Section 6, we offer a summary, our concluding remarks, and future lines of work.

1.2 The Node, Edge, and Arc Routing Problem

The Node, Edge, and Arc Routing Problem (NEARP) is defined on a connected multigraph $G = (N, E, A)$, where N is the set of nodes, E is the set of undirected edges, and A is the set of directed arcs. Let c_e denote the non-negative traversal cost for

$e \in E \cup A$, also known as deadheading cost, i.e., the cost for traversing the edge/arc without servicing it. The traversal cost is zero for nodes. Let $N_R \subseteq N$ be the set of required nodes, and let q_i denote the demand and p_i the processing cost of node $i \in N_R$. Similarly, let E_R and A_R be the set of required edges and arcs, respectively, and let q_e and p_e denote the demand and processing cost of $e \in E_R \cup A_R$, respectively. The processing cost is the total cost that accrues when the required edge or arc is serviced. It is the sum of the traversal and servicing costs. To follow the convention of Prins and Bouchenoua (2004), we only report the total traversal cost of a solution. A fleet of identical vehicles each with capacity Q is initially located in a special depot node, here denoted node 1. It is assumed that the size of the fleet is unbounded.

The goal is to identify a number of tours for the vehicles such that 1) every node $i \in N_R$, every edge $e \in E_R$, and every arc $e \in A_R$ is serviced by exactly one vehicle, 2) the sum of demands serviced by each vehicle does not exceed Q, and 3) the total cost of the tours is minimized.

The total servicing cost for any feasible solution to a given NEARP is constant. Hence, we do not need to consider servicing costs in our lower bound procedure. Also, the convention for reporting results on the CBMix benchmark is such that the constant sum of servicing costs has been subtracted. We introduce the concepts of servicing and processing cost here to be compatible with Prins and Bouchenoua, and to prepare for extensions to the NEARP with temporal constraints. Moreover, some heuristics, such as greedy construction heuristics, may use servicing costs.

1.3 Lower Bound for NEARP

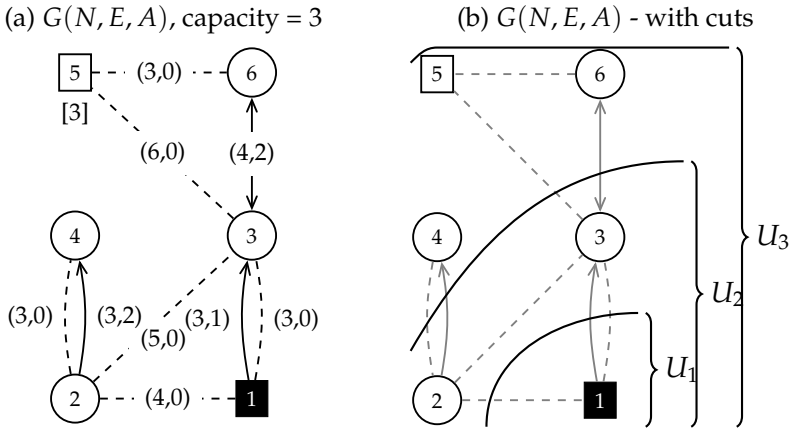
The algorithm is a further development of the Multiple Cuts Node Duplication Lower Bound (MCNDLB) for the CARP, Wøhlk (2006). We start by giving an intuitive description of the structure of the algorithm followed by a small example, and then provide a

formal description.

For notational reasons, in the description of the algorithm we will assume that the graph is simple, i.e., that there is at most one required link between any pair of nodes. For $e = (i, j) \in E \cup A$, we use the notation c_{ij}, q_{ij}, and p_{ij} to denote traversal cost, demand, and processing cost, respectively. The algorithm can, however, easily be extended to the non-simple case.

In the following we use $SPL(i, j)$ to denote the cost of a shortest path in G from i to j. Let $N' \subset N$ be a subset of the nodes. We define $\delta^-(N') = \{e = (i, j) \in E \cup A | i \in N \setminus N'$ and $j \in N'\}$ to be the set of links entering N' and $\delta^+(N') = \{e = (i, j) \in E \cup A | i \in N'$ and $j \in N \setminus N'\}$ to be the set of links leaving N'. Note that due to the existence of undirected edges, $\delta^-(N')$ and $\delta^+(N')$ are not necessarily disjoint. Finally, we define $\delta(N') = \delta^-(N') \cup \delta^+(N')$ to be the set of links connecting N' to the remaining graph. Finally, for any set of nodes, U, we use $G(U)$ to denote the graph induced by U.

Figure 1.1: NEARP example



Note: Node 1 is the depot node, a circle represents a node without demand, and a square is a node with demand, the quantity of the demand is given in brackets. The lines are dashed for non-demand edges/arcs and solid for those with demand. The direction of the demand arcs is given by the arrows. The deadheading cost and demand quantity are given in parenthesis for all edges/arcs

Starting with $U_1 = \{1\}$, we consider mutually disjoint cuts $(U_k, N \setminus U_k)$ such that

$U_1 \subset U_2 \subset \ldots \subset U_k \subset U_{k+1}$. For each such cut, U_k, the graph induced by $N \setminus U_k$ will consist of one or more connected components, $G'_s = (N'_s, E'_s, A'_s), s = 1, \ldots, t$, as illustrated in Figure 1.1(b). The number of vehicles needed to service the demand in G'_s, and the demand of links connecting G'_s to U_k can be estimated by $p_s = \lceil (\sum_{i \in N'_s} q_i + \sum_{(i,j) \in E'_s \cup A'_s \cup \delta(N'_s)} q_{ij}) / Q \rceil$, which is a simple lower bound for the bin-packing problem.

Ideally, each vehicle would service the demand of an edge or arc when entering G'_s and when leaving G'_s. When this is not the case, we say that the vehicle is using a deadheading link. Such links can be either links without demand or links with demand not currently being serviced. We estimate the number of deadheading links (entering arcs, leaving arcs, and undirected edges) needed for all vehicles to both enter and leave G'_s. With this, we can estimate the cost of servicing demand in G'_s and demand of links connecting G'_s to U_k by constructing a node duplicated network and letting m_s be the value of a minimum cost perfect matching in this network. We do this for all the connected components and hence, $L = \sum_{s=1}^{t} m_s$ estimates the cost of servicing everything outside $G(U_k)$.

To estimate the cost of servicing demand in $G(U_k)$, we use the minimum cost $\overline{c}^{\,u}_s$ of a link between U and each component, G'_s and multiply this by the number of deadheading links needed to connect the two: r^u_s. Iterating over all the mutually disjoint cuts and all the connected components of these, we can estimate the cost of servicing the demand in $G(U_k)$ as $L1 = \sum_{j=1}^{k-1} \sum_{s=1}^{t} \overline{c}^{\,u}_s r^u_s$. For each of these cuts, $L + L1$ is a lower bound on the cost, and the algorithm selects the cut with the highest value. Note that in the algorithm, the calculations become more complex than outlined above due to the existence of both directed and undirected links.

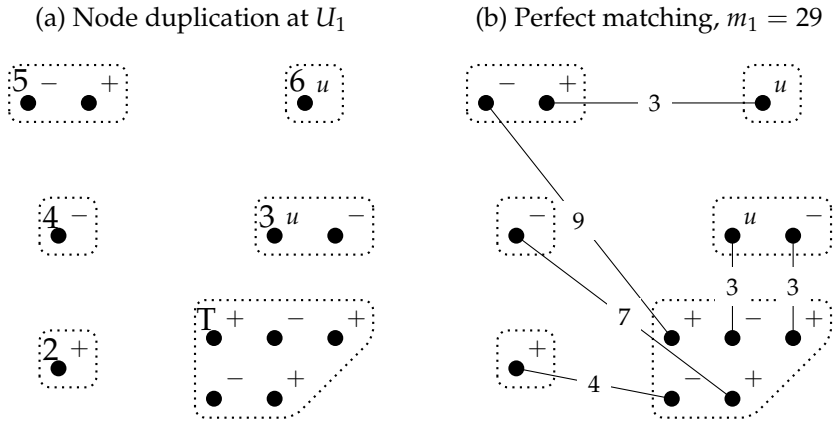
1.3.1 Lower Bound - Example

In this section we provide an example of the algorithm applied to a simple NEARP instance, presented in Figure 1.1. The algorithm performs a main iteration for each

cut, see Figure 1.1(b). Thus we start with U_1 in the first iteration, then U_2. We do not consider U_3 because it is the full graph, which is also the stopping criterion.

With $U_1 = \{1\}$ there is only one connected component. The node duplicated network is shown in Figure 1.2. It is constructed by making a copy of a node every time it is incident to a demand edge/arc. If the demand is outgoing, it gets a positive polarity; if it is incoming, it gets a negative polarity; and if it is an edge, it gets a neutral charge. The demand $2 \rightarrow 4$ results in a copy of node 2 with a positive charge and a copy of node 4 with a negative charge. If a node has a demand but is not incident to any nodes, e.g., node 5, two copies of that node are made, i.e., one positive and one negative.

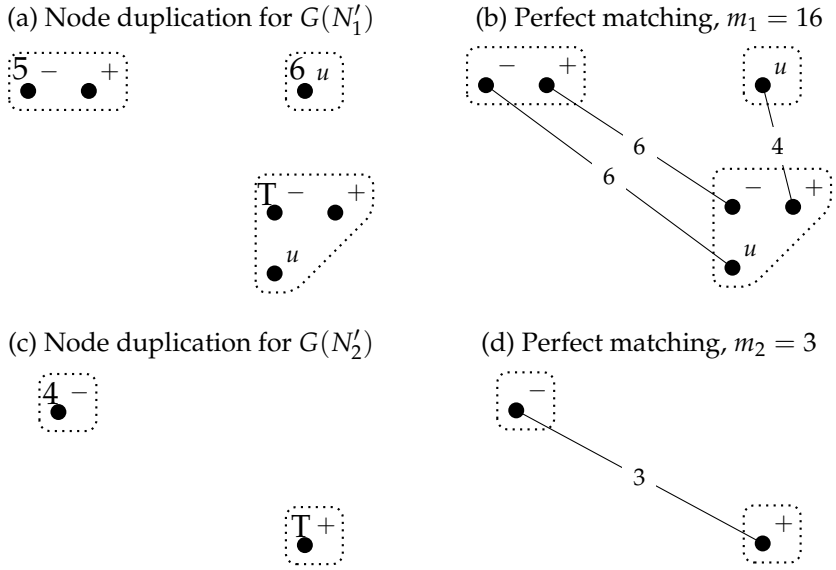
Figure 1.2: NEARP - Node duplicated network (U_1)



The set T in the node duplicated network represents copies of the set U, the number of nodes herein is determined by p_s. For each vehicle we must enter and leave the area T once, thus we add a negative and a positive charged node. For each demand entering T from the outside we remove a positive node and vice versa if we leave T. If this is an edge we turn a positive node into a neutral node and delete a negative node.

Based on the node duplicated network illustrated in Figure 1.2(a), we make a complete graph and add cost to each edge. In short, the cost is the shortest path given by Figure 1.1, with some exceptions, among others that a negative node cannot be

Figure 1.3: NEARP - Node duplicated network from U_2



matched to another negative node, the same applies to the positive nodes, whereas neutral nodes can be matched to other neutrals, positives, or negatives. A node cannot be matched to the node opposite the demand that created the node. Nodes in T cannot be matched to each other.

We now perform a minimum cost perfect matching. The result is illustrated in Figure 1.2(b). The cost of the matching is $L = m_1 = 29$. From Figure 1.1 we know that servicing the required arcs/edges will cost us 10. We can then calculate the first iteration in the lower bound; $L2 = 29 + 10 = 39$. Finally, we update $L1$ for the use in the next iteration as $L1 = \overline{c}_1^- \cdot r_1^- + \overline{c}_1^+ \cdot r_1^+ + \overline{c}_1^u \cdot r_1^u = 3 \cdot 2 + 3 \cdot 3 + 3 \cdot 0 = 15$.

In the next iteration, we apply the second cut of Figure 1.1 (b) and we now use U_2 represented by the set T. $G(N \setminus U_2)$ consists of two components with $N_1' = \{5, 6\}$ and $N_2' = \{4\}$. The node duplicated networks for $G(N_1)$ and for $G(N_2)$ are shown in Figures 1.3(a) and 1.3(c), respectively. The corresponding minimum cost perfect matchings are shown in Figures 1.3(b) and 1.3(d), respectively. The cost of the matchings is $L = m_1 + m_2 = 16 + 3 = 19$. Thus the lower bound of the second iteration is

$L2 = \max\{39, 19 + 15 + 10\} = 44$.

As $U_3 = N$ we terminate the algorithm and return the result from the second iteration (44) as the best lower bound. The optimal solution to the NEARP instance in Figure 1.1 is 46 which gives a gap of $\frac{46-44}{0.5\cdot(44+46)} = 4.44\%$.

1.3.2 Lower Bound - Algorithm

A detailed pseudo-code for the Multiple Cuts Node Duplicated Lower Bound for the NEARP is given in Algorithm 1. The main part of the algorithm lies in the construction of the matching network $G_s^M(N_s^M, E_s^M)$ in line 22. Algorithm 2 performs this task.

We let the node set N_s^M consist of three disjoint sets, S, T, and X, where S consists of copies of nodes from N_s', T consists of copies of nodes in U, and X can be considered to be extra copies of nodes in U and will be added later.

In Algorithm 2 line 2, we first consider the degree information of nodes $i \in N$: Let $D^-(R,i)$ be the number of required arcs entering node i, $D^+(R,i)$ the number of required arcs leaving node i, let $D^u(R,i)$ be the number of required edges incident to node i, and let $D(R,i)$ be the total number of required edges and arcs incident to node i. In Algorithm 2 line 8, for each node i in N_s', we add $D(R,i)$ nodes to S and call these nodes the family of i, denoted by $\chi(i)$. We say that the nodes in $\chi(i)$ are copies of i. Given a node j in $\chi(i)$, we refer to i as the origin of j, denoted by $\omega(j)$.

In Algorithm 2 line 15 we call Algorithm 3, which partitions S into three disjoint subsets γ^-, γ^+ and γ^u. For each node i in N_s', we consider the family $\chi(i)$ consisting of $D(R,i) = D^-(R,i) + D^+(R,i) + D^u(R,i)$ nodes. Of these, we associate $D^-(R,i)$ nodes with γ^-, $D^+(R,i)$ with γ^+, and $D^u(R,i)$ with γ^u.

In line 17, we consider the nodes in $N_s' \cap N_R$ for which $D(R,i) = 0$, i.e., required nodes without incident required arcs or edges. Note that these nodes were not considered above. For each such node, i, we add two nodes to S and call these the family of i, denoted by $\chi(i)$. We add one of the nodes to γ^- and the other to γ^+. As above, we call

Algorithm 1 MCNDLB algorithm

initialize $U = \{1\},\ L = 0,\ L1 = 0,\ L2 = 0$
while $U \neq N$ **do**
  $N' = N \setminus U$
  $G'(N')$ *//G' is the graph induced by N'*
  $t =$ number of connected components of G'
  $G'_s = (N'_s, E'_s, A'_s),\ 1 \leq s \leq t$ *//Each component is denoted by G'_s*
  for $s = 1$ to t **do**
    //Number of vehicles needed to service the demand of nodes, edges, and arcs in G'_s and $\delta(N'_s)$
    $p_s = \lceil (\sum_{i \in N'_s} q_i + \sum_{(i,j) \in E'_s \cup A'_s \cup \delta(N'_s)} q_{ij})/Q \rceil$
    //Number of required edges and arcs in cutset
    $\psi_s^u = |\{(i,j) \in \delta(N'_s) \cap E_R\}|$
    $\psi_s^- = |\{(i,j) \in \delta^-(N'_s) \cap A_R\}|$
    $\psi_s^+ = |\{(i,j) \in \delta^+(N'_s) \cap A_R\}|$
    //Number of deadheading edges and arcs needed in cutset
    $r_s^- = \max\{0, p_s - (\psi_s^- + \psi_s^u)\}$
    $r_s^+ = \max\{0, p_s - (\psi_s^+ + \psi_s^u)\}$
    $r_s^u = \max\{0, 2p_s - (\psi_s^u + \psi_s^- + \psi_s^+ + r_s^- + r_s^+)\}$
    //Minimum cost of edges and arcs in cutset
    $\overline{c}_s^{\,-} = \min_{(i,j) \in \delta^-(N'_s)} c_{ij}$
    $\overline{c}_s^{\,+} = \min_{(i,j) \in \delta^+(N'_s)} c_{ij}$
    $\overline{c}_s^{\,u} = \min_{(i,j) \in \delta(N'_s)} c_{ij}$
    $G_s^M =$ constructNodeDuplicatedNetwork(G, G'_s) *//See Algorithm 2*
    $m_s =$ value of minimum cost perfect matching in G_s^M
  end for
  $L = \sum_{s=1}^{t} m_s$
  $L2 = \max\{L2\,,\ L + L1 + \sum_{(i,j) \in E_R \cup A_R} c_{ij}\}$
  $L1 = L1 + \sum_{s=1}^{t} (r_s^u \cdot \overline{c}_s^{\,u} + r_s^+ \cdot \overline{c}_s^{\,+} + r_s^- \cdot \overline{c}_s^{\,-})$
  $U' = \{i \in N : i$ is adjacent to a vertex in $U\}$
  $U = U \cup U'$
end while
return $L2$

Algorithm 2 constructNodeDuplicatedNetwork(G, G'_s)

initialize $S = \emptyset$, $T = \emptyset$, and $X = \emptyset$
for all $i \in N$ **do**
  $D^-(R,i) = |\{(j,i) \in A_R\}|$ //*Required arcs entering node i*
  $D^+(R,i) = |\{(i,j) \in A_R\}|$ //*Required arcs leaving node i*
  $D^u(R,i) = |\{(i,j) \in E_R\}|$ //*Required edges incident to node i*
  $D(R,i) = D^-(R,i) + D^+(R,i) + D^u(R,i)$
end for
for all $i \in N'_s$ **do** //*Populate S*
  for $n = 1$ to $D(R,i)$ **do**
    Add new node j to G_s^M
    $S = S \cup j$
    $\chi(i) = \chi(i) \cup j$ //*These nodes are the family of i*
    $\omega(j) = i$ //*i is the origin node of j, denoted by* $\omega(j)$
  end for
  assignDirectionS($\gamma^-, \gamma^+, \gamma^u, D^-(R,i), D^+(R,i), D^u(R,i)$)
end for
for all $i \in N'_s \mid D(R,i) = 0$ **do** //*Required nodes not incident to required arcs or edges*
  Add new nodes j and k to G_s^M
  $S = S \cup j \cup k$
  $\chi(i) = \chi(i) \cup j \cup k$ //*These nodes are the family of i*
  $\omega(j) = i, \omega(k) = i$
  $\gamma^- = \gamma^- \cup j, \gamma^+ = \gamma^+ \cup k$
end for
for $n = 1$ to $2p_s - |\delta(N'_s) \cap \{E_R \cup A_R\}|$ **do** //*Populate T*
  Add new node j to G_s^M
  $T = T \cup j$
end for
assignDirectionT($\tau^-, \tau^+, \tau^u, r_s^-, r_s^+, r_s^u$)
for $n = 1$ to cardinalityX(S, T) **do** //*Populate X*
  Add new node j to G_s^M
  $X = X \cup j$
end for
Make G_s^M a complete undirected graph
assignDemands($G_s^M, G'_s, G, \chi(), q, \gamma^-, \gamma^+, \gamma^u$)
for all $(i,j) \in E_s^M$ **do** //*Set cost of edges*
  Set c_{ij} //*According to equation 1.1*
end for
return G_s^M

Algorithm 3 assignDirectionS($\gamma^-, \gamma^+, \gamma^u, D^-(R,i), D^+(R,i), D^u(R,i)$)

1: **for** $n = 1$ to $D^-(R,i)$ **do**
2: $\quad \gamma^- = \gamma^- \cup j \mid j \in \chi(i) \setminus (\gamma^-)$
3: **end for**
4: **for** $n = 1$ to $D^+(R,i)$ **do**
5: $\quad \gamma^+ = \gamma^+ \cup j \mid j \in \chi(i) \setminus (\gamma^- \cup \gamma^+)$
6: **end for**
7: **for** $n = 1$ to $D^u(R,i)$ **do**
8: $\quad \gamma^u = \gamma^u \cup j \mid j \in \chi(i) \setminus (\gamma^- \cup \gamma^+ \cup \gamma^u)$
9: **end for**

Algorithm 4 assignDirectionT($\tau^-, \tau^+, \tau^u, r_s^-, r_s^+, r_s^u$)

1: **for** $n = 1$ to r_s^- **do**
2: $\quad \tau^- = \tau^- \cup j \mid j \in T \setminus (\tau^-)$
3: **end for**
4: **for** $n = 1$ to r_s^+ **do**
5: $\quad \tau^+ = \tau^+ \cup j \mid j \in T \setminus (\tau^- \cup \tau^+)$
6: **end for**
7: **for** $n = 1$ to r_s^u **do**
8: $\quad \tau^u = \tau^u \cup j \mid j \in T \setminus (\tau^- \cup \tau^+ \cup \tau^u)$
9: **end for**

these nodes copies of i and for a node j in $\chi(i)$, we say that i is the origin of j, denoted by $\omega(j)$.

The set T consists of $2p_s - |\delta(N_s') \cap \{E_R \cup A_R\}|$ nodes, which can be considered copies of nodes in U. Because we know the minimum number of deadheading edges needed in U, we can partition T into three disjoint subsets τ^-, τ^+ and τ^u, where the values of r_s^-, r_s^+, and r_s^u determine the number of nodes in each of the three subsets, respectively. This is handled in lines 24 through 27.

To finalize the construction of G_s^M, let the number of nodes in X be determined by Algorithm 6, which is called from Algorithm 2 in line 29. X can be considered to be extra copies of nodes in U. Nodes in X are not connected to nodes in T. There are now enough nodes in $T \cup X$ for every node in S to be matched to one of these at cost $m^-(i)$ or $m^+(i)$. X is necessary because for any two nodes, i and j in S, it might

Algorithm 5 assignDemands($G_s^M, G_s', G, \chi(), q, \gamma^-, \gamma^+, \gamma^u$)

1: **for all** $(i,j) \in E_s' \cap E_R \mid i,j \in S$ **do** *//Assign demand of required edges*
2: $q_{kl} = q_{ij} \mid k \in \chi(i) \cap \gamma^u$ **and** $l \in \chi(j) \cap \gamma^u$ *//Each k or l can only be assigned 1 demand*
3: **end for**
4: **for all** $(i \to j) \in A_s' \cap A_R$ **do** *//Assign demand of required arcs*
5: $q_{kl} = q_{ij} \mid k \in \chi(i) \cap \gamma^+$ **and** $l \in \chi(j) \cap \gamma^-$ *//Each k or l can only be assigned 1 demand*
6: **end for**
7: **for all** $i \in N_s' \cap N_R \mid D(R,i) = 0$ **do** *//Assign demand of required nodes*
8: $q_{kl} = q_i \mid k \in \chi(i) \cap \gamma^+$ **and** $l \in \chi(i) \cap \gamma^-$ *//Each k or l can only be assigned 1 demand*
9: **end for**

Algorithm 6 cardinalityX(S, T)

1: **if** $|S| - |T| > 0$ **then**
2: $x = |S| - |T|$
3: **else if** $|S| - |T| \bmod 2 <> 0$ **then**
4: $x = 1$
5: **else**
6: $x = 0$
7: **end if**
8: **return** x

be cheaper to match both i and j to something in U instead of matching them to each other, illustrating the vehicle driving back to subgraph U and then returning to S. Note that although the triangle inequality may not apply in the original graph, it does apply in this matching network as long as no edges with cost infinity are involved.

The demand of required arcs and edges in G_s' is assigned to edges in G_s^M as explained in Algorithm 5 which is called from Algorithm 2 in line 34. These assignments are done in such a way that no node in N_s^M is chosen more than once and no demand in G_s' is assigned more than once. All other edges in E_s^M have zero demand.

Consider the s'th component, represented by the graph $G_s' = (N_s', E_s', A_s')$. For every node i in N_s' set $m^-(i) = \min_{u \in U} SPL(u,i)$ and similarly $m^+(i) = \min_{u \in U} SPL(i,u)$,

i.e., $m^-(i)$ and $m^+(i)$ are the lengths of a shortest path from any node in U to i and from i to any node in U, respectively. In lines 35 through 37, the costs of edges (i,j) in E_s^M are set by c_{ij} in equation 1.1.

$$c_{ij} = \begin{cases} \infty & \text{if } q_{ij} > 0 \\ \\ 0 & \text{if } i,j \in S \text{ and } \omega(i) = \omega(j) \text{ and } i \in \gamma^- \text{ and } j \notin \gamma^- \\ 0 & \text{if } i,j \in S \text{ and } \omega(i) = \omega(j) \text{ and } i \in \gamma^+ \text{ and } j \notin \gamma^+ \\ 0 & \text{if } i,j \in S \text{ and } \omega(i) = \omega(j) \text{ and } i \in \gamma^u \\ \\ \infty & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^- \text{ and } j \in \gamma^- \\ SPL(i,j) & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^- \text{ and } j \notin \gamma^- \\ \infty & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^+ \text{ and } j \in \gamma^+ \\ SPL(j,i) & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^+ \text{ and } j \notin \gamma^+ \\ SPL(j,i) & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^u \text{ and } j \in \gamma^- \\ SPL(i,j) & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^u \text{ and } j \in \gamma^+ \\ \min\{SPL(i,j), SPL(j,i)\} & \text{if } i,j \in S \text{ and } \omega(i) \neq \omega(j) \text{ and } i \in \gamma^u \text{ and } j \in \gamma^u \\ \\ \infty & \text{if } i,j \in T \\ \infty & \text{if } i \in S \cap \gamma^+ \text{ and } j \in T \cap \tau^+ \\ m^-(i) & \text{if } i \in S \cap \gamma^+ \text{ and } j \in T \setminus \tau^+ \\ \infty & \text{if } i \in S \cap \gamma^- \text{ and } j \in T \cap \tau^- \\ m^+(i) & \text{if } i \in S \cap \gamma^- \text{ and } j \in T \setminus \tau^- \\ \min\{m^-(i), m^+(i)\} & \text{if } i \in S \cap \gamma^u \text{ and } j \in T \\ \infty & \text{if } i \in T \cap \tau^+ \text{ and } j \in S \cap \gamma^+ \\ m^-(j) & \text{if } i \in T \setminus \tau^+ \text{ and } j \in S \cap \gamma^+ \\ \infty & \text{if } i \in T \cap \tau^- \text{ and } j \in S \cap \gamma^- \\ m^+(j) & \text{if } i \in T \setminus \tau^- \text{ and } j \in S \cap \gamma^- \\ \min\{m^-(j), m^+(j)\} & \text{if } i \in T \text{ and } j \in S \cap \gamma^u \\ \\ 0 & \text{if } i,j \in X \\ m^-(i) & \text{if } i \in S \cap \gamma^+ \text{ and } j \in X \\ m^+(i) & \text{if } i \in S \cap \gamma^- \text{ and } j \in X \\ \min\{m^-(i), m^+(i)\} & \text{if } i \in S \cap \gamma^u \text{ and } j \in X \\ \infty & \text{if } i \in T \text{ and } j \in X \\ \infty & \text{if } i \in X \text{ and } j \in T \end{cases} \tag{1.1}$$

In order to tighten the bound, consider every pair of demand edges, (i,j) and (k,l) in E_s^M. If $q_{ij} + q_{kl} > Q$, we set the cost of the edges $(i,k), (i,l), (j,k)$, and (j,l) to ∞, since (i,j) and (k,l) cannot be serviced on the same tour. For the sake of simplicity, the example given in Section 1.3.1 does not include this part.

1.3.3 Correctness of the Lower Bound

Since the bound is an extension of the MCNDLB for the CARP, which was proven to be valid in Wøhlk (2006), we focus on the changes that are made to the original algorithm.

The first change occurs in the calculation of p_s, i.e., the number of vehicles needed to service component s and the links connecting it to U in Algorithm 1 line 9. In the original algorithm the demand was summed over all demand edges. Because, in the NEARP, we have both required edges, arcs, and nodes, clearly the summation should be over all of these.

In Algorithm 1 lines 11 through 13, we calculate the required number of deadheading links. In the original algorithm, this was calculated as $r_s = \max\{0\,,\, 2p_s - q_s\}$. Needing at least p_s vehicles, each of which must both enter and leave the component, and having r_s required edges in the cut, this is clearly correct. For the NEARP, we first consider entering vehicles. Note that we must have at least p_s of these.

We have ψ_s^- entering arcs and ψ_s^u edges in the cut. Hence, up to $\psi_s^- + \psi_s^u$ vehicles can use these existing links and we need to construct $\max\{0\,,\, p_s - (\psi_s^- + \psi_s^u)\}$ deadheading entering arcs. The argumentation for arcs leaving the component is symmetrical. Needing at least $2p_s$ links in total, we now add the number of undirected edges corresponding to the difference between $2p_s$ and the sum of all required links (arcs and edges) and the number of deadheading arcs added to the network. Thereby the correctness of lines 11 through 13 has been shown. With these definitions in place, it follows directly that the estimate for servicing everything inside U, $L1$, in Algorithm 1 lines 15 through 17 is correctly generalized to the NEARP.

It only remains to show that the construction of the matching network in Algorithm 2 leads to a valid estimate for servicing G'_s and the cutset. The structure of the matching network is similar to the one in the original bound. For each original node, we add $D(R, i)$ nodes to S in the matching network. This is exactly the number of times we must either enter or leave the node due to arc and edge requirements. Clearly, we may

partition these into nodes that represent entering, leaving, and undirected demand. We use the same number of nodes in the sets T and X as in the original algorithm, but again, for the set T, we can partition the nodes into sets based on the knowledge described above. For required nodes with no adjacent required links, it is clearly legal to add two nodes to S - one for entering and one for leaving.

The assignment of required edges is done precisely as in the original algorithm, except that now we take the direction of arcs into account when selecting the nodes in each family to be matched. Furthermore, for required nodes we legally select the edge between the two copies of the original node to absorb the demand. As in the original algorithm, the cost of all these demand-assigned edges is set to infinity to prevent them from being used in the matching.

The remaining cost structure is far more complex in this algorithm than in the original one. This is due to the partitioning of families into entering, leaving, and undirected sets. When two nodes are in the same family, the cost of the edge connecting them is zero if it is possible to enter through one of the copies and leave through the other. Otherwise, the cost is set to infinity to prevent this connection from being used in the matching. For two nodes in different families, the cost is also infinite if either both nodes are entering nodes or both are leaving nodes, as this combination is illegal. If the combination is legal, the cost between such nodes corresponds to the cost of a shortest path between the origins of the nodes, while taking possibility of directions into account.

When considering a node i in S and a node j in T or in X, we use the different estimates of $m(i)$ as in the original algorithm, except that again, we need to take into account the different combinations of entering and leaving, making the expression less pretty. Connections internal to T and X are handled as in the original algorithm. As can be concluded from the above argumentation, the algorithm presented in this chapter indeed yields a feasible lower bound for the NEARP.

1.3.4 Lower Bound - Extensions

To strengthen the quality of the bound, for each of the nodes of U' on line 28, the node is added to U and we skip back to line 2. When we reach line 28 again the node is once more removed from U before the next node is added and we repeat the procedure. We only move to line 29 after all nodes of U' have been examined. This can strengthen the quality of the bound in that part of the matching. This procedure has been proven valid for similar lower bound procedures in Ahr (2004); Breslin and Keane (1997). When testing the addition of nodes from U' to U, the number of nodes added jointly as well as their combination influence the quality of the bound. Unfortunately, the best number and best combination cannot easily be predicted beforehand. We have chosen to add the nodes individually.

1.4 New NEARP Benchmarks

Only one set of test instances exists for the NEARP: the CBMix instances in Prins and Bouchenoua (2004). These instances are all based on graphs with a grid structure. To ensure more diversity in the test platform for future algorithm developments and for testing the lower bound algorithm described above, we present two new benchmarks. The first, called BHW, is based on classical CARP instances from the literature. The second, called DI-NEARP, is based on real-life instances of an industrial application. We adopt the convention for the CBMix benchmark, i.e., only the total traversal cost of a solution is reported. The instance definition files and numerical results for all three benchmarks are found at SINTEF's NEARP website (SINTEF, 2013).

1.4.1 The BHW Instances

This benchmark is generated from benchmark instances for the CARP. More specifically, we have used a selection of instances from the Gdb (Baker et al., 1983), Val (Be-

navent et al., 1992), and Egl (Eglese and Li, 1996) benchmarks.

For each instance, we have kept the underlying graph structure, the existing demand, and the vehicle capacity. We have made the following modifications to the instances: Some undirected edges have been replaced by directed arcs. If the edge was required, the demand is transmitted to the arc. Other undirected edges have been replaced by two directed arcs, one in each direction. If the edge was required, the demand is either transferred to one of the arcs or both arcs have been made required, each with a demand equal to the demand of the edge. Finally, some edges have been left unchanged. Furthermore, some of the nodes are made required.

Table 1.1 gives the most important properties for each instance. The first column states the name of the instance and the second provides a reference to the underlying CARP instance. The next three columns give the total number of nodes, undirected edges, and directed arcs in the graph. The following three columns give the same information for required entities. The next column states the vehicle capacity. Note that the vehicles are assumed to be identical. The remaining six columns provide statistical information regarding required entities. Pairwise, these columns provide the mean and standard deviation of the demand of the required nodes, edges, and arcs in the graph. Note that only the required entities are included in these calculations. All instances have relatively sparse networks, as they simulate real-life situations. The depot is located in node 1 in all BHW instances.

1.4.2 The DI-NEARP Instances

This benchmark is defined from six real-life cases from the design of carrier routes for home delivery of subscription newspapers and other media products in the Nordic countries. The company Distribution Innovation AS (DI) (Distribution Innovation, 2012) operates a web-based solution for design, revision, management, and control of carrier routes. Route design and revision are based on electronic road and address

data provided by commercial GIS vendors. Sophisticated models for travel and service time are utilized. The Spider VRP solver provided by SINTEF (SINTEF, 2012) is integrated in the solution.

