KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Decomposition Approaches for Optimization Problems

Joris Kinable

Supervisors:
Prof. dr. P. De Causmaecker
Prof. dr. F. Spieksma
Prof. dr. ir. G. Vanden Berghe

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

December 2014

Decomposition Approaches for Optimization Problems

Joris KINABLE

Examination committee:
Prof. dr. ir. H. Van Brussel, chair
Prof. dr. P. De Causmaecker, supervisor
Prof. dr. F. Spieksma, supervisor
Prof. dr. ir. G. Vanden Berghe, supervisor
Prof. dr. ir. M. Bruynooghe
Prof. dr. R. Leus
Prof. dr. K. Sörensen
(Faculty of Applied Economics, University of Antwerp)
Prof. dr. M. Trick
(Tepper School of Business, Carnegie Mellon University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

December 2014

© 2014 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Joris Kinable, Celestijnenlaan 200A box 2402, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

ISBN 978-94-6018-923-4

D/2014/7515/143

Preface

> Two roads diverged in a wood, and I—
> I took the one less traveled by,
> And that has made all the difference.
>
> from *The Road Not Taken* by *Robert Frost* (1874–1963)

The last 6 years, including 4 years as a PhD researcher, I spent most of my time abroad, living in different places, experiencing new cultures and connecting with new people. Two years I have stayed in Finland and Norway, three years in Belgium, another year in the USA, and in between I have visited China, Ireland, Poland, Colombia, France, and Great Britain. Each time you leave a place, you inevitably leave something behind, but, in return you carry new memories and experiences back home. I could not have undertaken this incredible journey without the support of colleagues, friends, family, and relatives. This section is dedicated to the people who supported me throughout the course of my PhD.

Following traditional academic etiquette, I would first and foremost like to thank my supervisors and jury members for their useful comments and suggestions with respect to my dissertation. I am particularly grateful towards my supervisors, Greet Vanden Berghe, Frits Spieksma, and Patrick De Causmaecker for their support during the last four years. Your doors were always open; your differences in vision and interests often provided for interesting and diverse insights. A warm word goes out to Greet, whom, over the years, I have got to appreciate as a person who is strongly concerned with the well-being of the people in her group.

Next I would like to thank Prof Michael Trick for providing me the opportunity to spend a year at Carnegie Mellon University. Despite the many obligations you had as Senior Associate Dean, you always managed to find time to provide me with excellent feedback. I sincerely thank you for your hospitality. Similarly I thank Prof Willem-Jan van Hoeve. Next to the many academic meetings we had over the past year, I thoroughly enjoyed dining, cooking and even gardening with you.

Finally I thank Erik Van Achter for his feedback regarding the textual quality of my publications.

Moving on to the colleagues section, I have to thank three groups: CODeS in Ghent, my colleagues at the Faculty of Economics in Leuven, and my colleagues at the Tepper School of Business, Carnegie Mellon University. Many of you I got to know as colleagues, co-authors and eventually as friends. On the CODeS side, I would explicitly like to thank Tony, Wim, Jannes, Pieter, Joris, Jan and Faysal. On the Leuven side there are Kris, Fabrice, Ann, Yannick, Mieke, Bart, Marjolein, Dries, Dennis, Jeroen and Roel. Some of you contributed directly to this dissertation, others indirectly through the many, often entertaining discussions we have had over the past couple of years, both at work in Belgium, at a conference in China, or even at a beer tasting in Leuven.

A dedicated section should be reserved for my friends at Carnegie Mellon University. A year full of new inter-cultural experiences, very long nights at work, and exhausting weekends of outdoor activities. Yang, Diana, Andre, Max, Spyros, Thiago, Siddharth, Vince and Wenting. Thank you for the colorful summer you've provided me. We've enjoyed many activities together, from having lunch in the park overlooking downtown Pittsburgh to visiting classical concerts, ice skating, climbing, crane bird folding, swimming and barbecuing in Moraine and Raccoon state parks, rafting in Ohiopyle, trips to Yellow Stone, San Francisco, and New York. I sincerely enjoyed these moments together.

Over the years, friends come and go, but some will stay with you for a very long time. You pretty much cannot get rid of them. Kai, Ralph, Guy, and Jan-Willem. Thank you for your support, any place, any time.

Lastly, I would like to thank my family, my parents Karin and Dirk, my grandparents and of course my sister Els. I explicitly acknowledge the support I received from my father, who always motivated me to pursue this PhD.

My final words go to Karin, who has patiently been here, all the way throughout my PhD.

Joris Kinable Ghent, November 2014

Abstract

This dissertation encompasses the development of decomposition approaches for a variety of both real-world and fundamental optimization problems. Many optimization problems comprise of multiple interconnected subproblems, often rendering them too large or too complicated to solve as a single integral problem. Decomposition approaches are required to deal with these problems efficiently. By decomposing a problem into multiple subproblems, efficient dedicated procedures can be employed to solve the subproblems independently. Furthermore, often strong bounds on the optimal solutions can be derived by exploiting structures in the underlying subproblems.

This work primarily focuses on analyzing and identifying problem components to decompose a problem into multiple, easier-to-solve, subproblems. The actual decompositions are obtained through mathematical techniques such as Column Generation and Benders decomposition, thereby relying on Integer Programming, Constraint Programming, heuristic and combinatorial procedures to solve the resulting subproblems. Each solution method is developed with scalability and extendability in mind, while simultaneously making the methods sufficiently robust to account for changes to the original problem definitions. Moreover the decomposition strategies are designed to preserve a notion of optimality, thereby providing insight into the quality of a solution.

From an application point of view, the present work is centered around four routing and scheduling problems: the School Bus Routing Problem (SBRP), the Concrete Delivery problem (CDP), the Time-Dependent TSP (TD-TSP) and the Balanced TSP (BTSP). For each of these problems, decomposition strategies have been developed. The SBRP and BTSP are solved via a branch-and-price framework; lower bounds on the SBRP are derived through Lagrangian Relaxation. A Benders decomposition is developed for the CDP. The subproblems resulting from the Benders decomposition are efficiently solved through Integer and Constraint programming, in combination with a fast scheduling heuristic. Finally, a generic, robust Constraint Programming approach, strengthened with Multivariate Decision Diagrams, is implemented for

the TD-TSP. To improve domain propagation, bounds derived from alternative problem relaxations are incorporated in the CP search through an additive bounding procedure. To validate the aforementioned solution approaches, experiments are conducted on real-world or simulated data.
By decomposing a problem, techniques from various interdisciplinary domains can be combined into an integrated solution approach. Correlations between the problems under consideration as well as the proposed solution methodologies provide insight as to the applicability, limitations and the intuition behind the various techniques. It are exactly these insights that ultimately will lead to fully automated problem solvers, capable of analyzing and decomposing optimization problems without human interference.

List of Symbols

ARL	Allocation Routing Location
BBC	Branch-Bound-Cut
BEFS	Best First Search
BFS	Breadth First Search
BPC	Branch-Price-Cut
BTSP	Balanced Traveling Salesman Problem
CDP	Concrete Delivery Problem
CG	Column Generation
CP	Constraint Programming
CPM	Column Pool Manager
CRSP	Capacitated Ring-Star Problem
CVRP	Capacitated Vehicle Routing Problem
D-W	Dantzig-Wolfe
DFJ	Dantzig Fulkerson Johnson
DFS	Depth First Search
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
GA	Genetic Algorithm
HC	Hamiltonian Circuit
IPS	Interior Point Stabilization

LAR	Location Allocation Routing
LB	Lower Bound
LD	Lagrangian Dual
LP	Linear Program
LPM	Linear Program Master
LR	Lagrangian Relaxation
MDD	Multivariate Decision Diagram
MILP	Mixed Integer Linear Program
MIP	Mixed Integer Program
MP	Master Problem
MV-TPP	Multiple Vehicle Traveling Purchaser problem
NN	Nearest Neighbor
OR	Operations Research
PMSP	Parallel Machine Scheduling Problem
PTS	Proximal Type Stabilization
RMC	Ready Mixed Concrete
RMP	Restricted Master Problem
SBRP	Schoolbus Routing Problem
SD	Steepest Descent
SP	subproblem
TD-TSP	Time Dependent Traveling Salesman Problem
TSP	Traveling Salesman Problem
UB	Upper Bound
VNS	Variable Neighborhood Search

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last two decades, tremendous progress has been made in the field of Operations Research (OR). This progress can partially be ascribed to significant gains in computational power. However, a far stronger driving force is generated by large companies which have gradually started to embrace OR, realizing the enormous savings in resources that can be achieved through optimization. As a natural consequence, many of today's optimization challenges find their origins in both industrial and commercial applications.

Unlike many of the traditional, fundamental optimization problems which often have nice mathematical properties or clean structures, these new optimization challenges frequently consist of several interconnected subproblems and potentially exhibit a large number of side constraints. Despite the fact that there exist highly efficient algorithms for many of the fundamental optimization problems such as the traveling salesman, knapsack, matching, and bin packing problems, it is often impossible to adapt these algorithms to problems with side constraints without drastically decreasing their performance. Moreover, developing dedicated algorithms for every optimization problem is expensive, time-consuming, requiring expert knowledge, and the resulting algorithms are often difficult to maintain or extend. Indeed, there is a demand for more robust, modular approaches enabling rapid development of optimization strategies. In this perspective, decomposition approaches, the focal point of this thesis, may provide a viable outcome.

In any decomposition, the original problem is divided into multiple, easier-to-solve subproblems. A solution to the original problem can be acquired by iteratively or sequentially solving the resulting subproblems. Specifically in the context of large, complex optimization problems, decompositions offer a number of advantages. When the original problem is divided into smaller

subproblems, each subproblem may be solved by efficient, dedicated procedures which can employ different optimization techniques independently. Similarly, various optimization problems often share some common component solvable by the same algorithm; isolating this component through decomposition could facilitate efficient reuse of code.
By decomposing a problem, different structural properties of the optimization problem can be captured. The latter observation is used to derive strong bounds on the optimal solutions. Examples can be found in literature where problems are decomposed through Column Generation; the resulting decomposition often yields stronger bounds than any of the bounds obtainable through alternative Mixed Integer Programming (MIP) models formulated over the entire problem. In fact, many of these MIP formulations suffer from large numbers of conditional constraints, resulting in weak Linear Programming relaxations and consequently weak bounds. Finally, symmetrical structures in MIP formulations frequently introduce repetition in branch-and-bound based search procedures employed by most Integer and Constraint programming solvers, thereby significantly reducing their effectiveness. Decomposition methods may again be used to mitigate these issues.
In this dissertation, decomposition techniques are studied for various optimization problems, mainly in the area of routing and scheduling. Each decomposition is developed with the following key design aspects in mind:

- How can a problem be decomposed without breaking its *structure*?
- How to *communicate* the results from one subproblem to another?
- Is the decomposition approach *robust*, i.e., can the approach accommodate modifications to the original problem specifications?
- Does the decomposition preserve a *notion of optimality*, i.e., is it possible to derive bounds on the optimal solution?

The dissertation is structured into five chapters, each centered around a different optimization problem or a different decomposition technique. The chapters are largely self-contained and can be read independent of each other.
Chapter 2 considers the School Bus Routing Problem (SBRP), a Vehicle Routing Problem involving the selection of bus stop locations, student assignment to buses, and routing of the school buses to the pickup locations of the students. Early attempts to solve SBRP are based on sequential (iterative) procedures where the selection, assignment and routing subproblems are solved independently. Although it is appealing to decompose a problem directly towards its natural components, this often leads to suboptimal results, or is simply not possible because the subproblems cannot be solved independently.

In this thesis, a Column Generation procedure is presented to solve the SBRP. The problem is decomposed into a master problem and a subproblem. In the subproblem, bus schedules are generated. Each schedule defines a bus route, starting and ending at the school, plus a description of the students that should be picked up along the route and their respective pickup locations. In turn, the master problem selects a set of compatible bus schedules such that each student is transported to school. The resulting solution procedure is capable of obtaining provable optimal solutions for large SBRP instances through the use of a branch-and-price framework. Particular emphasis is put on a number of techniques to improve the efficiency of the column generation procedures, through, for instance, the use of lower bounding procedures and stabilization methods.

Chapters 3 and 4 revolve around a Concrete Delivery Problem (CDP): a difficult problem involving both vehicle routing and scheduling. In CDP concrete delivery trucks are routed back and forth between concrete production stations and construction sites. Each construction site has limited processing capacity. Hence, when multiple truck loads are required, the trucks need to be sequenced, such that deliveries for the same customer do not overlap in time. However, due to the fact that concrete is a perishable product, the temporal spacing between two consecutive deliveries may not exceed a predefined amount of time to prevent the first batch of concrete from hardening before the next batch arrives. The deliveries are performed by a heterogeneous fleet of vehicles. Although the demand of the customers is known a priori, it is unknown how many deliveries are required to fulfill the customer's demand as this depends on the actual capacities of the vehicles performing the deliveries. Many variations of the CDP exist in literature, each dealing with different constraints, or treating the problem from a different perspective. An extensive comparison of these variations is provided in Chapter 3. In addition, to support future comparison of different solution methodologies, Chapter 3 presents a generalized version of the CDP, preserving the main characteristics of the existing problem variations. Exact methods based on Constraint Programming and Mixed Integer Programming as well as a number of heuristic approaches are studied for the problem at hand, and are compared on a large data set of problem instances.

The exact and heuristic methods presented in Chapter 3 are able to swiftly obtain good primal solutions, but establishing the quality of these solutions proves difficult due to the lack of strong bounds. One perhaps logical solution would be to attempt the same approach presented for the SBRP in Chapter 4 to the CDP problem, thereby generating individual delivery schemes for the vehicles, and using a centralized mechanism to select a set of compatible schedules. It is however not evident how to select or generate a set of compatible schedules efficiently due to the numerous temporal and delivery constraints. Therefore, we propose a different integrated solution approach for CDP based

on a logic based Benders decomposition. Similar to the Column Generation procedure for SBRP in Chapter 2, this approach decomposes the problem into a master problem and a subproblem. The master problem selects the customers concrete is delivered to. For these customers, the subproblem attempts to generate a feasible delivery schedule. Cuts are generated and added to the master problem in absence of such a schedule. Many of the algorithms developed in Chapter 3 can be efficiently reused in the Benders framework to solve the resulting master and subproblem.

The last two chapters, Chapters 5 and 6 treat two, more fundamental optimization problems: the Time-Dependent Traveling Salesman Problem (TD-TSP) and the Balanced TSP (BTSP). The TD-TSP asks to sequence a number of jobs, while minimizing the total sum of setup times. For each pair of jobs, a sequence dependent setup time is specified which does not only depend on the jobs, but also on their relative positions in the sequence. In Chapter 5 we develop a Constraint Programming based approach for the TD-TSP. Despite the fact that CP is more of a programming paradigm than an actual decomposition approach, we can still interpret it as such because of the way constraints are treated in CP. Each constraint captures some combinatorial structure of the problem, and implements its own independent filtering mechanism which removes inconsistent domain values. The constraints are propagated one by one until some fixed point is reached; communication between the constraints is solely based on the domains of the variables. Traditionally, variable domains bare little structural information as they are represented by flat sets of numbers. To strengthen the CP formulation for the TD-TSP, Chapter 5 proposes to incorporate Multivariate Decision Diagrams (MDDs) into the CP model. We show how the MDDs can be used to derive bounds on the optimal solutions for TD-TSP, to prune the search space, and to guide the CP search. Moreover, we show how MDDs can be used to consolidate the CP model by integrating structural information from other problem relaxations such as Linear Programming relaxations through the use of additive bounding.

Chapter 6 finally focuses on the k-Balanced Traveling Salesman Problem (k-BTSP) which seeks to find a route through a number of cities such that, when the edges in the tour are partitioned into k edge-disjoint groups, the total length of the edges in each group is approximately equal. The k-BTSP belongs to a particularly challenging class of optimization problems where equity or fairness in resource distribution is important. We present and compare two Mixed Integer Programming formulations. One of the formulations is solved through a traditional branch-bound-cut approach, whereas the other relies on a branch-price-cut framework. In the latter case, a decomposition is used which exploits an underlying combinatorial structure of the BTSP.

Foundations

This Section provides theoretical background for a number of commonly used decomposition methods. We focus primarily on the foundations of these methods, the intuition behind them, their applicability and some of their limitations. The purpose of this section is to give the unfamiliar reader a better understanding of the variety of available decomposition approaches, their mechanics and their uses. It is not our intent to give an exhaustive overview, nor do we treat all special cases or pitfalls inherently related to the implementation of these methods.
The content of Section *Column Generation* is based on Hooker (2013), Lübbecke (2010), Barnhart et al. (1998). Hooker (2013), and Wolsey and Nemhauser (1988) have been used for the Section *Benders Decomposition*. Finally, Section *Lagrangian Relaxation and Decomposition* is based on the work by Reeves (1993).