The GIS road network data may have been improved by the user through manual editing due to errors or lack of detail. All delivery points are geocoded, and the enhanced road network data are transformed into an internal graph representation in Spider. The basic node routing problem cases typically have a large number of points. Through road topology based aggregation heuristics in Spider, the original problem has been transformed to a NEARP with side constraints. The graph topology of the instances is taken directly from the Spider graph.

Data for the six instances was retrieved from the DI web server in 2011. These particular cases only have required edges and nodes, but no required arcs. The edges have symmetrical travel costs. The travel and service costs are set to the travel and service times calculated by the models in the DI system, as there is a close correlation between total route duration and the actual cost of the route plan. The index of the depot node is given explicitly.

In the industrial case, vehicle capacity is not constraining, but there is a constraint on route duration. We have transformed the duration constraint to a capacity constraint and selected four different values for capacity that produce a reasonable range for the number of routes, including the actual number from the real application. Hence, the DI-NEARP benchmark consists of 24 instances. They are named DI-NEARP-n*r*-Q*q*k, where r is the total number of required nodes, edges, and arcs and q is capacity in thousands. Table 1.2 gives the most important properties for each instance. The structure of this table is similar to that of Table 1.1 except for the second column which is not relevant for the DI-NEARP instances.

1.5 Computational Results

We have implemented the lower bound algorithm in two versions: one version where all nodes neighboring U are added at once, and one version where the addition of each node is tested separately before all nodes are added, as explained in Section 1.3.4. In this section, we report our experimental results for both implementations, while referring to the latter as AD1. All lower bound calculations are performed on a PC with an Intel Core 2 Duo CPU, running at 2.53 GHz and with 2GB of RAM.

The results obtained for the three benchmark sets are given in Tables 1.3, 1.4, and 1.5. In each table, the second column provides the best known upper bound for the instance, hereafter referred to as B^U. For the CBMix instances these are obtained from Prins and Bouchenoua (2004), Kokubugata et al. (2007), and Hasle et al. (2012). For the BHW and the DI-NEARP instances, the first upper bounds were obtained with the Spider solver (Hasle et al., 2012). Remember that the cost reported is the total traversal cost.

For each of the two lower bound versions, we give the obtained value of the lower bound algorithm B^L, and the relative optimality gap G^O, i.e., the percentage gap from the best known upper bound to the lower bound, as calculated by the following formula:

$$G^O = \frac{B^U - B^L}{(B^U + B^L)/2} 100$$

Finally, we provide the running time of the lower bound algorithm in seconds. We imposed a time limit of 10,000 seconds. For the large-size DI-NEARP instances, the calculation of the AD1 lower bound was not completed within this time limit. Hence, the AD1 column has been omitted in Table 1.5.

For the CBMix benchmark, the gaps vary between 0.0% and 39.7% with an average of 23.1%. The variation is larger for the BHW benchmark, where the minimum, maximum, and average gaps are 0.0%, 54.0%, and 24.2%, respectively. The average gap for

the large-size DI-NEARP instances is 27.8%, with a minimum of 7.0% and a maximum of 54.8%.

Optimal solutions for CBMix12 and CBMix23 were reported in Bosco et al. (2013). For three BHW instances: BHW2, BHW4, and BHW6, the lower bound proves optimality of the best known solutions. For all other instances, the optimal solution is unknown. It is therefore not possible to determine if the size of the gap is mainly explained by the quality of the lower bound or by the quality of the best known solutions. It is, however, noted that for the CARP, the corresponding lower bound closes the gap to the optimal solution for 1/3 of the benchmark instances, Wøhlk (2006). Along with the fact that only few algorithms for the construction of feasible solutions have been developed for the NEARP, this leads us to believe that a large portion of the gap may be explained by the quality of the upper bound.

The quality of the lower bound can be measured in two ways. For a solution for NEARP to be feasible it must satisfy two general conditions, 1) the flow balance, which enforces that each vehicle traverses the graph such that whenever it arrives at a node it will also leave the node; 2) the packing constraints which enforce that each vehicle does not exceed its capacity Q.

The MCNDLB lower bound provides solutions where the flow balance is satisfied to some extent due to perfect matching in the first iteration which enforces the necessary number of deadheading links. However, the bound only avoids 2-cycles and the path is thus not elementary. The other discrepancy is that the cost of these deadheadings in the succeeding cuts may be underestimated. We sum the costs of the cheapest way to cross each cutset, but not the cheapest way to cross all the cutsets. Hence, the bound does not foresee if a more expensive link should have been used.

As regards the packing constraints, we believe that these contribute the most to the quality of the gap. If we assume that the capacity Q accommodates all demands, such that only one vehicle would be required, the packing constraint would not contribute

to the gap. If we inspect the solutions to the DI-NEARP instances it is obvious that the gap decreases when the capacity of the vehicles increases. This is because the packing constraints become less important.

In the first iteration of the MCNDLB algorithm, there is a stronger focus on flow balance than packing. As cuts are added in subsequent iterations, emphasis is progressively shifted from flow balance towards packing.

1.6 Summary and Conclusion

The VRP literature has often been criticized for being based on idealized assumptions that render the proposed models inadequate for real life applications. With only a few exceptions, there has been a strict separation of node routing and arc routing problems in the literature. Prins and Bouchenoua (2004) argued that there are real-life applications that can neither be adequately modeled as pure arc routing problems, nor as pure node routing problems.

In this chapter, we have reinforced the claims of Prins and Bouchenoua and argued that the NEARP represents an important, new dimension of VRP model richness. We have also argued that the tradition of modeling applications such as newspaper delivery, mail delivery, and communal waste collection as arc routing problems is problematic. For real-life, large-size instances of such applications, where demand is basically located in nodes, abstraction techniques such as aggregation of demand may be needed to provide high-quality solutions. Reasonable aggregation heuristics will typically produce instances with demand on nodes, edges, and arcs.

The main contribution of this chapter is to provide (to the best of our knowledge) the first lower bound procedure for the NEARP. Also, we provide two new sets of test instances: the BHW benchmark derived from 20 well-known CARP instances, and the DI-NEARP benchmark with 24 instances derived from real-life data from carrier

routing of subscription newspapers and other media products. The new benchmarks complement the grid-based CBMix benchmark proposed by Prins and Bouchenoua, for which two other papers also provide upper bounds. For the BHW and DI-NEARP benchmarks, the first upper bounds have been produced by Hasle et al. (2012), so we now have lower and upper bounds for all test instances. For three instances, the gaps have been closed. The average gap is rather large: 25.1%. The lower bound procedure is based on a high quality lower bound for the CARP. This fact, and the limited amount of work on approximate methods for the NEARP, give us reason to believe that the main reason for the large gaps is the quality of the upper bound.

The NEARP is a theoretically interesting problem with high industrial relevance. The numerical results presented here should motivate the research community to develop better heuristics and exact algorithms that take advantage of the structure of this important problem. Moreover, NEARP extensions should be proposed on the basis of industrial aspects.

1.7 Acknowledgments

The authors would like to thank the company Distribution Innovation AS for giving us access to real-life case data from their newspaper delivery routing system. The work presented here has been funded by the Research Council of Norway as a part of the Effekt project (contract number 187293/I40, SMARTRANS), and the DOMinant II project (contract number 205298/V30, eVita).

Table 1.1: Details on the new BHW instances.

Instance	Basis Instance	Total number			Required			Capacity	Node demand		Edge demand		Arc demand	
		Nodes	Edges	Arcs	Nodes	Edges	Arcs		Mean	Std.	Mean	Std.	Mean	Std.
BHW1	GDB 1	12	11	22	7	11	11	5	1.0	0.0	1.0	0.0	1.0	0.0
BHW2	GDB 6	12	0	25	4	0	25	5	1.0	0.0	-	-	1.0	0.0
BHW3	GDB 12	13	8	30	5	8	7	35	7.2	3.5	8.0	4.1	9.0	2.3
BHW4	GDB 22	11	0	44	6	0	44	27	5.0	2.4	-	-	4.0	2.7
BHW5	Val 7a	40	0	132	30	0	132	200	8.5	4.3	-	-	8.0	3.3
BHW6	Val 7a	40	37	58	15	37	58	200	7.3	4.8	8.0	2.5	7.0	3.0
BHW7	Val 10d	50	0	194	35	0	194	75	9.1	4.6	-	-	7.0	2.3
BHW8	Val 10d	50	0	194	20	0	97	75	7.6	3.3	-	-	7.0	1.7
BHW9	Val 10d	50	26	142	10	26	142	75	8.6	3.1	6.0	1.6	7.0	2.3
BHW10	Egl e1 A	77	0	196	40	0	102	305	30.0	13.3	-	-	28.0	15.3
BHW11	Egl e1 A	77	0	196	20	0	51	305	28.3	11.1	-	-	28.0	10.8
BHW12	Egl s1 B	140	0	380	40	0	75	150	29.7	11.1	-	-	18.0	6.9
BHW13	Egl s1 B	140	0	380	25	0	150	150	29.0	11.2	-	-	18.0	9.7
BHW14	Egl e4 C	77	0	196	25	0	196	130	31.2	13.0	-	-	25.0	20.7
BHW15	Egl e4 C	77	0	196	30	0	98	130	33.0	11.9	-	-	25.0	14.6
BHW16	Egl s4 C	140	0	380	30	0	380	120	24.4	16.9	-	-	22.0	16.9
BHW17	Egl s4 C	140	0	380	50	0	190	120	22.3	7.1	-	-	22.0	11.9
BHW18	Egl e3 B	77	0	196	20	0	174	190	25.0	8.1	-	-	25.0	19.6
BHW19	Egl e3 B	77	0	196	20	0	87	190	24.1	9.0	-	-	25.0	13.8
BHW20	Egl s2 A	140	51	278	50	51	192	235	19.0	9.1	23.0	13.4	20.0	13.3

Table 1.2: Details on the DI-NEARP instances.

Instance	Number of			Required			Capacity	Node demand		Edge demand		Arc demand	
	Nodes	Edges	Arcs	Nodes	Edges	Arcs		Mean	Std.	Mean	Std.	Mean	Std.
DI-NEARP-n240-Q2k	563	815	0	120	120	0	2000	34.2	45.6	78.2	82.7	-	-
DI-NEARP-n240-Q4k	563	815	0	120	120	0	4000	34.2	45.6	78.2	82.7	-	-
DI-NEARP-n240-Q8k	563	815	0	120	120	0	8000	34.2	45.6	78.2	82.7	-	-
DI-NEARP-n240-Q16k	563	815	0	120	120	0	16000	34.2	45.6	78.2	82.7	-	-
DI-NEARP-n422-Q2k	710	871	0	302	120	0	2000	24.4	39.1	74.7	102.1	-	-
DI-NEARP-n422-Q4k	710	871	0	302	120	0	4000	24.4	39.1	74.7	102.1	-	-
DI-NEARP-n422-Q8k	710	871	0	302	120	0	8000	24.4	39.1	74.7	102.1	-	-
DI-NEARP-n422-Q16k	710	871	0	302	120	0	16000	24.4	39.1	74.7	102.1	-	-
DI-NEARP-n442-Q2k	761	917	0	294	148	0	2000	26.1	31.7	69.1	46.9	-	-
DI-NEARP-n442-Q4k	761	917	0	294	148	0	4000	26.1	31.7	69.1	46.9	-	-
DI-NEARP-n442-Q8k	761	917	0	294	148	0	8000	26.1	31.7	69.1	46.9	-	-
DI-NEARP-n442-Q16k	761	917	0	294	148	0	16000	26.1	31.7	69.1	46.9	-	-
DI-NEARP-n477-Q2k	667	837	0	203	274	0	2000	16.9	35.8	68.4	73.9	-	-
DI-NEARP-n477-Q4k	667	837	0	203	274	0	4000	16.9	35.8	68.4	73.9	-	-
DI-NEARP-n477-Q8k	667	837	0	203	274	0	8000	16.9	35.8	68.4	73.9	-	-
DI-NEARP-n477-Q16k	667	837	0	203	274	0	16000	16.9	35.8	68.4	73.9	-	-
DI-NEARP-n699-Q2k	982	1103	0	335	364	0	2000	57.3	67.2	176.1	175.5	-	-
DI-NEARP-n699-Q4k	982	1103	0	335	364	0	4000	57.3	67.2	176.1	175.5	-	-
DI-NEARP-n699-Q8k	982	1103	0	335	364	0	8000	57.3	67.2	176.1	175.5	-	-
DI-NEARP-n699-Q16k	982	1103	0	335	364	0	16000	57.3	67.2	176.1	175.5	-	-
DI-NEARP-n833-Q2k	1120	1450	0	347	486	0	2000	31.9	72.7	104.2	216.9	-	-
DI-NEARP-n833-Q4k	1120	1450	0	347	486	0	4000	31.9	72.7	104.2	216.9	-	-
DI-NEARP-n833-Q8k	1120	1450	0	347	486	0	8000	31.9	72.7	104.2	216.9	-	-
DI-NEARP-n833-Q16k	1120	1450	0	347	486	0	16000	31.9	72.7	104.2	216.9	-	-

Table 1.3: Results obtained for the CBMix instances.

Instance	Best Known	Lower Bound			Lower Bound AD1		
	B^U	B^L	G^O (%)	CPU (s)	B^L	G^O (%)	CPU (s)
CBMix1	2589	2409	7.2	1.0	2409	7.2	3.1
CBMix2	12220	9742	22.6	76.7	9742	22.6	353.4
CBMix3	3643	3014	18.9	7.5	3014	18.9	30.6
CBMix4	7583	5302	35.4	20.9	5302	35.4	118.8
CBMix5	4531	3747	18.9	3.8	3789	17.8	13.1
CBMix6	7087	4983	34.9	16.2	5201	30.7	43.1
CBMix7	9607	7296	27.3	58.7	7296	27.3	193.6
CBMix8	10524	7956	27.8	33.4	7956	27.8	196.8
CBMix9	4038	3460	15.4	2.5	3460	15.4	7.8
CBMix10	7582	6409	16.8	37.5	6432	16.4	113.0
CBMix11	4494	2998	39.9	4.6	3031	38.9	43.9
CBMix12*	**3138**	**3138**	0.0	2.1	**3138**	0.0	12.9
CBMix13	9110	6489	33.6	19.4	6524	33.1	238.3
CBMix14	8566	5719	39.9	15.7	5731	39.7	107.5
CBMix15	8280	6270	27.6	10.9	6318	26.9	64.3
CBMix16	8886	7416	18.0	24.5	7416	18.0	172.6
CBMix17	4037	3654	10.0	1.8	3654	10.0	22.0
CBMix18	7098	6089	15.3	25.7	6089	15.3	120.9
CBMix19	16347	11065	38.5	110.5	11143	37.9	549.6
CBMix20	4844	3400	35.0	2.3	3452	33.6	15.7
CBMix21	18069	12474	36.6	61.8	12474	36.6	221.5
CBMix22	1941	1825	6.2	1.8	1825	6.2	4.8
CBMix23*	**780**	667	15.6	0.1	667	15.6	0.8

Table 1.4: Results obtained for the BHW instances.

Instance	Best Known	Lower Bound			Lower Bound AD1		
	B^U	B^L	G^O (%)	CPU (s)	B^L	G^O (%)	CPU (s)
BHW1	337	324	3.9	0.3	324	3.9	1.3
BHW2*	**470**	**470**	0.0	0.4	**470**	0.0	0.9
BHW3	415	326	24.0	0.2	326	24.0	0.5
BHW4*	**240**	**240**	0.0	0.5	**240**	0.0	3.8
BHW5	506	498	1.6	5.4	502	0.8	52.1
BHW6*	**388**	**388**	0.0	2.9	**388**	0.0	32.3
BHW7	1094	930	16.2	41.7	930	16.2	347.2
BHW8	672	644	4.3	6.8	644	4.3	118.8
BHW9	913	791	14.3	28.0	791	14.3	346.5
BHW10	8556	6810	22.7	21.6	6810	22.7	123.1
BHW11	5021	3986	23.0	6.9	3986	23.0	40.0
BHW12	11042	6346	54.0	33.4	6346	54.0	207.7
BHW13	14510	8746	49.6	86.3	8746	49.6	576.7
BHW14	25194	17762	34.6	113.0	17762	34.6	737.5
BHW15	15509	12193	23.9	20.7	12193	23.9	214.2
BHW16	44527	26014	52.5	787.2	26014	52.5	4905.3
BHW17	26768	15396	53.9	162.7	15396	53.9	900.6
BHW18	15833	11202	34.3	77.9	11202	34.3	435.3
BHW19	9424	7065	28.6	14.1	7080	28.4	101.6
BHW20	16625	10730	43.1	269.9	10730	43.1	1388.4

Table 1.5: Results obtained for the DI-NEARP instances.

Instance	Best Known	Lower Bound		
	B^U	B^L	G^O (%)	CPU (s)
DI-NEARP-n240-Q2k	24371	16376	39.2	368
DI-NEARP-n240-Q4k	18352	14362	24.4	311
DI-NEARP-n240-Q8k	15937	13442	17.0	324
DI-NEARP-n240-Q16k	14953	13116	13.1	334
DI-NEARP-n422-Q2k	18990	11623	48.1	1571
DI-NEARP-n422-Q4k	15987	11284	34.5	1337
DI-NEARP-n422-Q8k	14627	11220	26.4	1049
DI-NEARP-n422-Q16k	14357	11198	24.7	1702
DI-NEARP-n442-Q2k	51656	35068	38.3	1689
DI-NEARP-n442-Q4k	45605	33585	30.4	1715
DI-NEARP-n442-Q8k	44652	32985	30.1	1736
DI-NEARP-n442-Q16k	42797	32713	26.7	1816
DI-NEARP-n477-Q2k	23124	19722	15.9	1572
DI-NEARP-n477-Q4k	20198	18031	11.3	1574
DI-NEARP-n477-Q8k	18561	17193	7.7	1582
DI-NEARP-n477-Q16k	18105	16873	7.0	1575
DI-NEARP-n699-Q2k	59817	34101	54.8	7249
DI-NEARP-n699-Q4k	40473	26891	40.3	6921
DI-NEARP-n699-Q6k	30992	23302	28.3	7133
DI-NEARP-n699-Q8k	27028	21967	20.7	7400
DI-NEARP-n833-Q2k	56877	32435	54.7	8239
DI-NEARP-n833-Q4k	42407	29381	36.3	8739
DI-NEARP-n833-Q8k	35267	28453	21.4	8675
DI-NEARP-n833-Q16k	33013	28233	15.6	8157

Chapter 2

A Branch-and-Cut-and-Price Algorithm for the Mixed Capacitated General Routing Problem

History: *This chapter has been prepared in collaboration with Jens Lysgaard and Sanne Wøhlk.*

A Branch-and-Cut-and-Price Algorithm for the Mixed Capacitated General Routing Problem

Lukas Bach, Jens Lysgaard, and Sanne Wøhlk

Cluster for Operations Research And Logistics,

Department of Economics and Business,

Aarhus University, Denmark

Abstract

In this chapter, we consider the Mixed Capacitated General Routing Problem which is a combination of the Capacitated Vehicle Routing Problem and the Capacitated Arc Routing Problem. The problem is also known as the Node, Edge, and Arc Routing Problem. We propose a Branch-and-Cut-and-Price algorithm for obtaining optimal solutions to the problem and present computational results based on a set of standard benchmark instances.

Keywords: Mixed Capacitated General Routing Problem; Branch-and-Cut-and-Price; Vehicle Routing; Node Routing; Arc Routing; General Routing; VRP; CARP; NEARP; MCGRP; Bound

2.1 Introduction

The Capacitated Vehicle Routing Problem (CVRP) was first presented in Dantzig and Ramser (1959) and has been studied extensively ever since. The Capacitated Arc Routing Problem (CARP) was presented more than twenty years later by Golden and Wong

(1981). Both problems find many applications in real life. The CVRP models problems of delivery of products and other situations where specific points have to be visited. The CARP, on the other hand, models situations where street segments need to be visited. Such problems are often found in snow removal and refuse collection, for instance.

In this chapter, we study a combination of the two types of problems. In the Mixed Capacitated General Routing Problem (MCGRP), nodes as well as directed arcs and undirected edges require visits from a fleet of vehicles. The MCGRP was first introduced in Pandit and Muralidharan (1995) and was rediscovered by Prins and Bouchenoua (2004) under a different name: The Node, Edge, and Arc Routing Problem (NEARP).

The MCGRP more specifically models situations where street segments as well as specific points need service. Applications are found within refuse collection where household waste is collected along the streets whereas industrial sites and supermarkets need to be considered as single locations due to the large amount of waste in these locations. And due to sheer size, large arc routing problems in real life waste management, snow removal, or mail delivery cannot be handled efficiently. One option for solving such problems is to contract neighborhoods of 10-20 arcs and edges into a single node, while leaving larger streets and highways as arcs and edges. The result of this process is a MCGRP.

Until now only one exact algorithm for the MCGRP has been presented, namely Bosco et al. (2013). Inspired by Fukasawa et al. (2006) we present the first Branch-and-Cut-and-Price algorithm for the problem. The main idea is to combine two different mathematical models for the problem, where one model involves cut generation and the other involves column generation. The purpose is to present a robust model where cuts can easily be added in a Branch-and-Price setup. However in the current implementation we only add cuts to the root node of the branching tree.

In Section 2.2, we present relevant literature regarding exact optimization of the CVRP and the CARP as well as the limited literature regarding the MCGRP. In Section 2.3, we first define the problem formally and introduce some notations. Next we present the two models by which our model is composed, and finally, we present our combined model. After presenting the combined model, we show how some valid inequalities originally presented for the CARP and for the single vehicle general routing problem can be extended to the MCGRP. In Section 2.4, we present the details of our Branch-and-Cut-and-Price algorithm and in Section 2.5, we provide our computational results. Finally, conclusions are drawn in Section 2.6

2.2 Related Literature

As the MCGRP is a combination of node routing and edge/arc routing, we first consider the literature presenting exact solutions of the CVRP and the CARP, respectively. We then turn our attention to previous work related to the MCGRP. The aim is not to provide a full review of all papers published within the area, but rather to describe the state of the art through a representative sample.

Many papers have presented exact algorithms for the CVRP. A major breakthrough was the robust Branch-and-Cut-and-Price algorithm by Fukasawa et al. (2006). This algorithm simultaneously generates columns based on a set partitioning formulation of the problem and cuts based on a flow formulation. The cuts are based on valid inequalities presented in Letchford et al. (2002) and Lysgaard et al. (2004). Our modeling approach is inspired by Fukasawa et al. (2006).

Baldacci et al. (2008) present an exact set partitioning algorithm for the CVRP, where the LP relaxation of the master problem is solved by sequentially solving three problems. Each of these problems is obtained by tightening the former with a new class of valid inequalities or by requiring routes to be elementary. In Baldacci et al. (2011),

this algorithm is improved by a new route generation method and the algorithm is extended to solve the vehicle routing problem with time windows. Baldacci and Mingozzi (2009) extend the exact algorithm of Baldacci et al. (2008) to solve a broader class of vehicle routing problems, including a heterogeneous fleet of vehicles, multiple depots, and site dependence.

Laporte (2009) and Baldacci et al. (2012) provide recent surveys on the CVRP and on exact approaches in particular. Baldacci et al. (2012) compare the exact methods of Lysgaard et al. (2004); Fukasawa et al. (2006); Baldacci et al. (2008, 2011). Their results show that, on average, the algorithm by Baldacci et al. (2011) is slightly better than the algorithms of Baldacci et al. (2008) and Fukasawa et al. (2006) and that it is, on average, faster than the latter two, also when taking into account the difference in computer speeds. The algorithm by Lysgaard et al. (2004) is outperformed by the others, but is used as part of the algorithm by Fukasawa et al. (2006).

Belenguer and Benavent (1998) suggest an IP model for the CARP based on 0-1 variables x_{ep}, representing whether or not vehicle p services edge e, and variables y_{ep} representing the number of times vehicle p traverses edge e without servicing it. They study the polyhedron of the CARP based on this model and present several facets for this polyhedron. Belenguer and Benavent (2003) further extend the work to an aggregated model using variables x_e to represent the number of times the edge e is traversed without service by any vehicle. They use this model in a cutting plane algorithm for obtaining lower bounds. Due to the aggregation, the algorithm cannot be used for obtaining feasible CARP solutions, but it produces high quality lower bounds.

Longo et al. (2006) solve the CARP by using a transformation to the CVRP and use the Branch-and-Cut-and-Price algorithm by Fukasawa et al. (2006) to solve the transformed problem. Bartolini et al. (2013) solve the CARP by transforming it to a CVRP and use a series of cut and column generations which is similar to the strategy used by Baldacci et al. (2008).

Bode and Irnich (2012) present an exact algorithm for the CARP based on a Cut-First Branch-and-Price-Second strategy. Their algorithm solves nearly all unsolved benchmark instances for the problem. In their algorithm, they use the pricing strategy suggested by Letchford and Oukil (2009) and use algorithms to identify violated inequalities. They use algorithms by Letchford et al. (2008) for Odd Edge Cutset, by Belenguer and Benavent (2003) and Ahr (2004) for Capacity inequalities, and by Belenguer and Benavent (2003) for Disjoint-path inequalities. Furthermore, the paper provides a full survey of exact solutions of the CARP and a detailed discussion of different modeling and solution approaches for the problem.

Bode and Irnich (2013a) present a comprehensive analysis of the use of different pricing strategies within the Cut-First Branch-and-Price-Second algorithm of Bode and Irnich (2012). In Bode and Irnich (2013a) methods for handling (k_1, k_2)-loop elimination in a SPPRC are investigated. The k_1 refer to cycle elimination with respect to eliminating task loops to model elementary routes, and k_2 refer to eliminating cycles respecting non-follower constraints. Bode and Irnich (2013b) introduce a SPPRC algorithm with (k,2)-loop elimination for efficiently handling the non-follower constraints arising in the Branch-and-Price algorithm presented in Bode and Irnich (2012), this SPPRC algorithm allows for ($k > 3$)-cycle elimination.

The MCGRP was first presented by Pandit and Muralidharan (1995). They motivate the introduction of this problem by real life applications and solve the problem, including both capacity and time duration constraints, using a route-first-partition-second algorithm. Gutiérrez et al. (2002) present a cluster-first-route-second algorithm for the same problem and show that their algorithm outperforms that of Pandit and Muralidharan (1995).

Four papers have considered the MCGRP from a metaheuristic point of view. Prins and Bouchenoua (2004) present a Memetic algorithm and Kokubugata et al. (2007) suggest a relatively simple algorithm based on Simulated Annealing. Dell'Amico et al.

(2012) present an Iterated Local Search algorithm for the MCGRP. Hasle et al. (2012) solve the problem using their Spider algorithm (Hasle and Kloster, 2007) which is a combination of Iterated Local Search and Variable Neighborhood Descent with a large number of parameter options.

To obtain lower bounds, Gaze (2013) extends the column generation approach by Letchford and Oukil (2009) to the MCGRP and uses it to solve the root node of a Branch-and-Price tree. Bach et al. (2013) present a lower bounding algorithm for the MCGRP based on the construction of a matching network in subgraphs obtained by a series of mutually disjoint cuts.

The only published exact algorithm for the MCGRP is by Bosco et al. (2013). They present an IP model based on variables x_{ij}^k for the MCGRP. Furthermore, they extend the capacity constraints and the odd edge cutset constraint of Belenguer and Benavent (1998) to the MCGRP and incorporate them in a Branch-and-Cut algorithm. Bosco et al. (2013) test five heuristic procedures and use the best of these to initialize their algorithm. Their algorithm can solve small instances to optimality within a time limit of 6 hours.

Three sets of benchmark instances are available for the MCGRP: 23 CBmix instances by Prins and Bouchenoua (2004), 20 BHW instances, and 24 DI-NEARP instances by Bach et al. (2013). The instances are available at SINTEF (2013) along with a table of best known upper and lower bounds for each instance. For the CBmix instances, the best known feasible solution is provided by Prins and Bouchenoua (2004), Kokubugata et al. (2007), Hasle et al. (2012), and Bosco et al. (2013) for 12, 8, 2, and 1 instances, respectively. For these instances, Bosco et al. (2013) and Gaze (2013) each provide one best known lower bound and Bach et al. (2013) provide the remaining 21. The two remaining sets of benchmark instances are newer and have been used in fewer experiments. For all these instances, Hasle et al. (2012) provide the best known solution. The best known lower bound is provided by Bach et al. (2013) for all instances except

one which is provided by Gaze (2013). Two CBmix instances and two BHW instances have been solved to optimality. None of the DI-NEARP instances have been solved optimally so far.

2.3 Model Formulation

The Mixed Capacitated General Routing Problem (MCGRP) is defined on a connected multi-graph $G = (N, E, A)$, where N is the set of nodes, E is the set of undirected edges, and A is the set of directed arcs. Let c_e (c_a) denote the non-negative traversal cost for $e \in E$ ($a \in A$). Let $N_R \subseteq N$ be the set of required nodes, i.e., nodes which require service. For these nodes, let q_i denote the size of the demand and p_i the processing cost of node $i \in N_R$. Similarly, let E_R and A_R be the set of required edges and arcs, respectively, and let q_e (q_a) and p_e (p_a) denote the demand and processing cost of $e \in E_R$ ($a \in A_R$), respectively. The processing cost is the additional cost incurred when the required edge or arc is serviced. Hence, the total cost of servicing an edge (arc) is $c_e + p_e$ ($c_a + p_a$).

Node $1 \in N$ denotes the depot node and holds a fleet of K identical vehicles, each of capacity W. We define $N' = N \setminus \{1\}$. The goal is to construct tours for the vehicles, starting and ending in the depot, such that each required node, arc, and edge is serviced by exactly one vehicle, the vehicle capacity is respected, and the total traversal cost is minimized. To follow the convention of Prins and Bouchenoua (2004), we only report the total traversal cost of a solution since the processing cost is constant.

2.3.1 Notation and Definitions

We denote an edge e as an unordered pair $\{i, j\}$. Arcs a are denoted by (i, j). We refer jointly to any arc or edge as a link. For any set $S \subseteq N'$, we define the graph $G(S)$

induced by S as the graph with node set S, edge set $E(S) = \{e = \{i,j\} \in E | i \in S, j \in S\}$, and arc set $A(S) = \{a = (i,j) \in A | i \in S, j \in S\}$.

For any set $S \subset N$, we define the following cutsets: The set of edges crossing the cut $\delta^u(S) = \{e = \{i,j\} \in E | i \in S, j \in N \setminus S \text{ or } i \in N \setminus S, j \in S\}$, the set of outgoing arcs $\delta^+(S) = \{a = (i,j) \in A | i \in S, j \in N \setminus S\}$, and the set of entering arcs $\delta^-(S) = \{a = (i,j) \in A | i \in N \setminus S, j \in S\}$. We define the joint cutset $\delta(S)$ as the total set of links in any of the cutsets:

$$\delta(S) = \delta^u(S) \cup \delta^+(S) \cup \delta^-(S) \subseteq E \cup A$$

Given a set $S \subseteq N'$, the total demand in $G(S)$ and $\delta(S)$ is denoted $Q(S)$ and is given by $Q(S) = \sum_{e \in E(S) \cup A(S) \cup \delta(S)} q_e + \sum_{i \in S} q_i$. To service this demand, at least $\gamma(S) = \lceil Q(S)/W \rceil$ vehicles are needed. In other words, at least $\gamma(S)$ vehicles must enter the set S.

When S consists of a single node i, we simplify notation and write i instead of $\{i\}$ in all the above definitions. Throughout the chapter, R used as subscript of any set, denotes the required entities of that set.

2.3.2 Two-Index Model

Bosco et al. (2013) presented the only known mathematical model for the MCGRP. Their model is based on three-index variables x_{ij}^k and y_{ij}^k for service and deadheading (i.e., traversing without servicing), respectively. For the CARP, Belenguer and Benavent (1998) presented a two-index formulation which is only efficient for small fleet sizes (less than six vehicles), see Bode and Irnich (2012) and references herein.

In this section, we present a two-index model for the MCGRP. To extend the model by Belenguer and Benavent (1998) is not straightforward due to the fact that the MCGRP contains directed arcs and required nodes. Therefore, we seek further inspiration in the models presented by Corberán et al. (2003, 2006) for single vehicle problems on mixed graphs.

We define the following variables:

$$\tilde{x}_{ek} = \begin{cases} 1 & \text{if vehicle } k \text{ services edge } e \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, \forall e \in E_R$$

$$\tilde{y}_{ak} = \begin{cases} 1 & \text{if vehicle } k \text{ services arc } a \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, \forall a \in A_R$$

$$\tilde{z}_{ik} = \begin{cases} 1 & \text{if vehicle } k \text{ services node } i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, \forall i \in N_R$$

For all edges $e \in E$ and all arcs $a \in A$ we define the following variables for each vehicle $k \in K$:

$$\begin{aligned} x_{ek} &= \text{total number of times vehicle } k \text{ deadheads edge } e \\ y_{ak} &= \text{total number of times vehicle } k \text{ deadheads arc } a \end{aligned}$$

To simplify the model, we introduce the following notation $\forall k \in K$:

$$\begin{aligned} \hat{x}_k(\delta^u(S)) &= \sum_{e \in \delta^u(S)} (\tilde{x}_{ek} + x_{ek}) \\ \hat{y}_k(\delta^+(S)) &= \sum_{a \in \delta^+(S)} (\tilde{y}_{ak} + y_{ak}) \\ \hat{y}_k(\delta^-(S)) &= \sum_{a \in \delta^-(S)} (\tilde{y}_{ak} + y_{ak}) \\ \hat{y}_k(\delta^\pm(S)) &= \hat{y}_k(\delta^+(S)) + \hat{y}_k(\delta^-(S)) \end{aligned}$$

(IP 2-index)

$$\text{minimize} \quad \sum_{k\in K}\Big(\sum_{e\in E_R} c_e\tilde{x}_{ek} + \sum_{e\in E} c_e x_{ek} + \sum_{a\in A_R} c_a\tilde{y}_{ak} + \sum_{a\in A} c_a y_{ak}\Big)$$

$$\text{s.t.} \quad \sum_{k\in K} \tilde{z}_{ik} = 1 \;\forall i \in N_R \tag{2.1}$$

$$\sum_{k\in K} \tilde{x}_{ek} = 1 \;\forall e \in E_R \tag{2.2}$$

$$\sum_{k\in K} \tilde{y}_{ak} = 1 \;\forall a \in A_R \tag{2.3}$$

$$\sum_{e\in E_R} q_e\tilde{x}_{ek} + \sum_{a\in A_R} q_a\tilde{y}_{ak} + \sum_{i\in N_R} q_i\tilde{z}_{ik} \le W \;\forall k \in K \tag{2.4}$$

$$\hat{x}_k(\delta^u(i)) + \hat{y}_k(\delta^{\pm}(i)) \ge 2\tilde{z}_{ik} \;\forall i \in N_R, \forall k \in K \tag{2.5}$$

$$\hat{x}_k(\delta^u(S)) + \hat{y}_k(\delta^{\pm}(S)) \ge 2\tilde{x}_{fk} \forall S \subseteq N', \forall f \in E_R(S), \forall k \in K \tag{2.6}$$

$$\hat{x}_k(\delta^u(S)) + \hat{y}_k(\delta^{\pm}(S)) \ge 2\tilde{y}_{fk} \forall S \subseteq N', \forall f \in A_R(S), \forall k \in K \tag{2.7}$$

$$\hat{x}_k(\delta^u(S)) + \hat{y}_k(\delta^{-}(S)) \ge \hat{y}_k(\delta^{+}(S)) \;\forall k \in K, \forall S \subset N \tag{2.8}$$

$$\sum_{e\in\delta^u(i)} x_{ek} + \sum_{a\in\delta^+(i)\cup\delta^-(i)} y_{ak} \text{ is } \begin{cases} \text{even} & \text{if } |\delta_R(i)| \text{ even} \\ \text{odd} & \text{if } |\delta_R(i)| \text{ odd} \end{cases} \quad \forall k \in K, \forall i \in N \tag{2.9}$$

$$\tilde{z}_{ik} \in \{0,1\} \;\forall i \in N_R, \forall k \in K \tag{2.10}$$

$$\tilde{x}_{ek} \in \{0,1\} \;\forall e \in E_R, \forall k \in K \tag{2.11}$$

$$\tilde{y}_{ak} \in \{0,1\} \;\forall a \in A_R, \forall k \in K \tag{2.12}$$

$$x_{ek} \ge 0, \text{integer} \;\forall e \in E, \forall k \in K \tag{2.13}$$

$$y_{ak} \ge 0, \text{integer} \;\forall a \in A, \forall k \in K \tag{2.14}$$

In (IP 2-index), the objective function minimizes the total traversal cost. Constraints (2.1), (2.2), and (2.3) ensure that every required item is serviced, whereas constraint (2.4) is the vehicle capacity constraint. Constraint (2.5) ensures that the vehicle which services a required node also visits the node. (2.6) and (2.7) are the corresponding constraints for edges and arcs. These constraints are motivated by Belenguer and Benavent

(1998). The balance constraint (2.8) ensures, a feasible balance between incoming and outgoing arcs for each set, such that when taking the undirected edges into account, we ensure an equal possible number of entries and exits. Constraint (2.9) forces the use of an even number of connections incident to each node. Constraints (2.8) and (2.9) are inspired by Corberán et al. (2003). The combination of these two constraints ensures that the routes are connected whereas constraints (2.6) and (2.7) prevent the occurrence of subtours. Finally, (2.10) - (2.14) define the variable domains.

2.3.3 Set Partitioning Model

In order to construct a Set Partitioning model for the MCGRP, we decompose model (IP 2-index) using Dantzig-Wolfe decomposition, Dantzig and Wolfe (1960, 1961).

For each vehicle $k \in K$, we define the polytope $\mathcal{P}_k$ given by constraints (2.4) - (2.14) in model (IP 2-index) with k fixed. Because the vehicles in K are identical, we have $\mathcal{P}_1 = \mathcal{P}_2 = \cdots = \mathcal{P}_{|K|}$. We use the short notation $\mathcal{P} = \mathcal{P}_k, \forall k \in K$. $\mathcal{P}$ defines the feasible region of the pricing problem in our decomposition.

We let Ω be the set of extreme points in $\mathcal{P}$. Such a point corresponds to a route for a single vehicle with the following requirements: 1) The vehicle must start and end in the depot, 2) The vehicle must service one or more required entities, 3) Each entity is serviced at most once on the route, 4) Any deadheading between two serviced entities, and between the depot and the first (last) entity serviced must follow a shortest path, and 5) The vehicle capacity must be respected.

The third requirement states that the route must be elementary with respect to serviced entities. In practice, this requirement is often relaxed to simplify the column generation process for the pricing-problem. In this case, the requirement is handled in a branching process.

It is essential that the deadheading occurs along a shortest path. If this requirement is relaxed, the problem is unbounded from above. An alternative is to leave the feasible

region unbounded and include extreme rays consisting of small deadheading loops. See Bode and Irnich (2012) for a detailed description of this.

We let c_r be the total traversal cost of the route and define the following parameters:

$$\begin{aligned} h_i^r &= 1 \text{ if route } r \text{ services node } i, \text{ and 0 otherwise.} \\ s_e^r &= 1 \text{ if route } r \text{ services edge } e, \text{ and 0 otherwise.} \\ t_a^r &= 1 \text{ if route } r \text{ services arc } a, \text{ and 0 otherwise.} \end{aligned}$$

To construct the Master Problem, we define, for each route $r \in \Omega$, the variable

$$\lambda^r = \begin{cases} 1 & \text{if route } r \text{ is used in the solution} \\ 0 & \text{otherwise} \end{cases}$$

The Master Problem can then be stated as

(IP Master)

$$\begin{aligned} \text{minimize} \quad & \sum_{r \in \Omega} c_r \lambda^r \\ \text{s.t.} \quad & \sum_{r \in \Omega} h_i^r \lambda^r = 1, \forall i \in N_R && (2.15) \\ & \sum_{r \in \Omega} s_e^r \lambda^r = 1, \forall e \in E_R && (2.16) \\ & \sum_{r \in \Omega} t_a^r \lambda^r = 1, \forall a \in A_R && (2.17) \\ & \sum_{r \in \Omega} \lambda^r \leq |K| && (2.18) \\ & \lambda^r \geq 0, \text{integer } \forall r \in \Omega && (2.19) \end{aligned}$$

In (IP Master), constraints (2.15), (2.16), and (2.17) ensure that every required node, edge, and arc is serviced by exactly one vehicle route, and constraint (2.18) ensures that no more than $|K|$ vehicles are used. For the LP relaxation of (IP Master), let $\boldsymbol{\pi} = [\pi_i]$, $\boldsymbol{\mu} = [\mu_e]$, and $\boldsymbol{\rho} = [\rho_a]$ be the vectors of dual variables corresponding to constraints (2.15), (2.16), and (2.17), respectively. Finally, let σ be the dual variable corresponding

to constraint (2.18). Then the Pricing Problem is given as

(IP Pricing)

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E_R} (\tilde{c}_e - \mu_e)\tilde{x}_e + \sum_{e \in E} c_e x_e + \sum_{a \in A_R} (\tilde{c}_a - \rho_a)\tilde{y}_a \\ & + \sum_{a \in A} c_a y_a + \sum_{i \in N_R} (c_i - \pi_i)\tilde{z}_i - \sigma \\ \text{s.t.} \quad & (2.4) - (2.14) \end{aligned}$$

In (IP Pricing), the objective function minimizes the reduced cost of the vehicle route generated.

2.3.4 Aggregated Model

Belenguer and Benavent (2003) present a one-index model for a relaxation of the CARP. The model is based on aggregated variables defined as

$$x_e = \sum_{k \in K} x_{ek} \; \forall e \in E$$

Intuitively, these variables count the number of times an edge is deadheaded by any vehicle. The drawback of defining a model based on such aggregated variables is that an optimal solution to the model may not necessarily correspond to a feasible solution to the CARP. The advantage, on the other hand, is that the number of variables is smaller than in the two-index model.

Belenguer and Benavent (2003) present several valid inequalities based on these aggregated variables and use them in a Cutting Plane algorithm that successfully obtains lower bounds for the CARP.

We use the idea of aggregated variables in our combined model which is presented in Section 2.3.5. In Section 2.3.6 we show how several of the valid inequalities presented in Belenguer and Benavent (2003) can be extended to the MCGRP, and in Section 2.4.1 we explain how some of these valid inequalities are used in our algorithm.

2.3.5 Combined Model

In this section, we present our combined model which is the model used in our Branch-and-Cut-and-Price algorithm. The purpose is to construct a model where valid inequalities can be added as cuts to the model without changing the structure of the pricing problem used to generate columns. The combined model is inspired by the modeling technique in Fukasawa et al. (2006) for the CVRP.

We use two types of variables in our combined model. From (IP Master), we use the route variables:

$$\lambda^r = \begin{cases} 1 & \text{if route } r \text{ is used in the solution} \\ 0 & \text{otherwise} \end{cases}$$

From the aggregated model, we use the aggregated deadheading variables:

$$\begin{aligned} x_e &= \text{total number of times edge } e \text{ is deadheaded} \\ y_a &= \text{total number of times arc } a \text{ is deadheaded} \end{aligned}$$

In a standard Set Partitioning model, the pricing problem, defined by (IP Pricing), returns a vector containing the values of $([h_i^r], [s_e^r], [t_a^r], 1)$ along with the cost c_r. In our combined model, we let the pricing problem, defined by (IP Pricing), return a vector containing the values of $([u_e^r], [w_a^r], [h_i^r], [s_e^r], [t_a^r], 1)$ along with the cost c_r where

$$\begin{aligned} u_e^r &= \text{number of times the route of column } r \text{ deadheads edge } e \\ w_a^r &= \text{number of times the route of column } r \text{ deadheads arc } a \\ h_i^r &= 1 \text{ if route } r \text{ services node } i, \text{ and 0 otherwise.} \\ s_e^r &= 1 \text{ if route } r \text{ services edge } e, \text{ and 0 otherwise.} \\ t_a^r &= 1 \text{ if route } r \text{ services arc } a, \text{ and 0 otherwise.} \end{aligned}$$

Note that in the implementation the latter three are changed to contain the number of times route r services the nodes i, edges e, and arcs a respectively.

With this vector, we can state a combined model for the MCGRP where constraints (2.22) - (2.26) are identical to the complete set of constraints in (IP Master), i.e., constraints (2.15) - (2.19). Constraints (2.20) and (2.21) connect the route variables λ^r to the aggregated deadheading variables x_e and y_a such that x_e (y_a) equals the value of q_e^r (h_a^r) from the active route columns.

(IP Combined)

$$\text{minimize} \quad \sum_{e \in E} c_e x_e + \sum_{a \in A} c_a y_a$$

$$\text{s.t.} \quad \sum_{r \in \Omega} u_e^r \lambda^r - x_e = 0, \forall e \in E \tag{2.20}$$

$$\sum_{r \in \Omega} w_a^r \lambda^r - y_a = 0, \forall a \in A \tag{2.21}$$

$$\sum_{r \in \Omega} h_i^r \lambda^r = 1, \forall i \in N_R \tag{2.22}$$

$$\sum_{r \in \Omega} s_e^r \lambda^r = 1, \forall e \in E_R \tag{2.23}$$

$$\sum_{r \in \Omega} t_a^r \lambda^r = 1, \forall a \in A_R \tag{2.24}$$

$$\sum_{r \in \Omega} \lambda^r \leq |K| \tag{2.25}$$

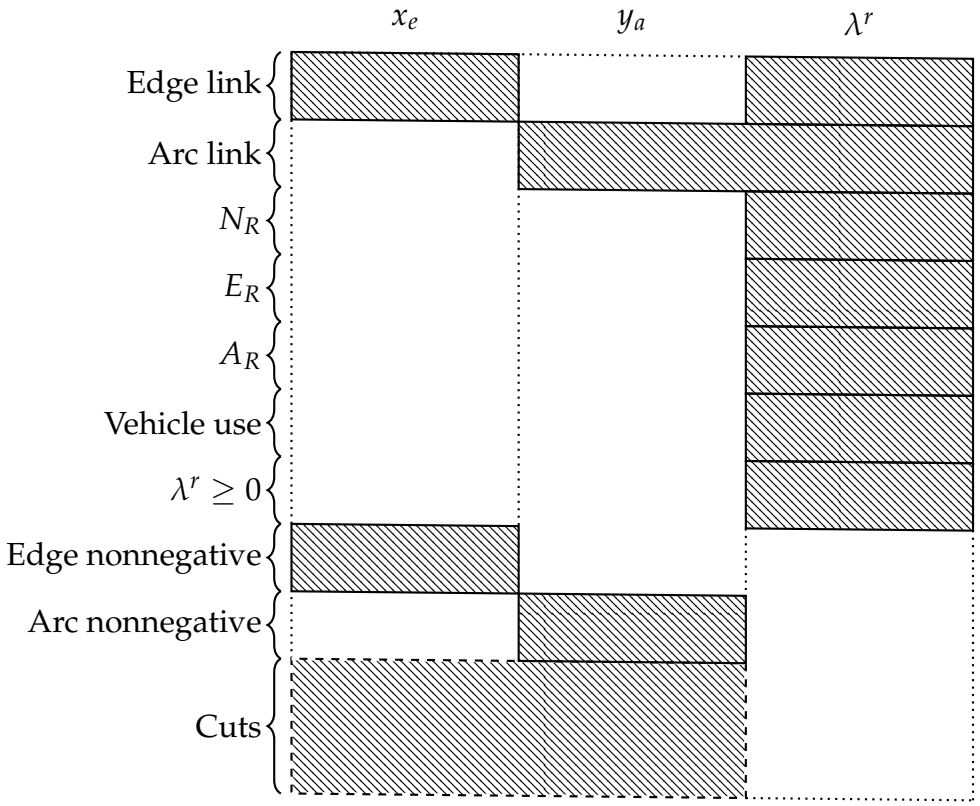
$$\lambda^r \geq 0, \text{integer } \forall r \in \Omega \tag{2.26}$$

$$x_e \geq 0, \text{integer } \forall e \in E \tag{2.27}$$

$$y_a \geq 0, \text{integer } \forall a \in A \tag{2.28}$$

Figure 2.1 shows the structure of (IP Combined). The structure clearly shows that the pricing problem is only affected by the top part of the model and remains unaffected by valid inequalities added to the model as a cut using only the aggregated variables x_e and y_a. Therefore, the valid inequalities from Belenguer and Benavent (2003) which are based on aggregated variables can, when extended to the MCGRP, be added to (IP Combined) without affecting the structure of the pricing problem used for column generation.

Figure 2.1: Structure of (IP Combined)



Due to constraints (2.20) and (2.21), the objective function of the pricing problem changes slightly compared to (IP Pricing). For the LP relaxation of (IP Combined), let $\boldsymbol{\phi} = [\phi_e]$ and $\boldsymbol{\psi} = [\psi_a]$ be the vectors of dual variables corresponding to constraints (2.20) and (2.21). The traversal costs are now associated with the x_e and y_a variables and thus not directly included in the pricing problem.

As for (IP Master), we let $\boldsymbol{\pi} = [\pi_i]$, $\boldsymbol{\mu} = [\mu_e]$, and $\boldsymbol{\rho} = [\rho_a]$ be the vectors of dual variables corresponding to constraints (2.22), (2.23), and (2.24), respectively. Finally, let σ be the dual variable corresponding to constraint (2.25). Then the Pricing Problem is given as

(IP C-Pricing)

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E_R} (-\mu_e)\tilde{x}_e + \sum_{e \in E} (-\phi_e)x_e + \sum_{a \in A_R} (-\rho_a)\tilde{y}_a \\ & + \sum_{a \in A} (-\psi_a)y_a + \sum_{i \in V_R} (-\pi_i)\tilde{z}_i - \sigma \\ \text{s.t.} \quad & (2.4) - (2.14) \end{aligned}$$

2.3.6 Valid Inequalities

The most commonly used cuts found in the literature are the *Capacity constraints* similar to those of Belenguer and Benavent (2003). For any set $S \subseteq N'$, these constraints ensure that a sufficient number of vehicles enters and leaves the induced graph $G(S)$. For the MCGRP, the capacity constraints are given by.

$$\sum_{e \in \delta^u(S)} x_e + \sum_{a \in \delta^+(S) \cup \delta^-(S)} y_a \geq 2\gamma(S) - |\delta_R(S)| \;\; \forall S \subseteq N' \tag{2.29}$$

The *Odd Edge Cutset constraints* from Belenguer and Benavent (2003) ensure that for any set S, if the cutset contains an odd number of required links, then at least one deadheading will be performed. For the MCGRP, the Odd Edge Cutset constraint is

$$\sum_{e \in \delta^u(S)} x_e + \sum_{a \in \delta^+(S) \cup \delta^-(S)} y_a \geq 1 \;\; \forall S \subseteq N' \text{ with } |\delta_R(S)| \text{ odd} \tag{2.30}$$

In some cases, (2.30) is violated while (2.29) is respected, and in other cases, the opposite is true. This shows that both inequalities are useful.

In (IP 2-index), constraint (2.8) is a balance constraint which ensures a feasible balance between incoming and outgoing arcs for a set of nodes. These constraints are also

valid in our combined model. In (2.31) and (2.32) we state them in a form suitable for (IP Combined) and refer to them as *Balance inequalities*.

$$\sum_{e \in \delta^u(S)} x_e + \sum_{a \in \delta^-(S)} y_a + |\delta_R^u(S) \cup \delta_R^-(S)| \geq \sum_{a \in \delta^+(S)} y_a + |\delta_R^+(S)|, \forall S \subseteq N' \tag{2.31}$$

$$\sum_{e \in \delta^u(S)} x_e + \sum_{a \in \delta^+(S)} y_a + |\delta_R^u(S) \cup \delta_R^+(S)| \geq \sum_{a \in \delta^-(S)} y_a + |\delta_R^-(S)|, \forall S \subseteq N' \tag{2.32}$$

In Belenguer and Benavent (2003), the authors present *Disjoint path inequalities #1 (DP1)* for the CARP. We now describe how these can be extended to work for the MCGRP. For proof of validity, we refer the reader to Belenguer and Benavent (2003).

For $S \subseteq N'$ with $2\gamma(S) \geq \delta_R(S)$, let $E' \subseteq E(N \setminus S)$ and $A' \subseteq A(N \setminus S)$. Define $T \subseteq N \setminus S$ to be the set of nodes which are endpoints of an edge or arc in $\delta(S)$.

Define

$$\alpha(S) = \begin{cases} \max\{2\gamma(S) - |\delta_R(S)|, 1\} & \text{if } |\delta_R(S)| \text{ is odd} \\ \max\{2\gamma(S) - |\delta_R(S)|, 0\} & \text{if } |\delta_R(S)| \text{ is even} \end{cases}$$

The matching network, $\tilde{G}(\tilde{N}, \tilde{E}, \tilde{A},)$ is defined as follows: If $\alpha(S) = 0$, let $\tilde{N} = N \setminus S$, let $\tilde{E} = E(N \setminus S)$, and let $\tilde{A} = A(N \setminus S)$. Alternatively, if $\alpha(S) > 0$, let $\tilde{N} = \{N \setminus S\} \cup \{s\}$, where s is an artificial node. Let $\tilde{E} = E(N \setminus S)$, and let $\tilde{A} = A(N \setminus S) \cup \{(i, s)\ \forall i \in T\}$.

The depot node has a supply of $2\gamma(S)$ units. Each node, $i \in T$ has a demand equal to the total number of edges and arcs in $\delta_R(S)$ incident to i. Finally, if s is defined, it has a demand of $\alpha(S)$.

Define the costs in $\tilde{G}$ as

$$\tilde{c}_e = \begin{cases} q_e & \text{if } e \in E' \\ 0 & \text{otherwise} \end{cases}$$

and

$$\tilde{c}_a = \begin{cases} q_a & \text{if } a \in A' \\ 0 & \text{otherwise} \end{cases}$$

Define the flow capacity as

$$\tilde{u}_e = \begin{cases} 1 & \text{if } e \in E' \cap E_R \\ 0 & \text{if } e \in E' \setminus E_R \\ \infty & \text{otherwise} \end{cases}$$

and

$$\tilde{u}_a = \begin{cases} 1 & \text{if } a \in A' \cap A_R \\ 0 & \text{if } a \in A' \setminus A_R \\ \infty & \text{otherwise} \end{cases}$$

Let $F(S, E', A')$ be the cost of a minimum cost flow in $\tilde{G}$. For the MCGRP, the *Disjoint path inequalities #1* are given by

Disjoint path inequalities #1; Let $S \subset N'$ with $2\gamma(S) \geq |\delta_R(S)|$ and let $E' \subseteq E(N \setminus S)$ and $A' \subseteq A(N \setminus S)$. If $F(S, E', A') + Q(S) > \gamma(S)W$ then

$$2(\sum_{e \in E'} x_e + \sum_{a \in A'} y_a) + (\sum_{e \in \delta^u(S)} x_e + \sum_{a \in \delta^+(S) \cup \delta^-(S)} y_a) \geq \alpha(S) + 2 \tag{2.33}$$

2.4 Branch-and-Cut-and-Price Algorithm

This sections describes our Branch-and-Cut-and-Price algorithm in detail. Firstly, in Section 2.4.1, we describe our algorithms for identifying valid inequalities. In Section 2.4.2, we describe our branching strategy, and in Section 2.4.3, we describe our procedure for generating columns. Finally, in Section 2.4.4, we describe stabilization techniques for our algorithm.

2.4.1 Cut Generation

We presented a number of valid inequalities for the MCGRP. In this section, we describe algorithms for the cuts used in our Branch-and-Cut-and-Price algorithm. When the initial set of columns has been generated, we use two algorithms for adding constraints to the root node.

In Algorithm 7, we consider single sets that consist of a single node i. We notice that if $Q(i)$ exceeds the vehicle capacity, the Capacity constraint can be useful. If an odd number of required links are adjacent to i, the Odd Edge Cutset constraint will help guide the solution. Finally, when i is unbalanced with respect to incoming and outgoing arcs, the Balance inequalities can be useful. Our computational experiments indicate that all three types of cuts help improve the running time of our algorithm.

Algorithm 7

1: **for** $i \in N'$ **do**
2: Add (2.29), (2.31), and (2.32) based on $S = \{i\}$ to the model
3: **if** $|\delta_R(i)|$ is odd **then**
4: Add (2.30) based on $S = \{i\}$ to the model
5: **end if**
6: **end for**

In Algorithm 8, we check the Capacity constraints and the Balance inequalities for a number of mutually disjoint cuts $(U_h, N \setminus U_h)$ with $U_1 = \{1\}$ and $U_1 \subset U_2 \subset U_3 \subset U_4$. Note that Bach et al. (2013) use precisely the same cuts to identify a similar capacity constraint in the combinatorial lower bound for the same problem. Furthermore, the algorithm also checks the inequalities for a number of cuts obtained by progressively adding the nodes in $U_{h+1} \setminus U_h$ to U_h. Finally, Algorithm 8 also checks the Odd Edge Cutset constraint for each of the cuts considered.

Algorithm 8

Let $U = \{1\}$
Set $S = N \setminus U$
while $S \neq \emptyset$ **do**
  Let $V \subseteq S$ be the set of nodes i that are adjacent to a node j in U
  for $i \in V$ **do**
    Set $U = U \cup \{i\}$
    Set $S = S \setminus \{i\}$
    Let $G_l^s = (S_l, E_l(S), A_l(S))$ be the connected components of $G(S) = (S, E(S), A(S))$
    for each G_l^s **do**
      Add (2.29), (2.31), and (2.32) based on S_l to the model
      if $|\delta_R(S_l)|$ is odd **then**
        Add (2.30) based on S_l to the model
      end if
    end for
  end for
end while

2.4.2 Branching

Branch-and-Price algorithms for routing problems often apply the branching strategy known as follow-on branching, originally based on Ryan-Foster branching. The idea is that, for a pair of required entities a and b, two branches are created in the branching tree. In the left branch (1-branch), the connection between a and b is fixed, i.e., if a is serviced, then b must be serviced immediately after and if b is serviced, it must be preceded by a. In the right branch (0-branch) this connection is forbidden, i.e., b may not be serviced immediately after a.

In our Branch-and-Cut-and-Price algorithm, we branch on follow-on in a slightly different way. Because we work directly on the original sparse graph as suggested by Letchford and Oukil (2009). Bode and Irnich (2012) suggest a branching scheme for follow-on branching on this sparse graph for the CARP. We use a similar follow-on branching scheme. The details on how the follow-on branching is handled in the pricing algorithm is described in Section 2.4.3.

Let $a, b \in N_R \cup E_R \cup A_R$ be two required tasks. We consider three branches. In the 1-branch, we fix the connection between a and b such that a must always be serviced immediately before b and furthermore we require that any deadheading between a and b occurs along a shortest path using costs from the original graph. This branch also includes the situation where no deadheading between a and b is needed because they are connected.

In the 0-branch, any solution where b follows immediately after a is forbidden. This effectively removes 1-cycles as 0-branches are introduced when necessary. This can be compared to Boland et al. (2006); Righini and Salani (2008) where resources are introduced ad-hoc to solve the Elementary-SPPRC.

In the 'middle' branch, we fix the connection between a and b and consider all solutions containing deadheading using a path longer than a shortest path between a and b. Clearly, no solution obtained in the middle branch can be optimal. Therefore this branch can be discarded without further investigation.

We create only the 1-branch and the 0-branch in our implementation. It is, however, important for us to discard solutions contained in the middle branch because in the modification of the network we take advantage of the fact that deadheading always occurs via a shortest path with respect to the real costs c_e and c_a, regardless of the values of the dual variables at a specific stage of the algorithm. We explain this further in Section 2.4.3. We use a depth first strategy in the branching tree where the 1-branch is prioritized.

2.4.3 Column Generation

To generate columns, we consider (IP C-Pricing). We solve it as a Shortest Path Problem with Resource Constraints (SPPRC) using a labeling algorithm. See Desrochers and Soumis (1988); Irnich and Desaulniers (2005) for more on the SPPRC, and for an implementation in a column generation setup, see Kallehauge et al. (2005). As in Letch-

ford and Oukil (2009); Bode and Irnich (2012), we work directly on the original graph G which is sparse for the MCGRP. To solve the SPPRC we require the dual variables $\boldsymbol{\phi} < 0$ and $\boldsymbol{\psi} < 0$ to avoid infinite deadheading cycles. This is further explained in Section 2.4.4.

Because all traversal costs in our model are associated with the x_e and y_a variables, the pricing problem (IP C-Pricing) contains no such cost. Therefore, the graph we use for column generation has costs $-\phi_e$ and $-\psi_a$ for deadheading edges and arcs, respectively, and a cost of $-\pi_i$, $-\mu_e$, and $-\rho_a$ for servicing nodes i, edges e, and arcs a, respectively. We define $SP(i,j)$ to be the set of links on a shortest path from i to j in the original network with respect to the original traversal costs c_e and c_a.

Each label has two resources, a capacity resource that is constrained by the maximum capacity of the vehicle, and a 'tabu' resource. A label L_a dominates a label L_b with respect to capacity if $Capcity(L_a) \leq Capcity(L_b)$. The 'tabu' resource, is more complicated as it is a list of entities we cannot service next. If $Tabu(L_a) = \emptyset$ it can dominate any other label with respect to the 'tabu' resource. If $Tabu(L_a) \neq \emptyset$ then all entities in the 'tabu' resource $Tabu(L_a)$ must be present in $Tabu(L_b)$ before label L_a can dominate label L_b. For an example $Tabu(L_a) = \{1,3,5\}$ can dominate $Tabu(L_b) = \{1,3,4,5\}$, but not $Tabu(L_c) = \{3,4,5,7\}$. We will further explain the 'tabu' resource in the following section.

Let $i \in N$ and let L_i be a label on i. We extend L_i as follows. For all $a = (i,j) \in \delta^+(i) \cap \{A \setminus A_R\}$ we create a deadhead extension of L_i to j, i.e., an extension where the arc a is deadheaded. For all $a = (i,j) \in \delta_R^+(i)$ we create a deadheading extension of L_i to j and if it is resource feasible with respect to capacity, we also create a service extension of L_i to j. In the latter, the arc a is serviced. For edges in $\delta(i)$, we extend in a similar way. For any of the above extensions of edges and arcs, if node $j \in N_R$, i.e., the node we extend to is required, we create a copy of the extended labels where we further service j if this is resource feasible with respect to capacity. The maximum

number of new labels created from L_i on node j for each edge or arc leaving node i is therefore four.

Modifying the Graph

In our pricing problem we must handle the branching rules described in Section 2.4.2. We first describe how the individual follow-on branchings are handled and then we describe the handling of multiple interacting branching rules.

We first consider the 0-branch rule forbidding the service of a required task $a \in N_R \cup E_R \cup A_R$ to follow immediately after another required task $b \in N_R \cup E_R \cup A_R$. Note that this is the only rule in effect in the following example. We model this as a 'tabu' resource denoted as $Tabu(L) = a$ where a is the required task and L a label as follows.

If a is a node $i \in N_R$, we simply set a mark on node i stating that labels L_i servicing i may not be extended to b. Such a label will have a 'tabu' resource $Tabu(L_i) = b$. If a is an arc $(i, j) \in A_R$, we set a similar mark on the arc. When a label is service extended along a there are two cases. For the extension which does not service the node j we set $Tabu(L_i) = b$. For the extension which service j, we set $Tabu(L_j) = \emptyset$ as we have now serviced another task. The latter is only relevant if node j is required. If j was required and also equal to b, then the service of j following a is not allowed and the extension is not performed. If a is an edge $\{i, j\} \in E_R$, we set a similar mark on the edge and treat it like the arcs keeping in mind that it can be serviced in either direction. The label L will always hold the 'tabu list' of the latest serviced entity. If we extend without any service involved the 'tabu' resource of the previous label is copied to the new label. If we service extend, the'tabu' resource of the previous label is replaced by the 'tabu' resource of the last required entity.

Consider now a 1-branch rule requiring entity b to follow immediately after entity a. We will first assume that both entities are arcs. Let $a = (i_a, j_a) \in A_R$ and

$b = (i_b, j_b) \in A_R$. We (temporarily) insert a new required arc (i_a, j_b) with demand equal to $q_a + q_b$ and cost equal to $\rho_a + \sum_{e \in SP(j_a, i_b)} \phi_e + \sum_{a \in SP(j_a, i_b)} \psi_a + \rho_b$. Furthermore, we (temporarily) deactivate the arcs a and b as required arcs, but keep them as deadheading arcs. Note that due to our branching rule, we only have to consider solutions where all deadheading is performed using shortest paths in the original network, independently of any dual cost of the deadheading edges and arcs. All shortest paths can therefore be calculated a priori.

If a and b are required nodes, we follow the same procedure and insert an arc between the nodes. The nodes are not discarded. Instead, a mark is set in node a stating that the new arc where b is serviced at the end must be used if a has been serviced. If a is a required node and b is an edge/arc, then a rule is inserted in node a requiring the new arc connecting the edge/arc b and the node a must be used if a is serviced. Similarly if a is an edge/arc and b is a node then a is replaced by a new arc connecting it to b while forcing service upon arrival.

If both a and b are edges, four new arcs with capacity $q_a + q_b$ are created to handle the flexibility of traversal of edges. 1) Arc (i_a, j_b) with cost $\mu_a + \sum_{e \in SP(j_a, i_b)} \phi_e + \sum_{a \in SP(j_a, i_b)} \psi_a + \mu_b$. 2) Arc (i_a, i_b) with cost $\mu_a + \sum_{e \in SP(j_a, j_b)} \phi_e + \sum_{a \in SP(j_a, j_b)} \psi_a + \mu_b$. 3) Arc (j_a, j_b) with cost $\mu_a + \sum_{e \in SP(i_a, i_b)} \phi_e + \sum_{a \in SP(i_a, i_b)} \psi_a + \mu_b$. 4) Arc (j_a, i_b) with cost $\mu_a + \sum_{e \in SP(i_a, j_b)} \phi_e + \sum_{a \in SP(i_a, j_b)} \psi_a + \mu_b$. Again the two original edges a and b are deactivated as required edges, but kept as deadheading edges. In case only a or b is an edge, only two of these new edges are created.

As the algorithm proceeds, several such 1-branch rules may result in several of the created arcs needing to be concatenated. For arcs, the concatenation is easily handled. If a, b, and c are all arcs and rules $a \rightarrow b$ and $b \rightarrow c$ are to be concatenated. Then we insert a new required arc (i_a, j_c) with demand equal to $q_a + q_b + q_c$ and cost calculated in the same way as when merging two arcs. As for the case with two arcs, we disable all other required arcs involving the arcs being merged, a, b, and c. This can easily be

extended to a case containing four or more arcs.

When merging edges or a combination of arcs and edges, there is a risk that the number of new arcs will explode. To avoid this, we use the dynamic programming approach presented in Wøhlk (2005) for finding a shortest path through such edges while taking the different orientations into account. This method results in at most four new arcs independently of the number of connections being concatenated. If we assume a and c to be edges and b to be either an edge or arc, and rules $a \rightarrow b$ and $b \rightarrow c$ are to be concatenated. Then the new arcs inserted will be (i_a, j_c), (i_a, i_c), (j_a, j_c), and (j_a, i_c) using the shortest path between the first and last required edge/arc being concatenated. This implies a service direction for any edges in between the first and last edge/arc, which is determined by the optimal orientation of those.

If any combination of branch rules involves the same required node no concatenation is necessary. For example, if a and c are edges/arcs and b is a node, with rules $a \rightarrow b$ and $b \rightarrow c$. Then, the first rule states that a label extended along arc a will end in a service node b, and the second rule will deem that any label servicing node b must be extended along the arc inserted by rule $b \rightarrow c$.