Column Generation

Often, Linear and Mixed Integer Programming models have a large number of variables, but only a limited number of constraints. Instead of solving these large models for all variables at once, which may prove intractable, a reduced problem defined over a subset of the variables may be considered instead, as most of the variables in the original model will be zero in the optimal solution. Column Generation (CG) is an iterative procedure used to solve these reduced problems. During every iteration, a check is performed to test whether adding any of the absent variables to the reduced model could potentially improve its objective value. The procedure terminates when no more such variables exist, thereby obtaining a provable optimal solution to the original problem.
The first applications of CG date back to the 1960's where CG was used in the context of Dantzig-Wolfe decomposition (D-W decomposition). D-W decomposition is a technique used to split optimization problems into a master problem and a subproblem which are then solved through CG. Often the decomposed problem can be solved much more efficiently. Moreover, when the procedure is terminated prematurely, for example due to a time limit, the current best solution is still primal feasible to the original problem. In what remains we will elaborate on the concept of CG in the context of D-W decomposition. The same ideas are used in Chapter 2 and Appendix A, albeit on different problem formulations.
Given the following general Linear Program (LP). For simplicity we assume

that the solution space represented by this LP is bounded and non-empty:

$$\begin{aligned} min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & Dx \geq d \\ & x \geq 0 \end{aligned} \tag{1.1}$$

Let $X = \{x \in \mathbb{Q}_+ | Dx \geq d\}$. From Minkowski's theorem we know that each $x \in X$ can be written as a convex combination of extreme points $\{y^k | k \in K\}$ and extreme rays $\{w^l | l \in L\}$, i.e.,:

$$x = \sum_{k \in K} \lambda_k y^k + \sum_{l \in L} \mu_l w^l \tag{1.2}$$

where $\sum_{k \in K} \lambda_k = 1$ and $\lambda_k, \mu_l \geq 0$ for all $k \in K, l \in L$. Substituting 1.2 into 1.1, thereby eliminating constraints $Dx \geq d$, yields:

$$\begin{aligned} min \quad & \sum_{k \in K} \lambda_k c^T y^k + \sum_{l \in L} \mu_l c^T w^l \\ \text{s.t.} \quad & \sum_{k \in K} \lambda_k A y^k + \sum_{l \in L} \mu_l A w^l \geq b \\ & \sum_{k \in K} \lambda_k = 1 \\ & \lambda_k, \mu_l \geq 0 \qquad\qquad \forall k \in K, l \in L \end{aligned} \tag{1.3}$$

This formulation, also known as extended formulation or Dantzig-Wolfe formulation, typically has a vast number of variables (columns), but usually only a limited number of constraints (rows). Instead of solving this problem directly, which is simply intractable due to the potentially large number of variables, the problem is solved only for a small subset of the extreme points and rays. The resulting, reduced problem, is often referred to as the Restricted Master Problem (RMP), whereas the original problem is called the Master Problem (MP). Note that in the RMP, the variables λ_k, μ_l corresponding with the extreme rays and points that are absent, are equal to zero.
When the RMP is solved, a solution is obtained for which, by definition of the MP and RMP, the following holds: $v(RMP) \geq v(MP)$, where $v(\cdot)$ gives the value of the optimal solution.
Recall that, at each iteration, the well-known Simplex algorithm, used to solve LPs, searches for non-basic variables with negative reduced cost to bring into the

basis, thereby (potentially) improving the objective value. For an optimization problem stated as $min\{c^T x | Ax \leq b, x \in R^n\}$ the reduced cost of a variable x_j is given by $c_j - u^T A_j$, where u is the vector of dual variables, and A_j the j-th column in A. An optimal solution is obtained when there are no more variables with negative reduced cost. The same technique can be applied to verify whether the solution of RMP can be improved through the addition of extra λ_k, μ_l variables.
Associate dual variables u, α with the constraints in formulation (1.3). The reduced cost of a variable λ_k, $k \in K$ is given by $c^T y^k - u^T A y^k - \alpha$. To verify whether there exists a variable λ_k currently absent in the RMP, and having a negative reduced cost, the following auxiliary problem, commonly referred to as the pricing problem, can be solved:

$$\begin{aligned} min \quad & c^T y - u^T A y - \alpha \\ \text{s.t.} \quad & Dy \geq d \\ & y \geq 0 \end{aligned} \tag{1.4}$$

If (1.4) yields a solution $\bar{y}$ with an objective value strictly smaller than 0, we have found a variable with negative reduced cost. Then a new variable λ_k, corresponding with the extreme point $\bar{y}$ is added to the RMP, with column $(A\bar{y}, 1)$ and cost $c^T \bar{y}$.
In a similar fashion we can search for μ_l, $l \in L$ variables with negative reduced cost by solving the following auxiliary problem:

$$\begin{aligned} min \quad & c^T w - u^T A w - \alpha \\ \text{s.t.} \quad & Dw \geq d \\ & w \geq 0 \end{aligned} \tag{1.5}$$

When the outcome of this problem has an objective $-\infty$ for a solution $\bar{w}$ a new variable μ_l, corresponding with the extreme ray $\bar{w}$ is added to RMP, with column $(A\bar{w}, 0)$ and cost $c^T \bar{w}$.
After adding one or more variables with negative reduced cost to the RMP, the RMP is re-solved and the process is repeated. This procedure is commonly referred to as *Column Generation*. The process terminates when no more variables with negative reduced cost can be identified. Logically, it follows that at this point $v(RMP) = v(MP)$ holds. Although it is not known beforehand how many Column Generation iterations are required before optimality is established, typically this number is fairly limited and significantly smaller than $|K| + |L|$.
Dantzig-Wolfe decomposition can be particularly effective if the original

constraint matrix has a so-called block diagonal structure, in which several blocks of constraints on set disjoint variables are linked together by some complicating constraints, e.g.,:

$$\begin{array}{lccccccccc} min & c^1x_1 & + & c^2x_2 & + & \dots & + & c^px_p & & \\ \text{s.t.} & D^1x_1 & & & & & & & \geq & d^1 \\ & & & D^2x_2 & & & & & \geq & d^2 \\ & & & & & \ddots & & & & \\ & & & & & & & D^px_p & \geq & d^p \\ & A^1x_1 & + & A^2x_2 & + & \dots & + & A^px_p & \geq & b \\ & x & \geq & 0 & & & & & & \end{array} \tag{1.6}$$

where $Dx \geq d$, $D = \begin{bmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^p \end{bmatrix}$, $d = \begin{bmatrix} d^1 \\ d^2 \\ \vdots \\ d^p \end{bmatrix}$ are the easy constraints and $Ax \geq b$, $A = \begin{bmatrix} A^1 & A^2 & A^3 & A^4 \end{bmatrix}$ are the complicating constraints. The D-W decomposition of such a problem results in p pricing problems which can be solved independently of one another. These block structures are frequently encountered in set-cover formulations for Vehicle Routing and Multi-Machine scheduling problems. An example of this can be found in Chapter 2 and in Appendix A.
Remarks:

- The D-W procedure outlined above is described in the context of LP; nevertheless, the procedure can be generalized to MILP.
- Next to block-diagonal matrix structures, there are several other special structures which can be exploited in D-W decomposition, see (Bergner et al., 2014).

Benders decomposition

Benders decomposition is a mathematical approach that exploits the fact that fixing a number of difficult variables in a mathematical model may simplify the problem considerably. The decomposition approach divides a problem into a master problem (MP) and a subproblem (SP), which are solved iteratively. The MP, considering a subset of the variables, is solved first. Next, the subproblem is solved for the remaining variables, while temporarily fixing the variables' values of the MP. Finally, based on the outcome of the SP, one or more cuts are generated and added to the MP, thereby effectively preventing the MP from

revisiting similar areas of the search space. Classical Benders decomposition (Benders, 1962), considers linear programming subproblems where cuts are derived from their dual solution. This approach is often interpreted as D-W decomposition applied to the dual of the problem.
Classical Benders decomposition can be applied to MILPs of the form:

$$\begin{aligned} z = min \quad & c^T x + d^T y \\ \text{s.t.} \quad & Ax + By \geq b \\ & x \in X \subseteq \mathbb{Z}_+^n \\ & y \geq 0 \end{aligned} \tag{1.7}$$

Again, for simplicity we assume that (1.7) is bounded and feasible. When the x variables are fixed to a solution $\bar{x}$, i.e., $x = \bar{x}$, formulation (1.7) simplifies to an LP, formally known as the *subproblem*:

$$\begin{aligned} min \quad & c^T \bar{x} + d^T y \\ \text{s.t.} \quad & By \geq b - A\bar{x} \\ & y \geq 0 \end{aligned} \tag{1.8}$$

Let $\bar{z}$ be the value of the optimal solution to (1.8). When solving (1.8) for a given $\bar{x}$, two scenarios exist:

1. Formulation (1.8) yields a feasible solution $\bar{y}$ for a fixed $\bar{x}$. Solution $(\bar{x}, \bar{y})$ is primal feasible to the original problem (1.7), with objective value $\bar{z}$.
2. Formulation (1.8) is infeasible, i.e., there does not exist a single y such that $(\bar{x}, y)$ is feasible in the original problem space.

The dual of (1.8) is given by:

$$\begin{aligned} max \quad & u^T(b - A\bar{x}) + c^T \bar{x} \\ \text{s.t.} \quad & u^T B \leq d^T \\ & u \geq 0 \end{aligned} \tag{1.9}$$

Whenever the dual problem (Formulation (1.9)) is infeasible, i.e. there does not exist a solution satisfying the constraints in the dual, then, for any value of $\bar{x}$, the subproblem (Formulation (1.8)) must be infeasible or unbounded. Consequently, the original problem (Formulation (1.7)) must also be infeasible

or unbounded. Hence, in what remains we will assume that the solution space defined by the constraints in the dual (1.9) is non-empty.
Assume that (1.8) yields a feasible solution $\bar{y}$, then by strong duality, formulation (1.9) must have a (finite) extreme point solution $\bar{u}$, s.t. $c^T\bar{x}+d^T\bar{y} = c^T\bar{x}+\bar{u}^T(b-A\bar{x})$. By weak duality it follows that $\underbrace{c^Tx + d^Ty}_{z} \geq c^Tx+\bar{u}^T(b-Ax)$. Finally this leads to the *Benders cut*, also known as an *optimality cut*, $z \geq c^Tx+\bar{u}^T(b-Ax)$, where z is a lower bound on the optimal solution value of (1.7). Informally this cut states that each time variables x are fixed to $\bar{x}$, the objective value of formulation (1.7) will be at least $\bar{z}$.
On the other hand, when the subproblem (1.8) cannot be solved for a given $\bar{x}$, i.e., the subproblem is infeasible, then its dual (1.9) must be either infeasible or unbounded (positive infinity). By the aforementioned assumption that the solution space defined by formulation (1.9) is non-empty, the dual cannot be infeasible, so it must be unbounded, i.e., $\bar{u}^T(b - A\bar{x}) > 0$ for some extreme ray solution $\bar{u}$. We can then generate a *feasibility cut* $\bar{u}^T(b - Ax) \leq 0$ which effectively states that x cannot be equal to $\bar{x}$.
Model (1.7) can now be reformulated, thereby eliminating the y variables, as:

$$\begin{aligned} min \quad & z \\ \text{s.t.} \quad & z \geq c^Tx + (u^k)^T(b - Ax) && \forall k \in K \\ & (u^l)^T(b - Ax) \leq 0 && \forall l \in L \\ & x \in X \end{aligned} \tag{1.10}$$

where K and L are resp. the extreme points and extreme rays corresponding with dual formulation (1.9). Model (1.10) is known as the master problem.
As with the extended formulation (1.3) obtained through D-W decomposition, formulation (1.10) cannot be solved in its entirety due to the fact that their may be a vast number of constraints. Therefore, the master problem (1.10) is solved only for a subset of its constraints. After solving the master problem, the subproblem is solved, and cuts (constraints) are added to the master problem. This iterative procedure terminates whenever the optimal value of the master problem equals the largest finite objective value obtained for the subproblem.
Similar to D-W decomposition, Benders decomposition can be particularly

effective if the constraint matrix has some special structure, for example:

$$
\begin{aligned}
& min\, c^T x + d^1 y_1 + d^2 y_2 + \ldots + d^p y_p \\
& \\
\text{s.t. } & A^1 x + B^1 y_1 & & & & \leq b_1 \\
& A^2 x + & B^2 y_2 & & & \leq b_2 \\
& \vdots & & \ddots & & \\
& A^p x + & & & B^p y_p & \leq b_p \\
& x \in X & & & & \\
& y \geq 0 & & & &
\end{aligned}
\tag{1.11}
$$

The Benders decomposition of such a problem results in p LPs which can be solved independently of one another. These problem structures arise in for example stochastic programming problems where the variables x represent first-stage decisions, and the y variables represent second-stage decisions under different scenarios for x (Conforti et al., 2014).

A disadvantage of the Classical Benders decomposition approach is that the subproblem needs to be an LP. In more recent work, e.g., Geoffrion (1972) and Hooker (2007), the Benders decomposition approach has been generalized to a broader class of problems, no longer requiring the subproblem to be linear. Nevertheless, the exact same principles apply: a master problem, a relaxation of the original problem, and a subproblem are solved iteratively, while generating cuts along the way. Hooker (2007) introduced the concept of Logic Based Benders decomposition. In contrast to the approach introduced by Benders (1962), cuts are not necessarily obtained from the dual formulation of a linear subproblem, but through the so-called inference dual. Whenever the subproblem is a feasibility problem, the inference dual is a condition which, when satisfied, implies that the master problem is infeasible (Rasmussen and Trick, 2007). This condition can then be used to obtain Benders cuts to cut off infeasible solutions. A particular case of Logic Based Benders decomposition, frequently referred to as Combinatorial Benders decomposition, is discussed by Codato and Fischetti (2006) where it is applied to MILPs involving large numbers of logical implications (big-M constraints). Whenever a particular assignment of variable values in the MP renders the SP infeasible, a Combinatorial Benders cut is generated and added to the master problem, thereby ensuring that at least one of the variables in the master problem changes its value. Note that this approach only works for binary variables or integer variables with a small domain. Logic Based Benders decomposition is used to solve the Concrete Delivery Problem in Chapter 4.

Lagrangian Relaxation and Decomposition

When the constraints in an MILP can be partitioned in a group of 'nice' constraints and 'complicating' constraints, Lagrangian Relaxation (LR) may be used to simplify the problem. LR removes the complicating constraints and introduces a penalty function in the objective function to penalize violation of the complicating constraints. LR can be used to compute strong bounds on the objective value of a MILP; it can be shown that the LR is at least as strong as the Linear Relaxation of the problem (Geoffrion, 1974). In some cases, the LR bound equals the optimal solution of the MILP (Wolsey and Nemhauser, 1988). Given the following Integer Problem where $Ax \geq b$ are the complicating constraints and $Bx \geq d$ the easy constraints:

$$\begin{aligned} min \quad & cx \\ \text{s.t.} \quad & Ax \geq b \\ & Bx \geq d \\ & x \in \mathbb{Z}^n \end{aligned} \tag{1.12}$$

The Lagrangian Relaxation $LR(\lambda)$ for $\lambda \geq 0$ is given by:

$$\begin{aligned} LR(\lambda) = min \quad & cx + \lambda(b - Ax) \\ \text{s.t.} \quad & Bx \geq d \\ & x \in \mathbb{Z}^n \end{aligned} \tag{1.13}$$

Note that (1.13) is a relaxation of the original problem (1.12) because (1) a negative term ($(b - Ax) \leq 0$) is added to the objective function and (2) some constraints have been removed. In fact, $LR(\lambda)$ is a relaxation of the original problem for all $\lambda \geq 0$. The problem is to find the strongest such relaxation, i.e., $LD = min_{\lambda \geq 0} LD(\lambda)$. Problem LD is called the Lagrangian Dual of the original problem with respect to constraints $Ax \geq b$. Geoffrion (1974) showed that LD is at least as strong as the LP relaxation of the original problem. A solution $\bar{x}$ to a problem $LR(\lambda)$ for some $\lambda \geq 0$ is optimal in the original problem if (1) $\bar{x}$ satisfies all the constraints in the original problem and (2) if $c\bar{x} = c\bar{x} + \lambda(b - A\bar{x})$, i.e., if $\lambda(b - A\bar{x}) = 0$.

Finally, in line with our discussion of decomposition approaches, one can also decompose a problem through Lagrangian Decomposition. Observe that problem

(1.12) can be rewritten as:

$$\begin{aligned} min \quad & cx \\ \text{s.t.} \quad & Ax \geq b \\ & By \geq d \\ & x = y \\ & x, y \in \mathbb{Z}^n \end{aligned} \tag{1.14}$$

Dualizing $x = y$ yields:

$$\begin{aligned} min \quad & cx + \lambda(x - y) \\ \text{s.t.} \quad & Ax \geq b \\ & By \geq d \\ & x, y \in \mathbb{Z}^n \end{aligned} \tag{1.15}$$

Clearly (1.15) decouples into two problems which can be solved independently:

$$\begin{aligned} min \quad & (c + \lambda)x \\ \text{s.t.} \quad & Ax \geq b \\ & x \in \mathbb{Z}^n \end{aligned} \tag{1.16}$$

$$\begin{aligned} min \quad & -\lambda y \\ \text{s.t.} \quad & By \geq d \\ & y \in \mathbb{Z}^n \end{aligned} \tag{1.17}$$

The sum of the objective values of these two problems is a valid lower bound on (1.12) for any value of λ. The resulting Lagrangian Decomposition bound is often stronger than the standard Lagrangian Relaxation bound. Moreover, the resulting subproblems may be easier to solve, and can capture different structural characteristics of the problem.
LR is applied in Chapter 2 to compute lower bounds on the objective value of the School Bus Routing Problem.

Chapter 2

The Schoolbus Routing Problem

Abstract

The School Bus Routing Problem (SBRP), a generalization of the well-known Vehicle Routing Problem, involves the routing, planning and scheduling of public school bus transportation. The problem can be decomposed into several subproblems, including bus stop selection, assigning students to buses, and determining the bus routes. This work presents an exact branch-and-price framework for the SBRP, with a strong emphasis on efficiency issues inherently related to column generation.
Experiments in this chapter are conducted on a data set of 128 SBRP instances. Many of these instances are solved optimally; for the remaining instances, strong lower and upper bounds have been derived, thereby improving upon some of the best results published in related work. Both lower bounds computed on the optimum solution, as well as stabilization added to the column generation procedure significantly improve the performance of the branch-and-price framework.

The content of this chapter is based on joint work with F.C.R. Spieksma and G. Vanden Berghe, see Kinable et al. (2014a)

2.1 Introduction

Many primary and secondary schools in Europe organize school bus transportation services for commuting students. The organization constitutes a challenging task both from a planning and a budgetary perspective. A typical school bus planning problem entails selecting appropriate bus stops reachable by the students, assigning students to the available buses, and determining the necessary bus routes. Possible locations for the bus stops are usually restricted by local policies and legislations such as the maximum walking distance to the stop, or safety regulations. From an optimization point of view, different goals can be aspired including minimizing the total travel distance or the number of buses, balancing bus loads or keeping the travel time spent by the students to a minimum. This vehicle routing problem is commonly referred to as the School Bus Routing Problem (SBRP), see Section 2.2 for a precise description.
The SBRP is part of the class of capacitated vehicle routing problems in which a set of bus tours has to be designed, each passing through a number of bus stops. The tours have to be disjoint except in the depot node. Students are assigned to stops on the tours; students can only be assigned to stops they can reach, and the total number of students assigned to stops on a single route cannot exceed the bus capacity.
Among the many variants that exist in the domain of schoolbus routing (see Park and Kim (2010), and Section 2 for an overview), we focus here on a single-school SBRP, without time windows. The main contribution of this chapter is to present a branch-and-price framework based on a set covering formulation of the SBRP. We provide an in-depth discussion on the design of the Branch-and-Price framework, thereby focusing on a number of choices made in the implementation in order to improve the efficiency of the framework. We discuss:

- lower bounds on the optimal integer solutions
- a comparison of two distinct stabilization approaches to reduce degeneracy
- effective pruning mechanisms
- a column pool manager

We demonstrate the performance of our branch-and-price algorithm on two benchmark sets: one set containing traditional SBRP instances, and a newly generated set of instances.
The remainder of this chapter is structured as follows: first, in Section 2.2, an overview of related work on the SBRP is given. Next, Sections 2.3, 2.4 introduce the column generation procedure. The latter procedure is then integrated in a branch-and-price framework which we discuss in Section 2.5.

To improve the framework's efficiency, several extensions are implemented, including stabilization (Section 2.4.2), a column manager (Section 2.4.3), and a pruning mechanism (Sections 2.5.3, 2.5.4). Finally, the resulting algorithm is tested on a series of 128 SBRP instances. The results are presented in Section 2.6. Section 5.10 provides the conclusions.

2.2 Problem description and related research

The SBRP can be defined as follows. We are given a set of bus stops V (including the school) with a distance for each ordered pair of stops, as well as a set of students S. For each student $s \in S$, a set $V_s \subseteq V$ is given that represents the set of stops to which the student can be assigned. Assigning student s to a stop in V_s is called a *feasible* assignment. There is a fleet of identical vehicles available, each with capacity Q. A *route* is a sequence of stops ending with the school. Students assigned to stops in a route are picked up by the vehicle performing that route. The problem is to find a feasible assignment of students to stops, and to find routes for the vehicles, such that (i) the capacity of each vehicle is respected, (ii) each student is picked up, and (iii) total length of the routes is minimized.