Having considered the interaction of multiple 1-branch rules, let us now consider any combination of 0-branch rules. Here, if entities a, b, and c are in the branch rules $a \not\rightarrow b$ and $b \not\rightarrow c$, then a will have a 'tabu list' of b, and b will have a 'tabu list' of c. If rules $a \not\rightarrow b$ and $a \not\rightarrow c$, then a will have a 'tabu list' of $\{b, c\}$.

Finally, we address the cases where 1-branch and 0-branch rules interact. For branch rules $a \rightarrow b$, $b \not\rightarrow c$, and $c \not\rightarrow d$, no concatenation is necessary. The new arc representing $a \rightarrow b$ will be given a 'tabu list' with c, and $c \not\rightarrow d$ will be unaffected. Thus with regard to interactions all 1-branches are first concatenated and then the relevant 0-branches are applied.

The major weakness of this approach to branching is the 0-branches. When the size of the 'tabu' lists is larger than one, they become more complex to dominate and thus

affect the speed of the algorithm.

2.4.4 Stabilization

There are two main reasons for using stabilization in an algorithm based on column generation.

Firstly, the values of the dual variables can fluctuate quite significantly in a column generation algorithm, especially in a Set Partitioning Formulation. This may cause the pricing problem to generate columns which are barely useful. In some situations, it is possible to stabilize the process by imposing bounds on the values of the dual variables. This often results in an algorithm which converges faster towards the final values of the dual variables, see du Merle et al. (1999).

Secondly, in some cases the dual variables take undesirable values which negatively impact the structure of the pricing problem. Bounding the allowed values of the dual variables can eliminate this problem.

In (IP Combined), the values of $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ can vary freely because the corresponding constraints are equality constraints. In the pure model as stated in (IP Combined), the values of these dual variables can be kept non-positive by stabilization, but cuts are added to the model and these may cause dual variables to obtain positive values. Assume that e is an edge for which $\phi_e > 0$. This means that the contribution of the deadheading variable x_e to the objective function is $-\phi_e < 0$. Hence, the pricing algorithm can produce columns of cost $-\infty$ by repeated deadheading of e. We relax constraints (2.20) and (2.21) to $\leq$. This forces $\phi_e \leq 0 \; \forall e \in E$ and $\psi_a \leq 0 \; \forall a \in A$ and stabilizes our model by ensuring that the above situation is avoided.

Intuitively, this relaxation means that we can pay for deadheading an edge e (or an arc) by increasing the value of x_e without using the edge in a route variable λ_r. We say that we pay the price c_e for not having to use e in a route. Note that the very purpose of our valid inequalities is to force the use of deadheading variables. It is important

to note that, in the pricing problem (IP C-Pricing), constraints (2.8) and (2.9) ensure that every route generated is connected. Therefore, in any integer solution, constraints (2.20) and (2.21) will hold with equality and we pay for exactly those edges (and arcs) used in the route.

Bode and Irnich (2012) discuss an alternative approach with the implementation of simple cycle variables in the master problem, to address the issue of infinite cycles for the CARP. Their approach is based on dual-optimal inequalities presented by Ben Amor et al. (2006). In their approach it is necessary to ensure non-negative reduced cost for deadheading on edges $e \in E$. The variables z_e represent the simple deadheading cycles (e, e), i.e., nodes (i, j, i). The cost on this cycle is $2c_e$, twice the cost of deadheading e, the addition of these variables to the master problem forms dual inequalities $\beta_e \leq c_e, \forall e \in E$, where β_e is the sum of all dual variables related to edge e. This has the effect of restricting the reduced cost of deadheading $\tilde{c}_e$ to $\tilde{c}_e \geq 0$. This in turn corresponds to our approach where $\phi_e \leq 0$, forces the reduced cost of the edge to be non-negative.

The comparison of the approaches is not as straightforward with respect to simple tours for arcs. The approach of Bode and Irnich (2012) implies that the sum of the reduced costs along the shortest deadheading cycle is non-negative. For edges, eliminating the simple cycles (e, e) ensure that there are no negative deadheading cycles if the graph contains only edges and nodes. There are many ways to form the shortest deadheading cycle for an arc $a = (i, j)$, it could be $b = (j, i)$. But, ensuring $\psi_a + \psi_b \leq 0$ does not guarantee that other negative reduced cost deadheading cycles involving arc a do not exist. Assume, we have two additional arcs $c = (j, k), d = (k, i)$, then (a, c, d) form a deadheading cycle. If ψ_b is negative, ψ_a can be positive, and therefore even with $\psi_c \leq 0$ and $\psi_d \leq 0$ we can have $\psi_c + \psi_d + \psi_a > 0$, which forms a negative reduced costs cycle.

In our approach we have all $\phi_e \leq 0 \ \forall e \in E$ and $\psi_a \leq 0 \ \forall a \in A$ which guarantee

that there are no negative reduced cost cycles. There is however some slack when comparing to the approach of using the deadheading cycles. An implementation using the simple cycle variables of Bode and Irnich (2012) extended to the edge and arc setup would possibly improve our method. However, this has not been considered.

2.5 Computational Results

The results presented in Tables 2.1-2.6 are all computed on an HP EliteBook with an Intel Core 2 Duo CPU P8700 @2.53 GHz and 4 GB of RAM. The algorithm is implemented in C++ as a sequential algorithm, and we use IBM ILOG CLPLEX 12.4 to solve the LP-relaxation. When comparing run time, we only compare to Bosco et al. (2013) who presented the only existing exact algorithm for the problem. They report results computed using a PC equipped with 2 Intel Xeon Quad Core CPUs @3.0 GHz with 6 GB RAM.

We mark optimal solutions with '*' instead of their gap. We use '‡'('†') to indicate that we reached a time limit of 3(6) hours. If no results were reported by Bosco et al. (2013) or no results obtained by our algorithm we use a '-'. In the tables, 'Root' refers to the objective value of the LP-relaxation in the root node of the branching tree. 'Nodes' are the number of branching nodes generated and 'CG' is the total number of columns generated. We refer to the results obtained in this chapter as BLW. For the best known solutions we have marked the solution with a letter indicating the corresponding reference. We use A for Prins and Bouchenoua (2004), B for Kokubugata et al. (2007), C for Hasle et al. (2012), D for Bach et al. (2013), E for Bosco et al. (2013), and F for Gaze (2013). For both upper and lower bounds, we indicate with bold if our result is better than the previous best known values which are obtained from the NEARP webpage (SINTEF, 2013). The gap is calculated as $(UB - LB)/((UB + LB)/2)$ unless stated otherwise..

Table 2.1: Computational results for CBMix

Instance	Best known: UB	LB	Gap[1]	BLW: UB	LB	Root	Gap[1]	Time	Nodes	CG
CBMix1	2589[A]	2513[F]	2.98	**2569**	**2547**	2516.5	0.86	†	10814	106454
CBMix2	12220[B]	9742[D]	22.57	-	**11487**	11486.3	-	†	14	35577
CBMix3	3643[C]	3014[D]	18.90	3684	**3514**	3500.2	4.72	†	2870	108494
CBMix4	7583[A]	5302[D]	35.41	**7582**	**7300**	7249.6	3.79	†	3670	47692
CBMix5	4531[B]	3789[D]	17.84	5548	**4387**	4379.6	23.37	†	266	16661
CBMix6	7087[A]	5201[D]	30.70	7643	**6738**	6737.4	12.59	†	160	22230
CBMix7	9607[C]	7296[D]	27.34	-	**9046**	9043.2	-	†	44	31163
CBMix8	10524[B]	7956[D]	27.79	12114	**9976**	9975.7	19.36	†	578	43538
CBMix9	4038[A]	3460[D]	15.42	4044	**3837**	3789.6	5.25	†	7780	38373
CBMix10	7582[A]	6432[D]	16.41	7614	**7343**	7324.0	3.62	†	2380	36900
CBMix11	4494[A]	3031[D]	38.88	-	**4318**	4317.3	-	†	14	10633
CBMix12	3138[E]	3138[D]	*	3138	3138	3138.0	*	2037.2	610	31311
CBMix13	9110[A]	6524[D]	33.08	-	**8681**	8678.5	-	†	138	22713
CBMix14	8566[A]	5731[D]	39.66	-	**8205**	8204.5	-	†	122	18139
CBMix15	8280[B]	6318[D]	26.88	8355	**8013**	7922.1	4.18	†	7464	32021
CBMix16	8886[B]	7416[D]	18.03	-	**8446**	8445.4	-	†	10	21475
CBMix17	4037[A]	3654[D]	9.96	-	**3943**	3941.4	-	†	82	9131
CBMix18	7098[B]	6089[D]	15.30	7137	**6856**	6845.6	4.02	†	1308	100641
CBMix19	16347[B]	11143[D]	37.86	-	**15628**	15621.3	-	†	222	45013
CBMix20	4844[A]	3452[D]	33.56	5068	**4647**	4640.3	8.67	†	254	40030
CBMix21	18069[B]	12474[D]	36.64	18201	**17295**	17259.7	5.10	†	2344	76287
CBMix22	1941[A]	1825[D]	6.16	1941	**1905**	1896.0	1.87	†	1476	46446
CBMix23	780[A]	780[E]	*	780	716	677.3	8.56	†	30758	224494

We have tested our algorithm on a total of 215 benchmark instances. Table 2.1 shows the results obtained for the set of 23 CBMix instances provided by Prins and Bouchenoua (2004). These instances are the oldest and have therefore been used in several papers, in particular with regards to upper bounds. For these instances, we improve the best known upper bound for two instances and the best known lower bound for 21 instances. For 10 instances, our upper bound is worse than the best known, and for 8 instances, our algorithm does not reach a feasible solution. The main reason for this is that our algorithm does not include a construction heuristic to provide an initial solution. For the instances where an upper bound is reached, the gap

Table 2.2: Computational results for BHW instances

Instance	Best known:			BLW:						
	UB	LB	Gap[1]	UB	LB	Root	Gap[1]	Time	Nodes	CG
BHW1	337[C]	324[D]	3.93	337	**337**	332.0	*	987.9	5190	16171
BHW2	470[C]	470[D]	*	470	470	470.0	*	0.2	10	416
BHW3	415[C]	405[F]	2.44	415	**415**	401.6	*	29.8	270	2786
BHW4	240[C]	240[D]	*	240	240	240.0	*	15.6	20	5256
BHW5	506[C]	502[D]	0.79	-	500	500.0	-	†	6	104203
BHW6	388[C]	388[D]	*	-	385	385.0	-	†	110	349404
BHW7	1094[C]	930[D]	16.21	-	**1047**	1046.3	-	†	172	70895
BHW8	672[C]	644[D]	4.26	-	**665**	664.1	-	†	42	67386
BHW9	913[C]	791[D]	14.32	-	**858**	857.4	-	†	80	56247
BHW10	8556[C]	6810[D]	22.73	-	**8310**	8304.2	-	†	54	25927
BHW11	5021[C]	3986[D]	22.98	-	**4690**	4689.5	-	†	98	18099
BHW12	11042[C]	6346[D]	54.01	-	**10605**	10603.2	-	†	50	10681
BHW13	14510[C]	8746[D]	49.57	-	**13952**	13948.4	-	†	84	20978
BHW14	25194[C]	17762[D]	34.60	-	**24377**	24375.3	-	†	130	13238
BHW15	15509[C]	12193[D]	23.94	16929	**15130**	15124.4	11.2	†	544	17727
BHW16	44527[C]	26014[D]	52.49	-	**42506**	42486.0	-	†	372	38926
BHW17	26768[C]	15396[D]	53.94	-	**25570**	25567.1	-	†	142	16363
BHW18	15833[C]	11202[D]	34.26	16774	**14840**	14837.9	12.2	†	348	34111
BHW19	9424[C]	7080[D]	28.41	10942	**9197**	9193.9	17.3	†	708	44800
BHW20	16625[C]	10730[D]	43.10	-	-	-	-	†	0	43701

between upper and lower bound is significantly improved.

The BHW instances (Bach et al., 2013) were presented later than the CBMix instances and have thus been used in fewer papers. Table 2.2 shows the results obtained for these 20 instances. We prove optimality of 4 instances. For two of these instances, optimality is proved for the first time. For the remaining instances, we reach our time limit of 6 hours. For the majority of these instances, we improve the lower bound but do not reach a feasible solution. The DI-NEARP instances also presented in Bach et al. (2013) are larger than the remaining instances and have not been used in our tests.

Bosco et al. (2013) present a Mggdb set of 138 instances in total divided into 6 subsets. Results obtained for these instances are presented in Tables 2.3 - 2.5. Out of the 138 instances, we prove optimality for 115 instances. For 31 of these, optimality is proved

Table 2.3: Computational results for Mggdb $\{0.25, 0.30\}$

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.25_1	280^E	280^E	*	280	251.0	*	79.4	280	280	275.8	*	57.8	890	4981
mggdb_0.25_2	349^E	339^F	2.91	349	317.0	5.6	†	352	**346**	341.8	1.7	‡	12250	65283
mggdb_0.25_3	278^E	278^E	*	278	253.5	*	923.9	278	278	270.4	*	601.4	2474	16939
mggdb_0.25_4	289^E	289^E	*	289	272.0	*	22.7	289	289	285.5	*	0.8	24	750
mggdb_0.25_5	394^E	384^F	2.57	394	364.0	6.4	†	394	**394**	391.1	*	2.3	90	1474
mggdb_0.25_6	292^E	292^E	*	292	284.0	*	14.9	292	292	292.0	*	0.2	2	477
mggdb_0.25_7	290^E	290^E	*	290	275.0	*	40.4	290	290	287.4	*	0.4	22	569
mggdb_0.25_8	356^F	328^F	8.19	-	-	-	-	**336**	**333**	332.0	0.9	‡	12776	56292
mggdb_0.25_9	331^F	305^F	8.18	-	-	-	-	**309**	**308**	303.4	0.3	‡	10504	41103
mggdb_0.25_10	265^E	265^E	*	265	265.0	*	2.8	265	265	265.0	*	1.2	6	1345
mggdb_0.25_11	356^E	347^F	2.56	356	345.0	3.1	†	356	**349**	347.9	2.0	‡	934	132538
mggdb_0.25_12	459^E	456^F	0.66	459	400.0	3.7	†	459	**459**	454.5	*	4.9	32	2614
mggdb_0.25_13	388^E	388^F	*	388	374.0	3.4	†	388	388	385.6	*	41.4	32	3984
mggdb_0.25_14	107^E	107^E	*	107	107.0	*	1.4	107	107	107.0	*	2.3	14	2753
mggdb_0.25_15	55^E	55^E	*	55	55.0	*	0.5	55	55	55.0	*	3.2	14	3235
mggdb_0.25_16	98^E	98^E	*	98	98.0	*	5.0	98	98	97.0	*	7121.4	4844	99008
mggdb_0.25_17	71^E	71^E	*	71	71.0	*	1.0	71	71	71.0	*	23.4	74	8637
mggdb_0.25_18	144^E	139^E	3.53	144	139.0	3.5	†	144	**140**	139.7	2.8	‡	404	177308
mggdb_0.25_19	53^E	53^E	*	53	47.0	*	1.1	53	53	52.5	*	0.1	8	216
mggdb_0.25_20	116^E	116^E	*	116	116.0	*	7.8	116	116	116.0	*	6.3	6	2966
mggdb_0.25_21	146^E	145^E	0.69	146	145.0	0.7	†	146	**146**	146.0	*	25.9	10	6981
mggdb_0.25_22	172^F	160^F	7.23	-	-	-	-	**160**	160	160.0	*	104.3	14	15487
mggdb_0.25_23	234^F	181^F	25.54	-	-	-	-	**181**	181	181.0	*	1008.2	34	34933

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.30_1	273^E	273^E	*	273	238.3	*	4279.5	273	273	264.4	*	76.2	1346	4930
mggdb_0.30_2	301^E	294^F	2.35	301	270.0	5.7	†	301	**301**	295.9	*	229.5	1076	9509
mggdb_0.30_3	270^E	270^E	*	270	253.0	*	4447.9	270	270	267.2	*	1.1	62	810
mggdb_0.30_4	260^E	260^E	*	260	231.0	*	39.7	260	260	252.5	*	4.4	168	1992
mggdb_0.30_5	388^E	384^F	1.04	388	369.0	2.8	†	388	**388**	388.0	*	0.7	12	862
mggdb_0.30_6	276^E	276^E	*	276	235.0	*	443.4	276	276	268.4	*	198.6	2034	8079
mggdb_0.30_7	273^E	273^E	*	273	228.5	*	2870.3	273	273	256.8	*	1553.9	5984	20544
mggdb_0.30_8	352^F	329^F	6.75	-	-	-	-	**331**	328	321.6	0.9	‡	7666	45312
mggdb_0.30_9	290^F	278^F	4.23	-	-	-	-	**281**	**281**	275.4	*	3435.3	3668	24939
mggdb_0.30_10	242^E	242^E	*	242	228.0	*	12.1	242	242	237.5	*	439.9	866	20738
mggdb_0.30_11	387^E	382^F	1.30	387	381.0	1.6	†	-	382	381.3	-	‡	42	18809
mggdb_0.30_12	467^E	462^F	1.08	467	395.0	6.4	†	467	**467**	461.6	*	2.0	42	1153
mggdb_0.30_13	486^E	479^F	1.45	486	447.0	8.0	†	**483**	**483**	476.3	*	83.9	44	3617
mggdb_0.30_14	101^E	101^E	*	101	98.0	*	184.7	101	101	100.0	*	1.9	14	2237
mggdb_0.30_15	44^E	44^E	*	44	44.0	*	0.5	44	44	44.0	*	2.3	16	2684
mggdb_0.30_16	105^E	105^E	*	105	105.0	*	2.8	105	105	105.0	*	11.7	0	4782
mggdb_0.30_17	65^E	65^E	*	65	65.0	*	0.6	65	65	65.0	*	10.3	20	5360
mggdb_0.30_18	144^E	144^E	*	144	144.0	*	1.6	144	144	144.0	*	217.7	30	31897
mggdb_0.30_19	51^E	51^E	*	51	50.0	*	0.4	51	51	51.0	*	0.2	14	418
mggdb_0.30_20	94^E	94^E	*	94	91.0	*	27.3	94	94	91.8	*	10.7	28	2606
mggdb_0.30_21	121^E	121^F	*	121	120.0	0.8	†	121	121	120.5	*	89.5	16	9909
mggdb_0.30_22	156^F	151^F	3.26	-	-	-	-	**153**	**153**	152.2	*	374.4	26	18563
mggdb_0.30_23	188^F	167^F	11.83	-	-	-	-	**167**	167	166.9	*	677.8	20	24901

[1] Gap calculated as $(UB - LB)/((UB + LB)/2)$

[2] Bosco et al. (2013) do not report the explicit lower bound, but report a gap from cplex . Thus we have copied this from their article.

Table 2.4: Computational results for Mggdb $\{0.35, 0.40\}$

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.35_1	252^E	252^E	*	252	232.0	*	601.0	252	251	242.4	0.4	‡	1188	8183
mggdb_0.35_2	284^E	284^E	*	284	271.0	*	1435.2	284	284	284.0	*	1.4	2	451
mggdb_0.35_3	243^E	243^E	*	243	208.0	*	1701.0	243	243	235.8	*	185.7	1156	7941
mggdb_0.35_4	242^E	242^E	*	242	180.0	*	27.6	242	242	235.0	*	3.7	162	1500
mggdb_0.35_5	309^E	299^F	3.29	309	282.0	6.1	†	309	**309**	302.7	*	37.5	588	3748
mggdb_0.35_6	262^E	262^E	*	262	235.0	*	1552.4	262	262	246.8	*	1241.9	6988	17251
mggdb_0.35_7	272^E	272^E	*	272	247.0	*	412.7	272	272	266.0	*	68.6	590	5524
mggdb_0.35_8	338^F	311^F	8.32	-	-	-	-	**316**	**315**	309.8	0.3	‡	17850	58174
mggdb_0.35_9	275^F	260^F	5.61	-	-	-	-	**266**	**265**	259.1	0.4	‡	11504	49403
mggdb_0.35_10	268^E	268^E	*	268	258.0	*	813.6	268	267	265.8	0.4	‡	6206	160987
mggdb_0.35_11	303^E	303^E	*	303	293.0	*	3875.7	313	301	300.0	3.9	‡	116	30081
mggdb_0.35_12	461^E	461^E	*	461	369.0	*	5229.9	461	461	461.0	*	2.4	28	1502
mggdb_0.35_13	417^E	417^F	*	417	402.0	3.0	†	417	417	416.9	*	5.3	2	2676
mggdb_0.35_14	84^E	84^E	*	84	81.0	*	369.3	84	84	83.5	*	0.8	4	1446
mggdb_0.35_15	44^E	44^E	*	44	44.0	*	0.7	44	44	44.0	*	5.8	42	3738
mggdb_0.35_16	75^E	75^E	*	75	71.5	*	2271.4	75	75	73.2	*	193.9	266	13687
mggdb_0.35_17	62^E	62^E	*	62	62.0	*	0.9	62	62	62.0	*	5.1	12	3355
mggdb_0.35_18	135^E	135^E	*	135	135.0	*	0.4	135	135	135.0	*	81.2	36	15302
mggdb_0.35_19	47^E	47^E	*	51	51.0	*	0.2	51	**51**	51.0	*	0.1	10	298
mggdb_0.35_20	96^E	96^E	*	96	93.0	*	75.8	96	96	93.5	*	27.6	60	3844
mggdb_0.35_21	120^E	118^E	1.68	120	117.5	2.1	†	120	**120**	119.0	*	60.0	30	8532
mggdb_0.35_22	146^F	138^F	5.63	-	-	-	-	**139**	**139**	138.2	*	191.5	24	12882
mggdb_0.35_23	233^F	178^F	26.76	-	-	-	-	**179**	**179**	178.5	*	304.7	70	20049

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.40_1	279^E	279^E	*	279	248.3	*	392.9	279	279	269.9	*	93.0	1118	6369
mggdb_0.40_2	308^E	302^E	1.97	308	281.0	1.9	†	308	**308**	297.8	*	570.7	2770	14542
mggdb_0.40_3	225^E	225^E	*	225	208.0	*	88.2	225	225	221.1	*	3.7	166	2045
mggdb_0.40_4	238^E	238^E	*	238	205.0	*	9.8	238	238	235.0	*	0.2	6	339
mggdb_0.40_5	344^E	334^F	2.95	344	289.0	7.0	†	344	**344**	339.9	*	2.8	44	1347
mggdb_0.40_6	270^E	270^E	*	270	247.0	*	418.9	270	270	266.4	*	10.4	316	2746
mggdb_0.40_7	282^E	282^E	*	282	215.0	*	7631.9	282	275	267.0	2.5	‡	2734	7606
mggdb_0.40_8	349^F	322^F	8.05	-	-	-	-	**333**	**326**	320.4	2.1	‡	4454	25787
mggdb_0.40_9	280^F	269^F	4.01	-	-	-	-	**275**	**273**	265.6	0.7	‡	12806	49058
mggdb_0.40_10	191^E	191^E	*	191	181.0	*	3.5	191	191	188.0	*	137.1	226	12953
mggdb_0.40_11	283^E	277^F	2.14	283	270.0	2.6	†	294	277	275.8	6.0	‡	358	48206
mggdb_0.40_12	412^E	412^E	*	412	314.0	*	10608.8	412	412	399.8	*	3.6	78	1935
mggdb_0.40_13	405^E	394^E	2.75	405	373.0	7.3	†	405	**405**	394.7	*	177.6	88	3426
mggdb_0.40_14	62^E	62^E	*	62	58.0	*	76.0	62	62	62.0	*	0.5	2	1025
mggdb_0.40_15	37^E	37^E	*	37	37.0	*	0.2	37	37	37.0	*	2.2	4	2763
mggdb_0.40_16	84^E	84^E	*	84	84.0	*	0.7	84	84	84.0	*	25.9	8	6859
mggdb_0.40_17	65^E	65^E	*	65	65.0	*	0.5	65	65	65.0	*	3.2	12	3118
mggdb_0.40_18	119^E	119^E	*	119	114.0	*	45.3	119	118	118.0	0.8	‡	1874	176869
mggdb_0.40_19	38^E	38^E	*	38	38.0	*	0.6	38	38	38.0	*	0.1	0	147
mggdb_0.40_20	94^E	94^E	*	94	92.0	*	15.9	94	94	94.0	*	5.4	6	2200
mggdb_0.40_21	104^E	104^E	*	104	100.3	*	207.6	104	104	103.9	*	40.9	10	7194
mggdb_0.40_22	129^F	129^F	*	-	-	-	-	129	129	128.2	*	116.5	14	11723
mggdb_0.40_23	193^F	160^F	18.70	-	-	-	-	**160**	160	160.0	*	198.3	2	16465

[1] Gap calculated as $(UB - LB)/((UB + LB)/2)$

[2] Bosco et al. (2013) do not report the explicit lower bound, but report a gap from cplex . Thus we have copied this from their article.

Table 2.5: Computational results for Mggdb $\{0.45, 0.50\}$

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.45_1	259^E	259^E	*	259	211.0	*	283.4	259	259	256.5	*	2.0	104	1038
mggdb_0.45_2	298^E	292^F	2.03	298	279.0	4.8	†	298	**298**	291.6	*	403.9	2800	10502
mggdb_0.45_3	237^E	237^E	*	237	210.0	*	246.1	237	237	235.4	*	0.7	32	711
mggdb_0.45_4	228^E	228^E	*	228	186.3	*	10.3	228	228	222.3	*	1.3	54	911
mggdb_0.45_5	350^E	347^F	0.86	350	309.0	4.4	†	350	**350**	346.4	*	4.0	128	1744
mggdb_0.45_6	218^E	218^E	*	218	182.0	*	109.0	218	**218**	215.8	*	1.5	74	1195
mggdb_0.45_7	243^E	243^E	*	243	171.0	*	121.2	243	243	224.6	*	968.3	4996	19344
mggdb_0.45_8	316^F	296^F	6.54	-	-	-	-	**296**	296	295.2	*	3.5	8	1360
mggdb_0.45_9	284^F	274^F	3.58	-	-	-	-	**277**	**277**	269.4	*	7207.2	7564	31022
mggdb_0.45_10	214^E	214^E	*	214	196.0	*	84.9	214	214	212.2	*	30.6	176	4992
mggdb_0.45_11	297^E	285^F	4.12	297	278.0	4.4	†	301	**289**	287.7	4.1	‡	140	34429
mggdb_0.45_12	393^E	386^F	1.80	393	321.0	5.5	†	393	**393**	377.5	*	6.3	94	2064
mggdb_0.45_13	407^F	403^F	0.99	423	371.0	10.2	†	423	**423**	422.4	*	2.2	4	1930
mggdb_0.45_14	66^E	66^E	*	66	58.0	*	102.7	66	66	63.5	*	2.4	30	2530
mggdb_0.45_15	34^E	34^E	*	34	34.0	*	0.5	34	34	34.0	*	1.7	4	1834
mggdb_0.45_16	70^E	70^E	*	70	62.0	*	238.2	70	70	70.0	*	12.0	8	4359
mggdb_0.45_17	53^E	53^E	*	53	53.0	*	1.7	53	53	53.0	*	5.5	14	3263
mggdb_0.45_18	123^E	115^E	6.72	123	112.0	7.2	†	123	**121**	120.3	1.6	‡	1004	217854
mggdb_0.45_19	48^E	48^E	*	48	40.0	*	0.9	48	48	48.0	*	0.1	6	108
mggdb_0.45_20	78^E	78^E	*	78	76.0	*	7.2	78	78	77.3	*	2.3	4	1855
mggdb_0.45_21	122^E	122^E	*	122	120.0	*	10530.8	122	122	122.0	*	5.9	6	3145
mggdb_0.45_22	137^F	135^F	1.47	-	-	-	-	**136**	**136**	134.8	*	121.5	16	12040
mggdb_0.45_23	146^F	142^F	2.78	-	-	-	-	**144**	**144**	143.8	*	354.0	6	24243

Instance	Best known:			Bosco et al. (2013):				BLW:						
	UB	LB	Gap[1]	UB	Root	Gap[2]	Time	UB	LB	Root	Gap[1]	Time	Nodes	CG
mggdb_0.50_1	214^E	214^E	*	214	178.0	*	86.8	214	214	207.3	*	7.5	252	2170
mggdb_0.50_2	269^E	259^F	3.79	269	233.0	6.8	†	269	**269**	259.5	*	40.7	890	3799
mggdb_0.50_3	218^E	218^E	*	218	184.0	*	399.7	218	218	205.6	*	22.4	642	3038
mggdb_0.50_4	219^E	219^E	*	219	172.0	*	16.4	219	219	214.8	*	0.5	22	524
mggdb_0.50_5	292^E	292^E	*	292	226.0	*	2889.3	292	292	290.5	*	1.7	56	1154
mggdb_0.50_6	276^E	276^E	*	276	228.0	*	1171.8	276	276	270.9	*	9.4	304	2138
mggdb_0.50_7	265^E	265^E	*	265	237.0	*	379.0	265	265	258.7	*	15.7	512	2873
mggdb_0.50_8	336^F	305^F	9.67	-	-	-	-	**310**	**309**	299.1	0.3	‡	8658	33848
mggdb_0.50_9	275^F	260^F	5.61	-	-	-	-	**265**	259	255.7	2.3	‡	1450	15389
mggdb_0.50_10	194^E	194^E	*	194	190.0	*	3.7	194	194	194.0	*	0.9	2	1111
mggdb_0.50_11	275^E	267^F	2.95	275	249.0	4.2	†	-	266	265.2	-	‡	30	11762
mggdb_0.50_12	445^E	445^E	*	445	352.0	*	13492.7	445	445	436.5	*	8.5	82	2570
mggdb_0.50_13	259^E	259^F	*	259	214.0	8.5	†	259	259	257.8	*	43.1	12	2620
mggdb_0.50_14	75^E	75^E	*	75	71.0	*	20.6	75	75	74.3	*	2.0	6	2269
mggdb_0.50_15	37^E	37^E	*	37	37.0	*	0.4	37	37	37.0	*	0.5	2	1115
mggdb_0.50_16	66^E	66^E	*	66	62.0	*	36.3	66	66	65.0	*	14.7	10	4130
mggdb_0.50_17	53^E	53^E	*	53	53.0	*	0.9	53	53	53.0	*	5.1	20	3637
mggdb_0.50_18	121^E	117^F	3.36	121	111.0	6.6	†	122	117	116.7	4.2	‡	450	164158
mggdb_0.50_19	44^E	44^E	*	44	39.7	*	0.3	44	44	44.0	*	0.1	6	259
mggdb_0.50_20	81^E	81^E	*	81	75.0	*	20.7	81	81	79.0	*	11.0	44	2175
mggdb_0.50_21	86^0	86^E	*	86	80.0	*	15691.3	86	86	83.9	*	54.3	50	8143
mggdb_0.50_22	124^F	123^F	0.81	-	-	-	-	**123**	123	123.0	*	54.6	4	9219
mggdb_0.50_23	132^F	125^F	5.45	-	-	-	-	**125**	125	124.6	*	203.9	12	15498

[1] Gap calculated as $(UB - LB)/((UB + LB)/2)$

[2] Bosco et al. (2013) do not report the explicit lower bound, but report a gap from cplex . Thus we have copied this from their article.

Table 2.6: Computational results for Mgval 0.50

Instance	Best known: UB	LB	Gap[1]	Bosco et al. (2013): UB	Root	Gap[2]	Time	BLW: UB	LB	Root	Gap[1]	Time	Nodes	CG
mgval_0.50_1A	145^E	137^E	5.67	145	137.0	*	6.3	-	**138**	138.0	-	†	50	87399
mgval_0.50_1B	170^E	170^E	*	170	170.0	*	2.9	170	170	170.0	*	459.5	16	14113
mgval_0.50_1C	-	-	-	-	-	-	-	**270**	**253**	249.0	6.5	†	3690	48445
mgval_0.50_2A	248^E	248^E	*	248	248.0	*	0.6	351	234	233.5	40.0	†	94	72624
mgval_0.50_2B	284^E	262^E	8.06	284	262.0	*	279.0	-	**274**	273.8	-	†	50	18625
mgval_0.50_2C	-	-	-	-	-	-	-	**473**	**453**	447.3	4.3	†	1214	9069
mgval_0.50_3A	75^E	72^E	4.08	75	72.0	*	1.6	-	**73**	72.1	-	†	16	21992
mgval_0.50_3B	107^E	99^E	7.77	107	99.0	*	801.4	-	**101**	100.3	-	†	54	9287
mgval_0.50_3C	137^E	117^E	1.47	137	117.0	7.9	†	137	**137**	135.1	*	9.8	74	1582
mgval_0.50_4A	350^E	346^E	1.15	350	346.0	*	144.9	-	346	346.0	-	†	6	69311
mgval_0.50_4B	413^E	361^E	13.44	413	361.0	4.2	†	-	**400**	398.1	-	†	6	42986
mgval_0.50_4C	488^E	442^E	9.89	488	442.0	9.1	†	-	**472**	471.2	-	†	42	44254
mgval_0.50_4D	-	-	-	-	-	-	-	-	**565**	563.6	-	†	86	23885
mgval_0.50_5A	367^E	360^E	1.93	367	360.0	*	1296.0	-	360	359.4	-	†	6	83686
mgval_0.50_5B	378^E	348^E	8.26	378	348.0	3.7	†	-	**365**	364.6	-	†	14	38474
mgval_0.50_5C	459^E	417^E	9.59	459	417.0	8.4	†	-	**449**	448.6	-	†	66	55025
mgval_0.50_5D	-	-	-	-	-	-	-	**551**	**527**	525.3	4.5	†	342	57215
mgval_0.50_6A	210^E	199^E	5.38	210	199.0	*	123.2	-	198	197.3	-	†	62	56140
mgval_0.50_6B	210^E	193^E	8.44	210	193.0	*	368.1	-	**206**	205.4	-	†	58	22099
mgval_0.50_6C	-	-	-	-	-	-	-	-	**281**	279.7	-	†	44	3442
mgval_0.50_7A	248^E	243^E	2.04	248	243.0	*	126.8	-	237	236.2	-	†	30	90805
mgval_0.50_7B	276^E	272^E	1.46	276	272.0	*	1429.5	-	271	268.9	-	†	40	53373
mgval_0.50_7C	-	-	-	0	0.0	-	-	-	**306**	304.6	-	†	72	14621
mgval_0.50_8A	388^E	382^E	1.56	388	382.0	1.4	†	-	**386**	385.6	-	†	8	57372
mgval_0.50_8B	-	-	-	-	-	-	-	-	**343**	342.5	-	†	46	67123
mgval_0.50_8C	-	-	-	-	-	-	-	**502**	**485**	483.9	3.4	†	942	61141
mgval_0.50_9A	306^E	306^E	*	306	306.0	*	4.1	-	305	304.1	-	†	12	167515
mgval_0.50_9B	278^E	262^E	5.93	278	262.0	3.5	†	-	**267**	266.7	-	†	8	50355
mgval_0.50_9C	301^E	279^E	7.59	301	279.0	7.4	†	-	**283**	281.9	-	†	18	59138
mgval_0.50_9D	-	-	-	-	-	-	-	-	**349**	348.9	-	†	50	25058
mgval_0.50_10A	385^E	378^E	1.83	385	378.0	1.3	†	-	-	-	-	†	0	116781
mgval_0.50_10B	369^E	364^E	1.36	369	364.0	1.4	†	-	-	-	-	†	0	88414
mgval_0.50_10C	406^E	389^E	4.28	406	389.0	4.2	†	-	**396**	395.7	-	†	16	58839
mgval_0.50_10D	-	-	-	-	-	-	-	-	**436**	433.3	-	†	132	51311

[1] Gap calculated as $(UB - LB)/((UB + LB)/2)$

[2] Bosco et al. (2013) do not report the explicit lower bound, but report a gap from cplex . Thus we have copied this from their article.

for the first time. We improve 38 best known lower bounds and 25 upper bounds. For many of the instances solved to optimality by both our algorithm and the one by Bosco et al. (2013), our computation time is lower, but there are also instances where the opposite is true. 24 of these instances are not solved by Bosco et al. (2013) because they require more vehicles than their algorithm can handle.