Notice that the description of our problem does not provide the locations of the students. Indeed, we only know for each student the set of stops to which this student can be assigned. This feature distinguishes our problem from the more general Multiple Vehicle Traveling Purchaser problem (MV-TPP, see Riera-Ledesma and Salazar-González (2012)) where a location for a student, and its induced distance to a stop, is used to include assignment costs in the objective of the problem. These assignment costs capture the cost of assigning a student to a stop, and may represent walking distance. Of course, MV-TPP is more general since, by having assignment costs in $\{0, \infty\}$, an instance of SBRP arises. However, the setting without assignment costs is conform the situation faced by a bus company designing routes (see Schittekat et al. (2013) for more details), where it is stipulated that any assignment of students to stops should satisfy a maximum walking distance. Thus, from the point of view of the bus company, no optimization of the walking distances is required; it is only required that they do not exceed this maximum walking distance.

Obviously, SBRP is not a new problem. Indeed, many variations have been proposed in the literature. Here it is not our ambition to give an overview; instead we restrict ourselves to discussing the main solution approaches. For a recent overview, we refer to (Park and Kim, 2010); a discussion of the MV-TPP and related models can be found in (Riera-Ledesma and Salazar-González, 2013).

Due to its composite nature, the earliest papers discussing school bus routing

attempted to solve the problem via decomposition. The selection, assignment and routing problems are solved independently and the results are then merged into a feasible SBRP solution. These attempts can be roughly classified into two groups (Park and Kim, 2010): Location Allocation Routing (LAR) and Allocation Routing Location (ARL). In the LAR class (e.g., Bodin and Berman (1979); Desrosiers et al. (1986); Dulac et al. (1980)), first the stops are determined and students are assigned to those stops, after which the necessary bus routes are generated. A disadvantage of this approach is that the first two subproblems, selection and allocation, are solved independently of the routing problem, often resulting in excessive and suboptimal routes (Park and Kim, 2010). To counter this problem, the ARL strategy has been proposed effectively changing the order in which the subproblems are treated (Bowerman et al., 1995; Chapleau et al., 1985). First the students are assigned to buses, thereby taking capacity constraints into consideration. Then the bus stops are selected and the bus routes are created. Although this approach resolves some of the issues inherent to LAR (Bowerman et al., 1995), assigning students to buses before the stop locations have been decided upon may still lead to suboptimal schedules. The problems surrounding the decomposition of the SBRP as demonstrated by the discussions on LAR and ARL strategies motivate the use of a more integrated approach that treats the SBRP as an integral problem instead of the sum of several subproblems.

Schittekat et al. (2013) describe a metaheuristic for the SBRP. By comparing their results with a lower bound, they show that the metaheuristic is capable of efficiently producing high-quality solutions for the instances generated.

Riera-Ledesma and Salazar-González (2012) propose a cutting plane algorithm for MV-TPP. The method is based on an integer programming formulation that uses, among other variables, a binary variable for each pair of stops. They report extensive computational results solving instances with up to 125 stops and 125 students. In a recent follow-up paper, Riera-Ledesma and Salazar-González (2013) use a set covering formulation, together with cuts from (Riera-Ledesma and Salazar-González, 2012) to construct a branch-and-cut-and-price algorithm for the MV-TPP. Although their work represents a formidable step forward in our ability to solve instances of MV-TPP, it is good to note that in most of their instances the number of students does not exceed the number of stops. In our experience however (see Schittekat et al. (2013); Park et al. (2012)), typical instances of the SBRP feature many more students than stops. One goal of our work is to find out how a branch-and-price approach fares upon such instances.

Another problem related to SBRP is the m-Capacitated Ring-Star Problem (m-CRSP) (Baldacci et al., 2007). In m-CRSP, one has to find m paths (rings), starting and ending in a depot, and traversing through a number of customers and steiner nodes. The paths have to be disjoint, except for the depot node. Each customer in the graph needs to be either part of a ring, or must be assigned

to a node which is part of a ring. The total number of customers in a single ring, plus the number of customers assigned to it cannot exceed a predefined capacity Q. The m-CRSP considers both assignment and routing costs. Assignment costs are incurred whenever a customer is assigned to another node in a ring. Routing costs are incurred for the edges that are part of a ring.
Let us clarify the relation between on the one hand, MV-TPP and its special case SBRP, and, on the other hand m-CRSP. Clearly, as described in (Riera-Ledesma and Salazar-González, 2012), an algorithm for MV-TPP can be used to solve instances of m-CRSP. To see this, imagine that each customer in m-CRSP becomes a student plus a stop in MV-TPP, while a steiner node in m-CRSP becomes a stop. Next, a solution to the resulting instance of MV-TPP which consists of routes that visit stops to which students have been assigned, is easily casted as a solution to m-CRSP. The reverse is true as well: MV-TPP is a special case of m-CRSP. For each student in MV-TPP, a customer is created, and for every stop a steiner node. Routing costs between the steiner nodes are identical to the routing costs between the stops in MV-TPP. Routing costs of edges incident to customers are set to infinity. In a similar fashion, assignment costs are determined. All assignment costs are set to infinity, except for certain customer-steiner node pairs: for a student $s \in S$ and V_s, the assignment costs are set to 0 for the corresponding customer-steiner node pairs. When the optimal solution to the constructed m-CRSP yields an objective value smaller than infinity, a feasible solution to MV-TPP follows directly. Note that m-CRSP requires m, the number of rings, as input. This value is not known for MV-TPP, but is bounded from above. Hence, an algorithm for m-CRSP can be used to solve the MV-TPP by means of binary search on the number of vehicles. Exact methods for m-CRSP based on integer programming formulations have been presented in (Baldacci et al., 2007) (Branch-and-Cut) and (Hoshino and de Souza, 2009) (Branch-and-Cut-and-Price).

2.3 Set covering formulation of SBRP

We use the following MIP formulation of the SBRP, which we will denote as the Master Problem (MP). Table 2.1 describes the necessary variables and parameters. For a traditional three-index MIP formulation of the SBRP, we refer to (Schittekat et al., 2013; Riera-Ledesma and Salazar-González, 2013).

Variable/parameter	Description
z_p	1 if bus schedule $p \in P$ is used, 0 otherwise
V	Set of bus stops (including the school v_0)
S	Set of students
V_s	$V_s \subseteq V \backslash \{school\}$: the set of stops student $s \in S$ can reach.
P	Set of all bus schedules.
t_{sp}	1 if student s is picked up in bus schedule p, 0 otherwise
r_{vp}	1 if stop v is part of bus schedule p, 0 otherwise
Q	Maximum capacity of the buses.
δ_p	Cost induced by schedule p
L, U	Lower and upper bound on the number of buses

Table 2.1: Parameters and variables defining the SBRP

$$MP: \ min \quad \sum_{p \in P} \delta_p z_p \tag{2.1}$$

$$\text{s.t.} \quad \sum_{p \in P} t_{sp} z_p \geq 1 \qquad \forall s \in S \tag{2.2}$$

$$\sum_{p \in P} r_{vp} z_p \leq 1 \qquad \forall v \in V \tag{2.3}$$

$$\sum_{p \in P} z_p \leq U \tag{2.4}$$

$$\sum_{p \in P} z_p \geq L \tag{2.5}$$

$$z_p \in \{0, 1\} \qquad \forall p \in P \tag{2.6}$$

In this formulation, p is an index corresponding with a bus route. More precisely, p defines a complete, valid, *bus schedule*: an ordered sequence of stops the bus driver should visit (ending at the school), and the specific students that should be picked up at the corresponding stops. The set of all feasible bus schedules is denoted by P. As mentioned before, we assume that all buses are identical, and have a capacity Q. The cost δ_p associated with each bus schedule is the travel

distance required to visit all stops on the schedule (notice that these travel distances do not necessarily satisfy the triangle inequality). Constraints (2.2), (2.3) respectively ensure that each student is picked up at some stop, and that no stop is visited more than once. Constraints (2.2) are commonly referred to as Set Cover constraints. Constraint (2.4) enforce bounds on the number of buses used in the solution. Observe that formulation (2.1)-(2.6) is quite flexible in the sense that it can accommodate all kinds of potential constraints on individual routes. For instance, upper bounds on the length of a route, or on the number of stops within a route, are easily incorporated.

2.4 Column Generation

Solving problem MP (Section 2.3) requires an exponentially large set of columns P. When the integrality constraints are replaced by the weaker constraints $z_p \geq 0$, we obtain a relaxation, denoted LPM, which we will solve via a column generation (CG) procedure (see (Chvátal, 1983; Vanderbeck, 1994; Wolsey, 1998) for details on CG). Instead of solving LPM directly, a reduced version called the Restricted Master Problem (RMP) is solved, where the set of columns P is replaced by a subset $P' \subseteq P$, $|P'| \ll |P|$. Subset P' contains an initial feasible solution which is produced by the heuristic proposed in Schittekat et al. (2013). Subsection 2.4.1 describes the pricing problem and provides two algorithms to solve it. The performance of the CG procedure is improved by using stabilization and by adding a column pool manager. Subsection 2.4.2 compares two popular stabilization approaches, whereas Subsection 2.4.3 elaborates on the column pool manager.

2.4.1 Pricing Problem

The pricing problem, obtained from the dual formulation of problem MP (2.1)-(2.6) is as follows:

$$\exists p : q_p < 0 \tag{2.7}$$

$$q_p = \delta_p + \sum_{v \in V} r_{vp} w_v - \sum_{s \in S} t_{sp} u_s + m - n \tag{2.8}$$

where w_v, u_s, m, n are the dual variables associated with constraints (2.3), (2.2), (2.4), (2.5), respectively.
Throughout this chapter, p^* denotes the column which yields the most negative

value for q_p, i.e., $p^* = \underset{p \in P}{\operatorname{argmin}}\{q_p\}$, and $q^* = q_{p^*}$ is the corresponding optimal objective value of the pricing problem.
Informally, the pricing problem involves finding a path ending at the school such that q_p yields a negative value. When the school is decoupled into a start and end node, the problem becomes the well-known Elementary Shortest Path Problem with Resource Constraints (ESPPRC) (Desaulniers et al., 2005). Since ESPPRC is NP-hard, we use two different procedures to solve the pricing problem: a fast heuristic, and a slower but exact labeling algorithm. First, the heuristic attempts to quickly find several columns with negative reduced cost. When this attempt fails, the exact labeling algorithm takes over. Due to its exact nature, the labeling algorithm is guaranteed to find a negative reduced cost column if such a column exists. Ideally, the number of times the exact algorithm is invoked is kept to a minimum as it is relatively expensive to execute.

A local search heuristic

Our local search algorithm is initialized with a randomly generated feasible bus route, i.e., an ordered sequence of stops. The algorithm iteratively attempts to improve the solution by exploring neighboring solutions. The set of feasible neighbor solutions is defined as the union of the following three neighborhoods:

1. Insert neighborhood - an unvisited stop is inserted in the route.
2. Remove neighborhood - a stop is removed from the route.
3. Swap neighborhood - Two stops in the route are swapped, thereby changing the order in which stops are visited.

For each route, a student assignment problem has to be solved. Due to the presence of constraint (2.3) in the MP, certain students are forced to be part of the route; a student $s \in S$ must be picked up whenever the route visits all stops in V_s. When there is residual bus capacity, additional students are added to the schedule, thereby maximizing the total dual price collected by the students. The objective value of a solution is calculated via equation (2.8). A neighboring solution is selected over another solution if its objective value is better; only improving moves are allowed. The algorithm terminates when there are no more improving moves available, i.e., when a local optimum is reached.
It is known that the column generation procedure can be accelerated when the pricing problem returns multiple columns at once. Therefore, the heuristic is executed several times to obtain multiple solutions. To prevent the heuristic

from returning the same solution twice, we initialize it with different routes, and we only accept neighborhood moves resulting in solutions which are sufficiently distinct from earlier returned solutions; a particular solution is considered sufficiently distinct from another solution whenever the stops along its route are not a subset of the stops visited by the other solution, or when the number of stops visited along its route differs by at least 50%.

Labeling algorithm

To prove optimality of the column generation procedure, an exact pricing algorithm is required. We solve the pricing problem to optimality using a dynamic programming based approach: a labeling algorithm. In related works, such approaches have been successfully applied to problems such as the Pickup and Delivery Problem with Time Windows (Ropke and Cordeau, 2009; Dell'Amico et al., 2006), the Vehicle Routing Problem with Time Windows (Desaulniers et al., 2005), and the Capacitated Location-Routing problem (Albareda Sambola, 2003). Alternatively, one may use the q-route approach as presented in Fukasawa et al. (2006), as has been demonstrated for the m-CRSP in Hoshino and de Souza (2012).

Consider the weighted, undirected, graph $G = (V', E)$, $V' = (V \cup \{t\})$, where V represents the set of bus stops, t a copy of the school vertex v_0, and $E = (V \times V) \cup (V \backslash v_0 \times \{t\})$ the set of edges. The weight on an edge $(i, j) \in V \times V$ equals c_{ij} whereas the weights on the edges (i, t), $i \in (V \backslash v_0)$ are equal to c_{iv_0}. To solve the pricing problem, the algorithm searches for a simple path from v_0 to t with negative reduced cost.

The labeling algorithm starts by generating the shortest possible partial path: $[v_0]$. At each consecutive iteration, a partial path p' is extended to each stop in V not already present in p'. For efficiency reasons, partial paths are not stored in their entirety; instead labels are used. Each label ℓ is associated with a vertex $v(\ell) \in V$, and has a reference (pointer) to a preceding label $p(\ell)$, except if the label is associated with vertex v_0. Each label ℓ uniquely identifies a partial path from v_0 to $v(\ell)$, i.e a path $p_\ell = [v(\ell), v(p(\ell)), v(p(p(\ell)))..., v_0]$ can be reconstructed simply by following the pointers to the preceding labels. Let $S(\ell)$ be a set of students that are assigned to a stop in path p_ℓ, and let $V(\ell)$ be the set of stops in p_ℓ, i.e., $V(\ell) = \{v(\ell), v(p(\ell)), v(p(p(\ell)))..., v_0\}$. The set $S(\ell)$ is calculated by selecting at most Q students, having the highest dual price; of course, a student $s \in S$ can only be selected if $V_s \cap V(\ell) \neq \emptyset$. Moreover, if $V_s \backslash V(\ell) = \emptyset$, then student s must be in $S(\ell)$ due to Constraint (2.3). Finally, for each partial path denoted by label ℓ, $c(\ell)$ is the accumulated cost defined by Equation (2.8).

When a partial path would be extended to all its unvisited neighbors, the

algorithm would simply be an inefficient enumeration approach. To circumvent this issue, several restrictions are applicable:

1. A path can only be extended to the school if the resulting path is of negative reduced cost.
2. A path may not be extended to a vertex if this would result in a cycle.
3. A path is not extended to a vertex if a lower bound proves that it can never become a negative reduced cost path. For any path identified by label ℓ, a lower bound on the reduced cost can be computed by taking into account: 1) $c(\ell)$, 2) a lower bound on cost required to complete the path to vertex t, 3) an upper bound on the dual prices that can be collected by students on the complete route.
4. A path is not extended if it is dominated by another path, i.e., if there exists a more cost-effective route.

To determine whether a path having label ℓ_1 dominates another path having label ℓ_2, the following domination criteria are used:

$$v(\ell_1) = v(\ell_2) \tag{2.9}$$

$$c(\ell_1) \leq c(\ell_2) \tag{2.10}$$

$$V(\ell_1) \subseteq V(\ell_2) \tag{2.11}$$

The above conditions intuitively state that for two paths, leading to the same stop, the path identified by label ℓ_1 dominates the path having label ℓ_2 if its reduced cost is better, and if path ℓ_2 has access to the same students as path ℓ_1, i.e., $S(\ell_1) \subseteq \{s \in S : V_s \cap V(\ell_2) \neq \emptyset\}$.
Clearly, it is pointless to extend a dominated label. Moreover, as a logical consequence, all successors of a dominated label are also suboptimal. For this reason, in our implementation we also maintain pointers from a label to its direct successors such that we can delete or modify successors in case one of their predecessors turns out to be suboptimal (for more details, we refer to a discussion on label correcting algorithms, e.g., Ahuja et al. (1993)).
The labeling algorithm can be accelerated by adding an admissible heuristic. Given the sequence of stops that make up the partial route, the heuristic first computes an upper bound on the maximal dual price that can be incurred by the students. Next, the heuristic computes a lower bound on the total distance of the complete route, that is the distance traveled so far plus the minimum distance required to extend the route to node t. Finally, via equation (2.8) the heuristic assesses whether extending the label to a path with negative reduced

cost is attainable.
The labeling algorithm terminates as soon as there are no more labels to extend. If, during the course of the algorithm, no path has been found, we can positively attest that no negative reduced cost path exists.
It should be pointed out that the order in which labels are being processed is of importance. Preferably, labels which at a later point in time appear to be dominated by some other label are not extended. The latter would require that the algorithm recomputes a substantial part of the search tree. In our implementation we experimented with three orderings in which the labels are processed: breadth first search (BFS), depth first search (DFS) and best first search (BEFS). In BFS the oldest unextended label gets extended first. In DFS, the newest generated label is always extended first. Finally, BEFS extends the label representing the path with the best objective value. Our experiments showed that BFS slightly outperformed DFS which in turn outperformed BEFS.
Several extensions and improvements for the labeling algorithm have been proposed in related works, some of which we have implemented to accelerate the pricing procedure.
The labeling algorithm should not necessarily search for a column with the best objective value. Instead of performing an exhaustive search, the algorithm could be easily modified such that it returns a negative reduced cost column as soon as it finds one. Notice that this modification still enables us to solve the column generation procedure to optimality.
A second modification amounts to changing the nodes to which a label can be extended. In the previous section, the exact labeling procedure always extends a partial path which ends in a node $v \in V$ to all its unvisited neighbors. Similar to (Dell'Amico et al., 2006), we also implemented a k-Nearest Neighbor (k-NN) variant where a node is only extended to its k nearest neighbors. Naturally, for $k = |V|$, the search neighborhood becomes exact. We start the algorithm for $k = 2$. When no solution has been found, we expand the search to $k = 4$ and finally to $k = |V|$. This procedure is motivated by the fact that the total travel distance is minimized for each bus tour. As a consequence of this approach, the average number of labels generated during the labeling procedure decreased for a number of instances.

2.4.2 Stabilization

Column generation is susceptible to degeneracy (Desaulniers et al., 2005), a process which decreases the convergence speed of the column generation algorithm. During several iterations, new columns are added to the master problem, but no improvements are being made in terms of the objective function

(plateau effect). Also, once the objective function gets closer to its optimal value, the convergence speed decreases drastically. Sometimes many additional iterations are needed to complete the process (tailing-off effect) (Desaulniers et al., 2005). The slow convergence speed is partially attributed to degeneracy in the primal. In addition, it is known that strong oscillations in the dual variables play an important role. In a typical column generation process, it is frequently observed that some dual variables pick up most of the dual price, whereas the remaining variables have a near-zero value. The unbalanced distribution of dual prices regularly causes the pricing problem to generate redundant columns which will never be part of an optimal solution; an example of which is given in (Rousseau et al., 2007).
To counter these issues, stabilization procedures have been developed in an attempt to guide the search faster towards a global optimum. Many different stabilization approaches exist, proximal-type (Merle et al. (1997); Amor and Desrosiers (2006)), Bundle-type (Briant et al. (2008)), and Iterior Point stabilization (Rousseau et al. (2007)) are the most prominent ones; detailed discussions and comparisons of these methods can be found in (Lübbecke, 2010; Amor et al., 2009; Lübbecke and Desrosiers, 2002). In this work, we compare two of the most popular stabilization approaches, namely Du Merle's 3-piecewise proximal type stabilization (Merle et al., 1997), as well as Interior Point Stabilization (Rousseau et al., 2007). In the next two subsections, we apply both methods to the SBRP formulation. Numerical experiments are reported in Section 2.6.

3-piecewise stabilization

In proximal-type stabilization methods, fluctuations in the dual solutions are suppressed by limiting the Euclidean distance between consecutive dual solutions. In general terms, the idea can be described as follows. First a good dual solution is estimated, around which a hypercube is centered, thereby marking a trust area in which dual points generated by the RMP must lie. Penalties are incurred when the points fall outside the allotted trust region. Next, the stability center is re-adjusted whenever a better estimate of the optimal dual solution is available. Similarly, the hypercube can be resized and the penalty functions adjusted.
In this work we will focus on a specific proximal-type stabilization approach proposed by Merle et al. (1997): 3-piecewise stabilization. To obtain a 3-piecewise stabilized column generation formulation, we have added stabilization variables y^- and y^+ to constraints (2.2), (2.3) and the objective function. The resulting stabilized formulation becomes:

$$min \quad \sum_{p\in P} \delta_p z_p - \sum_{s\in S} \delta_s^- y_s^- + \sum_{s\in S} \delta_s^+ y_s^+ -$$

$$\sum_{v\in V} \delta_v^- y_v^- + \sum_{v\in V} \delta_v^+ y_v^+ \tag{2.12}$$

$$\text{s.t.} \quad \sum_{p\in P} t_{sp} z_p - y_s^- + y_s^+ \geq 1 \qquad \forall s \in S \tag{2.13}$$

$$\sum_{p\in P} r_{vp} z_p + y_v^- - y_v^+ \leq 1 \qquad \forall v \in V \tag{2.14}$$

$$L \leq \sum_{p\in P} z_p \leq U \tag{2.15}$$

$$y_i^- \leq \epsilon_i \qquad \forall i \in V \cup S \tag{2.16}$$

$$y_i^+ \leq \epsilon_i \qquad \forall i \in V \cup S \tag{2.17}$$

$$z_p, y_i^+, y_i^- \geq 0 \qquad \forall p \in P, i \in V \cup S \tag{2.18}$$

where δ^+, δ^-, ϵ are predefined positive parameters. Note that when $\delta^+ = \delta^- = \epsilon = 0$, the original LPM (Section 2.3) appears. Writing down the dual gives us:

$$max \sum_{s\in S} (u_s - \epsilon_s t_s^- - \epsilon_s t_s^+) -$$

$$\sum_{v\in V} (w_v + \epsilon_v t_v^- + \epsilon_v t_v^+) - Um + Ln \tag{2.19}$$

$$\text{s.t.} - \sum_{v\in V} r_{vp} w_v + \sum_{s\in S} t_{sp} u_s - m + n \leq \delta_p \qquad \forall p \in P \tag{2.20}$$

$$-u_s - t_s^- \leq -\delta_s^- \qquad \forall s \in S \tag{2.21}$$

$$u_s - t_s^+ \leq \delta_s^+ \qquad \forall s \in S \tag{2.22}$$

$$-w_v - t_v^- \leq -\delta_v^- \qquad \forall v \in V \tag{2.23}$$

$$w_v - t_v^+ \leq \delta_v^+ \qquad \forall v \in V \tag{2.24}$$

$$t^+, t^-, u, w \geq 0 \tag{2.25}$$

where u, w, m, n, t^+, t^- are the dual variables associated with Constraints (2.13)-(2.17).
As can be observed from the dual formulation, the addition of stabilization parameters restricts the domains of the dual variables u_s, w_v to $\delta_s^- \leq u_s \leq \delta_s^+$, $s \in S$, resp. $\delta_v^- \leq w_v \leq \delta_v^+$, $v \in V$; deviation of these intervals is penalized by $\epsilon_i t_i$, $i \in V \cup S$ in the dual objective. Note that the stabilized formulation does not change the pricing problem (Equation 2.8). However, an optimal solution to the stabilized LPM is only obtained if there are no more columns with negative reduced cost, and $y_i^- = y_i^+ = 0, \forall i \in V \cup S$. Several different update procedures for δ and ϵ are proposed in (Merle et al., 1997; Oukil et al., 2007). However, during our experiments, we obtained the best results with the following update procedure. At iteration $j+1$ set:

$$\delta_s^- := u_s^j - \delta_s \qquad \forall s \in S \qquad (2.26) \qquad \delta_v^- := w_v^j - \delta_v \qquad \forall v \in V \qquad (2.28)$$

$$\delta_s^+ := u_s^j + \delta_s \qquad \forall s \in S \qquad (2.27) \qquad \delta_v^+ := w_v^j + \delta_v \qquad \forall v \in V \qquad (2.29)$$

where u_s^j resp. w_v^j are the dual values, of variables u_s, w_v, obtained at iteration j. At iteration $j = 0$, we use the dual values obtained from equations (2.2) resp. (2.3). Initially, we set $\delta_i = 0.5$, $\epsilon_i = 0.1$, $\forall i \in S \cup V$. Whenever, at iteration j, the pricing problem does not find any more negative reduced cost columns, we update the values of δ and ϵ as follows:

$$\epsilon_i := \frac{\epsilon_i}{2} \qquad \forall i \in U \cup V \qquad (2.30)$$

$$\delta_i := \begin{cases} \max\{0.1, \frac{1}{\delta_i}\} & \text{if } \delta_i^- \leq u_s^j(w_v^j) \leq \delta_i^+ \\ \min\{1, 2\delta_i\} & \text{otherwise} \end{cases} \qquad \forall i \in U \cup V \qquad (2.31)$$

Since ϵ is decreased at every update cycle, the y variables gradually dissipate. The above update procedure yielded significantly better results than a procedure which updates δ and ϵ during every column generation iteration.

Interior Point Stabilization

A major disadvantage of proximal-type stabilization methods is the extensive number of parameters that have to be tuned. To our knowledge, there is no consensus on how to select the initial parameters nor how to update them. Furthermore, not every choice of parameters works well for each instance.
A different stabilization approach, which circumvents the issue of parameter selection, is Interior Point Stabilization (IPS) (Rousseau et al., 2007). IPS attempts to achieve a better dual value distribution by generating a dual solution of the RMP that is an interior point of the optimal dual face rather

than an extreme point. The latter is achieved by taking a convex combination of several different optimal dual solutions. Experiments have shown that this approach decreases the amount of columns needed to prove optimality. However, the main argument against the use of IPS is that it requires the RMP to be solved multiple times during each column generation iteration to obtain the interior points. Nevertheless, in some occasions, better results are reported compared to proximal approaches, e.g., in Dell'Amico et al. (2006).
Let P^* be the set of columns for which $z_p > 0$, i.e., a subset of columns that are in the basis. Furthermore, let V^* and S^* be the sets of students resp. stops for which constraints (2.3) resp. (2.2) are *not* tight. By complementary slackness conditions, the optimal face of the dual polyhedron (D^*) corresponding to Equations (2.1)-(2.5) is given by the following constraints (Rousseau et al., 2007):

$$\sum_{s \in S} t_{sp} u_s - \sum_{v \in V} r_{vp} w_v - m + n \leq \delta_p \qquad \forall p \in P \backslash P^* \tag{2.32}$$

$$\sum_{s \in S} t_{sp} u_s - \sum_{v \in V} r_{vp} w_v - m + n = \delta_p \qquad \forall p \in P^* \tag{2.33}$$

$$u_s = 0 \qquad \forall s \in S^* \tag{2.34}$$

$$u_s \geq 0 \qquad \forall s \in S \backslash S^* \tag{2.35}$$

$$w_v = 0 \qquad \forall v \in V^* \tag{2.36}$$

$$w_v \geq 0 \qquad \forall v \in V \backslash V^* \tag{2.37}$$

$$m = 0 \qquad \text{if Eq. (2.4) is tight} \tag{2.38}$$

$$m \geq 0 \qquad \text{otherwise} \tag{2.39}$$

$$n = 0 \qquad \text{if Eq. (2.5) is tight} \tag{2.40}$$

$$n \geq 0 \qquad \text{otherwise} \tag{2.41}$$

An extreme point of this polyhedron can be obtained by using an arbitrary objective function:

$$max \quad \sum_{s \in S} \mu_s u_s - \sum_{v \in V} \mu_v w_v - Um + Ln \tag{2.42}$$

Here, the vector μ is randomly generated with each term uniformly chosen from the interval $[0, 1]$. Let D^μ denote the complete LP (Eq (2.32))-(2.42)) . The

dual of D^μ, P^μ, becomes:

$$min \quad \sum_{p \in P} \delta_p z_p \tag{2.43}$$

$$\text{s.t.} \quad \sum_{p \in P} r_{vp} z_p \leq \mu_v \qquad \forall v \in V \backslash V^* \tag{2.44}$$

$$\sum_{p \in P} t_{sp} z_p \geq \mu_s \qquad \forall s \in S \backslash S^* \tag{2.45}$$

$$L \leq \sum_{p \in P} z_p \leq U \tag{2.46}$$

$$z_p \geq 0 \qquad \forall p \in P \backslash P^* \tag{2.47}$$

$$z_p \in \mathcal{R} \qquad \forall p \in P^* \tag{2.48}$$

When solving P^μ for different μ, several extreme points are obtained. To obtain distant points, when we solve P^μ we also solve $P^{-\mu}$.
Experiments revealed that for many choices of μ, P^μ is infeasible, implying that D^μ is unbounded. The latter can be attributed to Equation (2.44); when comparing to Equation (2.3) in the original formulation, Equation (2.44) is stronger when $\mu_v < 1$. Fixing $\mu_v = 1 \ \forall v \in V$ resolved the issue while still generating sufficiently distinct extreme points. We would like to point out that whenever D^μ is unbounded, it would be possible to generate useful points by taking any feasible extreme point and determining the recession direction. The result is a ray which can in turn be used to compute an interior point of D^*.

2.4.3 Column Pool Manager

During each iteration, the pricing problem returns one or more columns, thereby increasing the size of the master problem. When the total number of columns in the RMP becomes too large, evaluating the master problem becomes a time consuming process. A Column Pool Manager (CPM) is used to reduce the number of 'active' columns in the RMP (see Barnhart et al. (1998)).
The CPM associates a timer with each column in the master problem. Every CG iteration, the timers are increased by one. The timer of a column $p \in P$ is set to zero when the column is part of the basis, i.e., when a non-zero value is associated with z_p. Once the number of active columns exceeds a certain threshold (2000 in our implementation), the patterns unused the longest are

removed from the master problem and stored in a pool. Each iteration, before the pricing problem is invoked, the pool is queried to determine whether certain columns need to be moved back into the master problem. The latter is necessary to prevent invoking an expensive pricing problem, which could yield the same columns. When the lookup procedure returns columns, the pricing problem is skipped. Note that the initial columns are never removed from the active set of columns because they ensure feasibility which could otherwise not be assured in a stabilized master problem.
To keep the pool size reasonable, similarly to the active columns, a timer is associated with the columns in the pool. Once the size of the pool exceeds 3000 columns, the oldest columns are permanently deleted.
In practice CPM is a powerful mechanism to limit the number of active columns. Obviously, it is desirable to choose the number of allowed columns in the active set as low as possible. However, when this number is set too low, patterns are frequently swapped in and out of the pool. When a column re-enters the active set via a lookup operation, the pricing problem is skipped and hence no new columns are generated. This reduces the amount of fresh information introduced to the master problem. As a consequence, when too many lookups are performed, the convergence of the column generation algorithm stalls. To counter this swapping behavior, the CPM keeps track of the number of successful lookups in a given time window. When this number exceeds a threshold, the number of allowed active columns is increased.

2.5 Branch and Price

Since the integrality constraints of the MP are relaxed in the LPM, it is possible that the optimal solution to the LPM is fractional. A branch-and-price framework is needed to obtain integer solutions. In this section, the necessary branching rules are described, as well as bounds used to prune parts of the branch tree.

2.5.1 Branching rules

Our branching strategy is based on the following lemma:

Lemma 1. *Suppose that $z = \{z_1, z_2, ..., z_k\}$ is a feasible solution to LPM (Section 2.3), is fractional, and has cost $c(z)$. Then, either an integral solution exists with cost $c(z)$, or there exists an edge $e \in V \times V$ such that:*

$$0 < \sum_{p \in P : e \in p} z_p < 1$$

Proof. We will show that if $\sum_{p \in P: e \in p} z_p \in \{0, 1\}$ for each edge $e \in V \times V$, then an integral solution with cost $c(z)$ exists, thereby proving the lemma. Recall that a bus schedule $p \in P$ specifies a set of stops, as well as, for each of these stops, a set of students. Consider two schedules i and j that have a positive value in our current solution (i.e., $0 < z_i, z_j < 1$), and that both have an edge in common (such a pair of schedules must exist, otherwise $\sum_{p \in P: e \in p} z_p \in \{0, 1\}$ cannot hold for each edge e).
Consider the set of stops in schedule i (say stop set S_i) and the set of stops in schedule j (say stop-set S_j). Suppose S_i and S_j do not coincide, i.e., suppose $S_i \neq S_j$. Then stops $t, u, v, w \in V$ exist s.t. (t, u) is a route segment of both schedule i, j, (u, v) is a route segment of schedule i, and (u, w) is a route segment of schedule j. Recall that we assumed that for each edge $e \in V \times V$, $\sum_{p \in P: e \in p} z_p \in \{0, 1\}$ holds. Hence, for each route segment $e \in \{(t, u), (u, v), (u, w)\}$ we must have that $\sum_{p \in P: e \in p} z_p = 1$. That however is in violation with constraint (2.3) for node $u \in V$. It follows that the set of stops visited by schedule i and schedule j coincide. This implies that the fractional solution $z = \{z_1, z_2, ..., z_k\}$ consists of schedules whose stop-sets are either identical or disjoint. In other words, we can identify a partition of $\{1, \ldots, k\}$ into α subsets such that subset $K_j \subseteq \{1, \ldots, k\}$ contains schedules with an identical stop-set, $j = 1, \ldots, \alpha$. Now, define for each $s \in S, j = 1, \ldots, \alpha$ the *presence* of student s in subset K_j as:

$$\sum_{\substack{l \in K_j \\ l \text{ contains } s}} z_l = \beta_{js}$$

Since z is a feasible solution to LPM, and in particular satisfies Constraint (2.2), we know that the β_{js} satisfy:

$$\sum_{j=1}^{\alpha} \beta_{js} = 1 \qquad \forall s \in S \tag{2.49}$$

$$\sum_{s \in S} \beta_{js} \leq Q \qquad \forall j = 1, \ldots, \alpha \tag{2.50}$$

Due to integrality of Q, it is well-known that an integral solution to this system must exist, thereby ensuring that a student is either present completely in some subset K_j, or not at all. The existence of an integral solution z follows with cost $c(z)$. □

In the following we describe the two branching rules that we use. Branching rules are used to cut off the current fractional solution, and to partition the remaining solution space Z_u into two subsets. After a finite number of branchings, either

an optimal integral solution is obtained, or Z_u is exhausted, proving that no feasible integral solution exists. Notice that the above lemma guarantees that the presence of a fractional solution either gives us a way to find an integral solution, or we can identify a fractional edge on which we can branch. The latter observation is used for the second branching rule described below.

Given a fractional solution $z = \{z_1, z_2, ..., z_k\}$, where z_p, $p = 1, \ldots, k$ is the variable associated with bus schedule $p \in P$. If $b = \sum_{p \in P} z_p$ is fractional, we can *branch on the number of buses* b, thereby creating two branches: one where the number of buses used is at least $\lceil b \rceil$ and one with at most $\lfloor b \rfloor$ buses. Implementing this rule is straightforward as it simply amounts to updating U and L. Furthermore, all columns generated at the parent node can be reused at the child nodes.

The first branching rule is not sufficient to guarantee an integral solution. Therefore, as a second branching rule, we *branch on an edge*: two stops are visited consecutively in a single route or not. Enforcing that stop $v_i \in V$ is not reachable from $v_j \in V$ and vice versa is achieved by removing the edge (v_i, v_j) from the underlying graph. Ensuring that the two stops occur consecutively in a route requires some more effort. To accomplish this, we choose to modify the pricing problems. In case of the heuristic pricing method, the branching rules are easily accommodated for in the neighborhood definitions. Similarly, in the labeling algorithm, a route cannot be extended if the resulting route violates the branching constraints.

Strictly speaking, the second branching rule renders the first rule redundant. However, computational experiments revealed that the use of the first branching rule has a positive effect on the size of the tree.

In contrast to branching on the number of buses, when branching on an edge, it is not possible to reuse all columns from the parent node. For each column generated in the parent node, one needs to check whether the column is in accordance with the newly created branching rule. Depending on the branching rule, the column can either be modified to meet the new branching constraints, or has to be removed altogether.

In our implementation, the branching rules are always employed in the order of description. When the branching framework branches on an edge, the most uncertain edge is selected, i.e., $\underset{e \in E}{\operatorname{argmin}}(|\sum_{p:e \in p} z_p - 0.5|)$. In case of ties, the edge that occurs in most patterns is used. For alternative, more sophisticated approaches to select a branching candidate, e.g., strong branching or branch decisions based on pseudo-costs, we refer the interested reader to the works of Martin (1999).

A final aspect of interest is the order in which branches are investigated. The branch-and-price tree is explored in a depth-first-search (DFS) manner, always starting with the most constrained branch, e.g., when branching on the number of buses, the tightest bound is expanded first. In case of the remaining two

branching rules, the algorithm always starts by investigating the branch which enforces the use of an edge (vertex). The choice for DFS is based on the observation by Martin (1999) that feasible solutions tend to lie deep in the branch tree. Furthermore, DFS based search strategies tend to use significantly less memory than alternative approaches such as for example breadth-first-search.

2.5.2 Pattern initialization

At each node of the branch-and-price tree, the master problem has to be initialized with a feasible set of columns. Alternatively, when such a set does not exist, one needs to prove infeasibility of the master problem. The root node can be straightforwardly initialized by any feasible solution to the SBRP, but finding an initial feasible solution for any of its siblings tends to be difficult. Moreover, proving that such a solution does not exist is a cumbersome task. Hence it would be beneficial when the master problem itself could be used to prove nonexistence of a feasible solution. With this goal in mind, artificial columns, which together meet all the constraints of the master problem (Constraints (2.3)-(2.6)), are introduced. Each artificial column has a cost strictly larger than the cost of the longest feasible route; hence, the master problem favors cheaper non-artificial columns over the artificial ones. When the column generation procedure terminates, and the resulting solution still contains artificial columns, one can indeed conclude that no feasible solution exists.
To generate the artificial columns, first L stops are selected which are used to generate simple school-stop-school paths (recall that L is the lower bound on the required number of buses). Next, each student is assigned to one of those paths, independent of whether the student can reach the path, as the latter is not imposed by any of the constraints in the master problem. Similarly, the bus capacity requirements can be ignored. Finally, a large cost is assigned to the artificial patterns. In our implementation, it usually only takes a few iterations before the artificial columns leave the basis of the RMP.