Bosco et al. (2013) also present a set of 204 Mgval instances, divided into 6 subsets.

The authors present results for 150 of these instances and discard the remaining 54 instances due to the required number of vehicles. We have chosen to focus on one of the subsets and present results for all instances in this set in Table 2.6. Out of the 34 instances in this set, we prove optimality for 2 instances, one of which is proven for the first time. We improve the lower bound for 24 instances and the upper bound for 4 instances. For 27 instances we are not able to find a feasible solution.

To summarize, we have presented benchmark results for 215 instances and have proven optimality for 122 of these instances.. For 34 instances optimality is proved for the first time. We improve 98 best known lower bounds and 31 best known upper bounds. For 50 of the instances, we are not able to obtain a feasible solution within our time limit. This is due to the fact that we do not initialize our algorithm with a feasible solution obtained from a construction heuristic. Implementing such a starting solution would have a positive effect on both computation time and the upper bounds obtained.

2.6 Conclusion and Future Research

We have presented the first Branch-and-Cut-and-Price algorithm for the MCGRP and compared our results to the only other existing exact algorithm for the problem, Bosco et al. (2013). The main conclusion based on our computational experiments is that our algorithm is in general superior to that algorithm. In particular, our algorithm is stronger for problems with many vehicles where the algorithm by Bosco et al. (2013) is vulnerable because it is based on a three-index model.

For the instances where our algorithm does not prove optimality within the time limit, the obtained lower bound is generally strong and we improve several best known lower bounds. The weak point in our algorithm is that it is only initialized with tours servicing a single demand entity. We do not use any construction heuristic for provid-

ing an initial solution. As a result, for the difficult instances, our algorithm struggles to obtain a feasible upper bound. This increases the running time of out algorithm. For comparison, Bosco et al. (2013) have strong focus on obtaining a good initial solution. An obvious improvement of our algorithm is to implement a construction heuristic to be used for initialization of the algorithm. Furthermore, it could be an advantage to use a meta heuristic to run in parallel to the exact algorithm and simultaneously update the best known upper bound while also adding columns to the exact algorithm.

In our algorithm, we only add valid inequalities to (IP Combined) in the root node of the branching tree. In Section 2.3.6 we presented four types of valid inequalities. To improve our algorithm, separation algorithms could be implemented to identify violated inequalities throughout the branching tree.

Chapter 3

Freight Railway Operator Timetabling and Engine Scheduling

History: *This chapter has been prepared in collaboration with Michel Gendreau and Sanne Wøhlk and most of the work has been done during a research visit at CIRRELT in Montreal. The chapter has been presented at: INFORMS Annual Meeting, 14-17 October, 2012, Phoenix, AZ, USA, Econometric Institute, Erasmus University Rotterdam, 22 February, 2013, Rotterdam, the Netherlands, and EURO-INFORMS Joint International Meeting, 1-4 July, 2013, Rome, Italy. The chapter has been submitted to European Journal of Operations Research and is undergoing a revision*

Freight Railway Operator Timetabling and Engine Scheduling

Lukas Bach[†], Michel Gendreau[‡] and Sanne Wøhlk[†]

[†]Cluster for Operations Research And Logistics,

Department of Economics and Business,

Aarhus University, Denmark

[‡]CIRRELT and MAGI,

École Polytechnique de Montréal,

Montréal, QC, Canada

Abstract

In this chapter we consider timetable design at a European freight railway operator. This is done by choosing the time of service for demands among discrete points of service within a time window. The objective in the model is to minimize cost while adhering to constraints regarding infrastructure usage, demand coverage, and engine availability. The model is solved by a column generation scheme where feasible engine schedules are designed in a labeling algorithm with time-dependent cost and service times.

Keywords: Freight Railway, Timetabling, Partial Integration, Routing, Scheduling, Optimization, Branch-and-Price

3.1 Introduction

In this chapter we consider the problem facing a freight railway operator who has to develop a yearly timetable that includes access to infrastructure. When developing the timetable, a railway operator has to apply for usage of railway infrastructure. In Europe, the organization RailNetEurope (RNE) works to harmonize the access to infrastructure in their 38 member countries. When designing their timetables the operators can apply for a train path, which is an origin and destination pair with given departure time and transit times. These paths can be designed and applied for in multiple ways.

The paths are designed either by RNE, in which case they are called catalogue paths, or the railway operator can design and apply for paths themselves, in which case they are called tailormade paths. By definition, the catalogue paths comply with the regulations such that the paths are feasible. This chapter focuses on the (RNE designed) catalogue paths. Thus the problem is to choose the set of paths that reduces the operating costs of the railway operator. We apply this problem to a case arising at a railway operator who manages the timetabling of unit trains.

DB Schenker Rail Scandinavia A/S (DBSRS) is a railway operator managing transports which originate from and/or have destination in a Scandinavian country. The core business of DBSRS is to provide engines and engine drivers to move customers' cars between stations according to long term contracts. More specifically DBSRS is assigned by its mother companies to handle timetabling and to allocate drivers and engines to the planned trips in the RNE corridor 1 area from Maschen (Hamburg) in the south to Hallsberg (in southern Sweden) in the north. Within the corridors, which are stretches of railway coordinated by RNE, time-slots are allocated by RNE.

In Section 3.2, we describe the problem and case in detail. Then in Section 3.3 we present the modeling of the problem and we describe our solution approach in Section 3.4. Computational experiments are provided in Section 3.5, and conclusions are drawn in Section 3.6.

3.2 Problem and Case Description

In this section, we describe the general problem and the case specific details from DBSRS and the network in which they operate. The overall goal of the problem is to minimize the total costs for the railway operator. When designing a timetable, the total costs include both engine usage, track usage, and other driving related costs. In the DBSRS case we have a given set of engines at hand and the opportunity costs for these engines are not fixed. Thus it is difficult to associate the engines with a fixed charge.

A review of railway routing and scheduling can be found in Cordeau et al. (1998). For more recent reviews of railway optimization, see Lusby et al. (2011); Cacchiani and Toth (2012).

The horizon for the timetable is weekly and has to be repeated for one year. The timetables are typically designed more than six months in advance. The contracts between DBSRS and their customers vary in length and scope, but, without loss generality the contracts all last for at least a year, which covers the timetabling period of a year. Hence, at the time of planning the demand is known.

In this chapter we only consider catalogue paths and thus do not design our own additional tailormade paths. Kuo et al. (2010) study the problem of implementing additional paths in a timetabling problem with elastic demands.

The DBSRS operates a network of 15 active stations that are connected in a tree like structure such that there is only one route between any two stations. The rail network is managed by a different infrastructure manager in each of the three countries in which DBSRS operates. Applications for time-slots are handled by the infrastructure manager during the spring the year before the tracks are actually needed. Basically this requires contracts with clients to be made almost a year in advance.

The cost structure for using the tracks differs from country to country in the region covered by DBSRS. The first cost category is quite simple and reflects the number of kilometers driven, whereas others involve fixed costs for passing specific points.

The costs of using tracks vary over time in some areas and are subject to a capacity charge. Capacity costs are designed to reduce traffic at certain times and are enforced in Denmark and Sweden. In Denmark, the charge applies if the train uses any part of the sections covered by the capacity charge within the time window from 7:00 to 18:59. Sweden has a similar system, but with time windows from 7:00-9:00 and from 16:00-18:00 for passage of a number of sections

Various problems with time-dependent costs have been studied over the past few years. Tagmouti et al. (2007) study a variant of the Capacitated Arc Routing Problem (CARP) where the cost of a route depends on the start time of the service and no waiting is allowed between demands. The problem is transformed to a node routing problem and solved using a column generation scheme. Black et al. (2013) solve a Price-Collecting Arc Routing Problem (PARP) where transit-time is included in the objective; thus with transit-time being time-dependent the cost becomes so as well. The problem is solved using Variable Neighborhood Search and Tabu Search. Both applications forbid waiting time between demands, whereas our application allows for waiting between demands.

The problem can be described as a timetabling problem where a set of heterogeneous engines have to serve a set of unit train demands that cannot be served simultaneously. Each demand is given a time-window during which the service of the demand has to start when a time-slot is available. The planning department at DBSRS currently has to cover approximately 230 demands per week.

The problem is related to Vehicle Routing Problems with Time Windows (VRPTW). The problem at hand shares the time window aspects of the VRPTW, but we allow only to service demand at discrete points during the time window. The access to infrastructure is not considered by the VRPTW. For an introduction to the VRPTW, see Desrosiers et al. (1995) and for a recent review of the VRPTW that focuses on heuristic methods to solve real-life instances we refer to Bräysy and Gendreau (2005a,b). When not solved

by heuristics, the VRPTW is often solved using column generation approaches, see e.g., Kallehauge et al. (2005). The Discrete Time Window Assignment Vehicle Routing Problem was introduced by Spliet and Desaulniers (2012) and this problem relates to our work as their model chooses among a discrete set of possible time windows. They solve the problem using a Branch-Price-and-Cut algorithm.

Each demand requires some preparation time at the station of origin and finalization time at the destination. This time is mainly used for shunting cars, but there may also be a safety margin that serves to prevent propagation of delays. The model handles multiple engine types. They differ as regards the demands they can serve, e.g., costs, and safety margins.

The transit times and time-slots in the DBSRS operational area on the main pathway of corridor 1 are identified using the information from RailNetEurope. This approach cannot be used to identify transit times and time-slots on legs outside the main pathway as the time-slots are not specified. Thus we design them by letting time-slots connect to the time-slots in the main pathway.

The model does not consider maintenance. In the DBSRS case, maintenance is carried out at 10,000 km intervals. Here the engines are taken out of service and replaced by spare engines that continue the engine rotation.

3.3 Modeling

The problem can be described as a timetabling problem where a set of heterogeneous engines have to serve a set of unit train demands that cannot be serviced simultaneously. Each demand is given a time window during which the service of the demand has to start when a time-slot is available. The time-slots are laid out by the infrastructure manager who sets them in advance. There is approximately one time-slot every half hour and with time windows of about 6 hours, there is a limited number of pos-

minimize	*Total costs*
subject to	*Demand coverage*
	Availability of engines
	Replication of weekly timetable
	Network capacity
	Time-slot compliance
	Time window compliance
	Flow conservation
	Engine type compliance
	Waiting time after demands
	Transit time between demands

Figure 3.1: Model overview

sible departure times. The planning horizon for the schedule is one week, whereas the rotation period of the engines can be as long as necessary. Cacchiani et al. (2008) consider timetabling in a similar corridor and Ziarati et al. (1997) solve an engine assignment problem with a heterogeneous fleet, but neither includes the path application concept considered in this chapter.

The goal of the model is to produce a weekly timetable that minimizes the total costs. An overview of the model can be seen in Figure (3.1). In the following we consider the implementation of the constraints and how to handle them.

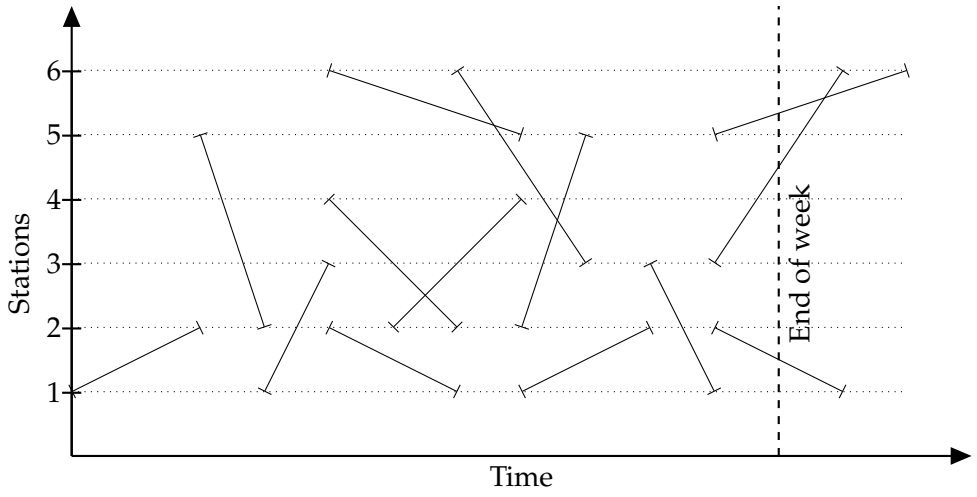
The demand coverage is simple as we are interested in covering all demands exactly once. Here we do not allow multiple coverings of the demand. Engines are divided into multiple categories and each category has a fixed number of engines. A cost can be associated with the usage of the different engine types.

We will handle two issues related to the replication of weekly timetables: Firstly, we need to define how to handle the circular nature of the weekly plan. Secondly, we have to decide on the design and length of the engine rotations. The lack of weekly time-periods with all engines grounded presents a problem. This forces the planning

period to be circular in the sense that an engine departing at the very end of the week will arrive at the beginning of the week, see figure 3.2.

Nahapetyan and Lawphongpanich (2007) solve a dynamic traffic assignment problem where they use a circular approach as opposed to starting and ending with an empty system. Caimi et al. (2011) use the Periodic Event Scheduling Model (PESP) on an infrastructure management level to generate feasible time-slot allocations. In the PESP, events are repeated over a time horizon, for instance 1 hour, such that a train is departing at the same minute every hour. Lindner and Zimmermann (2005) use the PESP model to create cost optimal train schedules for a railway operator. As the events in our case are periodic over a week, the time horizon in a PESP model would have to be set to 1 week. The PESP model considers the design of time-slots or at least the maximum time between different events such as two different trains departing from the same station and the headway between them. In our case this is handled by the choice of train paths and by avoiding overlaps.

Figure 3.2: Weekly timetable - example



Note: A line in the figure represents a demand in a timetable that is driven from the station at the beginning to the station at the end of the line.

The replication of weekly cycles can be handled in several ways; one approach is to let each engine follow a repetitive weekly plan such that each engine has the same schedule every week. Another way is to keep the weekly timetable but plan with a longer finite horizon for the individual engines, e.g., 4 weeks. We have chosen a weekly schedule that is not engine specific in the sense that it is not necessarily operated by the same engine every week.

For our approach to be viable we have to find a way to ensure that the different schedules fit together. We would like to avoid a complex formulation where we have to balance the weekly schedules with respect to both place and time. As regards place, there are no depots so therefore we treat all stations as depots such that we have a problem with multiple depots. This is important because an engine has to be ready at the end of one week at a place that fits with the starting place of the following week.

As the operation is continuous and does not have a fixed period without activity for engines or engine groups, there is no natural point that we could designate as the beginning of a week. To remove the time factor such that the start of the week can be kept fixed at the same time for all engines, we could choose a point during the week where no - or the least - demands cross and then balance the weekly schedules at this point.

Without the time factor we would face a problem related to defining the beginning / end of the week. In figure 3.2 we see an example of the end of week (eow) concept. It can be done in such a way that demands that start in the last part of a week continue into the next week which is essentially the beginning of the week. When we balance the number of weekly schedules that have a given station as their origin station with the number that have it as their destination station, the times of the origins must be before the times of the destinations in order to guarantee that engines are actually available.

To be able to remove the time factor we have to make sure that the first station on any schedule is serviced earlier than the last station on every schedule. To break the

circularity of the weekly timetable we propose to find the point during a week that interferes with the least demands. At this point we split all demands that intersect it in two parts, one from the origin to the eow and one from the start of week until arrival. We call these two parts partial demands. These two partial demands are linked to a dummy station at the chosen eow as shown in figure 3.3. Each dummy station is unique for a demand that uses a specific time-slot.

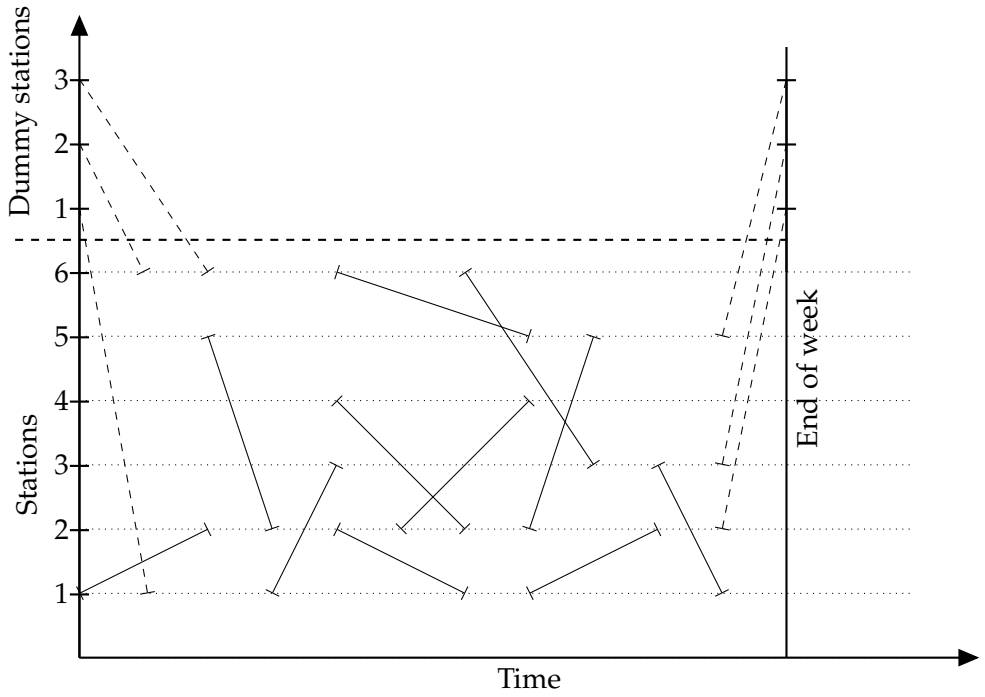
The dummy station ensures that there will be one weekly schedule that starts with the first of the partial demands and one that ends with the last partial demand. Whether these two partial demands are on the same schedule is not important. If they are on the same schedule, there will be one circular schedule that will be driven by the same engine each week. If they are not in the same schedule, they will still be driven by the same engine but now they force the two weekly schedules involving them to be merged into a rotation of at least two weeks. At this point the other origin and destination stations are irrelevant as they will be balanced accordingly.

Demands that do not cross the eow remain unchanged as it is certain that the first station on a schedule is the origin and the last is the arrival and that they can never cross the eow. Thus when the flows in and out of ordinary and dummy stations are balanced, we ensure that all schedules are able to be part of an engine rotation with a possible infinite time horizon.

After handling the replication of weekly timetables, we return to the remaining constraints. The network capacity allows only one train per time-slot. Hence two demands that use the same or have partially overlapping schedules would have to be planned such that no time-slot is used twice.

Paths and time window compliance are related in the way that the start of service of a demand has to be within the time window and within this time window we have to select a path which means that it is not enough just to be within the time window. Therefore any time window will have an associated set of paths that can be used.

Figure 3.3: Weekly timetable with dummy stations - example



Flow conservation ensures that the flows in and out of stations are balanced for all stations of the schedule except at the first and last.

Engine type compliance ensures that the right engine type is used for a given demand. For all demands a subset of the engine types can be used to service it. Thus it is important that we assign the right engine type to a demand to keep the timetable feasible.

Waiting and transit times between demands are treated somewhat similarly. Transit time between demands ensures that there is enough time to complete possible reposition trips between two demands. It also includes the necessary time to prepare for service of the following demand. Waiting time after demands handles the safety margins that are planned after the completion of each demand to avoid propagation of delays. This waiting time is set according to the policy of the railway operator.

3.4 Solution Approach

As described above the problem of developing the weekly timetable is subject to multiple sets of constraints. To combine them into a joint formulation of the problem would result in a complex formulation. Instead we choose a column generation approach. It allows us to split the problem in two and thus reduce the complexity. We use a Dantzig-Wolfe decomposition approach to split the problem into a master-problem and a pricing-problem. For more information on column generation, we refer the reader to Desrosiers and Lübbecke (2005) and Desrosiers et al. (1995).

minimize	*Total costs*	
subject to	*Demand coverage*	*Master*
	Availability of engines	
	Replication of weekly timetable	
	Network capacity	
	Time-slot compliance	*Pricing*
	Time window compliance	
	Flow conservation	
	Engine type compliance	
	Waiting time after demands	
	Transit time between demands	

Figure 3.4: Model overview

We split the model such that all constraints concerning the individual engine are dealt with in the pricing-problem. In the master-problem we consider the replication of weekly timetables, time-slot usage, availability of engines, and demand coverage, because these variables apply to multiple engines at once. By solving the pricing-problem, we can generate engine schedules that are combined into a weekly timetable in the master-problem. Figure 3.4 shows how the problem is split into master and

pricing-problems.

We use a delayed column generation approach and generate the columns to enter the basis, i.e., the column with the most favorable reduced cost, each time we solve this restricted master problem to optimality. We implement this with a branching method to form a Branch-and-Price setup.

3.4.1 Master-Problem

In this section, we first show the formulation of the full master-problem, then we deduce the restricted master-problem and the linear relaxation hereof. The full master-problem is formulated with the parameters given in table 3.1 and equations (3.1) to (3.6).

Table 3.1: Parameters

Parameter	Description
N	Set of all stations including dummy stations, indexed by n
D	Set of all demands, d
S	Set of all time-slots, s
E	Set of engine types, e
w_e	Availability of engines of type e
c_r	Cost of schedule r
α_r^d	Demand d is covered by schedule r, 1 = yes, 0 = no
β_r^n	Station n is used by schedule r, as origin station = 1, as destination = -1, as both = 0, or neither = 0
γ_r^s	Time-slot s is used by schedule r, 1 = yes, 0 = no
δ_r^e	Schedule r driven by engine type e, 1 = yes, 0 = no

As columns in the master-problem we use engine schedules generated in the pricing-problem. Thus selecting a column represents using the corresponding schedule for an engine. The only variable in the model, x_r, represents choosing this column. Here x_r equals 1 if a given column is used, and 0 otherwise. The set Ω is the set of columns

under consideration. Hence this set will be adjusted as we add and remove columns in the restricted master-problem.

$$\min \sum_{r \in \Omega} c_r x_r \tag{3.1}$$

$$\text{s.t.}, \sum_r \alpha_r^d x_r = 1, \forall d \in D \tag{3.2}$$

$$\sum_r \beta_r^n x_r = 0, \forall n \in N \tag{3.3}$$

$$\sum_r \gamma_r^s x_r \leq 1, \forall s \in S \tag{3.4}$$

$$\sum_r \delta_r^e x_r \leq w_e, \forall e \in E \tag{3.5}$$

$$x_r \in \{0; 1\}, \forall r \in R \tag{3.6}$$

The objective function is given in equation (3.1) where c_r is the total cost for the column r, corresponding to a given schedule. Thus we seek to minimize the total cost which is composed by a fixed engine usage cost and driving costs, both engine type specific.

In (3.2) we make sure that all demands are serviced by the engines. To ensure the necessary balance between the starting and ending stations of the engines we have the balance constraint (3.3). This constraint also balances the dummy stations such that an engine ending at a dummy station forces an engine to start at the same dummy station. In (3.4) we ensure that each time-slot is used at most once. Engine availability is ensured by (3.5), where we bound the number of each engine type used. Finally the binary constraint (3.5) forces each schedule to be used at most once.

To generate the columns in the pricing-problem we have to find columns with negative reduced costs and therefore we determine the reduced costs by using the duals from the master-problem to alter the costs in the pricing-problem such that a minimization of the pricing-problem results in the columns with the lowest reduced cost.

Let $\boldsymbol{\pi}, \boldsymbol{\rho}, \boldsymbol{\sigma},$ and $\boldsymbol{\tau}$ represent the vectors of the dual variables defined by constraints 3.2, 3.3, 3.4, and 3.5 respectively, e.g., $\boldsymbol{\pi} = [\pi_d], \forall d \in D$.

3.4.2 Pricing-Problem

In the pricing-problem, we handle the weekly schedule of a single engine. The goal is to choose a set of demands to form a schedule, subject to a set of constraints. The costs of the demands are original costs adjusted with the duals from the master-problem. The total cost in the pricing-problem then becomes the reduced cost for the corresponding column in the master-problem.

The constraints in the pricing-problem (Figure 3.4, Section 3.3) are: path compliance, time window compliance, engine type compliance, flow conservation, transit, and waiting times between demands.

The pricing-problem will be solved as a Shortest Path Problem with Resource Constraints, see Irnich and Desaulniers (2005), on a network representation of the demands. In section 3.4.2 we explain the design of the network. For each demand $i \in N$, there is a time window in which the service of the demand can be initiated. Within this time window the demands can be serviced only at specific time-periods, i.e., where a path is available.

In case of deadheading, we can choose to place waiting time either before deadheading at the destination of the previous demand and/or at the origin station of the following demand. Knowing that the deadheading cost is time-dependent, we must choose the optimal waiting time before starting the deadheading.

The pricing-problem is defined with the parameters given in table 3.2. Define the variables $y_{tij} \in \{0;1\}$ where $y_{tij} = 1$ if demand j is serviced immediately after demand i at time t, and $y_{tij} = 0$ otherwise. Further define $z_e \in \{0;1\}$ as $z_e = 1$ if engine type e is used, and $z_e = 0$ otherwise. Then the objective function is defined as shown in equation 3.7.

Table 3.2: Parameters

Parameter	Description
T	Set of all time-periods t in the model, T covers one week.
Δ	Resolution of time-periods in minutes, $\Delta = 1$ gives $\|T\| = 10080$.
N	Set of all stations including dummy stations, indexed by n
D	Set of all demands, d
S	Set of all time-slots, s
E	Set of engine types, e
S_{dt}	$S_{dt} \subset S$ contains the time-slots s which are used by demand d at time t, if it is not possible to start the demand at the given time $S_{dt} = \emptyset$.
g_{tij}	Transition time from demand $i \in D$ to $j \in D$ at time t
b_{tj}	If the time window for the demand $j \in D$ allows for start of service at time t, equal to 1 otherwise 0.
o_j	Origin station for demand $j \in D$.
a_j	Arrival station for demand $j \in D$.
ϕ_{tij}	Transit cost from demand $i \in D$ to $j \in D$ at time t.
v_e	Cost of using engine type e.
ψ_{tj}	Cost of demand $j \in D$ at time t.

$$\bar{c_r} = \sum_{e \in E} ((v_e - \tau_e) z_e) + \sum_{t \in T} \sum_{j \in D} \left((\psi_{tj} - \pi_j - \sum_{s \in S_{jt}} \sigma_s) \sum_{i \in D} y_{tij} + \sum_{i \in D} \phi_{tij} y_{tij} \right) \tag{3.7}$$
$$+ \sum_{j | y_{(\arg\min_{t | \sum_{j \in D} y_{t0j} = 1} \{t\}) 0j} = 1} \rho_{o_j} - \sum_{j | y_{(\arg\max_{t | \sum_{i \in D} y_{ti0} = 1} \{t\}) i0} = 1} \rho_{a_j}$$

The first term determines the cost of using the engine and is modified by the dual vector $\boldsymbol{\tau}$. Then we have the duals for servicing the demands $\boldsymbol{\pi}$ and the duals for using the time-slots $\boldsymbol{\sigma}$. The costs for using a station as origin station and arrival station are represented by the dual $\boldsymbol{\rho}$. We find the origin station o_j by finding the first demand of the week. Then we take the appropriate cost ρ_n for using this station n as origin station. We find the first time-period t where we start service for any demand by $\arg\min_{t | \sum_{j \in D} y_{t0j} = 1} \{t\}$. This is done by finding the first time-period, t, with $y_{t0j} = 1$ for

any j. We set $i = 0$ to represent the source / sink. For the last time-period, we return to the sink, thus we have $j = 0$ and maximize t instead to find the last time-period with a demand.

Network

The pricing-problem is normally defined on a graph, $G = (N, A)$, where N is the set of nodes (demands) and A is the set of arcs (transition between demands). Here, the constraints are handled by the graph where only feasible connections are possible. We solve the SPPRC using a labeling algorithm and hence we have a resource measuring time consumption. When extending from one demand to another we use the time consumption to determine the cost of the extension. As this cost is time-dependent and we can add waiting time before extending, we have different extension possibilities that - possibly - do not dominate each other. We do not know the optimal time within the time window to service demands with time-dependent costs and therefore - as a worst case - we would have to explore all possible service times leading to huge numbers of combinations of service time at demand 1, waiting time before deadheading, and service time at demand 2. The locally optimal decision on when to deadhead has to be part of a globally optimal solution. This cannot be guaranteed when deadheading and therefore we have to extend by deadheading in all ways that are non-dominated. We thus propose that we can revise the graph to reduce this to a local decision. In the original graph $G(N, A)$, we have a cost for deadheading to another time-dependent demand that can be performed at multiple times at time-dependent cost. If for every demand $j \in D$, we create a node in a new graph, for every time-period $t \in T$ with $b_{tj} = 1$, we have a set of nodes that each represent servicing the demand at a specific time. The cost of deadheading between two such nodes that service different demands can then be determined strictly locally by choosing the cheapest deadheading without affecting future decisions.

We obtain the aforementioned network by transforming the original graph using algorithm 9.

Algorithm 9 Extended graph

```
1: Set N' = {0}
2: for all j ∈ D do
3:    for all t ∈ T do
4:       if b_tj = 1 then
5:          i = new node(t, j)
6:          N' = N' ∪ {i}
7:       end if
8:    end for
9: end for
10: for all j ∈ D do
11:    populate succesorList(j)
12: end for
```

The algorithm simply creates a copy of each demand for every time-period within the time window with an available time-slot. Then finally, we update the successors for each demand. As we now have a fixed service time for each node, we can calculate the time when it is possible to make a feasible connection between any two nodes.

To allow for reposition before the first demand and after the last demand serviced, we insert reposition nodes just after the source and before the sink. The source and sink are only connected to the reposition nodes and thus any schedule will have to use a reposition at both ends; note, however, that the reposition can be to the same node as when the schedule ends and thus constitute a free reposition.

Labeling Algorithm

After transforming the network it appears to be an acyclic network where elementary paths can be obtained using a standard algorithm. But as multiple nodes represent the same demands, a path with more than one node representing the same demand constitutes a cycle and is thus a non-elementary path.

To find the elementary shortest path in this network we use the label correcting algorithm proposed in Feillet et al. (2004). Note that due to the acyclic nature of the graph, we do not have a time resource. As the demands are tied to multiple nodes, the resources represent demands and therefore multiple nodes compete for the same resource. The use of this algorithm therefore guarantees an optimal solution.

This approach can be slow because having one resource for each demand causes many labels to be non-dominated. Instead, as suggested by Boland et al. (2006); Righini and Salani (2008), we focus on a subset of the resources. We do this by taking a non-elementary path and adding as resources the demands that cause it to be non-elementary. Then the problem is solved until an optimal elementary path is obtained. We denote this method Delayed Resource Constraints (DRC).

In most cases the cycles prove to be 2-cycles. As suggested by Houck et al. (1980) it is relatively easy to remove them without deteriorating the asymptotic worst case running time. We denote this method 2-Cycle Elimination (2CE).

3.4.3 Branching Rules

We have two branching rules. Rule A considers fixing the time of servicing a demand d by splitting the time windows in two. Rule B is a follow-on branching that chooses a pair of demands, where one has to be performed immediately after the other.