2.5.3 Bounds

The efficiency of a branch-and-price framework is determined by two factors: the size of the tree and the processing time required by individual nodes. As discussed in Section 2.4.2, the processing time of a node is strongly affected by the tailing-off effect, which, in a Branch and Price framework, could occur at every node. Consequently, to speed up the processing times and to reduce the size of the search tree, bounds are used. Let Z_{MP} denote the value of

the optimal value of the master problem (Equations (2.1)-(2.6)) and UB an upper bound on Z_{MP}. Further, for some node u in the tree, let Z^u_{LMP} denote the optimal solution of the *relaxed* master problem at node u, $LB1^u$ a lower bound on Z^u_{LMP}, and Z^u_{RMP} the solution to the *restricted* master problem at node u at any given iteration. By definition, the following relations must hold: $UB \geq Z_{MP}$ and $Z^u_{RMP} \geq Z^u_{LMP} \geq LB1^u$. Computations at a node u can be terminated as soon as the gap between the lower bound at node u and the upper bound is closed, i.e., $LB1^u = UB$, independent of whether there still exist columns with negative reduced cost. Additionally, a node is pruned when its lower bound exceeds the upper bound UB. Consequently, the availability of strong bounds has a significant impact on the efficiency of our framework.
Any feasible integral solution to the master problem discovered at some node u is an acceptable upper bound on Z_{MP}. For SBRP, an upper bound UB is readily available at the root node because it is initialized with a feasible solution. In a branch-and-price tree, the root node is by definition the least constrained; each subsequent branching will add extra constraints to the model. Hence, the optimal objective value of a parent node is always lower or equal to the objective of any of its siblings (in case of minimization problems) and hence the optimal objective value of a node serves as a lower bound to its siblings. The lower bound inherited from a parent serves as an initial lower bound for its siblings. This lower bound can be tightened during the processing of the node using the following equation (Lasdon (1970)):

$$LB1^u = Z^u_{RMP} + Uq^* \tag{2.51}$$

Here, q^* is the optimal solution to the pricing problem as defined in Section 2.4 and U the upper bound on the number of buses (Table 2.1).

A second, stronger lower bound $LB2^u$ can be deduced via Langrangian Relaxation as demonstrated by Vanderbeck (1994). Given the original master problem LPM as presented in Section 2.4, we can relax complicating constraints (2.2), and (2.3) using the Lagrangian multipliers u_s and w_v respectively, and

drop the integrality constraints. The resulting problem becomes:

$$Z^u_{LMP} = \quad min \sum_{p\in P} \delta_p z_p + \sum_{s\in S} u_s(1 - \sum_{p\in P} t_{sp} z_p) - \sum_{v\in V} w_v(1 - \sum_{p\in P} r_{vp} z_p) \tag{2.52}$$

$$\text{s.t. } L \leq \sum_{p\in P} z_p \leq U$$

$$= \quad \sum_{s\in S} u_s - \sum_{v\in V} w_v + min \sum_{p\in P} [\delta_p - \sum_{s\in S} t_{sp} u_s + \sum_{v\in V} r_{vp} w_v] z_p \tag{2.53}$$

$$\text{s.t. } L \leq \sum_{p\in P} z_p \leq U$$

$$\geq \quad Z^u_{RMP} + Um - Ln + min \sum_{p\in P} [q_p - m + n] z_p \tag{2.54}$$

$$\text{s.t. } L \leq \sum_{p\in P} z_p \leq U$$

$$\geq \quad Z^u_{RMP} + Um - Ln + min\{L(q^* - m + n), U(q^* - m + n)\} \tag{2.55}$$

$$= \quad Z^u_{RMP} + Uq^* + (U - L)n \equiv LB2^u \tag{2.56}$$

Equation (2.54) follows from the Weak Duality Theorem and the fact that optimal Langrangian multipliers equal the dual variables of the Dual problem (Wolsey and Nemhauser, 1988).
When compared, the second lower bound $LB2^u$ (Equation (2.56)) is stronger than the first bound $LB1^u$ because $U - L \geq 0$ and $n \geq 0$.

For both bounds, it holds that the stronger the bounds L, U on the number of buses are, the tighter the lower bound will be. The latter also motivates the use of a branching rule that branches on the number of buses. When solving LPM (Section 2.4), Z^u_{RMP} decreases monotonically towards its optimal value Z^u_{LMP}. The lower bound $LB2^u$ also converges towards Z^u_{LMP} but not monotonically as $q*$ does not change monotonically at each successive iteration. As a consequence, the computed lower bound at the latest iteration might be lower (less tight) than a lower bound computed at an earlier iteration. It is therefore important to store the tightest lower bound.
A clear downside to calculating the lower bounds is the need for the optimal value of the pricing problem q^*. As elaborated in Section 2.4.1, the pricing problem is preferably solved by a fast (local search) heuristic, and not by the more expensive exact algorithm. The latter is worsened by the fact that one

might need to compute the lower bound at several iterations to obtain a tight bound on Z^u_{LMP}. Vanderbeck (1994) points out that instead of the optimal value of the pricing problem, also a lower bound could be used, but this usually leads to weaker bounds.

2.5.4 Branch and Price Implementation

This section briefly discusses the course of the branch-and-price algorithm.
At each node of the branch-and-price tree, a master problem is solved. The lower bound on the node is updated whenever the pricing problem is solved to optimality, which we enforce at least once every 100 iterations (Section 2.4.2). A node is pruned whenever the lower bound exceeds the upper bound. Once the node's master problem is solved to optimality, a branching decision is made. Three possible scenarios exist:

1. The solution is fractional (and feasible). A branch is created.
2. The solution is an integral solution. The upper bound on the global optimum is updated when possible.
3. The solution contains an artificial column. The solution is infeasible and the node is pruned.

As discussed in Section 2.4.2, the occurrence of the tailing-off effect has a large impact on the overall convergence speed, especially if this effect occurs at every node of the branch-and-price tree. In an attempt to counter this behavior, tailing-off is detected by measuring the relative difference (improvement) in the objective value of the master problem over a number of iterations. If the measured difference is less than a constant (0.1) during 20 iterations, we conclude that tailing-off occurs. Then a lower bound on the solution is computed and a branching decision is made. Note however that if the solution is integral or contains an artificial column, we cannot straightforwardly terminate computations. In such a case we store the state of the algorithm, including all the generated columns. Solving the node to optimality is then postponed until all remaining nodes have been evaluated. The advantage of this approach is that when a new (better) integral solution is discovered in the meantime, it might be possible to prune the node based on the improved bounds, which potentially saves time.
Finally, when a new incumbent optimal solution is discovered, all nodes are pruned, and a new branch-and-price tree is grown starting from the latter solution and the columns already generated for this node. The philosophy behind this pruning decision is again based on the assumption that a faster

converge of a branch-and-price algorithm is achieved with better initial solutions. In particular, this approach works well when the regular bound-based pruning is ineffective due to a relatively large gap between the optimal solutions to the MP and LPM respectively.

2.6 Computational Experiments

One aspect of instances of SBRP is that the number of students is typically much larger than the number of stops. This feature is not present in instances of the MV-TPP that have been used for computational testing (see Riera-Ledesma and Salazar-González (2012)); therefore, we used instances from Schittekat et al. (2013), and we generated new instances to focus on this property.
Computational experiments are conducted on two data sets. The first data set (Data Set I) contains the instances 1-95 from the data set used in Schittekat et al. (2013) [1] which have previously been used by Schittekat et al. (2013). The easiest instance has 5 stops and 25 students, whereas the hardest instances have 40 stops and 800 students. Each instance describes the distances between the stops, the maximum bus capacity and the maximum walking distance. The set of reachable stops for each student follows from its location and the allowed walking distance; the latter distance controls the average number of stops a student is able to reach. Routing occurs in the Euclidean plane. The second, newly generated data set (Data Set II) contains 34 instances. The problem sizes in this class range from 20-35 stops, and up to a maximum of 450 students. These instances are generated randomly as follows. For a given number of stops, a complete graph is created with distances on the edges uniformly selected from the interval $[1, 21]$. Similarly, the stops a student can reach are randomly selected such that the average number of reachable stops equals a predefined ratio. Notice that the distances in this class may violate the triangle inequality. The exact instance characteristics for both sets are summarized in Tables 2.2 (page 43), 2.3 (page 46): the instance ID (*ID*), the number of stops (*stop*), number of students (*stud*), bus capacity (*cap*) and the average number of stops in the vicinity of a student (*v/s*), i.e., $\frac{\sum_s |V_s|}{|S|}$.
Per instance, we performed a number of experiments, whose results are reported in Tables 2.2, 2.3 and are compared against the results published in Schittekat et al. (2013):

- LB_{old}: The best bound published in Schittekat et al. (2013).

[1] The instances are available online at http://antor.ua.ac.be/schoolbus-routing

- LB_{new}: The lower bound on an optimal solution obtained while solving the branch and price tree. Bold face entries indicate an improvement compared to LB_{old}.
- MH: The best solution obtained by the metaheuristic as published in Schittekat et al. (2013).
- $B\&P$: The branch-and-price solution. This column reports the best integer solutions which have been found within a time limit of 1 hour (3600000ms). Bold face entries indicate an improvement compared to MH.
- *gap*: The gap between $B\&P$ and LB_{new}, computed as: $100\frac{B\&P - LB_{new}}{B\&P}$.
- *nodes*: Number of nodes explored in the branch and price tree.
- *t(ms)*: Total computation time in ms.
- $t_{pricing}$*(ms)*: Amount of time spent solving pricing problems in ms.

For the majority of instances, our branch-and-price algorithm is able to find integer solutions, many of which have been proven to be optimal. Some of the largest instances we solved to optimality contain 40 stops and 800 students. Typically, the bounds computed are very strong, i.e., the average gap is less than 1%. These bounds are much stronger than the bounds obtained by solving the LP relaxation of the SBRP MIP model reported in (Schittekat et al., 2013). Instances with a high v/s ratio are usually solved faster compared to instances with a low v/s ratio. Solutions for these instances often exhibit routes with a small number of stops, resulting in faster pricing problems. Furthermore, the bounds for these instances are usually very strong. While comparing the effects of the number of stops and students in the instances, we noticed that instances with a large number of stops (and limited number of students) are significantly harder to solve than instances with a limited number of stops and a larger number of students; e.g., doubling the number of stops has much more impact on the runtime than doubling the number of students.
When comparing to Schittekat et al. (2013), 26 of the instances had their bounds improved, and better solutions were discovered for 9 instances. In addition, note that in Schittekat et al. (2013), the exact methods to compute bounds and optimal integer solutions were allotted a runtime of 2 hours, whereas in this work the runtime is limited to one hour. Using a traditional MIP model in Schittekat et al. (2013), 43 instances were solved to optimality. When comparing their heuristic solutions to the lower bound they computed, 3 more instances were solved to optimality, resulting in a total of 46 instances. Using our branch and price approach, a total of 68 instances are solved to optimality.
Although no solutions for Data Set II (Table 2.2) were published in Schittekat

et al. (2013), we were able to use their code to obtain heuristic solutions for these instances. Using the branch and price approach, better solutions were discovered for 9 instances. Optimality is attested for 25 of the instances. We again observed that instances with shorter routes are generally solved faster. In particular, if we reduce the vehicle capacity, we automatically obtain shorter routes and, hence, better run times. The latter is indeed confirmed when we compare the instances with vehicle capacities of 15 against instances with larger vehicle capacities.

In summary, the exact branch and price approach presented in this chapter is capable of producing strong lower and upper bounds for several SBRP instances. In fact, a significant number of instances previously published in literature had their bounds improved. When time is of the essence, and lower bounds are not necessarily of interest, e.g. in real-time scheduling applications, the metaheuristic presented in Schittekat et al. (2013) should however be preferred over the exact branch and price approach. Even with a limited amount of computational time, the metaheuristic is capable of obtaining strong, feasible solutions. Furthermore, for instances of over 40 stops, the branch and price approach was unable to improve upon the solutions returned by the metaheuristic.

As described in Section 2.4.2, degeneracy in column generation significantly affects the time required to solve a branch-and-price node. To test the effects of the stabilization mechanisms, we solved the root nodes of instances 11-51 (Data Set I) to optimality, while measuring the required number of iterations as well as computation times. Figure 2.1 (page 48) compares the different stabilization approaches: IPS stabilization with resp. 20 and 50 extreme points (IPS20, IPS50), Du Merle's proximal type stabilization (PTS), or no stabilization at all (No Stab). Only the PTS approach was capable of solving all instances within a time limit of 1 hour. Both PTS and IPS reduced the number of iterations required to solve the master problem. IPS20 needed 46% of the number of iterations required when no stabilization was used, while this number was 43% for IPS50, and 45% for PTS. However, looking from a time perspective, IPS20 required on average 314% of the computation time required when no stabilization was used, IPS50 needed 710%, whereas only 22% was required for PTS. Although IPS needs significantly more time to solve the instances, we must note that instance 50 could only be solved if either IPS or PTS was used. Concluding, IPS and PTS both reduce the number of iterations required, but, from a time perspective, PTS significantly outperforms IPS.

Table 2.2: Computational results Data Set I

ID	stop	stud	cap	v/s	LB_{old}	LB_{new}	MH	B&P	gap	nodes	$t(ms)$	$t_{pricing}(ms)$
1	5	25	25	1	141.01	141.01	141.01	141.01	0	1	1778	136
2	5	25	50	1	161.62	161.62	161.62	161.62	0	2	104	47
3	5	25	25	1.52	182.14	182.14	182.14	182.14	0	3	195	106
4	5	25	50	1	195.8	195.8	195.80	195.8	0	4	86	34
5	5	25	25	1.88	111.65	111.65	111.65	111.65	0	5	156	72
6	5	25	50	2.4	103.18	103.18	103.18	103.18	0	6	97	38
7	5	25	25	4.88	7.63	7.63	7.63	7.63	0	7	41	5
8	5	25	50	4.56	25.64	25.64	25.64	25.64	0	8	88	27
9	5	50	25	1	281.49	**286.68**	286.68	286.68	0	11	183	26
10	5	50	50	1.4	197.2	197.2	197.20	197.2	0	12	99	28
11	5	50	25	1.46	181.02	**193.55**	193.55	193.55	0	29	3035	493
12	5	50	50	1.4	215.86	215.86	215.86	215.86	0	30	72	21
13	5	50	25	1.96	130.53	130.53	130.53	130.53	0	31	245	61
14	5	50	50	2.74	96.26	96.26	96.26	96.26	0	32	110	40
15	5	50	25	4.8	12.89	12.89	12.89	12.89	0	33	260	65
16	5	50	50	4.42	30.24	30.24	30.24	30.24	0	34	145	52
17	5	100	25	1	360.35	360.35	360.35	360.35	0	35	94	1
18	5	100	50	1	290.67	**304.23**	304.23	304.23	0	38	111	12
19	5	100	25	1.55	255.93	**294.21**	294.21	294.21	0	41	1869	151
20	5	100	50	1.37	229.41	229.41	229.41	229.41	0	43	1627	226
21	5	100	25	2.8	134.95	134.95	134.95	134.95	0	44	535	84
22	5	100	50	1.84	139.87	**144.41**	144.41	144.41	0	47	12974	958
23	5	100	25	4.77	58.95	58.95	58.95	58.95	0	48	1388	215
24	5	100	50	4.58	39.44	39.44	39.44	39.44	0	49	791	287
25	10	50	25	1.22	242.85	242.85	242.85	242.85	0	50	471	297
26	10	50	50	1.2	282.12	282.12	282.12	282.12	0	51	323	178
27	10	50	25	1.6	244.54	244.54	244.54	244.54	0	52	809	450
28	10	50	50	1.26	288.33	288.33	288.33	288.33	0	53	632	444
29	10	50	25	3.96	108.95	108.98	108.98	108.98	0	57	5680	2279
30	10	50	50	2.86	157.48	157.48	157.48	157.48	0	58	407	310
31	10	50	25	9.2	32.25	32.25	32.25	32.25	0	59	430	292
32	10	50	50	8.96	36.66	36.66	36.66	36.66	0	60	260	183

Table 2.2: Computational results Data Set I

ID	stop	stud	cap	v/s	LB_{old}	LB_{new}	MH	B&P	gap	nodes	$t(ms)$	$t_{pricing}(ms)$
33	10	100	25	1.18	400.54	**403.18**	403.18	403.18	0	65	834	246
34	10	100	50	1.29	294.11	**296.53**	296.53	296.53	0	68	5398	2244
35	10	100	25	1.25	369.62	**388.87**	388.87	388.87	0	190	20578	5704
36	10	100	50	1.33	294.8	294.8	294.80	294.8	0	192	3030	1640
37	10	100	25	3.98	178.28	178.28	178.28	178.28	0	194	29681	5027
38	10	100	50	3.26	175.41	**175.96**	175.96	175.96	0	199	131052	30255
39	10	100	25	9.18	57.5	57.5	57.50	57.5	0	200	1819	1069
40	10	100	50	9.22	31.89	31.89	31.89	31.89	0	201	1522	941
41	10	200	25	1	735.27	735.27	735.27	735.27	0	202	21	1
42	10	200	50	1	506.06	506.06	512.16	**506.06**	0	204	73	14
43	10	200	25	1.85	463.78	**513**	513.00	513	0	261	124693	9716
44	10	200	50	1.28	458.17	**475.21**	475.21	475.21	0	371	69692	10891
45	10	200	25	3.53	331.49	**347.29**	347.29	347.29	0	374	30215	4061
46	10	200	50	3.66	194.66	194.66	217.46	217.46	11.71	136	3600000	155777
47	10	200	25	9.22	102.93	102.93	102.93	102.93	0	137	5890	2262
48	10	200	50	8.93	55.05	55.05	55.05	55.05	0	138	8201	3694
49	20	100	25	1.15	507.81	507.81	520.24	**507.81**	0	140	6002	5544
50	20	100	50	1.17	406.65	406.65	420.64	**406.65**	0	142	2920371	2918926
51	20	100	25	2.08	404.78	**419.17**	422.21	**419.17**	0	211	708642	478578
52	20	100	50	1.73	356.52	356.52	360.86	360.86	1.22	456	3600000	3560533
53	20	100	25	5.09	245.17	245.17	245.17	245.17	0	458	71459	37360
54	20	100	50	6.13	181.3	181.3	185.06	185.06	2.08	462	3600000	848080
55	20	100	25	17.36	52.52	52.52	52.52	52.52	0	463	16494	13825
56	20	100	50	18.46	19.05	19.05	19.05	19.05	0	464	2666	2334
57	20	200	25	1.2	851.98	**875.46**	903.84	**875.46**	0	734	1564365	99575
58	20	200	50	1.15	473.89	**476.05**	485.65	**476.05**	0	766	472158	317436
59	20	200	25	1.76	589.89	**597.99**	616.93	**606.8**	1.47	615	3600000	577259
60	20	200	50	1.78	451.09	451.09	462.31	462.31	2.49	809	3600000	819042
61	20	200	25	5.38	366.1	366.1	373.21	373.21	1.94	1062	3600000	437484
62	20	200	50	5.53	246.49	246.49	250.75	250.75	1.73	1389	3600000	625320
63	20	200	25	18	93.01	93.01	93.01	93.01	0	1391	62586	42405
64	20	200	50	17.97	45.4	45.4	45.40	45.4	0	1392	28623	18384

Table 2.2: Computational results Data Set I

ID	stop	stud	cap	v/s	LB_{old}	LB_{new}	MH	B&P	gap	nodes	$t(ms)$	$t_{pricing}(ms)$
65	20	400	25	1.46	1247.65	**1323.35**	1323.35	1323.35	0	1535	1446744	76934
66	20	400	50	1.22	709.87	**720.83**	733.54	**720.83**	0	1663	1058957	165084
67	20	400	25	2.48	911.06	911.06	975.12	975.12	7.03	1248	3600000	134663
68	20	400	50	1.89	599.12	599.12	614.67	614.67	2.6	1867	3600000	260149
69	20	400	25	4.58	756.04	**763.76**	763.76	763.76	0	1903	3111861	136752
70	20	400	50	7.45	298.05	298.05	298.47	298.47	0.14	1915	3600000	384688
71	20	400	25	18.15	239.58	239.58	239.58	239.58	0	1916	92356	41734
72	20	400	50	18.13	84.49	84.49	84.49	84.49	0	1917	190202	62401
73	40	200	25	1.4	787.14	787.14	831.94	831.94	5.69	1936	3600000	3527195
74	40	200	50	1.38		**549.64**	593.35	593.35	7.95	1463	3600000	3574732
75	40	200	25	2.57	696.04	696.04	728.44	728.44	4.65	1583	3600000	921613
76	40	200	50	2.7		**474.14**	481.05	481.05	1.46	1584	3600000	3543283
77	40	200	25	9.57	328.19	328.19	339.75	339.75	3.52	59	3600000	2415227
78	40	200	50	10.72	273.05	273.05	273.88	273.88	0.31	60	3600000	3568533
79	40	200	25	36.42	76.77	76.77	76.77	76.77	0	2028	137932	129914
80	40	200	50	36.3	58.46	58.46	58.46	58.46	0	2029	330240	322539
81	40	400	25	1.51	1307.52	1307.52	1407.05	**1394.23**	6.63	2289	3600000	605095
82	40	400	50	1.3		**820.52**	858.80	858.8	4.67	204	3600000	3541207
83	40	400	25	3.23	869.38	869.38	891.02	891.02	2.49	2341	3600000	671259
84	40	400	50	2.57		**721.75**	757.42	757.42	4.94	247	3600000	779834
85	40	400	25	10.02	575.66	575.66	586.29	586.29	1.85	294	3600000	789361
86	40	400	50	9.88		**392.06**	395.95	395.95	0.99	2400	3600000	2033063
87	40	400	25	35.74	195.33	195.33	195.33	195.33	0	2401	333985	304871
88	40	400	50	36.59	70.77	70.77	70.77	70.77	0	2402	449242	415835
89	40	800	25	1.33	2801.05	2801.05	2900.14	2900.14	3.54	166	3600000	467658
90	40	800	50	1.54	1280.51	1280.51	1345.70	1345.7	5.09	258	3600000	704129
91	40	800	25	2.71	2153.76	2153.76	2200.57	2200.57	2.17	286	3600000	494468
92	40	800	50	3.08	978.88	**978.9**	1025.16	1025.16	4.72	179	3600000	431057
93	40	800	25	9.63	1404.16	1404.16	1404.16	1404.16	0	303	542641	204234
94	40	800	50	10.92	613.72	613.72	616.58	616.58	0.47	1	3600000	3467511
95	40	800	25	36.11		**396.92**	396.92	396.92	0	305	790502	701400

Table 2.3: Computational results Data Set II

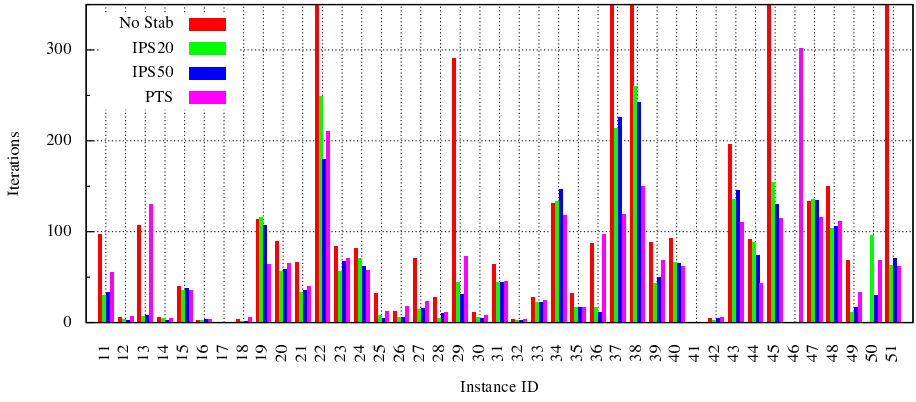
ID	stop	stud	cap	v/s	LB_{new}	MH	B&P	gap	nodes	$t(ms)$	$t_{pricing}(ms)$
1	20	100	15	4.02	73	73	73	0	520	17306	13925
2	20	100	20	4.02	60	60	60	0	33	315377	111698
3	20	100	25	4.02	55	55	55	0	42	84882	46638
4	20	100	15	7.72	71	71	71	0	522	13248	10472
5	20	100	20	7.72	49	49	49	0	35	34049	23610
6	20	100	25	7.72	42	42	42	0	44	33028	25316
7	20	200	15	4.16	238	238	238	0	523	3758	2133
8	20	200	25	4.16	102	104	102	0	47	969466	226204
9	20	200	15	8.52	224	224	224	0	1	11433	4487
10	20	200	25	8.52	86	86	86	0	48	15689	10673
11	25	150	15	4.13	115.2	120	120	4.17	464	3600000	974827
12	25	150	25	4.13	72.47	81	**74**	2.12	203	3600000	1113312
13	25	150	15	8.13	111	115	**111**	0	466	164844	148875
14	25	150	25	8.13	60	62	**60**	0	260	1838698	1140206
15	25	250	15	4.14	277	277	277	0	467	12768	7818
16	25	250	25	4.14	126.88	132	**131**	3.25	304	3600000	1276481
17	25	250	15	8.41	275	275	275	0	468	23223	18311
18	25	250	25	8.41	111	120	**111**	0	326	1068039	775015
19	30	300	15	3.99	326	326	326	0	469	55441	38709
20	30	300	25	3.99	151.89	172	**156**	2.71	348	3600000	3226403
21	30	300	15	8.48	315	317	**315**	0	473	175157	130120
22	30	300	25	8.48	123	134	**123**	0	356	2278841	2012475
23	30	350	15	3.91	439	439	439	0	474	19185	12401
24	30	350	25	3.91	186.14	200	**188**	1	401	3600000	1797273
25	30	350	15	8.39	439	440	**439**	0	478	92573	71374
26	30	350	25	8.39	162	169	**162**	0	404	1154063	924603
27	35	400	15	3.92	521	521	521	0	479	49677	35519
28	35	400	25	3.92	233.07	244	**237**	1.69	429	3600000	2282498
29	35	400	15	8.63	498	498	498	0	480	67755	53351
30	35	400	25	8.63	206	217	**207**	0.49	451	3600000	3026446
31	35	450	15	3.93	613	613	613	0	481	33458	17080
32	35	450	25	3.93	277.85	290	**282**	1.49	509	3600000	639639

Table 2.3: Computational results Data Set II

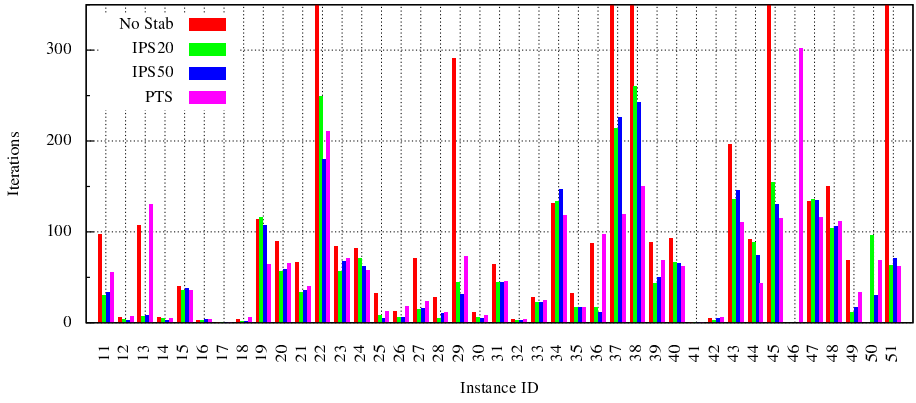
ID	stop	stud	cap	v/s	LB_{new}	MH	B&P	gap	nodes	$t(ms)$	$t_{pricing}(ms)$
33	35	450	15	8.63	608	608	608	0	482	78742	51914
34	35	450	25	8.63	256.14	260	**257**	0.33	516	3600000	3160984

2.7 Conclusion

In this chapter we studied the School Bus Routing Problem, a vehicle routing problem which encompasses selecting bus stops, assigning students to the selected stops, and finally determining the necessary bus routes, while minimizing the routing costs. We developed an exact algorithm for the SBRP based on a set covering formulation. This formulation enables us to solve a large number of the SBRP instances of the benchmark released by Schittekat et al. (2013) to optimality. For the remaining instances, lower bounds as well as some of the integer solutions have been improved. Also, when comparing to (Riera-Ledesma and Salazar-González, 2013), we observe that set-partitioning formulations solve instances with large number of students more efficiently than instances with a large number of stops. However, when the set covering formulation is strengthened with valid inequalities, the number of stops that can be handled increases significantly (Riera-Ledesma and Salazar-González (2013)). Thus, one possible extension is to incorporate the separation of valid inequalities.
In this chapter, we applied several techniques which have been reported to increase the performance of column generation, such as: exact and heuristic pricing algorithms, bounding procedures, a column pool manager, stabilization techniques, and a more rigid branching approach for the branch-and-price tree. Among these approaches, bounding procedures and stabilization cause the largest performance improvements. Two types of stabilization have been compared: interior point stabilization and Du Merle's proximal type stabilization. Although both reduce the number of required iterations considerably, the proximal type outperforms the interior point stabilization as the latter increased the computation time per iteration significantly.



(a) Iterations



(b) Time

Figure 2.1: Comparison of different stabilization approaches

Chapter 3

The Concrete Delivery Problem

Abstract

From an operational point of view, Ready-Mixed Concrete Suppliers are faced with challenging operational problems such as the acquisition of raw materials, scheduling of production facilities, and the transportation of concrete. This chapter is centered around the logistical and distributional part of the operation: the scheduling and routing of concrete, commonly known as the Concrete Delivery Problem (CDP). The problem aims at finding efficient routes for a fleet of (heterogeneous) vehicles, alternating between concrete production centers and construction sites, and adhering to strict scheduling and routing constraints. Thus far, a variety of CDPs and solution approaches have appeared in academic research. However, variations in problem definitions and the lack of publicly available benchmark data inhibit a mutual comparison of these approaches. Therefore, this work presents a more fundamental version of CDP, while preserving the main characteristics of the existing problem variations. Both exact and heuristic algorithms for CDP are proposed. The exact solution approaches include a Mixed Integer Programming (MIP) model and a Constraint Programming model. Similarly, two heuristics are studied: the first heuristic relies on an efficient best-fit scheduling procedure, whereas the second heuristic utilizes the MIP model to improve delivery schedules locally. Computational

The content of this chapter is based on joint work with T. Wauters and G. Vanden Berghe, see Kinable et al. (2014b).

experiments are conducted on new, publicly-accessible, data sets; results are compared against lower bounds on the optimal solutions.

3.1 Introduction

The Concrete Delivery Problem (CDP), a combination of a scheduling and a routing problem, encompasses the distribution of concrete to a number of construction sites. Each construction site $i \in C$, also referred to as customer, requests q_i tons of concrete. The concrete is transported by a non-homogeneous set of trucks K, each capable of delivering l_k, $k \in K$, tons of concrete. A time window $[a_i, b_i]$ is associated with each construction site $i \in C$, denoting the time interval during which the concrete can be delivered. Deliveries cannot commence before a_i and should be completed before b_i. Due to the hard time windows and capacity limitations of the trucks, it is often impossible to satisfy the demand of all customers (oversubscribed scheduling problem).
The amount of concrete requested by a single customer typically exceeds the capacity of a single truck. Hence, multiple deliveries may be required. Deliveries for the same customer cannot overlap in time, and have to take a maximum time lag γ into consideration; the time between two consecutive deliveries is limited to guarantee proper bonding between the two layers of concrete. Deliveries may not be preempted or split amongst multiple customers. The time required to perform a delivery is truck dependent, and will be denoted by p_k, $k \in K$.
The concrete is produced at several homogeneous production sites (P) which are located some distance away from the customers. The trucks start at a central depot and have to travel to a production plant, after which they have to unload their cargo at one of the construction sites. Loading the concrete and traveling between a pickup and delivery point $i \in P, j \in C$ takes a certain amount of time t_{ij}. Trucks are always filled to their maximum capacity. Whenever the total amount of concrete delivered at a construction site exceeds the customer's demand, the excessive amount becomes waste. A summary of the notation used throughout the paper is provided in Table 3.1.
The objective of the problem is to maximize the number of satisfied customers, weighted by their demand. A customer $i \in C$ is satisfied if at least q_i tons of concrete have been delivered. Since a concrete distributor has only finite capacity, it is natural that some customers cannot be serviced. Rejected customers are supposed to relax their request or to revert to a different supplier. Concrete distributors also have the option to temporarily increase capacity by hiring a subcontractor. Although not considered in this work, it would be straightforward to include services by subcontractors at higher cost in the objective function.
The CDP bares a strong resemblance with two well-studied NP-hard problems:

the Capacitated Vehicle Routing Problem (CVRP) with Time Windows and Split Deliveries and the Parallel Machine Scheduling Problem (PMSP) with Time-Windows. Although efficient exact and heuristic algorithms have been presented in the literature for both problems individually, it is not evident how to generalize these methods to CDP. In related works (see Section 3.2 for details), dedicated approaches are published for several variations of the CDP. Each variation has its own unique constraints, including constraints dealing with different types of concrete, equipment requirements (pumps, vehicle types), and customer demands (origin of the concrete, delivery speed). Next to the fact that some authors conduct their experiments on confidential data, the variations in their problem descriptions render it notoriously difficult to mutually compare their algorithms. This work addresses a more fundamental version of the CDP, which captures the commonalities amongst the existing variations of the problem while dropping some of the more exotic constraints. Nevertheless, the exact and heuristic approaches presented in this work are easily modified to incorporate the omitted constraints. To facilitate a future comparison of CDP implementations, we publish a large number of problem instances.
The chapter is structured as follows. First, Section 3.2 provides a detailed overview of related research. Next, in Section 3.3, three mathematical models for CDP are presented. The first model, based on Mixed Integer Programming (MIP), is derived from models for the CVRP and the PMSP. The second model extends the first model by incorporating the requirement that a vehicle may perform multiple deliveries to the same customer. Finally, the third model, as an alternative to the second model, is based on Constraint Programming. Section 3.4 proposes two heuristic approaches for the CDP. The first heuristic utilizes an efficient constructive procedure, which schedules customers one-by-one, following a best-fit policy. A meta-heuristic controls the exact order the customers are considered for scheduling by the constructive procedure. The second heuristic relies on the MIP model presented in Section 3.3. Instead of solving the entire model at once, the heuristic repeatedly fixes one part of the schedule while re-optimizing another part. To compare the various methods discussed in this work, Section 3.5 outlines two approaches to compute bounds on the optimal solutions. Computational results are presented in Section 3.6. Finally, Section 3.7 offers the conclusion.

3.2 Related Research

A number of related works involving several variations of the CDP exist in the literature. To obtain insight in the diversity in problem specifications, as well as in the solution approaches, we provide an extensive overview of the most recent works in this area. Table 3.2 summarizes for each of these papers the

Parameter	Description
P	Set of concrete production sites
C	Set of construction sites, also denoted as customers. $\|C\| = n$
V	$V = C \cup \{0\} \cup \{n+1\}$
$0, n+1$	resp. the start and end depots of the trucks.
K	Set of trucks
q_i	Requested amount of concrete by customer $i \in C$
l_k	Capacity of truck $k \in K$
p_k	Time required to empty truck $k \in K$
a_i, b_i	Time window during which the concrete for customer i may be delivered.
t_{ij}	Time to travel from i to j, $i, j \in V \cup P$
γ	Maximum time lag between consecutive deliveries.

Table 3.1: Parameters defining the CDP

main problem characteristics and solution approaches. The abbreviations used in the table are explained in Table 3.8. Omitted values in Table 3.2 indicate that a certain property is not applicable, is unknown or is not considered in this chapter.

	Solution method	Time Windows	Start/End location	Production depots	Loading, unloading	Fleet	Instrumentation	Deliveries	Objectives
Hertz et al. (2012)	m	sv	c	he	d	he		v,l,sp	u,v$_s$,v$_c$
Misir et al. (2011)	he	hs,p	p	he,s	d	he	✓	s,r,v,l,sp	t,w,d
Silva et al. (2005)	he	hd,p	p	ho	d	ho		l,sp	u,o
Asbach et al. (2009)	m,he	hd,p	m	he	d	he	✓	s,r,v,l,sp,ss	u,s
Yan and Lai (2007)	m	sd,p	p		f	ho		s,r,l,sp	t,d,op
Lin et al. (2010)	m	hd,sv,p	p	ho	d	ho		s,r,l,sp	t,d,u,b,s
Naso et al. (2007)	m,he	sd,p	p	he,s	d	ho		s,r,l,sp,ss	u,o,d
Schmid et al. (2009)	hy,he	sd	p	ho	d	he	✓	s,r,l,sp	t,d
Schmid et al. (2010)	m,hy	sd	p	ho	d	he	✓	s,r,l,sp	t,d
This work	m,cp,he,hy	hd	c	ho	f	he		s,r,l,sp	s

Table 3.2: Summary of the various CDP problems and solution approaches in related literature. For notation, refer to Section 3.8 on page 80

.

Schmid et al. (2009) and Schmid et al. (2010) consider a concrete delivery problem with soft time windows. In contrast to our work, their objective is to satisfy all customers, while minimizing total tardiness. Both works also consider orders requiring vehicles with special equipment to be present during the entire delivery process, e.g., a pump or conveyor. Hybridized solution approaches combining MIP models and Variable Neighborhood Search heuristics are proposed to solve the resulting optimization problems.
Schmid et al. (2009) propose a hybrid heuristic for a CDP with soft time windows, and obligatory deliveries. The heuristic resembles a traditional Column Generation (CG, for details refer to e.g., Desaulniers et al. (2005)) procedure. The columns (delivery patterns), describe feasible orderings of deliveries to satisfy specific customers. More precisely, a pattern describes exactly which vehicles perform deliveries for a customer, and when the deliveries should commence. A MIP approach, comparable to the master problem (MP) in a CG, selects a delivery pattern for each customer in such a way that the selected patterns together form a feasible solution, thereby satisfying all routing and scheduling constraints. Next, in contrast to a traditional CG where a pricing problem is utilized to generate new columns for the MP, new columns are created via a Variable Neighborhood Search (VNS) procedure which is initialized with the solution produced by the MIP model. The VNS procedure perturbates and re-optimizes the MIP solution until a new feasible schedule is obtained which is different from the original MIP solution. The resulting schedule is decomposed into delivery patterns and added to the set of existing patterns in the MIP. Finally, the MIP model is resolved and the procedure repeats until a termination criterion is met.
An alternative procedure to Schmid et al. (2009) has been published by Schmid et al. (2010). A VNS heuristic, very similar to the one from Schmid et al. (2009), is deployed to generate an initial solution. Then, the solution is inserted into a MIP model. Instead of solving the entire MIP model at once, a large number of variables are fixed to the values of the initial solution; only a small portion of the variables remain unfixed. The exact variables to fix are determined by a Very Large Neighborhood Search procedure. The MIP model is solved repeatedly, having different decision variables fixed and unfixed, always starting from the best incumbent solution. When compared mutually, Schmid et al. (2010) performs slightly better than Schmid et al. (2009) on small and medium sized instances, ranging from 27 to 60 orders.