We first consider branching rule A. For each column x_r, we have a vector v_r corresponding to N', indicating which nodes i in $G'(N', A')$ we visit. Hence $v_r^i = 1$ if node i is visited by schedule r, and zero otherwise. From this vector, we define the disjoint subsets $N'_d, \forall d \in D$ of all the nodes i that serve the demand d. Thus N'_d is the set of all the copies of a specific demand. We call this the node family of demand d. When splitting a time window we remove a subset of the nodes in N'_d. Instead of splitting it into two halves, we perform what we call a balanced split as follows:

From the master problem, we know that equation (3.8) must hold.

$$\sum_{r \in \Omega} \sum_{i \in N'_d} x_r v_r^i = 1, \forall d \in D \tag{3.8}$$

We define $\boldsymbol{\lambda}$ as the set $\{\lambda_i : i \in N'\}$. In equation (3.9), we define λ_i as the fraction of the demand d that is serviced by a specific node $i \in N'$. For any solution to be feasible, $0 \geq \lambda_i \leq 1$ must be true for all $i \in N'$. If $0 < \lambda_i < 1$ for any $i \in N'$, then the solution must be fractional as there can be at most one $i \in N'_d$ serviced by any schedule r in an integer solution.

$$\lambda_i = \sum_{r \in \Omega} x_r v_r^i, \forall i \in N' \tag{3.9}$$

In equation (3.10), we define $\boldsymbol{\lambda}^d$ as disjoint subsets of $\boldsymbol{\lambda}$ such that every λ_i that serves the same demand d is present in exactly one $\boldsymbol{\lambda}^d$.

$$\boldsymbol{\lambda}^d = \{\lambda_i : i \in N'_d\}, \forall d \in D \tag{3.10}$$

We know from equation (3.8) that equation (3.11) must be true.

$$\sum_{i \in \{i : \lambda_i \in \boldsymbol{\lambda}^d\}} \lambda_i = 1 \tag{3.11}$$

As an example of a fractional solution, we can have $\boldsymbol{\lambda}^{15} = \{0, \frac{1}{4}, 0, \frac{1}{4}, \frac{1}{2}, 0\}$. The goal of splitting up the time window, which is represented by $\boldsymbol{\lambda}^d$, is to equally divide the fractional values on the branches. Hence a left branch will be $\boldsymbol{\lambda}^{15} = \{0, \frac{1}{4}, 0, \frac{1}{4}\}$ and a right branch will be $\boldsymbol{\lambda}^{15} = \{\frac{1}{2}, 0\}$, which is what we call the balanced split. The corresponding nodes are then removed from N'_d and N' as well.

To choose the demand $d \in D$ on which to perform this split, we find the $\boldsymbol{\lambda}^d$ with the most non-zero λ_i. This means that $\boldsymbol{\lambda}^{15} = \{0, \frac{1}{4}, 0, \frac{1}{4}, \frac{1}{2}, 0\}$ has 3 non-zero values.

The second branching rule, rule B, is a follow-on procedure. When the branching opportunities for rule A are exhausted, we know that all demands are serviced at a fixed time but possibly by different engines. In this situation, the solution can be fractional if two or more demands are serviced by different schedules. We therefore wish

to force the demands into a given sequence that will be serviced by a single engine. As we know from rule A that all demands are fixed in time, we focus on the general network $G(N, A)$. We define $\boldsymbol{w}_r$ for the nodes serviced in $G(N, A)$ by a schedule r. This corresponds to the vector $\boldsymbol{v}_r$ for the nodes serviced in the graph $G'(N', A')$. The relation is shown in equation (3.12) and figure 3.5.

$$w_r^d = \sum_{i \in N'_d} v_r^i, \forall d \in D \tag{3.12}$$

From equation (3.12), we observe that w_r^d is the sum of the nodes in the node family of d. Notice that only one of these can be in a single schedule. We now have the vector $\boldsymbol{w}_r$ for each r. Hence we know whether the demands are serviced or not, i.e., $w_r^d = 1$ if demand d is serviced by schedule r, and $w_r^d = 0$ otherwise.

We then construct a vector $\boldsymbol{\eta}_r$ of the demand indices for demands with $w_r^d = 1$. This can be seen in equation (3.13). The relationship between $\boldsymbol{v}_r$, $\boldsymbol{w}_r$, and $\boldsymbol{\eta}_r$ is illustrated in figure 3.5.

$$\boldsymbol{\eta}_r = [d : w_r^d = 1], \forall d \in D \tag{3.13}$$

Figure 3.5: Relationship between $\boldsymbol{v}_r$, $\boldsymbol{w}_r$, and $\boldsymbol{\eta}_r$

$$\begin{array}{lccccccc}
\boldsymbol{v}_r = (& \underbrace{0,0,0,0,} & \underbrace{0,0,1,0,} & \underbrace{0,0,0,} & \underbrace{0,1,0,} & \underbrace{1,0,} & \underbrace{0,0,0} &) \\
\boldsymbol{w}_r = (& 0, & \underbrace{1,} & 0, & \underbrace{1,} & \underbrace{1,} & 0 &) \\
\boldsymbol{\eta}_r = (& & 2, & & 4, & 5 & &)
\end{array}$$

Note: $\boldsymbol{v}_r$ is the vector of nodes on a schedule r in the graph $G'(N', A')$, $\boldsymbol{w}_r$ is the vector of nodes in the graph $G(N, A)$ on the same schedule, and $\boldsymbol{\eta}_r$ are the node indices for the non-zero values in $\boldsymbol{w}_r$

In the vector $\boldsymbol{\eta}_r$, the first demand serviced by schedule r will be the demand $\eta_r[i = 1]$ and the last will be $\eta_r[i = \sum_{d \in D} w_r^d]$. As shown by Ryan and Foster (1981) we know that in any fractional solution, there will be a pair of demands serviced by different

schedules. Thus the sum of the x_r variables when they are serviced together must be between 0 and 1, see equation (3.14).

$$0 < \sum_{r \in \{r : w_r[d_1] = w_r[d_2] = 1\}} x_r < 1 \tag{3.14}$$

If all pairs of demand are 1 or 0, we have an integer solution. To get an integer solution, Ryan and Foster (1981) branch so that the equation (3.14) equals 1 or 0. If this holds for all pairs, it also holds for pairs that are serviced immediately after each other. We obtain the values for these immediately preceding pairs in equation (3.15), where p_{ij} is the flow on the corresponding arc (ij) in $G(N, A)$.

$$p_{ij} = \sum_{\substack{r \in \{r | \exists k : \eta_r[k] = i \\ \wedge\ \eta_r[k+1] = j\}}} x_r, \forall i \in D, j \in D \tag{3.15}$$

When branching, we set $p_{d_1,d_2} = 1$ on the left branch and $p_{d_1,d_2} = 0$ on the right branch. This forces d_2 to be visited right after d_1 - if visited at all. Thus we remove all connections to other demands from d_1. On the other branch, we forbid this connection. What we effectively do is to merge these two demands into one new demand.

When we determine two demands to branch on, we select the pair with $p_{d_1 d_2}$ closest to 1.

3.4.4 Node Selection Strategies

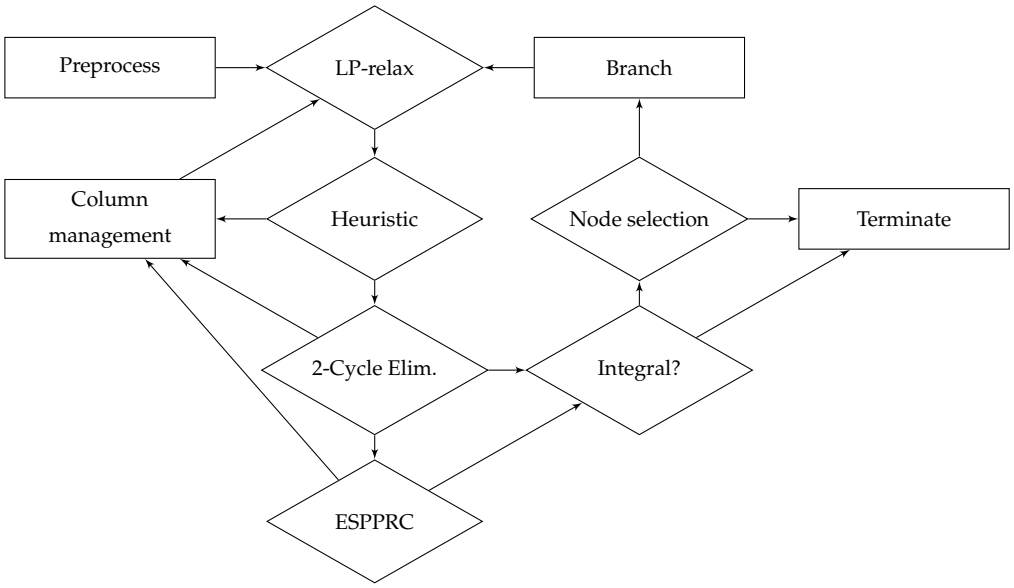
A depth first search is often used in a Branch-and-Price algorithm to allow the master-problem to remain as unchanged as possible when moving down the node tree. In a breadth first search, an additional number of columns has to be stored and reinserted at the nodes. This process increases the memory requirement significantly. Furthermore, the lower bound from the LP-relaxation in the root node to this particular problem is very strong and is either very close to the optimal integer solution or equal to. This indicates that it is favorable to look for an integer feasible solution with a depth first rather than a breadth first search.

To speed up the process we use a stronger branching rule until we have found the first integer feasible solution. This is a variation of the time window split where instead of splitting the time window evenly, we split it such that the single node with the highest fractional value is on one branch and the rest are on the other. This efficiently leads to the first integer solution, but yields an asymmetric branching tree. It takes very long time to close this tree because the remaining branches are very weak. Therefore we restart the branching tree after finding the first integer solution and return to the standard time window split.

Experiments with alternative node selection strategies on a significant set of instances prove the superiority of the combined strategy.

3.4.5 Implementation

Figure 3.6: Branch-and-Price algorithm



The overall Branch-and-Price algorithm can be seen in figure 3.6. The algorithm starts preprocessing by finding a point during the week with the smallest number of possible activities. We designate this as the end of week. This is often in the early hours

of Sunday. Then the LP-relaxation is solved with a set of columns allowing for an initial feasible solution. This set consists of expensive columns serving exactly 1 demand each and using no engines and time-slots. An initial solution from a heuristic is often used to speed up the solution of the root LP-relaxation for column generation. Time is not the most important factor here and the algorithm reaches the solution of the root node relatively fast and thus a good initial solution would not provide a significant overall speed up.

When the LP-relaxation has been solved, the dual values are passed to the pricing-problem. In all cases, we solve the pricing-problem with a labeling algorithm and use only elementary schedules. We first solve the pricing-problem with a heuristic that only allows for a waiting time between demands of 12 hours and 24 hours for the engines, irrespective of whether they deadhead or not. This procedure significantly reduces the number of possible connections and thus provides a significant speed up with high quality columns. When we cannot find any more elementary columns with negative reduced costs, we run the exact algorithm with 2-cycle elimination to ensure optimality. If the best solution to the pricing-problem is elementary and of non-negative reduced cost, we skip the next algorithm for the pricing-problem as we have an optimal solution to the LP-relaxation. The final ESPPRC algorithm for the pricing-problem ensures that we always find an elementary solution. The algorithm iteratively adds resource constraints to the problem depending on which nodes cause the schedule to be non-elementary. This process continues until the best solution is elementary. The algorithm is not used very often as the 2-cycle elimination is, in most cases, enough to avoid any length cycles.

If we have negative reduced cost schedules, we add columns to the master-problem. As we do not dominate labels on the sink node, a large number of columns can be added. Experiments with addition of different percentages of the possible columns yielded no clear result, but the trade-off is between adding all, i.e., one per node in the

pricing-problem, and only the best. Adding all gives the best information for the duals, but over time the LP-relaxation becomes overburdened by the high number of columns added. On the other hand adding only the best columns results in a fast LP-relaxation, but adds a large number of additional calls of the pricing-problem algorithm. Good results were obtained by adding a number of columns corresponding to at most 50% of the nodes in the pricing-problem and removing unused columns in the LP. It is important to note that we only add the elementary columns with a negative reduced cost, thus we often add less columns than dictated by the maximum.

It is necessary to remove unused columns because we insert a relatively high number of columns after each LP iteration. Columns that have not entered the basis for the first 100 LP iterations are removed from the master-problem. Experiments with different values did not lead to any decisive results. We need to keep enough information in the LP and balance this against the size of the LP. Notice, that to ensure that we always have a feasible solution to the LP-relaxation, the columns representing the initial solution are never removed.

3.5 Computational Experiments

Experiments have been carried out on an Intel Core i7 2.8 GHz and implemented with C++ as a sequential algorithm. We use ILOG CPLEX 12.4 to solve the LP-relaxation.

3.5.1 Data Instances

We have created a set of 23 instances based on actual demand data from DBSRS. The instances with demand equal to 228 hold the real case demands and all other instances are variations hereof. The network and path information is based on information collected from RailNetEurope. Instances with less than 228 demands have had demands removed. This has been done in two ways: in the (s)-instances, we have removed

demands pair-wise from the end of the week to preserve the density of the problem. Secondly, for the instances named (ex3) and (ex6), the time windows have been extended by 3 and 6 hours respectively. These instances are based on the (s)-instances such that they are similar in every way except for the time window length. In the (r)-instances, we randomly remove demands such that the symmetry is not kept and the demands are evenly spread over the week.

3.5.2 Results

Table 3.3: Solution times, homogeneous engines

D/C		Root time:				Time to IP:				Total time:			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	0.4	1.2	2.1	1.1	2.5	5.5	9.6	4.1	2.5	5.5	9.6	4.1
	27	0.6	1.2	2.2	0.9	2.6	5.6	11.9	2.7	2.6	5.6	11.9	2.7
	24	0.6	1.3	2.4	1.2	3	5.9	10.7	4.2	3	5.9	10.7	4.2
120	30	1.4	2.4	4.2	2.4	4.3	13.9	23.1	10.3	4.3	13.9	23.1	10.3
	27	1.3	2.8	4.2	2.5	7.1	14.4	23.7	10	7.1	14.4	23.7	10
	24	1.5	3.3	5.1	2.6	7.8	13.6	17.4	9.8	7.8	13.6	> 720	9.8
150	30	3.5	8.8	13.7	5.7	16.6	37.3	61.6	21.2	16.6	37.3	61.6	21.2
	27	3.6	10.4	15	5.5	19.7	48.2	78.7	20.9	19.7	48.2	78.7	20.9
	24	3.9	8.2	14.7	6.1	25.1	51.8	68.4	22.1	25.1	51.8	68.4	22.1
180	30	8.3	19.9	35.3	11.3	36.2	75.1	134.8	35	36.2	75.1	134.8	35
	27	7.9	22.1	37.9	10.1	44.6	107.8	140	42.7	44.6	107.8	140	42.7
	24	9.2	17.7	26.5	13.6	60.3	113.3	168.9	52.1	60.3	113.3	168.9	52.1
210	30	14.8	30.7	41.1	14.1	62.3	159.5	226.1	63.6	62.3	159.5	226.1	63.6
	27	13.4	34.6	45.2	17.5	67.5	219.4	261.6	49.8	126.4	219.4	261.6	49.8
	24	13.7	31	38.3	17.8	85.7	242.1	309.6	72.4	85.7	242.1	> 720	72.4
228	30	24.5	22.7	50.5	n/a	130.3	101.8	191.3	n/a	> 720	320.2	191.3	n/a
	27	33.3	21.9	65.2	n/a	168.4	147.1	375.8	n/a	168.4	147.1	> 720	n/a
	24	23.3	28.6	67.4	n/a	224.5	262.8	448.1	n/a	224.5	262.8	448.1	n/a

Note: Time is reported in minutes. Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column.

Table 3.4: Pricing-problem algorithms used, homogeneous engines

D/C		Heuristic				2-cycle elim.				ESPPRC			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	503	623	699	798	89	88	118	142	0	0	0	0
	27	470	666	820	885	61	87	146	174	0	0	0	2
	24	713	755	874	882	64	73	80	181	0	0	0	0
120	30	770	941	1070	1368	151	183	204	256	0	0	0	4
	27	974	1049	1234	1409	133	178	147	270	0	0	0	0
	24	1275	1606	3335	1394	110	99	347	263	0	0	0	0
150	30	1740	1997	2071	2059	194	287	230	329	0	0	0	2
	27	2321	3348	3687	2122	147	198	190	363	0	0	0	6
	24	2742	3380	3707	2381	142	146	182	325	0	0	0	6
180	30	3436	3487	4108	2864	276	252	283	457	0	0	0	4
	27	3653	4957	5477	3523	247	287	298	452	0	0	0	2
	24	5027	5063	5871	4469	201	181	190	563	0	0	0	4
210	30	4822	6233	6877	3805	346	384	406	503	0	0	0	5
	27	7360	7573	7925	3768	436	297	304	495	0	0	0	2
	24	5915	8778	15533	5990	243	310	571	502	0	0	0	1
228	30	23568	14053	8738	n/a	1271	972	484	n/a	8	8	22	n/a
	27	6957	7470	15072	n/a	526	543	605	n/a	1	3	6	n/a
	24	8251	8622	10256	n/a	428	450	365	n/a	0	2	0	n/a

Note: Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column.

The algorithm provides high quality integer solutions to real sized problem instances in about 4 hours. This is highly satisfactory considering the time horizons involved in the planning process. As this is carried out on a yearly basis and with long deadlines we are willing to allocate even more time to this process if necessary.

The algorithm is applied in two versions; a version using only 1 engine type, able to serve all demands and access relevant infrastructure, and a version using 3 engine types; one diesel engine and two different electrical engines of which one has a greater pulling power and thus has to be used on some specific demands. Hence some demands can be serviced only by the diesel engine, the strong electrical engine, by both

Table 3.5: Branching method and root gap, homogeneous engines

D/C		Branch A:				Branch B:				%-gap			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	52	48	62	72	16	25	31	43	0	0	0	0
	27	47	50	66	77	0	22	30	52	0	0	0	0
	24	53	44	52	76	4	18	21	55	0	0	0	0
120	30	83	88	98	96	50	59	73	75	0	0	0	0
	27	77	88	90	97	37	67	35	85	0	0	0	0
	24	69	63	153	91	24	22	31	68	0	0	0.02	0
150	30	96	109	116	124	45	77	37	107	0	0	0	0
	27	90	91	105	122	25	39	39	106	0	0	0	0
	24	88	94	104	120	31	30	35	87	0	0	0	0
180	30	131	115	124	153	67	59	59	124	0	0	0	0
	27	117	120	125	134	50	52	61	95	0	0	0	0
	24	114	107	117	133	53	42	39	90	0	0	0	0
210	30	148	144	158	170	72	76	86	134	0	0	0	0
	27	172	142	143	168	82	73	70	143	0.01	0	0	0
	24	145	140	250	167	62	69	58	113	0	0	0.01	0
228	30	448	299	159	n/a	92	94	97	n/a	0.01	0.01	0	n/a
	27	158	159	237	n/a	73	88	83	n/a	0	0	0.03	n/a
	24	162	156	148	n/a	78	70	85	n/a	0	0	0	n/a

Note: Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column. The gap is reported in percent and a rounding up function is used and therefore any fractional value $> 0 > 0$ and < 0.005 is reported as 0.01.

electrical engines, or by all 3 engine types.

First we discuss the results for the version with homogeneous engines, which can be seen in tables 3.3-3.5. For the heterogeneous engines the results can be seen in tables 3.6-3.8. For the root time, the table shows 4 columns, one for each of the instance types; (s), (ex3), (ex6), and (r). The main rows show the number of demands in the instances, and the 3 sub rows show the number of engines available in the experiments. Thus the leftmost cell with results shows the instance (s) with demand 90 and 30 engines available.

Table 3.6: Solution times, heterogeneous engines

D/C		Root time:				Time to IP:				Total time:			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	0.4	0.9	1.7	0.9	0.8	3.6	5	3.2	0.8	3.6	5	3.2
	27	0.5	1	2	0.8	1.7	3.9	7.1	3	1.7	3.9	7.1	3
	24	0.5	1	1.9	0.7	0.8	4.4	6.5	2.9	0.8	13.1	14.5	2.9
120	30	1.1	2.4	3.9	2	4.9	11.1	19.3	7.6	4.9	11.1	19.3	7.6
	27	1.1	3	4.8	2.1	5.6	13.3	22	7.6	5.6	13.3	29.6	7.6
	24	1.3	3.1	5.1	2.1	7.5	13.3	29.3	7.8	21.3	13.3	> 720	7.8
150	30	3.4	5.4	10.3	2.8	15.7	19.4	48.7	9.9	15.7	19.4	48.7	9.9
	27	3.7	6.8	11.5	3.6	18.8	27.2	58.9	12.2	18.8	27.2	58.9	12.2
	24	3.1	4.4	9.9	3.8	18.3	29.2	59.1	11.9	> 720	> 720	119.9	11.9
180	30	7.4	19.6	21.3	9.1	37.1	71.4	79.8	29.8	37.1	71.4	117.5	29.8
	27	6.6	23.3	21	8.2	40.8	100.5	105.4	32.4	40.8	100.5	105.4	32.4
	24	7.4	21.8	20.7	8.6	42.6	130.6	137.1	45.1	> 720	> 720	137.1	45.1
210	30	17	34.4	36.5	15.5	82.5	165.9	217.5	54.1	82.5	165.9	217.5	54.1
	27	12.9	53.4	46.8	17.1	118.1	291.7	270.6	69	118.1	1173	270.6	69
	24	13.1	30.6	41.3	18.1	135.6	222	313.2	91	> 720	> 720	313.2	> 720
228	30	25.4	25.1	56.5	n/a	189.1	188.6	382.3	n/a	189.1	188.6	382.3	n/a
	27	22.5	22.6	67.7	n/a	155.8	155.5	535.6	n/a	> 720	> 720	535.6	n/a
	24	22	23.2	51.6	n/a	296.1	302.3	692.8	n/a	> 720	> 720	> 720	n/a

Note: Time is reported in minutes. Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column.

Table 3.5 shows that for most instances the initial integer solution found is equal to the objective value of the root LP-relaxation and thus optimal. For some instances we find an integer solution very close to the root relaxation and thus have to continue branching. It takes considerable time to close the gap and prove optimality. The gap is relatively small in the range of <0.01%. To use a solution that may be sub-optimal, but with a very small gap is not a critical worst case scenario in practice. In a few cases the algorithm does not prove optimality within the 12 hour preset time limit. However, it finds a high quality integer solution for all instances within 4 hours.

As expected, table 3.3 clearly shows that the solution time decreases significantly

Table 3.7: Pricing-problem algorithm calls, heterogeneous engines

D/C		Heuristic				2-cycle elim.				ESPPRC			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	238	546	460	864	11	58	32	195	0	0	0	1
	27	427	623	716	796	41	47	53	172	0	0	0	2
	24	259	1605	1351	811	6	114	102	187	0	0	0	0
120	30	822	970	1052	1293	166	152	184	244	0	0	0	1
	27	950	1418	1519	1362	105	97	135	242	0	0	0	1
	24	2883	1342	23151	1456	226	71	2628	273	0	0	0	5
150	30	1971	2202	2312	2123	229	185	256	401	0	0	0	6
	27	2255	3208	3369	2343	141	152	200	408	0	0	0	7
	24	48053	54888	4621	2657	2518	2712	226	377	0	0	0	4
180	30	3751	3493	6608	2624	245	251	450	393	0	0	0	2
	27	3378	4653	5292	3276	216	231	248	498	0	0	0	3
	24	15776	7070	5392	4772	686	414	176	635	0	0	0	1
210	30	5124	6847	7451	4086	337	305	486	526	0	0	0	2
	27	5554	20186	7843	5577	292	887	284	607	0	0	0	8
	24	7792	10863	8758	6610	289	428	282	519	0	0	0	14
228	30	7897	7897	8911	n/a	452	452	411	n/a	3	3	5	n/a
	27	10436	10436	9923	n/a	258	258	412	n/a	4	4	1	n/a
	24	9213	8899	5402	n/a	195	216	119	n/a	7	9	6	n/a

Note: Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column.

as the instances get smaller. When we compare the (s)-instances with the (r)-instances, it can be seen that the latter have a faster runtime. This could be explained by the fact that these demands are more evenly spread over the week and thus the infrastructure becomes less restrictive. Solving the two expanded time window sets (ex3 & ex6) is more time-consuming as would be expected. This is mainly due to the fact that they result in much larger pricing-problems that have to be called more often.

For all instances the time to solve the root node is much shorter than acquiring a feasible integer solution. Combined with the fact that the root relaxation provides a very strong lower bound this means that we can evaluate the consequences of changes

Table 3.8: Branching method and root gap, heterogeneous engines

D/C		Branch A:				Branch B:				%-gap			
		s	ex3	ex6	r	s	ex3	ex6	r	s	ex3	ex6	r
90	30	8	51	28	77	0	1	0	59	0	0	0	0
	27	32	31	36	75	0	5	1	51	0	0	0	0
	24	4	62	55	74	0	3	0	54	0	0.03	0.03	0
120	30	86	83	100	95	63	42	50	68	0	0	0	0
	27	71	68	76	97	10	11	8	76	0	0	0.01	0
	24	103	53	4970	96	17	9	24	75	0.01	0	0.05	0
150	30	89	86	112	127	49	39	37	111	0	0	0	0
	27	81	85	86	124	25	20	20	107	0	0	0	0
	24	671	176	102	111	1014	18	11	79	0.09	0.04	0.01	0
180	30	117	99	217	145	63	44	63	100	0	0	0	0
	27	107	100	113	138	40	43	36	95	0	0	0	0
	24	329	212	100	144	3	24	13	87	0.03	0.01	0	0
210	30	141	126	147	163	59	61	65	128	0	0	0	0
	27	121	419	125	162	68	101	41	104	0	0.01	0	0
	24	157	210	136	226	45	146	52	63	0.01	0.03	0	0.01
228	30	145	145	147	n/a	74	74	71	n/a	0	0	0	n/a
	27	90	90	141	n/a	70	70	57	n/a	0.06	0.06	0	n/a
	24	103	102	52	n/a	87	43	32	n/a	0.02	0.03	0.1	n/a

Note: Each column shows the value for the instance type labeled at the top with demand and engines given in the leftmost column. The gap is reported in percent and a rounding up function is used and therefore any fractional value > 0 and < 0.005 is reported as 0.01.

rapidly for practical instances and later get the actual solution, if necessary.

Tables 3.6-3.8 hold numerical data for the heterogeneous engines and show that the runtime increases compared to the homogeneous case. Although the introduction of multiple engine types causes the runtime to increase, the average time to reach an integer solution increases from about 80 minutes to about 155 minutes. The runtime roughly doubles when we solve 3 pricing-problems instead of 1 and add additional complexity to the model. Note, however, that we do not solve all pricing-problems in every iteration unless necessary. Each individual pricing-problem is smaller than the

single pricing-problem.

When we compare tables 3.4 & 3.7 it seems that the two algorithms behave in the same way with respect to the pricing-problems being called. The heuristic is often enough to provide usable columns and the 2-cycle elimination algorithm is very effective. For all reduced sized (s, ex3, ex6)-instances, the 2-cycle elimination always ensures elementary columns. The reason for this is that the time window length compared to travel times does not allow for cycles longer than 2. In the remaining full size and (r)-instances, we have demands that cross the end of the week and therefore they are represented both in the beginning and in the end of the week. The demand can therefore be serviced either in the beginning of the period or towards the end. This could obviously be a cycle of much greater length and therefore it may be necessary to use the ESPPRC algorithm to ensure elementary columns in these cases.

Tables 3.5 & 3.8 show that branching method A is used more often than B, which is expected as it has first priority. Branching method B is not used in some instances. Those instances are all small and capacity constraints are not as binding as in the large instances.

3.6 Conclusions

In this chapter we have presented a model to solve the problem of creating a yearly timetable for a freight railway operator taking into consideration the infrastructure paths made available. We have proved that the model is able to provide high quality integer solutions to real-life sized instances. It solves relatively fast and thus leaves room for flexibility in problem size and time window lengths for practical applications when considering that we have a fairly long planning horizon. We have analyzed the running time effects of reducing the number of available engines and of increased time window sizes. The model also provides a fast lower bound that can be used to

assess the consequences of changes in the timetable, the consequences of reducing the number of engines available to assess the marginal cost of this, or the possibility of serving additional demands.

3.7 Acknowledgments

Part of this work has been carried out while Lukas Bach was visiting CIRRELT and we would like to thank CIRRELT for hosting this visit. We would also like to thank DB Schenker Rail Scandinavia for providing real-life test data. This work is partly sponsored by NordForsk project #25900. Finally, part of the data has been collected in cooperation with Anders Kastberg, whom we would like to thank for the collaboration.

Chapter 4

Integrating Timetabling and Crew Scheduling at a Freight Railway Operator

History: *This chapter has been prepared in collaboration with Dennis Huisman and Twan Dollevoet, the work has been performed during a period with numerous short term research visits at the Econometric Institute, Erasmus University Rotterdam. The chapter has been presented at: TRISTAN, 8th Triennial Symposium on TRANSPORTATION ANALYSIS, 9-14 June, 2013, San Pedro de Atacama, Chile, and OR 2013, the International Conference on Operations Research, 3-6 September, 2013, Rotterdam, the Netherlands.*

Integrating Timetabling and Crew Scheduling at a Freight Railway Operator

Lukas Bach[†], Dennis Huisman[‡*] and Twan Dollevoet[‡]

[†]Cluster for Operations Research And Logistics,
Department of Economics and Business,
Aarhus University, Denmark

[‡]Econometric Institute and
Erasmus Center for Optimization in Public Transport (ECOPT),
Erasmus School of Economics, Erasmus University Rotterdam,
Rotterdam, the Netherlands

[*]Process quality & Innovation,
Netherlands Railways, Utrecht, the Netherlands

Abstract

We investigate to what degree we can integrate a Train Timetabling / Engine Scheduling Problem with a Crew Scheduling Problem. In the Timetabling Problem we design a timetable for the desired lines by fixing the departure and arrival times. Also, we allocate time-slots in the network to secure a feasible timetable. Next, we assign engines in the Engine Scheduling Problem to the lines in accordance with the timetable. The overall integration is achieved by obtaining an optimal solution for the Timetabling / Engine Scheduling Problem. We exploit the fact that numerous optimal, and near optimal solutions exist. We consider all solutions that can be obtained from the optimal engine schedule by altering the timetable,

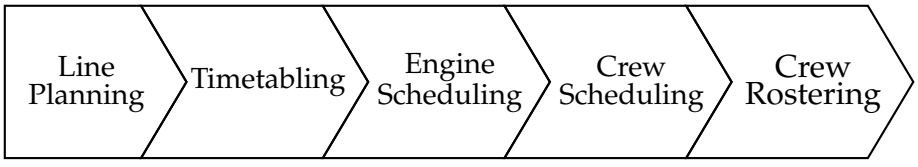
while keeping the order of demands in the schedules intact. The Crew Scheduling model is allowed to re-time the service of demands if the additional cost is outweighed by the crew savings. This information is implemented in a mathematical model for the Crew Scheduling Problem. The model is solved using a column generation scheme. Hereby it is possible for the Crew Scheduling algorithm to adjust the timetable and achieve a better overall solution. We perform computational experiments based on a case at a freight railway operator, DB Schenker Rail Scandinavia, and show that significant cost savings can be achieved.

Keywords: Railway Crew Planning, Vehicle and Crew Scheduling, Partial Integration, Time Windows, Branch-and-Price

4.1 Introduction

In this chapter we investigate to what extent we can integrate a Train Timetabling / Engine Scheduling Problem and a Crew Scheduling Problem (CSP). Our approach is inspired by a case at a freight railway operator, DB Schenker Rail Scandinavia (DBSRS).

The planning process at railway operators has been described at different levels of detail by, among others, Huisman et al. (2005), Caprara et al. (2007) and Lusby et al. (2011). In general, the planning process can be described in five phases as follows:



In the Timetabling Problem we design a timetable for the desired lines - from the Line Planning Problem - and fix the departure and arrival times. Also, we allocate

time-slots in the network to secure a feasible timetable. In the Engine Scheduling Problem we assign engines to the lines in accordance with the timetable. We ensure that this allocation produces a feasible plan. In Chapter 3 of this thesis we solve an integrated version of these two problems at a freight railway operator.

In this chapter we seek to integrate the Timetabling and Engine Scheduling phases with Crew Scheduling. The overall goal is to investigate whether it is favorable to leave these earlier stages open to adjustments in the Crew Scheduling phase. To this end, we first solve a CSP based on integration with the Timetabling and Engine Scheduling Problem (TESP). We seek to integrate by allowing changes to the timetable in the CSP phase. We do this while keeping the solution to the TESP feasible. Hence, we explore alternative feasible solutions. We then compare this to a completely sequential approach, where we solve the CSP in a classic approach without any link to the timetabling or engine scheduling phases.

The remainder of this chapter is structured as follows. In Section 4.2, we describe the problem and introduce case specific details, and in Section 4.3, we review the relevant literature. In Section 4.4, we explain the integration with the TESP and present our integrated model, we present the solution method hereto in Section 4.5. In Section 4.6, we discuss the results of our computational experiments. Finally, we present our conclusions and future research directions in Section 4.7.

4.2 Problem Description

The demand in this model is strictly unit train demand: Any demand is for a full train driving from an origin to a destination station. It is not possible to aggregate demand. For each of these demands, we have a fixed time window wherein the demand should be fulfilled. For the crew, a single demand can be split into one or more tasks at relief points along the route. In the case considered in this chapter there is a maximum of

228 demands per week.

We consider a planning horizon of one year during which the timetable has to be repeated each week. The timetable design is typically done more than half a year in advance. The contracts between DBSRS and their customers in general cover the timetabling period of a year.

The transit times and time-slots in the operational area of DBSRS are allocated by the infrastructure manager, who also assigns access to the network. We use these time-slots to model what times we can access the infrastructure. The time-slots are published with a time resolution of 1 minute and are available about every half hour in the main corridor. It is thus only possible to start usage of the tracks at discrete points in time.

We have a number of engines available and generate the same number of engine schedules, each containing a set of demands to perform during a week. A specific engine can perform different engine schedules from week to week. This can lead an engine to start and end at different stations in the beginning and end of the week. However, this is balanced over all engine schedules. For an engine schedule to be feasible, it must satisfy a set of constraints (mainly it can only use tracks when allowed to do so by a time-slot) and include buffer times at the beginning and end of demands, forcing the plan to be more robust.

The crew at DBSRS are divided into three major groups separated by the country in which they are employed. This means that they also have a different set of working regulations. Within these groups, the crew are furthermore assigned to crew bases within their country. The crew should always end their duty at their home base, but can travel anywhere otherwise allowed by the working regulations.