Hertz et al. (2012) discuss CDP with a strong emphasis on routing and vehicle constraints. In contrast to our work, scheduling and sequencing of the individual deliveries at the customers' side is not taken into account. Furthermore, no time windows on deliveries for a customer are imposed. Next to an exact MIP model, Hertz et al. (2012) propose a (heuristic) decomposition of their problem into two MIP subproblems: an assignment and a routing problem. First, the assignment

problem is solved, thereby assigning deliveries to vehicles such that all customers are satisfied. Next, for each vehicle, a routing problem is solved, determining the optimal route to perform the deliveries. To improve the solution quality, estimations of the travel costs incurred when assigning a specific delivery to a particular vehicle are added to the assignment problem.
Lin et al. (2010) present a MIP model, using a time-indexed notation. Some experiments are conducted and compared against man-made schedules from a Taiwanese RMC production company. No results with respect to computation times, bounds or comparative studies have been included; it is unclear whether the MIP model solved the problem instance(s) to optimality.
A generalization of the present work is published by Asbach et al. (2009). Next to the constraints discussed in Section 3.1, Asbach et al. (2009) consider vehicle synchronization at the loading depots, vehicles with specialized equipment, and additional time lags between deliveries for a single customer. In addition, Asbach et al. (2009) add constraints ensuring that some customers only receive concrete from a subset of production stations, and constraints limiting the time that concrete may reside in the vehicle's drum. Here we do not explicitly consider these constraints, as they can simply be incorporated in our model by modifying the arcs or costs in the underlying routing graph (Section 3.3). Asbach et al. (2009) formally define their problem using a MIP model, and establish that it is NP-Hard to solve by reduction to the Euclidean TSP problem. Next, a heuristic is proposed to solve the CDP because solving the MIP model is considered intractable. The heuristic, initialized with a feasible schedule, iteratively un-schedules and re-schedules one or two customers simultaneously. A (greedy) constructive procedure is used to reinsert the unscheduled customers into the schedule, in an attempt to obtain a better solution. This procedure is repeated until a local optimum is reached, or a pre-defined time limit is exceeded. The authors do not publish bounds on the optimal solutions, and the problem instances are inaccessible due to a non-disclosure agreement, rendering a fair comparison to this work virtually impossible.

While in this work we consider both the routing and scheduling aspects of the concrete production and distribution, Silva et al. (2005) focuses primarily on the scheduling problems at the concrete production sites. A single delivery takes a fixed amount of time which does not depend on the vehicle's location. Moreover, the fleet of vehicles is homogeneous, thereby simplifying the problem considerably as the exact number of deliveries required to meet the customer's demand is known beforehand. Silva et al. (2005) use a Genetic Algorithm (GA) to assign customer orders to concrete production centers. The orders are subdivided into a number of 'jobs'. Trucks located at the production sites are employed to deliver the concrete in a timely manner. Assigning deliveries to trucks is performed via an Ant Colony Optimization algorithm.
Naso et al. (2007) expand upon the work by Silva et al. (2005), adding a

significant number of additional hard and soft constraints to the problem. Contrary to our own work, deliveries are performed by a fleet of homogeneous vehicles. Furthermore, customers require an uninterrupted flow of concrete, meaning that the next truck must be available as soon as the previous truck finishes unloading. All orders must be fulfilled; unfulfilled orders are covered by external companies at a hire cost. Similar to Silva et al. (2005), Naso et al. (2007) employ a GA to assign orders originating from construction sites to nearby production centers. Each production center is equipped with a single loading dock, where the produced concrete is loaded into trucks. Since each dock can only handle a single truck at a time, sequencing of the trucks is required. This is performed via a constructive heuristic. Once a feasible loading sequence has been determined for each production center, another constructive heuristic is deployed to determine the schedules and routes for the available fleet of vehicles. This schedule must ensure that a vehicle is ready to transport the concrete to the customers' site as soon as the concrete for the delivery is available at the loading dock. If no vehicle can accommodate the delivery, an external vehicle is hired and a penalty is incurred. Computational experiments are performed on data from a Dutch concrete supplier. The performance of the heuristic is compared against four different scheduling policies utilized by human schedulers; no comparisons against Silva et al. (2005) are included in the paper.

Yan and Lai (2007) consider a heuristic approach to a CDP consisting of a single production station and a homogeneous fleet. Next to the travel time of vehicles, the authors include overtime of deliveries, and operation time of the production station and construction sites in their objective function. A MIP model is provided, modeled on a time-space network: nodes in the network encode both time and location information, whereas arcs represent translations in the time and space domains. A disadvantage of such a model is the fact that the continuous time domain is discretized in fixed intervals; taking the time intervals too small leads to an excessive number of nodes in the network, whereas large time intervals lead to inaccurate or suboptimal solutions. Solving the model using a MIP solver turned out intractable for their real-world problem instances. Hence, a heuristic decomposition approach is proposed instead. This approach first solves a relaxed version of the MIP model. The resulting solution is then used to simplify the original MIP model by fixing a number of the decision variables to their corresponding values in the relaxed solution. Finally, the simplified MIP model is solved. The authors claim that the resulting decomposition yields very good results when compared against a lower bound obtained by a MIP solver, or solutions produced by human schedulers. However, it must be note that their dataset is limited to just three instances; it is unclear how any related problems or approaches would fair upon such a decomposition.

Misir et al. (2011) propose a new hyper-heuristic to the CDP. The hyper-heuristic is initialized with both a feasible CDP solution and a set of heuristics. At each iteration, the hyper-heuristic selects a heuristic which is used to construct a new CDP solution based on the solution obtained during the previous iteration. The quality of the new solution is established, and the hyper-heuristic decides whether the new solution is accepted or rejected. The authors compare their work against four alternative hyper-heuristics. In Durbin and Hoffman (2008) a decision support system is presented which assists in the dispatching of trucks, creating delivery schedules, and determining whether new orders should be accepted. The system is designed in such a way that it can cope with uncertainty and changes in the schedule caused by order cancellations, equipment failures, traffic congestions. The scheduling and routing problems handled by the system are represented by time-space networks, and are solved through MIP.

In contrast to the previous, mainly Operations Research based approaches, Graham et al. (2006) treat CDP from a Machine Learning point of view. A neural network is used to predict the delivery rate of concrete at a production site, based on a set of known parameters such as the capacities of the trucks, their average inter arrival time, designation of the concrete, season, number of accepted and rejected deliveries, etc. Data from previous construction projects is used to train the model. Results from this approach may be used to improve the work flow, planning and resource utilization. As this work conceptually deviates from the other papers discussed in this section, we did not include it in Table 3.2.

3.3 Mathematical models

3.3.1 Integer Programming models

This section introduces two MIP models for CDP. The first one, CDP1, in section 3.3.1 shows how two well-known models for the Capacitated Vehicle Routing Problem (CVRP) with Time Windows and Split Deliveries, and the Parallel Machine Scheduling Problem (PMSP) with Time Windows and Maximum Time Lags can be combined to model the CDP. Although the resulting model supports split-deliveries, all deliveries for a single customer should be made by different vehicles; it is not possible that a single vehicle performs two or more deliveries to the same customer. A model (CDP2) which overcomes this limitation is presented in Section 3.3.1.

An Integer Programming Model for CDP

The concrete trucks start their trip at a central (source) depot and travel between production sites and customers. At the end of the day, the trucks return to an end (sink) depot (which may or may not be the same as the starting depot). This routing problem can be modeled on a directed, weighted graph $G(V, A)$, consisting of vertex set $V = \{0\} \cup C \cup \{n+1\}$, where vertices 0 and $n+1$ are resp. the source and sink depots. Vertices representing the production depots are omitted. Instead, the fact that a vehicle has to reload in between two deliveries has been accounted for in the arc costs. Optionally, a positive load time for the vehicles can be added to the arcs between two customers. The arc set A is defined as follows:

- the source, sink depots have outgoing resp. incoming edges to/from all other vertices.
- there is an arc (i, j) for all $i, j \in C, i \neq j$.

The arc costs are as follows:

- $c_{0,i} = \min_{p \in P} t_{0,p} + t_{p,i}$ for all $i \in C$
- $c_{i,j} = \min_{p \in P} t_{i,p} + t_{p,j}$ for all $i, j \in C$
- $c_{i,n+1} = t_{i,n+1}$
- $c_{0,n+1} = 0$

A solution to CDP consists of a selection of $|K|$ source-sink paths, that collectively satisfy the routing and scheduling constraints as outlined above.
The CDP is very similar to two well known combinatorial optimization problems: the CVRP with Time Windows and Split Deliveries, and the PMSP with Time Windows and Maximum Time Lags. However, there is a fundamental difference between the CDP and CVRP with split deliveries: in CVRP with split deliveries, the capacity of a single vehicle must be shared amongst all the customers visited by the vehicle, whereas this is not the case in CDP. Nevertheless, the model for a Capacitated Vehicle Routing Problem as presented by Desaulniers et al.

(2005) can be modified to meet the routing constraints of CPD as follows:

$$max \sum_{i \in C} q_i y_i \tag{3.1}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(i)} l_k x_{ijk} \geq q_i y_i \qquad \forall i \in C \tag{3.2}$$

$$\sum_{j \in \delta\{0\}} x_{0jk} = \sum_{i \in \delta\{n+1\}} x_{i,n+1,k} = 1 \qquad \forall k \in K \tag{3.3}$$

$$\sum_{j \in \delta^+(i)} x_{i,j,k} = \sum_{j \in \delta^-(i)} x_{j,i,k} \qquad \forall i \in C, k \in K \tag{3.4}$$

$$C_k^i + t_{ij} - M(1 - x_{ijk}) \leq C_k^j - p_k \qquad \forall i, j \in V, k \in K \tag{3.5}$$

$$a_i + p_k \leq C_k^i \leq b_i \qquad \forall i \in V, k \in K \tag{3.6}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in V, k \in K \tag{3.7}$$

$$C_k^i \in \mathbb{Z}_{\geq 0} \qquad \forall i \in V, k \in K \tag{3.8}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C \tag{3.9}$$

In this formulation, denoted as CDP-route, x_{ijk} is a binary variable indicating whether vehicle k travels from i to j. Integer variables C_k^i, $i \in C, k \in K$, record the time that vehicle k finishes its delivery to customer i. Binary variable y_i denotes whether customer $i \in C$ is serviced. For notation purposes, $\delta^-(\cdot)$ resp. $\delta^+(\cdot)$ denote the incoming resp. outgoing neighborhood sets. Constraint (3.2) ensures that sufficient concrete is delivered at construction site. Constraints (3.3)-(3.4) determine the shape of a feasible tour: a tour starts at the source depot, visits a number of pickup and delivery points and finally returns to the sink depot. Constraints (3.4) are the flow preservation constraints. A customer $i \in C$ cannot be visited before a_i, and deliveries have to be completed before b_i (Constraint (3.6)). Furthermore, between two consecutive visits, starting, processing and travel times have to be taken into account (Constraint (3.5)). The remaining constraints restrict the domains of the variables.

Although CDP-route captures many of the requirements of CDP, it cannot express the maximum time lag requirement between multiple deliveries for a single customer. The later concept however can be modeled using a flow based formulation originating in the PMSP problem. To this extent, the model of Bard and Rojanasoonthon (2006) is adapted, thereby obtaining CDP-schedule:

$$max \sum_{i \in C} q_i y_i \tag{3.10}$$

$$\sum_{k \in K} \sum_{l \in K_0 \setminus \{k\}} l_k z^i_{kl} \geq q_i y_i \qquad \forall i \in C \tag{3.11}$$

$$\sum_{l \in K_0} z^i_{k_0 l} = 1 \qquad \forall i \in V \tag{3.12}$$

$$\sum_{l \in K_0 \setminus \{k\}} z^i_{kl} = \sum_{l \in K_0 \setminus \{k\}} z^i_{lk} \qquad \forall k, l \in K, k \neq l, i \in V \tag{3.13}$$

$$C^i_k - M(1 - z^i_{kl}) \leq C^i_l - p_l \qquad \forall k, l \in K, k \neq l, i \in V \tag{3.14}$$

$$C^i_l - p_l \leq C^i_k + \gamma + M(1 - z^i_{kl}) \qquad \forall k, l \in K, k \neq l, i \in V \tag{3.15}$$

$$a_i + p_k \leq C^i_k \leq b_i \qquad \forall i \in V, k \in K \tag{3.16}$$

$$z^i_{kl} \in \{0, 1\} \qquad \forall i \in V, k, l \in K \tag{3.17}$$

$$C^i_k \in \mathbb{Z}_{\geq 0} \qquad \forall i \in V, k \in K \tag{3.18}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C \tag{3.19}$$

The binary variable z^i_{kl} is equal to one if truck $k \in K$ delivers its payload immediately before truck $l \in K$ to customer $i \in C$, zero otherwise. In this model, k_0 represents a dummy truck, $K_0 = K \cup \{k_0\}$.
Constraint (3.11) ensures that sufficient concrete is delivered to each customer. Constraints (3.12)-(3.13) sequence the trucks. A truck can only be used once for every customer, and whenever it is used, its delivery must be succeeded by another delivery (possibly a delivery by the dummy truck k_0)(Constraint (3.12)). The dummy truck must be scheduled (Constraint (3.12)). Constraints (3.13) are the flow preservation constraints. Together with Constraints (3.12), these constraints enforce that all deliveries are scheduled on a ring. Each delivery has exactly one successor and one predecessor. Constraints (3.14) link the completion time variables C^i_k and the sequence variables z^i_{kl}, thereby enforcing that deliveries do not overlap in time. Additionally, Constraints (3.15) enforce a maximum lag time between consecutive deliveries. Finally, Constraints (3.16) ensure that deliveries are scheduled within the customer's time window.
A feasible solution to CDP is obtained at the intersection of the CPD-route and CPD-schedule polytopes. Connecting the two polytopes is accomplished via

the linking constraints:

$$\sum_{l \in K_0 \setminus \{k\}} z^i_{kl} = \sum_{j \in N} x_{ijk} \qquad \forall k \in K, i \in C \tag{3.20}$$

Note that constraints (3.11), (3.16) in CDP-route are identical to Constraints (3.2), (3.6) in CDP-schedule respectively, and are dropped as a consequence. After removing redundant constraints and rewriting some constraints via the linking constraints, we arrive at a model for CDP which we denote as CDP1.

An Integer Programming model for CDP with revisits

A disadvantage of CDP1 is that a single truck cannot visit the same customer more than once. This restriction is unrealistic as often a small number of concrete trucks drive back and forth between a production depot and a construction site. It is however not obvious how to efficiently lift this restriction in CDP1. Clearly, when the binary variables x_{ijk} are replaced by equivalent integer variables indicating the number of times vehicle k travels from i to j, one can still distinguish the routes, but expressing the scheduling constraints becomes difficult. Two options exist: either the distinct trips made by a single vehicle are enumerated (e.g., vehicle k travels from i to j during trip t), or the deliveries to a specific customer are enumerated. The latter solution is applied in assignment-based formulations for scheduling problems (see e.g., Berghman et al. (2011)). An alternative IP formulation for CDP using this approach is presented by Asbach et al. (2009). This model, simplified and adjusted to our notation, is given below. Let the sets P, C be defined as before. In addition, for each customer $i \in C$, a new ordered set consisting of deliveries, $D^i = \{1, \ldots, n(i)\}$, is defined, where $n(i) = \lceil \frac{q_i}{\min\limits_{k \in K}(l_k)} \rceil$ is an upper bound on the number of deliveries required to customer i. As shorthand notation, d^i_j will be used to denote delivery j for customer i. A time window $[a_u, b_u]$ is associated with each delivery $u \in D^i$, $i \in C$, which is initialized to the time window of the corresponding customer $i \in C$, i.e., $[a_u, b_u] = [a_i, b_i]$ for all $i \in C, u \in D_i$. Finally, $D = \bigcup_{i \in C} D_i$ is the union of all deliveries.
Let $G(V, A)$ be the directed, weighted graph consisting of vertex set $V = \{0\} \cup D \cup \{n+1\}$. The arc set A is defined as follows:

- the source, sink depots have outgoing resp. incoming edges to/from all other vertices.
- a delivery node d^i_h has a directed edge to a delivery node d^i_j if $h < j$, $i \in C$, $h, j \in D_i$.

- there is a directed edge from d_u^i to d_v^j, $i \neq j$, except if d_v^j needs to be scheduled earlier than d_u^i.

The arc costs are as follows:

- $c_{0,d_j^i} = \min_{p \in P} t_{0,p} + t_{p,i}$ for all $d_j^i \in D$
- $c_{d_u^i,d_v^j} = \min_{p \in P} t_{i,p} + t_{p,j}$ for all $d_u^i, d_v^j \in D, d_u^i \neq d_v^j$
- $c_{d_j^i,n+1} = t_{i,n+1}$
- $c_{0,n+1} = 0$

The entire model, entitled CDP2, becomes:

$$max \sum_{i \in C} q_i y_i \tag{3.21}$$

$$\sum_{j \in \delta^+(0)} x_{0jk} = \sum_{i \in \delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K \tag{3.22}$$

$$\sum_{j \in \delta^-(i)} x_{j,i,k} = \sum_{j \in \delta^+(i)} x_{i,j,k} \qquad \forall i \in D, k \in K \tag{3.23}$$

$$S(i) \leq 1 \qquad \forall i \in D \tag{3.24}$$

$$S(j+1) \leq S(j) \qquad \forall i \in C, j \in \{1, \ldots, n(i)-1\} \tag{3.25}$$

$$\sum_{j \in C_i} l_k S(j) \geq q_i y_i \qquad \forall i \in C \tag{3.26}$$

$$C^i - M(1 - x_{ijk}) \leq C^j - p_k - c_{ij} \qquad \forall (i,j) \in A, i \neq 0, k \in K \tag{3.27}$$

$$C^0 - M(1 - x_{0jk}) \leq C^j - c_{0j} \qquad \forall (0,j) \in A, k \in K \tag{3.28}$$

$$C^i - p_k S(i) \geq a_i \qquad \forall i \in D \tag{3.29}$$

$$C^{j+1} - p_k S(j+1) - C^j \leq \gamma \qquad \forall i \in C, j \in \{1, \ldots, n(i)-1\} \tag{3.30}$$

$$C^{j+1} \geq C^j + p_k S(j) \qquad \forall i \in C, j \in \{1, \ldots, n(i)-1\} \tag{3.31}$$

$$a_i \leq C^i \leq b_i \qquad \forall i \in V \tag{3.32}$$

$$x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K \tag{3.33}$$

$$y_i \in \{0,1\} \qquad \forall i \in C \tag{3.34}$$

Here, $S(i) = \sum_{k \in K} \sum_{j \in \delta^+(i)} x_{ijk}$, for all $i \in D$, indicates whether delivery $i \in D$ is made by any vehicle. Binary variables x_{ijk} denote whether vehicle $k \in K$ travels from i to j, $i, j \in V$. Binary variables C^i record the time that delivery $i \in D$ is completed. In addition, C^{n+1} records the makespan of the schedule. Finally, boolean variables y_i, denote whether customer $i \in C$ is serviced.