For a duty to be feasible it must respect the rules in Table 4.1. For German drivers, a duty becomes a night duty if 3 or more hours of the duty are within night time. In contrast, for Danish drivers, it is a night duty if just 1 minute is within night time. In

Table 4.1: Duty related labor rules

	Germany	Denmark	Sweden
Min work hours	5	6	3
Max work hours	14	9	10
Max drive time	8	9	10[9]
Max drive time ND*	8	9[8]	10[8]
Max time without break	5.5	4.75	5
Length of break	(duty $<$ 8): 0.5	0.5	0.5
...	(duty $<$ 12): 0.75		
...	(duty $<$ 14): 2		
Night time	23:00 - 6:00	1:30 - 4:30	22:00 - 6:00

Note: *ND(Night Duty). Values in [] are for cross-border traffic, where it is more restricting than native rules.

Sweden there are only work time regulations related to a night duty and no fixed cost for a night duty. Instead, the cost of the actual time during night time is compensated.

If a driver crosses a border, EU rules are enforced. The rules for offsetting a night duty are only affecting Danish drivers, who with respect to this, now have the duty defined as a night duty similar to the German rules. For Danish and Swedish night duties, the maximum drive time is reduced to 8 hours. Further, the maximum drive time during the day becomes 9 hours.

As part of any duty there is a set of mandatory tasks. These are prepare and finalize duty tasks. To have a break, start, or end a duty, it is necessary to walk to/from a break room. This walking time is dependent on the station used, and it is added to the time of the task. There is also a maximum work time without a break. This ensures that a break is placed at a reasonable time during a duty. For all drivers the minimum break time to reset the time without break is 30 minutes. We refer to this as the minimum break. As it can be seen from Table 4.1, it is sometimes necessary to have at least one longer break for the duty to be feasible. We call this the required break.

The cost of a duty is modeled as a fixed part and variable part, this is roughly $\frac{1}{3}$ fixed

and $\frac{2}{3}$ flexible. Having some of the cost flexible allows for a more balanced solution, not maximizing duty length. How the cost should be distributed between fixed and flexible is a managerial problem, the model can be used with any combination of the two. The variable part consists of a time-dependent cost, costs for night duties, and costs for cross-border activities.

The problem then becomes to cover all crew tasks during a week at the minimum cost using the crew from the three different countries.

4.3 Literature

Many successful applications of Operations Research in the railway context have been discussed in the literature over the last decades (see Kroon et al. (2009) for an example). Traditionally, the planning process at a railway operator is decomposed into five levels. The Crew Scheduling Problem (CSP) is the fourth level and is usually solved when the timetable and the engine schedules are determined. In what follows, we review the recent literature on the CSP and on the integration of earlier levels with the CSP. For more information on the other scheduling levels, we refer to Caprara et al. (2007), Huisman et al. (2005), or Lusby et al. (2011) and references therein. We also refer to Chapter 3 for more on the Timetabling and Engine Scheduling Problem.

In the seminal study by Caprara et al. (1999), the CSP is modeled as a set covering problem and solved by column generation. In this approach, the columns represent feasible duties, and the rows correspond to the tasks that have to be performed. The master problem is solved by Lagrangean relaxation, while feasible duties are generated in the pricing problem by solving a resource-constrained shortest path problem.

Recent studies on the CSP mainly focus on developing acceleration techniques in order to solve large-scale problems. Elhallaoui et al. (2005) aggregate multiple tasks into one and thereby reduce the size of both the master and the pricing problems. By

updating the aggregation dynamically, the problem can still be solved to optimality.

Jütte and Thonemann (2012) divide the CSP into multiple regions and price the assignment of trips to these regions. This procedure is implemented by Jütte et al. (2011) for a real-world instance from DB Schenker. The authors show that large-scale instances can be solved in a reasonable computation time.

Several algorithms for the CSP have found their way to commercial decision support systems. PowerSolver is developed by Kwan and Kwan (2007) and applied to several instances from the UK. Abbink et al. (2011) describe LUCIA, a crew scheduling algorithm used by Netherlands Railways that can solve instances with tens of thousands of tasks. LUCIA is based on an algorithm for the crew rescheduling problem that was developed by Huisman (2007).

All the algorithms described so far deal with crew *planning*, where computation times of hours or days are acceptable. When disruptions occur in the daily operations, crew duties should be rescheduled in real-time. Potthoff et al. (2010) describe a method to reschedule duties within minutes for a case of Netherlands Railways. Similarly, Rezanova and Ryan (2010) develop a real-time crew rescheduling approach based on set partitioning and test it on real-life instances from Denmark. In this chapter we apply the CSP to the case from DBSRS. A particular difference is that we consider crew employed in different countries who thus work under different working regulations. The crew considered work in Sweden, Denmark, or Germany. The main working regulations are similar. Yet some differences have to be addressed. Papers considering train working regulations related to these countries include; Rezanova and Ryan (2010) for a Danish, and Jütte et al. (2011) for a German setting.

The integration of single-depot Vehicle and Crew Scheduling is studied in Freling et al. (2003). Huisman et al. (2005) extend their model to multiple depots and focus on a sub/extraurban transit system. They show that there are significant cost savings to be achieved. This conclusion is supported by Groot and Huisman (2008), Mesquita

et al. (2009), and Steinzen et al. (2010), among others. All these papers consider a full integration of the Vehicle and Crew Scheduling Problem.

Kliewer et al. (2012) extend the Vehicle and Crew Scheduling Problem with the addition of time windows for trips. By allowing to shift trips with up to 4 minutes in time, they achieve a further cost reduction. The approach is based on an integrated model presented by Steinzen et al. (2010) and tested on real-life data from public bus transportation. A similar approach to re-timing can be seen in Veelenturf et al. (2012), where a Crew Rescheduling Problem at Netherlands Railways is considered. Here, minor changes to the timetable result in improved solutions.

Gintner et al. (2008) present a partial integration of Vehicle and Crew Scheduling. The optimal vehicle flow can be decomposed into an optimal vehicle schedule in a number of different ways. All these alternative solutions to the Vehicle Scheduling Problem are implicitly explored in the crew scheduling phase. This chapter also partially integrates timetabling and vehicle scheduling with crew scheduling. Our approach allows to change the timetable, but leaves the order of the trips in the vehicle/engine schedule unchanged. In contrast to Kliewer et al. (2012), who also allow some degree of re-timing, our model is for a railway system whereas their model is for a public bus transit system. The most significant difference is that the access to the railway infrastructure has to be taken into account.

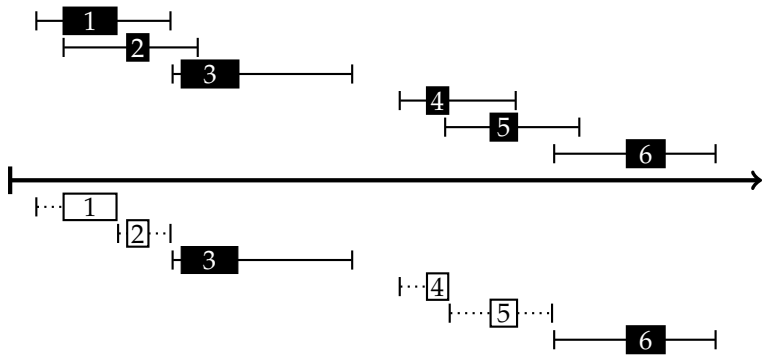
4.4 Methodology

Both the combined Timetabling and Engine Scheduling Problem (TESP) as well as the Crew Scheduling Problem are solved using a column generation approach. We compare the integrated approach to the sequential approach, where we solve the CSP using a timetable set by the TESP. By comparing this to the integrated approach, we can evaluate the quality and the possible improvements by integration.

4.4.1 Integration

When solving the TESP as in Chapter 3, there are a number of alternative feasible solutions. In the integrated approach, we seek to explore these by allowing re-timing of the individual demands. An engine schedule will remain feasible as long as the order of the demands is unchanged. In an optimal solution, a demand is serviced at a point in time within its time window, and by a specific engine schedule. An engine schedule services one or more demands. The demands in a schedule can be far apart time-wise, such that service at any point in time within their time window is allowed. If this is the case, we say that a demand can move freely within its time window. This holds for many demands. The others are either fixed or can use a restricted part of the time window. We will explain this in the following section. We always keep the TESP feasible. To maintain feasibility, we need to make sure that no more than one demand use the same time-slot. This approach explores many alternative solutions for the crew.

Figure 4.1: Time window reduction



To keep the schedules feasible, we adjust the time windows of the demands, maintaining the order of demands and thus keeping the schedules feasible. The algorithm we use to adjust the time windows keeps a time window unchanged if there is no overlap with the next or the previous time window. Otherwise we start from the time-slot used by the optimal solution and keep the next time-slot intact while there is no overlap. When it reaches an overlap to the next time window, the current time window is

cut. In Figure 4.1, the top part represents an engine schedule and the time windows of the demands. On the bottom part the adjusted time windows are shown.

4.4.2 Timetabling and Engine Scheduling Problem

The main objective in the Timetabling and Engine Scheduling Problem is to fix the timetable. The TESP is formulated and solved as in Chapter 3, and the solutions hereof are used as input for the integrated approach. No variables explicitly define the timetable. Instead, the timetable is extracted from the time-slots chosen in the model.

The TESP is formulated as a Set Partitioning Problem, with Ω being the set of engine schedules, r, under consideration. D is a set of all demands, d. N is the set of all stations, indexed by n. S is a set of all time-slots, s. E is the set of engine types, e. w_e is the engine availability for engine type e, given as a parameter.

$$\min \sum_{r \in \Omega} c_r x_r \tag{4.1}$$

$$\text{s.t.}, \sum_{r \in \Omega} \alpha_r^d x_r = 1, \quad \forall d \in D \tag{4.2}$$

$$\sum_{r \in \Omega} \beta_r^n x_r = 0, \quad \forall n \in N \tag{4.3}$$

$$\sum_{r \in \Omega} \gamma_r^s x_r \leq 1, \quad \forall s \in S \tag{4.4}$$

$$\sum_{r \in \Omega} \delta_r^e x_r \leq w_e, \quad \forall e \in E \tag{4.5}$$

$$x_r \in \{0,1\}, \quad \forall r \in \Omega \tag{4.6}$$

The following parameters are determined in the pricing problem: c_r is the cost of schedule r. The parameter α_r^d indicates if demand d is covered by schedule r. β_r^n is equal to 1 if a station n is used by schedule r as origin station, if it is used as destination station it is equal to -1. If a station is used as both origin and destination or neither, β_r^n

is equal to 0, i.e., the schedule is balanced with respect to this station. If time-slot s is used by schedule r, then γ_r^s is equal to 1. We set δ_r^e equal to 1 if schedule r is driven by engine type e. The integer program looks as follows:

The objective function (4.1) minimizes the cost of the selected engine schedules. In constraints (4.2) we make sure that all demands are serviced by the engines. To ensure a necessary balance between the starting and ending stations of the engines, we have the balance constraint, constraints (4.3). In constraints (4.4) we ensure that each time-slot is used at most once. Engine availability is ensured by constraints (4.5).

In the pricing problem we handle one weekly schedule of an engine in the network. To get the engine schedule, we solve a Shortest Path Problem. The start of service for the demand must be within its time window, which is defined by $[a_d, b_d]$ where a_d is the earliest start and b_d is the latest start of service. Within the time window demands can only be serviced when a path is available. Hence, service can be initiated at a set of discrete times t stored in the set U_d. The selected time of service is induced from the set of time-slots used by the schedule and is in this way connected to the master-problem. Furthermore, constraints considering shunting time, wait time after demands, deadheading, and flow conservation for intermediate stations are treated in the pricing problem.

4.4.3 Crew Scheduling Problem

The CSP is formulated as a Set Covering Problem, and solved by a column generation approach in which we generate duties ad-hoc in a pricing problem. From the TESP we have demands $d \in D$. Each d is a demand going from one station to another. On this route there can be one or more relief points for the duties. Thus d is split into one or more crew tasks. P is the set of all tasks, indexed by p. Ω is the set of all duties (columns) in the restricted master problem, indexed by r. The cost of a duty, r, is given by c_r, a_r^p indicates if a crew task p is covered by duty r.

$$\min \sum_{r \in \Omega} c_r y_r \tag{4.7}$$

$$\text{s.t.}, \sum_{r \in \Omega} a_r^p y_r \geq 1, \quad \forall p \in P \tag{4.8}$$

$$y_r \in \{0,1\}, \forall r \in \Omega \tag{4.9}$$

The objective function (4.7) minimizes the cost of the selected duties. Constraints (4.8) ensure that all tasks are covered by at least one duty. The duties are generated in multiple pricing problems by solving a Shortest Path Problem on a graph representing the tasks and adhering to the working regulations, see Section 4.4.5.

4.4.4 Integrated Model

Constraints (4.1)-(4.6) form the master problem for the TESP, and constraints (4.7)-(4.9) form the master problem for the CSP. To integrate the two problems, we must connect the timing of the engines with the crew duties. To include the engines, we take the set of demands D. For each d we take the corresponding set U_d and make a copy of d for each time t at which the demand can be performed. We assign these copies d' to the set D'_d. The set D'_d now contains a node for each time it is possible to initiate service of d. For notational convenience, we define D' as the union of the sets D'_d over $d \in D$.

We introduce the decision variable $x_{d'}$ which is equal to 1 if the demand is performed at the given time, 0 otherwise. Thus we also add the constraints in equations (4.10) to an integrated master problem. We hereby ensure that we service each demand exactly once.

$$\sum_{d' \in D'_d} x_{d'} = 1, \quad \forall d \in D \tag{4.10}$$

Each demand d corresponds to one or more tasks p. If we move service of a demand in

time, we also have to cover the crew tasks at different times. For each d' we also make a set of copies of the corresponding crew tasks p, so we now have a p' for each $d' \in D'_d$ contained in a set P'_p for all tasks $p \in P$. For each of these copies we add a decision variable $z_{p'}$, which is equal to 1 if the task is performed at the time the copy represents, or 0 otherwise. We also have to introduce constraints such that only one of these are used. Veelenturf et al. (2012) introduce successor sets to link tasks together. We do not need sets as we only have one feasible successor task, because we can only 'delay' the train initially, whereas they can delay the trains at intermediate stations. Therefore, we introduce $succ(p')$ which points to the next task of p', $succ(p') = \emptyset$ if there is no successor, i.e., it is the last task of the demand. The relationship is kept by constraints (4.11).

$$z_{p'} - z_{succ(p')} = 0, \forall p' \in \{P' : succ(p') \neq \emptyset\} \tag{4.11}$$

We link the demand copies $d' \in D'$ in the same way to the first task, so the time chosen for service with the engine always corresponds to the time chosen for the crew tasks. Hence, we define $task(d')$ as the first task p' corresponding to d'. Then we form constraints (4.12) which are similar to constraints (4.11).

$$x_{d'} - z_{task(d')} = 0, \quad \forall d' \in D' \tag{4.12}$$

We also let the sum of copies of the same task be equal to 1, so only one time for a task can be chosen. The constraint is given in equation (4.13).

$$\sum_{p' \in P'_p} z_{p'} = 1, \quad \forall p \in P \tag{4.13}$$

We let the parameter $\gamma_r^{p'}$ be equal to 1 if task copy p' is used by schedule r, 0 other-

wise. Let the parameter EDC be the engines' driver capacity such that $EDC = 2$ is interpreted as there can be 1 engine driver and 1 deadheading engine driver in the engine. Then we let constraints (4.14) ensure that $z_{p'}$ attains a positive value if the corresponding task copy is chosen.

$$EDCz_{p'} - \sum_{r \in \Omega} \gamma_r^{p'} y_r \geq 0, \quad \forall p' \in P' \tag{4.14}$$

After introducing constraints (4.10)-(4.14) and variables $z_{p'}$ and $x_{d'}$, it can be seen from equation (4.11) and (4.12) that $x_{d'}$ and all $z_{p'}$ corresponding to this demand copy will always be equal. This is because there is always one and only one successor for each p'. Because of this, we can remove the variables $z_{p'}$ from the model if we replace them with $x_{d'}$ in constraints (4.14). The addition of the parameter $dem(p')$, which points to the d' that a crew task is derived from, makes it possible to replace $z_{p'}$ with $x_{dem(p')}$ in equations (4.14).

From the train scheduling problem we know that some demands at a certain time use the same time-slots. When allowing changes to the optimal solution, we need to make sure that no more than one demand is using a time-slot. For all $s \in S$, we let M_s be the set of demand copies d' using the same time-slot s. To ensure that no more than one of the demand copies use the same time-slot, we add constraints (4.15).

$$\sum_{d' \in M_s} x_{d'} \leq 1, \quad \forall s \in \{S : |M_s| > 1\} \tag{4.15}$$

The objective function (4.16) together with constraints (4.17)-(4.22) form the integrated master problem. We explore sub-optimal solutions from the TESP. The costs of the underlying time-slots are time-dependent in cost. Hence, it is necessary to introduce the cost of choosing a sub-optimal solution, this is captured by $c_{d'}$.

$$\min \sum_{r \in \Omega} c_r y_r + \sum_{d' \in D'} c_{d'} x_{d'} \tag{4.16}$$

$$\text{s.t.}, \sum_{d' \in D'_d} x_{d'} = 1, \quad \forall d \in D \tag{4.17}$$

$$\sum_{d' \in M_s} x_{d'} \leq 1, \quad \forall s \in \{S : |M_s| > 1\} \tag{4.18}$$

$$\sum_{r \in \Omega} \alpha_r^p y_r \geq 1, \quad \forall p \in P \tag{4.19}$$

$$EDC x_{dem(p')} - \sum_{r \in \Omega} \gamma_r^{p'} y_r \geq 0, \quad \forall p' \in P' \tag{4.20}$$

$$y_r \in \{0, 1\}, \quad \forall r \in \Omega \tag{4.21}$$

$$x_{d'} \in \{0, 1\}, \quad \forall d' \in D' \tag{4.22}$$

A stronger bound can be achieved by adding the constraints in (4.23). The parameter EDC in (4.21) is of a big M type. In the relaxation this allows for fractional x variables. This indicates that tasks corresponding to the same demand are serviced at different times.

$$\sum_{r \in \Omega} \gamma_r^{p'} y_r - x_{dem(p')} \geq 0, \forall p' \in P' \tag{4.23}$$

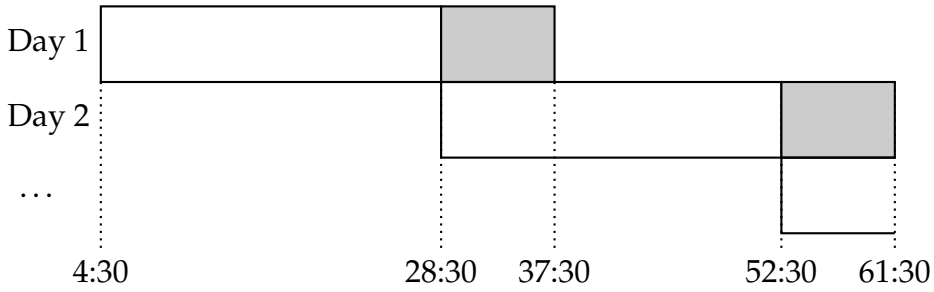
Equation (4.23) ensures that all tasks in a demand are serviced at least with the same fraction as the corresponding demand. However, this is not part of our model as it also slows down the solution process significantly.

4.4.5 Pricing Problem

The pricing problem for duty generation is modeled as a Shortest Path Problem with Resource Constraints (SPPRC) on a graph. Using a shortest path algorithm on a cyclic network is a problem. Abbink et al. (2011) deal with this by splitting the pricing problem into days of the week. This gives one pricing problem per day, beginning with the

first time-period of the day and ending at the latest plus the maximum work time allowed in a duty. In this way all possible duties are captured, and the individual pricing problems are now acyclic.

Figure 4.2: Splitting the week into days



Note: Example for the Danish labor rules where the maximum working time is 9 hours.

Splitting the pricing problem into individual days could be done at a number of arbitrary points during the week with 24 hours in between. Intuitively this would be at midnight which would separate the days. Figure 4.2 shows the days split at 04:30. Having multiple pricing problems also allows us to solve these in a parallel manner, not necessarily solving all the pricing problems to optimality in each iteration. In Section 4.5 the splitting of the pricing problem will be explained in greater detail.

4.5 Implementation

To establish a benchmark to see how much the solution to the CSP can be improved by our integration approach, we use our model in two ways. First, for the benchmark, we fix the x variables - representing the chosen timetable - to the solution provided by TESP. We then try to increase the lower bound by branching in a breadth first manner as we are in this case not interested in an integer solution, but in increasing the lower bound as much as possible.

The objective is to compare the cost of this benchmark solution to the cost of the integrated approach. For the comparison we use the total crew costs and the additional costs of choosing a sub-optimal engine schedule in the integrated approach.

In the integrated approach we let the timetable variables x flow freely within the adjusted time windows. To reach an integer solution, we first branch on the timetable variables, then we try to find an integer solution for the crew by performing a depth first search while branching on the tasks in the duties. In this section, we will cover the graphs for the pricing problems, the algorithm for finding the shortest path, and the branching approaches.

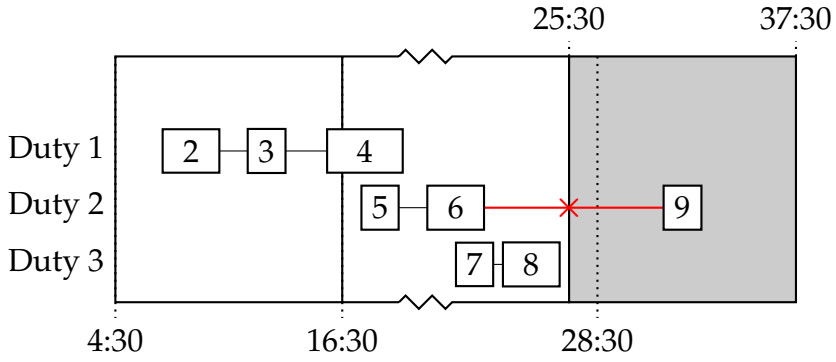
4.5.1 Pricing Problem Splitting

There are both different costs and additional working regulations depending on whether or not a duty is a night duty. To deal with this, we have to record whether a duty (for German and Danish drivers) is a night duty and apply the appropriate regulations. A resource could be added to the labels, recording whether we spend sufficient night time, and this resource could offset the night cost and the regulations. We can avoid this resource if we can a priori determine whether we are generating a night duty. We can do this if we, apart from splitting the pricing problem into days, also split it into day / night duties. By doing so, we avoid adding an extra resource.

Consider first the Danish rules for night duties. A Danish duty is a night duty if 1 or more minutes are spent between 1:30 and 4:30. If we split the day such that it begins at 4:30, then any duty should start between 4:30 and 28:30. As the maximal working time is 9 hours, duties can end at 37:30 at the latest. We now know that any task or arc crossing time 25:30 (24:00 + 1:30) will offset a night duty. As the graph is acyclic, this can indeed only happen once. In Figure 4.3, we depict the period between 4:30 and 37:30 for a specific day. We create two pricing problems for this instance. The first one contains only the nodes between 4:30 and 25:30. By construction, only day duties can

be generated in this pricing problem. Such a day duty can start and end anytime within this period. The second pricing problem will generate night duties only. It considers the period between 16:30 and 37:30. A duty can start in the period from 16:30 and 28:30 and should end after 25:30. Doing this we can be sure that the duty is a night duty.

Figure 4.3: Defining a night duty

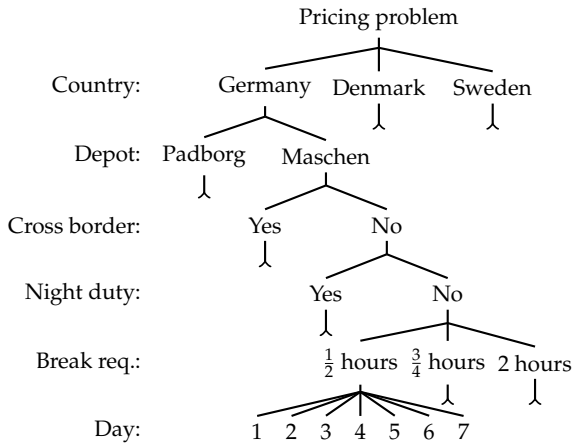


Note: Duties 1-3 contain a number of tasks. Duty 1 is a regular day duty, duty 2 is a night duty as it crosses 25:30. Duty 3 does not cross 25:30, so it is a day duty. As it ends before 25:30, it can only be generated in the pricing problem for day duties.

This can be extended to the German rules that define a night duty as a duty lasting 3 hours between 23:00 and 6:00. This is equivalent to spending one minute between 2:00 (23:00 + 3:00 - 24:00) and 3:00 (6:00 - 3:00). This holds because the minimum duty length is in any case more than 3 hours. Thus if we split the day at 3:00, we can apply a similar procedure as for the Danish rules, only adjusting the specific times at which duties can start and end.

Furthermore, the pricing problems are split by nationality and depot, such that any problem has a fixed start/end point and is governed by a given set of labor rules. For the German drivers where the length of the required break depends on the duty length, a separate pricing problem will be solved for the different possible lengths of the duty, i.e., less than 8, 12, and 14 hours. When drivers cross borders they must adhere to a set

Figure 4.4: Pricing problems



of EU working rules. By solving a separate pricing problem when we allow the drivers to cross borders, we still have a distinct set of working rules that does not change as a result of future decisions in the pricing problem.

In Figure 4.4 it can be seen how the final number of individual pricing problems increases by this method. In total with the rules applied, we get about 350 pricing problems.

4.5.2 Pricing Problem Graphs

In the pricing problem we solve the SPPRC on multiple graphs, which are generated as follows. The graphs consist of nodes representing tasks, each node has a specific work time (wt_i) and drive time (dt_i) given. The arcs connecting the nodes have work time (wt_{ij}) and drive time (dt_{ij}). They also have a break time (bt_{ij}) assigned, representing how long a break is possible between two tasks. A parameter, break at end (be_{ij}), indicates whether a break is at the end of an arc or in the beginning of it. If it is possible to place the break in either end, two parallel arcs are introduced. These parameters are important when calculating the time since break requirement.

For each individual pricing problem a graph is created with the nodes that fall within the time frame, regulations, and the geography of the problem. For example, a task is represented by a node if it is reachable from the depot within the maximum working time of the duty. If the problem contains cross border activities, only tasks where it is possible to connect to a cross border task are included. The same applies to night duties. Only tasks that can be part of a night duty are included. For non-night duties the opposite applies.

To reduce the probability of cycles in the duties, the following rules are enforced prescribing which nodes can be connected. First, we define a cycle as a duty performing the same task at two different times. This is an infeasible duty, but it can be generated because we do not enforce elementary duties. If we do not allow any node to connect to any other node that represents the same task at a different time, we reduce the number of infeasible connections. This can be expanded to disallow any connection from a node to any other nodes coming from the same engine demand but at a different time.

Finally, we can remove connections that will not be feasible with respect to the time-slots in the master problem. Because some copies of different demands might use the same infrastructure, only one of these can be chosen. Equivalently, servicing tasks from different demands that will use the same infrastructure simultaneously, cannot be part of an integer solution.

4.5.3 Resource Extension Functions

The Shortest Path Problem with Resource Constraints (SPPRC) is solved on the graphs described in Section 4.5.2. A labeling algorithm is used to solve the problem. We introduce 4 resources that are necessary in this problem: work time (WT), drive time (DT), time since last break (TSB), and break (B).

Resource extension functions from node i to j are then formulated as follows:

$$WT_j = WT_i + wt_{ij} + wt_j \tag{4.24}$$

$$DT_j = DT_i + dt_{ij} + dt_j \tag{4.25}$$

$$TSB_j = \begin{cases} wt_j & \text{if } be_{ij} = 1, bt_{ij} \geq minBreak \\ dt_{ij} + wt_j & \text{if } be_{ij} = 0, bt_{ij} \geq minBreak \\ TSB_i + wt_{ij} + wt_j & \text{otherwise} \end{cases} \tag{4.26}$$

$$B_j = \begin{cases} 1 & \text{if } bt_{ij} \geq reqBreak \\ B_i & \text{otherwise} \end{cases} \tag{4.27}$$

We do not extend a label if it is not resource feasible, i.e., if the resources exceed the given maximums. For WT, DT, and TSB the maximums are dependent on the regulations and are thus different among the graphs. Any of these three resources can dominate another resource of equal or larger size. Let us define the capacities for the resources as $maxWorkingTime$, $maxDrivingTime$, and $timeWithoutBreak$.

The break resource (B) is a binary resource equal to 1 if it is fulfilled. The break resource can only dominate if it is greater than or equal to the resource of another label.

Furthermore, if WT + “drive time to the depot" exceeds the maximum work time, then we do not extend the label. If the required break has not yet been held, we do not extend the label if WT + “drive time to the depot" + “time of the required break" exceeds the maximum work time.

When solving the pricing problems, a large number of labels are created and extended at each node. A label with a higher resource consumption in either of the resources WT, DT, or TSB cannot dominate a label that has a lower consumption of just one of the resources. As a consequence, many of the labels with high resource consumption cannot be dominated, even if they appear to be 'bad' labels. If we can relax the dominance criteria so more labels can be dominated at each node, we can speed up the algorithm.

A label $\boldsymbol{a} \succ \boldsymbol{b}$ if all resources in $\boldsymbol{a}$ are dominating the resources in $\boldsymbol{b}$. In some cases we can improve this dominance rule by expiring some resources. The resource drive time (DT) can be expired - or set equal to 0 - if $(maxWorkingTime - WT) \leq (maxDrivingTime - DT)$. The label then dominates any other label with respect to DT.

We can ignore the drive time and expire the resource because the duty will still be feasible if we drive for the remainder of the allowed working time. A similar argument holds for the time since break (TSB) resource: If $(maxWorkingTime - WT) \leq (maxTimeWithoutBreak - TSB)$, then we can expire the TSB resource in a similar fashion. These additional rules aid the dominance of more labels and thus speed up the algorithm.

4.5.4 Solving the Shortest Path Problem

Splitting the pricing problem into multiple smaller pricing problems creates a number of smaller problems to solve. In each iteration of the Branch-and-Price algorithm we have to prove that no duties with negative reduced costs exist. Hence, it is necessary to solve all the individual pricing problems. In each intermediate iteration in the column generation it is only necessary to generate some columns with negative reduced costs. We do not need all possible columns nor the best. Thus, only a subset of the pricing problems need to be solved in each iteration. This subset is solved in parallel as the individual pricing problems are independent.

4.5.5 Branching

We branch in two stages, first for the timetable variables x, and second for the duties y. To reach an integer timetable we fix one $x_{d'}, \forall d' \in D'_d$ to 1 for all $d \in D$. Our branching approach has two ways of fixing this. First, we fix exactly one of the variables to 1 on the left branch and 0 on the right branch. Second, we split the set D'_d in two by the fractional values of the $x_{d'}$ variables, such that the fraction is equally divided on both branches.

The branching procedure first checks whether any $x_{d'}$ variables satisfy $0.8 \leq x_{d'} < 1, \forall d' \in D'$. If this is the case, we select the one closest to 1 for branching. If none of the $x_{d'}$ variables satisfy the criterion, the second rule is applied. In the second rule the $d \in D$ with the largest number of non-zero variables $x_{d'}, \forall d' \in D'_d$ is branched on.

To find integer solutions for the duties, we employ a classic follow-on branching scheme based on Ryan-Foster branching (Ryan and Foster, 1981) on the crew task. The implementation of this is similar to the implementation used in solving the TESP in Chapter 3. The follow-on scheme ensures that a task a is always performed immediately before a task b on the left branch, and task b is never performed immediately after task a on the right branch.

4.6 Computational Experiments and Results

Experiments have been carried out on an Intel Core i7 2.8 GHz and implemented with C++ using ILOG CPLEX 12.4 as the linear programming solver, as a parallel algorithm using up to 8 threads.

In this section we will first discuss implementation details. Then, we describe the instances used, before we present results for the pure Crew Scheduling Problem (CSP), and then finally, we will compare these results to those of the Integrated Crew Scheduling Problem (I-CSP).

4.6.1 Implementation Details

Implementing and solving the model gives some challenges: Finding an optimal solutions to the I-CSP model in reasonable time is not possible. Therefore it is solved in two stages. First the timetable related variables are fixed, and then we move to the crew phase.

As we will address later, the LP-relaxation is very tight when the timetable is fixed. But letting the timetable be adjusted gives a quite weak LP-relaxation. Thus branching to get the fixed/integer timetable takes a long time. This is the main reason as to why we do not solve the model to optimality. Instead we branch on the timetable as described but add an acceptance criterion for accepting a node. We either pick the 1-branch if the new objective value is less than 0.01% worse than the predecessor. If this criterion is not met, we choose the branch with the lowest objective value. In this way we can reach a fixed/integer timetable in reasonable time and then move on to solving the crew part of the problem. This part is solved as for the pure CSP and is solvable in reasonable time as will be shown in Section 4.6.3.

4.6.2 Instances

Observing Table 4.2 we have the instances used to test the algorithms. These are based on the instances and the results from Chapter 3. We have tested on three main groups of instances. The main parameter is the number of demands, which is the size of the set D. The second parameter is the number of engines available. We have two different test sets: one with 24 and one with 30 engines. If two test instances are equal in all parameters except the engines available, we will expect more dense schedules or higher utilization. This leads to less possible changes to the timetable when we modify the time windows. The set D' holds the different time-wise possibilities for serving the demands. The third parameter is the average width of a time window. In the stan-

Table 4.2: Data instances

Instance	TW width	Engine: Avail.	$\|D\|$	$\|D'\|$	Crew: $\|P\|$	$\|P'\|$	Excl. sets
CSP-90-24-s	6	24	90	650	462	3,424	637
CSP-90-30-s	6	30	90	772	462	3,947	713
CSP-90-30-ex3	9	30	90	1,032	462	5,287	779
CSP-180-24-s	6	24	180	1,131	956	6,059	1,118
CSP-180-30-s	6	30	180	1,312	956	6,903	1,265
CSP-180-30-ex3	9	30	180	1,779	956	9,712	1,586
CSP-228-24-s	6	24	228	1,511	1,214	7,905	1,555
CSP-228-30-s	6	30	228	1,745	1,214	9,104	1,711
CSP-228-30-ex3	9	30	228	2,295	1,214	12,355	2,024

dard "s" instances it is 6 hours, and in the extended time window instances "ex3", the time windows are extended by 3 hours, to 9 hours in total. For each demand in D, as described earlier, we split the demand into one or more tasks at relief points. The set P consists of all these tasks. The set P' holds all the crew tasks at different times of service. Finally, in the table we have the Exclusion sets, which are sets containing more than one demand d' that cannot be part of the same solution.

4.6.3 Computational Results

In Tables 4.3 and 4.4 we report results for the CSP and I-CSP. In the tables the column headings are interpreted as follows: The gap is the gap from the integer solution to the lower bound for the crew phase (LB-C), to the root relaxation for the crew phase (Root-C), and to the root relaxation for the timetabling phase (Root-TT). An * means that an optimal solution was found. In Table 4.4 the gaps are given as the average gaps over all repetitions. Hence an * means that an optimal solution was found for each of the repetitions for that instance. We also report run times. Here Root-TT is the time to solve the timetabling root node. Int-TT / Root-C is the time to reach an integer timetable, which is the same as solving the root node for the crew phase (Root-C). Int-C

is the additional time it takes to reach an integer solution for the crew. In Table 4.3 the column (Columns) states the total number of columns generated at the root node and during branching. The column (Branch nodes) gives the number of nodes explored in the branching tree.

In Table 4.3, we present test results for the CSP. From the table we can see that the

Table 4.3: Crew Scheduling Problem

Instance	% Gap:		Columns:		Branch	Time (seconds):		
	LB-C	Root-C	Root	Branch	nodes	Root-C	Int-C	Total
CSP-90-24-s	*	0.0174	5,552	807	23	15	20	26
CSP-90-30-s	*	0.0153	5,304	378	7	17	23	23
CSP-90-30-ex3	*	*	5,591	0	1	26	27	27
CSP-180-24-s	0.0538	0.0841	10,840	398,644	4,093	29	117	†
CSP-180-30-s	*	0.0115	10,202	35,646	388	35	88	484
CSP-180-30-ex3	0.0661	0.0903	10,918	406,115	3,812	59	175	†
CSP-228-24-s	0.0047	0.0240	12,840	394,961	2,041	51	239	‡
CSP-228-30-s	0.0436	0.0736	12,471	677,403	5,465	56	187	‡
CSP-228-30-ex3	0.0644	0.0821	12,876	665,440	4,916	88	297	‡

* indicate a zero gap, i.e., an optimal solution is found.
† is maximum runtime (3 hours) for medium sized instances (demand = 180).
‡ is maximum runtime (9 hours) for large sized instances (demand = 228).

model can be solved to proven optimality in reasonable time for the small instances that can be solved in less than 30 seconds. For the medium sized instances we can prove optimality for one instance. In general we get integer solutions very fast. In the worst case this is 3 minutes. Considering the real-life sized instances, we obtain feasible solutions for all of them within 5 minutes. These results indicate that the model scales rather well. This is primarily due to the structure of the pricing problems that are split into days. Thus, adding extra demands can at worst affect two consecutive days. Hence, not all pricing problems increase in size when adding a demand.

Looking at the gaps, it can be seen that the root relaxation is a very strong lower

bound on the integer solution. The worst root gap is less than 0.1%, a gap that is further narrowed to less than 0.07% when branching. In all cases this gap is so small that the root can be used as a very precise proxy for the objective value of the integer CSP solution. We exploit this in the two-phase approach described in the previous section. The results presented in Table 4.3 are used as benchmark solutions for evaluating the quality of the I-CSP solutions.

Table 4.4: Integrated Crew Scheduling Problem

Instance	**% Improvement:**			**% Gap:**			**Time (seconds):**		
	Avg	**Min**	**Max**	**LB-C**	**Root-C**	**Root-TT**	**Root**	**Int-TT**	**Int-C**
CSP-90-24-s	10.10	9.16	11.09	0.011	0.046	14.444	344	1512	24
CSP-90-30-s	12.63	11.87	13.23	0.013	0.059	14.929	494	2106	48
CSP-90-30-ex3	12.89	12.26	13.84	0.047	0.079	15.388	1165	3218	152
CSP-180-24-s	7.59	7.48	7.79	0.149	0.168	14.759	1103	5453	580
CSP-180-30-s	8.38	8.02	8.64	0.146	0.157	15.526	1636	7430	492
CSP-180-30-ex3	9.88	9.59	10.24	0.288	0.289	16.675	4810	14857	1865
CSP-228-24-s	7.11	6.28	7.62	0.188	0.195	15.544	2216	10442	779
CSP-228-30-s	7.42	6.88	8.08	0.145	0.152	16.377	3655	14922	1742
CSP-228-30-ex3	9.12	8.58	9.74	0.212	0.219	17.633	9856	26140	645

In Table 4.4, we present test results for the I-CSP. The solution method in itself is deterministic. However, as the pricing problems are solved in parallel, columns are inserted in different orders. This leads to different dual values and as a consequence, to different columns being generated in the following iteration. When branching, different branching decisions can be made due to having an alternative solution to the LP-relaxation. Because the algorithm is terminated before we explore the entire search tree, we might obtain a different solution in each repetition. Therefore, the table is based on multiple runs for each of the test instances. The table shows 6 repetitions for each instance.

For all instances a significant cost reduction is achieved in both worst and best case. Compared to the CSP, this cost reduction clearly shows that adjusting the timetable at

the CSP phase is an advantage. Observing Table 4.4, it is clear that the cost reductions fluctuate for each instance, which is expected due to the heuristic nature of the solution method. We believe that this lies within an acceptable range. A longer runtime will lead to greater stability as more alternative timetables can be explored. When comparing improvements among different test instances, it should be considered that these are not necessarily comparable. Consider a given timetable with engine schedules and its CSP solution, in theory the timetable can be the timetable that is also optimal for the CSP. In this case we would have a 0% improvement for the I-CSP. This would not tell us anything about the quality of the I-CSP method but rather that we were lucky to have the optimal timetable. Thus, when evaluating the consistency of the method, we compare among single instances. When evaluating the quality of the method, we note that for all test instances we get significant improvements, which shows that the method is useful.

From Table 4.4 it can be seen that improvements for the I-CSP are between 6.28% and 13.84%. The smallest and largest average improvement are 7.11% and 12.89%, respectively. From the results it is clear that the I-CSP delivers significant improvements compared to the benchmark cases. Observing the data sets with 90 demands, it can be seen that the least flexible *CSP-90-24-s* is also the one with the lowest average improvement, compared to *CSP-90-30-s* and *CSP-90-30-ex3*. The tendency that the improvement is linked to the degree of flexibility is also present for the instances with 180 and 228 demands. It is a tendency that strongly suggests that the improvements found stem from the degree of flexibility.

Another tendency suggests a correlation between the size of the demand and the improvement. This can be explained by the complexity of the underlying Timetable and Engine Scheduling Problem. Here the schedules are less compact in the smaller instances, and thus there is more flexibility left for the I-CSP. It could also suggest that the method is able to exploit the flexibility better on smaller instances. From Table 4.2 it

can be seen that there is more flexibility for each crew task and demand for the smaller than the larger instances.

To ensure that the stability of the method is consistent, we have tested the I-CSP approach with further repetitions on selected instances *CSP-228-30-s* and *CSP-228-24-s*. Here 12 repetitions has been performed without changing the conclusions of Table 4.4.

4.7 Conclusions and Future Research

In this chapter we set out to investigate to what extent the Crew Scheduling Problem (CSP) can be integrated into the earlier planning stages at a freight railway operator. We have presented a method for integration and a model for the Integrated Crew Scheduling Problem (I-CSP). We have suggested a heuristic Branch-and-Price approach to solve the I-CSP. The suggested algorithm has been tested on a set of test instances derived from a real-life case at a freight railway operator. The results from the I-CSP have been compared to a set of benchmark results obtained by solving a standard CSP model. The comparisons show that the proposed I-CSP method gives improvements from 6.28% to 13.84% for the different instances.

There are several interesting paths for future research. First, it has been shown that the lower bound obtained from LP-relaxation to the I-CSP is weak and finding it is time consuming. This affects the branching process and makes it impossible to close the optimality gap for real-life instances. By identifying key demands where flexibility is important and demands where flexibility is less or not important, it could be possible to search among a larger set of the good solutions. This could be implemented by examining the quality of the duties and only allowing changes to demands covered by low quality duties.

Second, we have shown that added flexibility greatly benefits the overall solution. Additional flexibility can be introduced by the way the time windows are split. The

current algorithm ensures that the decision on re-timing of one demand is completely independent of others. If instead we allow the full, original, time window to be used, while adding precedence constraints on the demands in the same engine schedule, we would achieve additional flexibility.

Bibliography

Abbink, E., L. Albino, T. Dollevoet, D. Huisman, J. Roussado, and R. Saldanha (2011). Solving large scale crew scheduling problems in practice. *Public Transport 3*(2), 149–164.

Ahr, D. (2004). *Contributions to Multiple Postmen Problems*. Ph. D. thesis, University of Heidelberg.

Amberg, A. and S. Voss (2002). A hierarchical relaxations lower bound for the capacitated arc routing problem. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences 3*.

Bach, L., G. Hasle, and S. Wøhlk (2013). A lower bound for the node, edge, and arc routing problem. *Computers & Operations Research 40*(4), 943–952.

Baker, E. K., J. S. DeArmon, and B. L. Golden (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research 10*(1), 47–59.

Baldacci, R., N. Christofides, and A. Mingozzi (2008). An Exact Algorithm for the

Vehicle Routing Problem Based on the Set Partitioning Formulation with Additional Cuts. *Mathematical Programming 115*, 351–385.

Baldacci, R. and A. Mingozzi (2009). A Unified Exact Method for Solving Different Classes of Vehicle Routing Problems. *Mathematical Programming 120*, 347–380.

Baldacci, R., A. Mingozzi, and R. Roberti (2011). New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem. *Operations Research 59*(5), 1269–1283.

Baldacci, R., A. Mingozzi, and R. Roberti (2012). Recent Exact Algorithms for Solving the Vehicle Routing Problem under Capacity and Time Window Constraints. *European Journal of Operational Research 218*, 1–6.

Baldacci, R., P. Toth, and D. Vigo (2010). Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research 175*, 213–245.

Bartolini, E., J.-F. Cordeau, and G. Laporte (2013). Improved Lower Bounds and Exact Algorithm for the Capacitated Arc Routing Problem. *Mathematical Programming 137*(1-2), 409–452.

Belenguer, J. M. and E. Benavent (1998). The Capacitated Arc Routing Problem: Valid Inequalities and Facets. *Computational Optimization and Applications 10*, 165–187.

Belenguer, J. M. and E. Benavent (2003). A Cutting Plane Algorithm for the Capacitated Arc Routing Problem. *Computers & Operations Research 30*(5), 705–728.

Ben Amor, H., J. Desrosiers, and J. M. Valério de Carvalho (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research 54*(3), 454–463.

Benavent, E., V. Campos, A. Corberán, and E. Mota (1992). The capacitated arc routing problem: Lower bounds. *Networks 22*, 669–690.

Black, D., R. Eglese, and S. Wøhlk (2013). The time-dependent prize-collecting arc routing problem. *Computers & Operations Research 40*(2), 526–535.

Bode, C. and S. Irnich (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research 60*(5), 1167–1182.

Bode, C. and S. Irnich (2013a). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. forthcoming in: Transportation Science.

Bode, C. and S. Irnich (2013b). The shortest-path problem with resource constraints with $(k, 2)$-loop elimination and its application to the capacitated arc-routing problem. Technical Report LM-2013-01, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany.

Boland, N., J. Dethridge, and I. Dumitrescu (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters 34*(1), 58 – 68.

Bosco, A., D. Laganá, R. Musmanno, and F. Vocaturo (2013). Modeling and Solving the Mixed Capacitated General Routing Problem. *Optimization Letters 7*, 1451–1469.

Bräysy, O. and M. Gendreau (2005a). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science 39*(1), 104–118.

Bräysy, O. and M. Gendreau (2005b). Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science 39*(1), 119–139.

Breslin, P. and A. Keane (1997). The Capacitated Arc Routing Problem: Lower Bounds. Master's Thesis. *University College Dublin, Department of Management Information Systems*.

Cacchiani, V., A. Caprara, and P. Toth (2008). A column generation approach to train timetabling on a corridor. *4OR: A Quarterly Journal of Operations Research 6*(2), 125–142.

Cacchiani, V. and P. Toth (2012). Nominal and robust train timetabling problems. *European Journal of Operational Research 219*(3), 727 – 737.

Caimi, G., M. Fuchsberger, M. Laumanns, and K. Schüpbach (2011). Periodic railway timetabling with event flexibility. *Networks 57*(1), 3–18.

Caprara, A., M. Fischetti, and P. Toth (1999). A heuristic method for the set covering problem. *Operations Research 47*(5), 730–743.

Caprara, A., L. Kroon, M. Monaci, M. Peeters, and P. Toth (2007). Passenger railway optimization. In C. Barnhart and G. Laporte (Eds.), *Transportation*, Volume 14 of *Handbooks in Operations Research and Management Science*, pp. 129–187. Amsterdam: Elsevier.

Corberán, A., A. N. Letchford, and J. M. Sanchis (2001). A cutting plane algorithm for the general routing problem. *Mathematical Programming 90*, 291–316.

Corberán, A., E. Mota, and J. M. Sanchis (2006). A Comparison of Two Different Formulations for Arc Routing Problems on Mixed Grapgs. *Computers & Operations Research 33*, 3384–3402.

Corberán, A., I. Plana, and J. M. Sanchis (2007). A branch & cut algorithm for the windy general routing problem and special cases. *Networks 49*, 245–257.

Corberán, A., A. Romero, and J. M. Sanchis (2003). The Mixed General Routing Polyhedron. *Mathematical Programming 96*, 103–137.

Cordeau, J.-F., P. Toth, and D. Vigo (1998). A survey of optimization models for train routing and scheduling. *Transportation Science 32*(4), 380–404.

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

Dantzig, G. B. and P. Wolfe (1960). Decomposition Principle for Linear Programs. *Operations Research 8*, 101–111.

Dantzig, G. B. and P. Wolfe (1961). The Decomposition Algorithm for Linear Programs. *Econometrica 29*(4), 767–778.

Dell'Amico, M., J. D. Díaz, G. Hasle, and M. Iori (2012). An Adaptive Iterated Local Search for the Node, Edge, and Arc Routing Problem. *Submitted*.

Desrochers, M. and F. Soumis (1988). A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR 26*(3), 191 – 212.

Desrosiers, J., Y. Dumas, M. M. Solomon, and F. Soumis (1995). Chapter 2 time constrained routing and scheduling. In C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser (Eds.), *Network Routing*, Volume 8 of *Handbooks in Operations Research and Management Science*, pp. 35–139. Elsevier.

Desrosiers, J. and M. E. Lübbecke (2005). A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon (Eds.), *Column Generation*, pp. 1–32. Springer US.

Distribution Innovation (2012). Distribution Innovation home page. `http://www.di.no/?lang=en`. Accessed: 27/07/2012.

Doerner, K. and V. Schmid (2010). Survey: Matheuristics for rich vehicle routing problems. In M. Blesa, C. Blum, G. Raidl, A. Roli, and M. Sampels (Eds.), *Hybrid Metaheuristics*, Volume 6373 of *Lecture Notes in Computer Science*, pp. 206–221. Springer Berlin / Heidelberg.

Dror, M. (2000). *Arc routing: theory, solutions, and applications*. Kluwer Academic.

du Merle, O., D. Villeneuve, J. Desrosiers, and P. Hansen (1999). Stabilized column generation. *Discrete Mathematics 194*(1-3), 229 – 237.

Eglese, R. W. and L. Y. Li (1996). An interactive algorithm for vehicle routing for winter-gritting. *Journal of the Operational Research Society 47*, 217–228.

Elhallaoui, I., D. Villeneuve, F. Soumis, and G. Desaulniers (2005). Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research 53*(4), 632–645.

Feillet, D. (2010). A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR 8*(4), 407–424.

Feillet, D., P. Dejax, M. Gendreau, and C. Gueguen (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks 44*(3), 216–229.

Freling, R., D. Huisman, and A. P. M. Wagelmans (2003). Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling 6*(1), 63–85.

Fukasawa, R., H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck (2006). Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Mathematical Programming 106*(3), 491–511.

Gaze, K. (2013). Column generation for the mixed capacitated general routing problem, abstract, WARP 1.

Gendreau, M., J. Y. Potvin, O. Bräysy, G. Hasle, and A. Løkketangen (2008). Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography. In B. Golden, S. Raghavan, and E. Wasil (Eds.), *The Vehicle Routing Problem - Latest Advances and New Challenges*. Springer Verlag, Heidelberg.

Gintner, V., N. Kliewer, and L. Suhl (2008). A crew scheduling approach for public transit enhanced with aspects from vehicle scheduling. In M. Hickman, P. Mirchandani, and S. Voss (Eds.), *Computer-aided Systems in Public Transport*, Volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pp. 25–42. Springer Berlin Heidelberg.

Golden, B., S. Raghavan, and E. Wasil (2010). *The vehicle routing problem: Latest advances and new challenges*. Operations Research Computer Science Interfaces Series. Springer.

Golden, B. L. and R. T. Wong (1981). Capacitated Arc Routing Problems. *Networks 11*, 305–315.

Groot, S. and D. Huisman (2008). Vehicle and crew scheduling: Solving large real-world instances with an integrated approach. In M. Hickman, P. Mirchandani, and S. Voss (Eds.), *Computer-aided Systems in Public Transport*, Volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pp. 43–56. Springer Berlin Heidelberg.

Gutiérrez, J. C. A., D. Soler, and A. Hervás (2002). The capacitated general routing problem on mixed graphs. *Revista Investigacion Operacional 23*(1), 15–26.

Hasle, G. and O. Kloster (2007). Industrial vehicle routing. In G. Hasle, K.-A. Lie, and E. Quak (Eds.), *Geometric Modelling, Numerical Simulation, and Optimization - Applied Mathematics at SINTEF*, pp. 397–435. Springer.

Hasle, G., O. Kloster, M. Smedsrud, and K. Gaze (2012). Experiments on the node, edge, and arc routing problem. Technical report, SINTEF.

Houck, D., J.-C. Picard, M. Queyranne, and R. Vemuganti (1980). The travelling salesman problem as a constrained shortest path problem: Theory and computational experience. *OPSEARCH 17*, 93–109.

Huisman, D. (2007). A column generation approach for the rail crew re-scheduling problem. *European Journal of Operational Research 180*(1), 163–173.

Huisman, D., R. Freling, and A. P. M. Wagelmans (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science 39*(4), 491–502.

Huisman, D., L. G. Kroon, R. M. Lentink, and M. J. C. M. Vromans (2005). Operations Research in Passenger Railway Transportation. *Statistica Neerlandica 59*, 467–497.

Irnich, S. and G. Desaulniers (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon (Eds.), *Column Generation*, pp. 33–65. Springer US.

Jütte, S., M. Albers, U. W. Thonemann, and K. Haase (2011). Optimizing railway crew scheduling at DB Schenker. *Interfaces 41*(2), 109–122.

Jütte, S. and U. W. Thonemann (2012). Divide-and-price: A decomposition algorithm for solving large railway crew scheduling problems. *European Journal of Operational Research 219*(2), 214–223.

Kallehauge, B., J. Larsen, O. B. G. Madsen, and M. M. Solomon (2005). Vehicle routing problem with time windows. In G. Desaulniers, J. Desrosiers, and M. M. Solomon (Eds.), *Column Generation*, pp. 67–98. Springer US.

Kliewer, N., B. Amberg, and B. Amberg (2012). Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport 3*(3), 213–244.

Kokubugata, H., A. Moriyama, and H. Kawashima (2007). A practical solution using simulated annealing for general routing problems with nodes, edges, and arcs. In *Proceedings of the 2007 international conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics*, SLS'07, Berlin, Heidelberg, pp. 136–149. Springer-Verlag.

Kroon, L. G., D. Huisman, E. J. W. Abbink, P.-J. Fioole, M. Fischetti, G. Maróti, L. Schrijver, A. Steenbeek, and R. Ybema (2009). The new Dutch Timetable: The OR Revolution. *Interfaces 39*, 6–17.

Kuo, A., E. Miller-Hooks, and H. S. Mahmassani (2010). Freight train scheduling with elastic demand. *Transportation Research Part E: Logistics and Transportation Review 46*(6), 1057–1070.

Kwan, R. S. and A. Kwan (2007). Effective search space control for large and/or complex driver scheduling problems. *Annals of Operations Research 155*(1), 417–435.

Laporte, G. (2009). Fifty Years of Vehicle Routing. *Transportation Science 43*(4), 408–416.

Letchford, A. N., R. Eglese, and J. Lysgaard (2002). Multistart, Partial Multistar and the Capacitated Vehicle Routing Problem. *Mathematical Programming 94*, 21–40.

Letchford, A. N. and A. Oukil (2009). Exploiting Sparsity in Pricing Routines for the Capacitated Arc Routing Problem. *Computers & Operations Research 36*(7), 2320–2327.

Letchford, A. N., G. Reinelt, and D. Theis (2008). Odd Minimum Cut-Sets and b-matchings Revisited. *SIAM Journal on Discrete Mathematics 22*(4), 1480–1487.

Lindner, T. and U. Zimmermann (2005). Cost optimal periodic train scheduling. *Mathematical Methods of Operations Research 62*(2), 281–295.

Longo, H., M. P. de Aragão, and E. Uchoa (2006). Solving Capacitated Arc Routing Problem using a Transformation to the CVRP. *Computers & Operations Research 33*(6), 1823–1837.

Lusby, R., J. Larsen, M. Ehrgott, and D. Ryan (2011). Railway track allocation: models and methods. *OR Spectrum 33*(4), 843–883.

Lysgaard, J., A. N. Letchford, and R. W. Eglese (2004). A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem. *Mathematical Programming 100*(2), 423–445.

Martinelli, R., M. Poggi, and A. Subramanian (2011). Improved bounds for large scale capacitated arc routing problem. Technical report, Monografias em Ciencia da Computacao 14/11, Pontificia Universidade Catolica, Rio de Janeiro, Brazil.

Mesquita, M., A. Paias, and A. Respício (2009). Branching approaches for integrated vehicle and crew scheduling. *Public Transport 1*(1), 21–37.

Nahapetyan, A. and S. Lawphongpanich (2007). Discrete-time dynamic traffic assignment models with periodic planning horizon: system optimum. *Journal of Global Optimization 38*(1), 41–60.

Orloff, C. (1974). A fundamental problem in vehicle routing. *Networks 4*, 35–64.

Pandit, R. and B. Muralidharan (1995). A capacitated general routing problem on mixed networks. *Computers & Operations Research 22*(5), 465–478.

Potthoff, D., D. Huisman, and G. Desaulniers (2010). Column generation with dynamic duty selection for railway crew rescheduling. *Transportation Science 44*(4), 493–505.

Prins, C. and S. Bouchenoua (2004). A memetic algorithm solving the VRP, the CARP and general routing problems with nodes, edges and arcs. In W. Hart, N. Krasnogor, and J. Smith (Eds.), *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, pp. 65–85. Springer.

Rezanova, N. J. and D. M. Ryan (2010). The train driver recovery problem - a set partitioning based model and solution method. *Computers & Operations Research 37*(5), 845–856.

Righini, G. and M. Salani (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks 51*(3), 155–170.

Ryan, D. M. and B. A. Foster (1981). An integer programming approach to scheduling. In A. Wren (Ed.), *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pp. 269–280. North-Holland.

SINTEF (2012). Spider web pages. `http://www.sintef.no/Projectweb/Transportation-planning/Software/Spider/`. Accessed: 27/07/2012.

SINTEF (2013). NEARP web pages. `http://www.sintef.no/NEARP`. Accessed: 27/12/2013.

Spliet, R. and G. Desaulniers (2012). The discrete time window assignment vehicle routing problem. *Les Cahiers du GERAD-2012-81.*.

Steinzen, I., V. Gintner, L. Suhl, and N. Kliewer (2010). A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. *Transportation Science 44*(3), 367–382.

Tagmouti, M., M. Gendreau, and J.-Y. Potvin (2007, 8/16). Arc routing problems with time-dependent service costs. *European Journal of Operational Research 181*(1), 30–39.

Toth, P. and D. Vigo (Eds.) (2001). *The vehicle routing problem*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Veelenturf, L. P., D. Potthoff, D. Huisman, and L. G. Kroon (2012). Railway crew rescheduling with retiming. *Transportation Research Part C 20*, 95–110.

Wøhlk, S. (2005). *Contributions to Arc Routing.* Ph. D. thesis, University of Southern Denmark.

Wøhlk, S. (2006). New lower bound for the capacitated arc routing problem. *Computers & Operations Research 33*(12), 3458–3472.

Wøhlk, S. (2008). A decade of capacitated arc routing. In B. L. Golden, S. Raghavan, and E. A. Wasil (Eds.), *The vehicle routing problem: Latest advances and new challenges*. Springer.

Ziarati, K., F. Soumis, J. Desrosiers, S. Gélinas, and A. Saintonge (1997). Locomotive assignment with heterogeneous consists at CN North America. *European Journal of Operational Research 97*(2), 281–292.

Dansk Resumé

Struktur

Afhandlingen består af fire selvstændige kapitler. Hvert kapitel er præsenteret som en akademisk tidsskriftsartikel. Kapitlerne hænger sammen to og to via deres tema. De første to omhandler rutelægning i en blandet graf bestående af både orienterede- og ikke orienterede kanter samt noder. Disse kan alle have en efterspørgsel. Dette problem kaldes “the Mixed Capacitated General Routing Problem (MCGRP)". De sidste to artikler har også rutelægningselementer, om end i mindre grad. De består begge af en praksis anvendelse på et problem inspireret af en godstogsoperatør. Tre af de fire artikler deler metoden søjle generering, som er løst både eksakt og heuristisk.

The Mixed Capacitated General Routing Problem

Ofte bliver rutelægningsproblemer løst som et rent kant- eller noderutelægningsproblem. Et kantrutelægningsproblem opstår f.eks. når der skal saltes veje om vinteren. Her er alle veje, der skal saltes, repræsenteret som en kant. Det samme er ofte tilfældet, når der skal samles affald ind. Her repræsenteres en hel vej som en kant, fordi alle på vejen får tømt affald samtidig. I et noderutelægnings problem er efterspørgslen

repræsenteret som punkter i en graf. Det kan f.eks. være pakkeomdeling. I et blandet rutelægnings problem (MCGRP) er der en kombination af de to ovenstående eksempler. Det kan f.eks. være erhvervskunder, der skal have tømt en container, mens private kunder får tømt affald en hel vej på saamme tid. Dette problem er mere generelt og dækker til fulde begge de andre problemer.

Kapitel 1

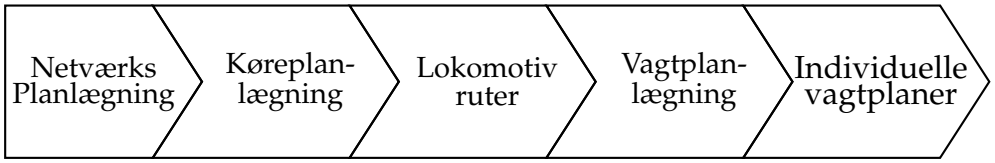
I kapitel et præsenterer vi, for MCGRP, som de første, en metode til at beregne en nedre grænse for omkostningerne for at servicere alle kunder i en blandet graf. Vi udvikler endvidere nye testeksempler til at evaluere kvaliteten af vores metode samt andre metoder, der løser selve rutelægningen.

Kapitel 2

I kapitel to arbejder vi med samme problem som i kapitel 1. Her foreslår vi en ny model. Modellen anvender metoden “Branch-and-Cut-and-Price" til at finde en optimal løsning til MCGRP. De hidtidige metoder foreslået i litteraturen er primært heuristiske metoder. Der findes kun én anden artikel, der løser problemet eksakt (Bosco et al., 2013). Vi foreslår en mere generel anvendelig løsnings metode end deres. For at evaluere kvaliteten af vores metode, sammenligner vi vores resultater med de bedste fra den akademiske litteratur. Vi er i stand til at levere løsninger, der generelt er bedre end (Bosco et al., 2013) og samtidig forbedre de bedst kendte løsninger for andre testeksempler fra litteraturen.

Køreplaner for godstogsoperatører

Når der skal planlægges hos en godstogsoperatør, sker dette ofte i sekventielle niveauer. I den eksisterende litteratur sker det ofte i følgende niveauer:



Vi ønsker i de følgende to kapitler at undersøge, i hvilken grad vi kan integrere nogle af disse sekventielle trin. I det første kapitel tager vi hensyn til lokomotivplanen, når der lægges køreplan. I det andet kapitel tager vi hensyn til vagtplanlægningen for lokomotivførere, når der lægges køreplan.

Kapitel 3

I kapitel 3 integrerer vi køreplanlægningen med den optimale udnyttelse af lokomotiver hos en godstogsoperatør. Vi udvikler en matematisk model til at løse problemet. Denne løses ved hjælp af metoden søjlegenerering. I kapitlet er vi i stand til at løse problemet for testeksempler baseret på data fra en godstogsoperatør. Den foreslåede metode er således i stand til at løse problemet og levere en optimal køreplan.

Kapitel 4

I kapitel 4 integrerer vi vagtplanlægningen for lokomotivførere med køreplanslægningen. Her vil vi undersøge, om dette er muligt ved at justere i en allerede fastlagt køreplan. Vi ønsker derved at opnå en mere effektiv udnyttelse af lokomotivførerne. Vagtplanlægning for lokomotivførere er komplekst. Dette skyldes, at det er underlagt et stor regelsæt, som skal overholdes. Vi foreslår en metode til at integrere de to problemstillinger og løser det med en heuristisk “Branch-and-Price" metode. Vi opnår gode resultater og viser, at der er en besparelse at opnå ved en bedre integration af de to problemer.

DEPARTMENT OF ECONOMICS AND BUSINESS

AARHUS UNIVERSITY
SCHOOL OF BUSINESS AND SOCIAL SCIENCES
www.econ.au.dk

PhD Theses since 1 July 2011

2011-4 Anders Bredahl Kock: Forecasting and Oracle Efficient Econometrics

2011-5 Christian Bach: The Game of Risk

2011-6 Stefan Holst Bache: Quantile Regression: Three Econometric Studies

2011:12 Bisheng Du: Essays on Advance Demand Information, Prioritization and Real Options in Inventory Management

2011:13 Christian Gormsen Schmidt: Exploring the Barriers to Globalization

2011:16 Dewi Fitriasari: Analyses of Social and Environmental Reporting as a Practice of Accountability to Stakeholders

2011:22 Sanne Hiller: Essays on International Trade and Migration: Firm Behavior, Networks and Barriers to Trade

2012-1 Johannes Tang Kristensen: From Determinants of Low Birthweight to Factor-Based Macroeconomic Forecasting

2012-2 Karina Hjortshøj Kjeldsen: Routing and Scheduling in Liner Shipping

2012-3 Soheil Abginehchi: Essays on Inventory Control in Presence of Multiple Sourcing

2012-4 Zhenjiang Qin: Essays on Heterogeneous Beliefs, Public Information, and Asset Pricing

2012-5 Lasse Frisgaard Gunnersen: Income Redistribution Policies

2012-6 Miriam Wüst: Essays on early investments in child health

2012-7 Yukai Yang: Modelling Nonlinear Vector Economic Time Series

2012-8 Lene Kjærsgaard: Empirical Essays of Active Labor Market Policy on Employment

2012-9 Henrik Nørholm: Structured Retail Products and Return Predictability

2012-10 Signe Frederiksen: Empirical Essays on Placements in Outside Home Care

2012-11 Mateusz P. Dziubinski: Essays on Financial Econometrics and Derivatives Pricing

2012-12 Jens Riis Andersen: Option Games under Incomplete Information

2012-13 Margit Malmmose: The Role of Management Accounting in New Public Management Reforms: Implications in a Socio-Political Health Care Context

2012-14 Laurent Callot: Large Panels and High-dimensional VAR

2012-15 Christian Rix-Nielsen: Strategic Investment

2013-1 Kenneth Lykke Sørensen: Essays on Wage Determination

2013-2 Tue Rauff Lind Christensen: Network Design Problems with Piecewise Linear Cost Functions

2013-3 Dominyka Sakalauskaite: A Challenge for Experts: Auditors, Forensic Specialists and the Detection of Fraud

2013-4 Rune Bysted: Essays on Innovative Work Behavior

2013-5 Mikkel Nørlem Hermansen: Longer Human Lifespan and the Retirement Decision

2013-6 Jannie H.G. Kristoffersen: Empirical Essays on Economics of Education

2013-7 Mark Strøm Kristoffersen: Essays on Economic Policies over the Business Cycle

2013-8 Philipp Meinen: Essays on Firms in International Trade

2013-9 Cédric Gorinas: Essays on Marginalization and Integration of Immigrants and Young Criminals – A Labour Economics Perspective

2013-10 Ina Charlotte Jäkel: Product Quality, Trade Policy, and Voter Preferences: Essays on International Trade

2013-11 Anna Gerstrøm: World Disruption - How Bankers Reconstruct the Financial Crisis: Essays on Interpretation

2013-12 Paola Andrea Barrientos Quiroga: Essays on Development Economics

2013-13 Peter Bodnar: Essays on Warehouse Operations

2013-14 Rune Vammen Lesner: Essays on Determinants of Inequality

2013-15 Peter Arendorf Bache: Firms and International Trade

2013-16 Anders Laugesen: On Complementarities, Heterogeneous Firms, and International Trade

2013-17 Anders Bruun Jonassen: Regression Discontinuity Analyses of the Disincentive Effects of Increasing Social Assistance

2014-1 David Sloth Pedersen: A Journey into the Dark Arts of Quantitative Finance

2014-2 Martin Schultz-Nielsen: Optimal Corporate Investments and Capital Structure

2014-3 Lukas Bach: Routing and Scheduling Problems - Optimization using Exact and Heuristic Methods

ISBN: 9788790117979