Constraints (3.22)-(3.24) are the common vehicle routing constraints, defining respectively the starting and ending location of a tour, flow preservation, and the number of times a delivery can be made. Furthermore, Constraints (3.25) orders the deliveries: a delivery d^i_{j+1} cannot me performed whenever delivery

d_j^i has not been made. This constraint, in conjunction with Constraint (3.30) implements the maximum time lag between consecutive deliveries. The sum of capacities of the vehicles performing the deliveries for customer $i \in C$ should cover the customer's demand (Constraint (3.26)). Constraints (3.27)-(3.32) enforce the necessary scheduling restrictions. A delivery cannot be made outside the customer's time window (Constraints (3.29),(3.32)), travel times need to be accounted for (Constraint (3.27),(3.28)). Finally, Constraints (3.31) ensure that deliveries to the same customer do not overlap in time.
Although this formulation allows a single vehicle to visit customers more than once, it has the clear disadvantage that each customer is represented by a possibly large set of customer nodes, especially when the bound $n(i)$, $i \in C$, is weak. One solution is to artificially limit the maximum value of $n(i)$, but this potentially cuts off the optimal solution.

A note on complexity

Theorem 1. *The CDP as defined by CDP2 (Section 3.3.1) is NP-Hard.*

Proof. The proof follows by reduction from the Hamiltonian Cycle (HC) problem. Let P be an instance of the HC problem, defined on the simple, undirected, incomplete graph $G(V, E)$ with vertex set V and edge set $E \subset V \times V$. Finding a HC in G is NP-Complete (Garey and Johnson, 1979). Let P' be an instance of CDP, as defined in CDP2 (Section 3.3.1). Instance P' has only one vehicle k_1, having capacity $l_{k_1} = 1$ and processing time $p_{k_1} = 0$. The vertex set of P', say V', is identical to the vertex set of P, i.e., $V = V'$. Designate an arbitrary vertex in V' as depot; this vertex serves both as starting and ending depot. The remaining $|V| - 1$ vertices become customers, each having a demand of 1. The arc set A of the routing graph $G'(V', A)$, is complete, i.e. there is an arc for every $i, j \in V$ pair. The weights of the arcs are defined as follows. The weight of an arc (i, j) resp. (j, i) is equal to 1 if there exists a corresponding edge $(i, j) \in E$; its weight is equal to 2 otherwise. Finally, the time windows for each customer are set to $[0, |V| - 1]$, and the time lag $\gamma = 1$. Observe that to satisfy the demand of a single customer, only one delivery is required, so the time lag does not pose any restrictions on the delivery schedule. Furthermore, in a delivery schedule where all customers are satisfied, vehicle k_1 must arrive at the last customer at time $|V| - 1$ at the latest. By construction of the routing graph, this is only possible when vehicle k_1 traverses arcs having weight 1 only. It is straightforward to show that iff there exists a HC in P, then there must exist a delivery schedule for P' satisfying the demand of all customers. As

each delivery for each customer may only be made once (Constraint 3.24), the resulting delivery route for k_1 must correspond with a HC. □

3.3.2 Constraint Programming model

Next to MIP, Constraint Programming (CP) offers an alternative means to model the CDP. Unfortunately, no universal CP language exist; the syntax and expressiveness is solver dependent. In the discussion below, we only use constraints present in IBM's CP optimizer. However, equivalent versions of these constraints exist in several other CP solvers.
Our model utilizes interval variables (Laborie and Rogerie, 2008; Laborie et al., 2009). An interval variable represents an interval during which an activity is performed. An activity, and its corresponding interval variable may be optional, i.e., there is no obligation to schedule the activity. More formally, following the notation from (Laborie and Rogerie, 2008), an interval variable α is a variable where domain $dom(\alpha)$ is a subset of $\{\perp\} \cup \{[s, e) | s, e \in \mathbb{Z}, s \leq e\}$. An interval variable is fixed if its domain is reduced to a singleton, i.e., if either $\perp$ is assigned to α (the interval is absent) or $[s, e)$ is assigned (the interval is present and fixed, its start time is s, its end time is e, and its durations is $e - s$). An absent interval variable is ignored by any constraint or expression. For example, a constraint specifying that an interval α needs to end before the start of an interval β is always satisfied if either of the intervals is absent.
For each interval variable α, a number of functions is defined: $presenceOf(\alpha)$ is a function that indicates whether α is present (value 1) or absent (value 0); $startOf(\alpha)$ returns the start time s of α when it is present, $endOf(\alpha)$ returns the end time e and $dur(\alpha)$ returns $e - s$. These functions can be used in other constraints. The solver detects an inconsistency and backtracks whenever it notices that $endOf(\alpha) - startOf(\alpha) \geq dur(\alpha)$ is not satisfied. In our model we use the shorthand $\alpha = \{r, d, t, o\}$ to express the constraints $startOf(\alpha) \geq s$, $endOf(\alpha) \leq e$, $dur(\alpha) \geq d$, $presenceOf(\alpha) = 1$ iff $o = oblig$ (it is unconstrained when $o = opt$). The remaining constraints used in this model and supported by IBM's CP optimizer are summarized in Table 3.3.

The CP model (Algorithm 8) relies on three hierarchical levels of optional interval variables which are linked via the span and alternative constraints (lines 8, 10):

- A variable c_i, for each customer $i \in C$ (line 3).
- A variable d^i_j for each possible delivery $j \in D_i$ to customer $i \in C$ (line 4).
- A variable $d^i_{j,k}$ for each vehicle $k \in K$ that may perform a specific delivery $j \in D_i$ for customer $i \in C$ (line 5).

Constraint	Description
span(α, B)	Interval α (if present) spans over all present intervals from the set of intervals B, i.e., the start and end of α coincides with resp. the first present interval and the last present interval in B. If α is not present, then neither are the intervals in B.
alternative(α, B)	If interval α is present, then exactly one of the intervals in set B is present. The start and end of interval α coincides with the start and end of the selected interval from set B.
endBeforeStart(α, β)	$endOf(\alpha) \leq startOf(\beta)$. Automatically satisfied if either of the intervals is absent.
startBeforeEnd(α, β, z)	$startOf(\alpha) + z \leq endOf(\beta)$. Automatically satisfied if either of the intervals is absent.
noOverlapSequence$(B, dist)$	Sequences the intervals in set B. Ensures that the intervals in B do not overlap. Furthermore, the two-dimensional distance matrix $dist$ specifies a sequence dependent setup time for each pair of activities. Absent intervals are ignored.
first(α, B)	If interval α is present, it must be scheduled before any of the intervals in B.
last(α, B)	If interval α is present, it must be scheduled after any of the intervals in B.

Table 3.3: Description of CP constraints

Algorithm 1: CP model for CDP.

`Variable definitions:`

1 $s = \{0, 0, 0, oblig.\}$
2 $t = \{0, \infty, 0, oblig.\}$
3 $c_i = \{r_i, d_i, 0, opt.\}$ $\quad \forall i \in C$
4 $d^i_j = \{r_i, d_i, 0, opt.\}$ $\quad \forall i \in C, j \in D_i$
5 $d^i_{j,k} = \{r_i, d_i, p_k, opt.\}$ $\quad \forall i \in C, j \in D_i, k \in K$

`Objective:`

6 **max** $\sum_{i \in C} q_i \cdot \text{presenceOf}(c_i)$

`Constraints:`

7 **forall** $i \in C$
8 $\quad \text{span}(c_i, \bigcup_{j \in D_i} d^i_j)$
9 $\quad$ **forall** $j \in D_i$
10 $\quad\quad \text{alternative}(d^i_j, \bigcup_{k \in K} d^i_{jk})$
11 $\quad \text{presenceOf}(c_i) = (\sum_{k \in K} l_k \sum_{j \in D_i} \text{presenceOf}(d^i_{jk}) \geq q_i)$
12 $\quad$ **forall** $j \in \{1, \ldots, n(i) - 1\}$
13 $\quad\quad \text{endBeforeStart}(d^i_j, d^i_{j+1})$
14 $\quad\quad \text{startBeforeEnd}(d^i_{j+1}, d^i_j, -\gamma)$
15 $\quad\quad \text{presenceOf}(d^i_{j+1}) \rightarrow \text{presenceOf}(d^i_j)$
16 **forall** $k \in K$
17 $\quad \text{noOverlapSequence}(\bigcup_{i \in C, j \in D_i} d^i_{jk} \cup s \cup t, dist)$
18 $\quad \text{first}(s, \bigcup_{i \in C, j \in D_i} d^i_{jk} \cup t)$
19 $\quad \text{last}(t, \bigcup_{i \in C, j \in D_i} d^i_{jk} \cup s)$

The Constraint on line 11 states the relation between a customer and the customer's deliveries: a customer may only be scheduled if sufficient concrete is delivered. The constraints 13 - 15 take care of the scheduling requirements. Finally, Constraints 17 - 19 deal with vehicle routing restrictions: for each vehicle, a Hamiltonian path is created which starts and ends at resp. the source depot s and the sink depot t, while respecting travel times between the deliveries. To improve constraint propagation, two implicit sets of constraints are added. Let $m(i) = \lceil \frac{q_i}{\max_{k \in K}(l_k)} \rceil$ be a lower bound on the number of deliveries required to satisfy customer $i \in C$. The following redundant constraints are added:

1 **forall** $i \in C$
2 **forall** $j \in \{1, \ldots, m(i)\}$
3 presenceOf(c_i)=presenceOf(d^i_j)
4 **forall** $j \in \{m(i), \ldots, n(i) - 1\}$
5 presenceOf(c_i)$\wedge$($\sum_{l \in \{1..j\}, k \in K} l_k \cdot$presenceOf($d^i_{lk}$)$< q_i$) $\rightarrow$ presenceOf(d^i_{j+1})

The constraint on line 3 states that if customer c_i, $i \in C$, is present, then at least $m(i)$ deliveries must be made. The constraint on line 5 ensures that if the total amount of concrete delivered in the first j deliveries to customer i does not suffice, at least one more delivery is made.

3.4 Heuristic models

3.4.1 Steepest Descent and Best Fit

The first heuristic is based on a steepest descent local search and a best fit constructive procedure. The constructive heuristic (Section 3.4.1) schedules the visits to the costumers one-by-one at the 'best' possible position, determined by several heuristic criteria, e.g., the start time of the visit, and the capacity of the vehicle. The order π the customers are processed by the constructive procedure is controlled by a Steepest Descent heuristic (Subsection 30).

Best fit constructive heuristic

Algorithm 2 (page 78) illustrates the best fit constructive heuristic. The following functions, omitting the implementation details for clarity, are used:

- $satisfied(\mathscr{S}, i)$: returns whether customer $i \in C$ is satisfied in solution $\mathscr{S}$, i.e., whether sufficient concrete is delivered for this customer.
- $earliestStartTimeOfVisit(\mathscr{S}, k, i)$: searches for the earliest moment in time vehicle $k \in K$ can schedule the next delivery to customer $i \in C$ given the current solution $\mathscr{S}$, and while taking travel times into consideration. Note that deliveries for the same customer cannot overlap, so the time returned is always later than the end time of the previous delivery for customer $i \in C$.

- $wasteOfVisit(\mathscr{S}, k, i)$: returns the amount of $waste$, $0 \leq waste < l_k$, a visit by vehicle $k \in K$ would introduce to customer $i \in C$, given the current solution $\mathscr{S}$.
- $endOfPreviousVisit(\mathscr{S}, i, t)$: returns the end time of the previous visit to customer $i \in C$ in $\mathscr{S}$. If this is the first visit to customer i then the value t is returned.
- $scheduleVisit(\mathscr{S}, i, bestK, start)$: schedules a visit to customer i in solution $\mathscr{S}$ using vehicle $bestK$ at time $start$.
- $attemptChaining(\mathscr{S}, i)$: this method is invoked when the next delivery for customer $i \in C$ cannot be scheduled due to the maximum time lag to the forgoing delivery. This method tries to shift all previous scheduled visits for customer $i \in C$ in solution $\mathscr{S}$ to a later point in time, taking into account the maximum timelag γ and the customer deadline b_i. Next it tries to schedule a new visit for customer $i \in C$ at the earliest available point in time, and returns true whenever this operation is successful.
- $clean(\mathscr{S})$: removes all the visits of unsatisfied customers from solution $\mathscr{S}$.

The algorithm iterates over all costumers in π. For each customer it searches for the best vehicles to schedule the customer's visits. First of all, to schedule a new visit, a feasible start time must be found which respects, 1) the travel times, 2) the maximum time lag γ with respect to the previous visit, and 3) the customer's deadline b_i. The best vehicle to perform a visit is selected by the following (ordered) list of criteria:

1. earliest available vehicle, counting from the moment the delivery may commence
2. minimize waste, i.e., the amount that the vehicle's capacity surpasses the remaining demand of the customer
3. in case of a draw, select the largest vehicle.

These criteria have been obtained experimentally.

Steepest Descent

The quality of the schedule produced by the best-fit heuristic is largely determined by the order π in which the customers are being processed. Different

solutions may be discovered when the best-fit procedure is invoked multiple times for different permutations of π. To this extent, a local search procedure is used which modifies the vector π. The initial ordering of π is determined according to the following criteria:

1. earliest deadline b_i
2. highest demand q_i
3. earliest release date a_i

These criteria and their order have been determined empirically. At each iteration, the steepest descent heuristic performs a full neighborhood search, i.e., all shifts and swaps of customers in π are considered.

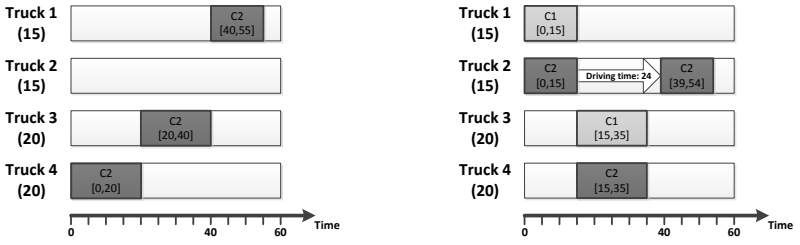
Heuristic limitations

A number of instances exist where the heuristic would never find the optimal solution, independent of the order π. An example of such an instance is depicted in Figure 3.1. Two customers (1 and 2) with demands $q_1 = 25$ and $q_2 = 45$ are scheduled using four vehicles (two with capacity 15 and two with capacity 20) in time windows $[a_1 = 0, b_1 = 40]$ and $[a_2 = 0, b_2 = 60]$. The maximum time lag $\gamma = 5$. The delivery times are equal to the vehicle capacities, i.e., $l_k = p_k$, for all $k \in K$. The travel times between customers are $t_{11} = 18, t_{12} = 37, t_{21} = 37, t_{22} = 24$; all other travel times are set to 0.
The heuristic fails to find the optimal solution due to the criteria used to determine the next delivery to a specific customer. As elaborated in the subsection 3.4.1, the vehicle performing the next delivery for a customer is selected according to its availability, the amount of waste it produces and its size. Only when the first criterion is replaced by 'select the vehicle which produces the largest amount of waste', the optimal solution for this instance is found. It may be clear however that such a criterion is undesirable for most instances.

3.4.2 Fix-and-Optimize heuristic

The pseudo code for the fix-and-optimize heuristic is provided in Algorithm 3. The following functions are used in this code:

- $satisfied(\mathscr{S}, i)$: returns whether customer $i \in C$ is satisfied in solution $\mathscr{S}$, i.e., sufficient concrete is delivered to this customer.



(a) Best SD-Heuristic solution (b) Optimal solution

Figure 3.1: Example where the SD-heuristic does not find the optimal solution.

- *calcOverlappingSet*(i): Calculates the set of customers $C' \subset C$ having an overlap between their delivery interval and that of customer $i \in C$. Note that $|C'| \geq 1$ as customer i must be included in C'.
- $fix(C', \mathscr{S})$: Fixes the variables for the customers in C' as stated in schedule $\mathscr{S}$. If customer $j \in C'$ is not satisfied in schedule $\mathscr{S}$, set $y_j = 0$, set $y_j = 1$ otherwise. In a similar fashion, the completion variables as well as a number of flow variables are fixed.
- *reOptimize*$(\mathscr{S})$: Solves the resulting MIP model and updates $\mathscr{S}$ accordingly.
- *release*(): Unfixes all fixed variables.

The algorithm is initialized with a feasible schedule $\mathscr{S}$ and a permutation π of customers not satisfied in $\mathscr{S}$. At each iteration of the algorithm, the fix-and-optimize heuristic attempts to optimize part of the schedule, using the MIP model CDP2 presented in Section 3.3.1. However, instead of solving the entire model at once, only a small part of the problem is optimized, as most of the variables are fixed. First, for a given customer $i \in \pi$, the heuristic determines the set of customers $C' \subseteq C$ having their delivery interval overlap with the interval $[a_i, b_i]$. Next, the heuristic fixes all y_i and C^i variables of the customers not in C' to their corresponding values in $\mathscr{S}$. Moreover, flow variables x_{ijk} that do not affect any of the possible deliveries to customers in C' are fixed as well. Finally, the heuristic solves the resulting MIP model, thereby rescheduling the customers in C', while leaving the deliveries to the other customers unchanged. The resulting schedule must be at least as good as the schedule obtained at the previous iteration of the algorithm.
Note that the above procedure can be used to:

1. improve an existing schedule, i.e., attempt to add unscheduled customers
2. construct a new schedule by starting from an empty schedule

Algorithm 3: Fix-and-Optimize heuristic.

input : A permutation of customers to schedule π, An initial schedule $\mathscr{S}$
output : A feasible CDP solution

```
1 foreach i ∈ π do
2     if ¬satisfied(𝒮,i) then
3         O=calcOverlappingSet(i);
4         C′ = {j ∈ C : j ∉ O};
5         fix(C′,𝒮);
6         reOptimize(𝒮);
7         release();
8 return 𝒮
```

3.5 Bounds

Upper bounds, required to assess the quality of the proposed algorithms, are obtained by solving the LP relaxation of CDP2 (Section 3.3.1). A number of cuts are added to strengthen the bound. These cuts are calculated by solving a number of small subproblems. First, a simple test is conducted to verify whether there exists at least one schedule satisfying a specific customer. The latter is achieved by solving the CDP2 (Section 3.3.1) as follows:

1. First, for a given customer $i \in C$, set $y_i = 1$, and set $y_j = 0$, for all $j \in C, j \neq i$.
2. Solve CDP2. Whenever the model turns out infeasible, there exists no schedule that accommodates customer i.

All customers who could not be accommodated in the schedule are permanently discarded. Next, the above procedure is repeated in a similar fashion, but this time for pairs of customers:

1. For every pair of customers $i, j \in C, i \neq j$, set $y_i = y_j = 1$, and set $y_k = 0$, for all $k \in C, k \notin \{i, j\}$.
2. Solve the MIP model. Whenever the model turns out infeasible, there exists no schedule in which both customers i and j are satisfied. Consequently, the valid inequality $y_i + y_j \leq 1$ may be added to the model.

Note that the above procedure identifies Minimum Infeasible Subsets (MIS) of size two. Clearly, the procedure may be repeated to identify larger Infeasible Subsets, this, however, quickly becomes computationally intractable.
After adding all cuts based on MIS of size two, the LP relaxation of the resulting model is computed, strengthened by a number of cuts automatically generated by IBM's Ilog Cplex solver version 12.5.1.
An alternative means to calculate a valid upper bound involves solving the following simple MIP model:

$$max \sum_{i \in C} q_i y_i \tag{3.35}$$

$$\sum_{i \in S} y_i \leq |S| - 1 \qquad \forall S \in \mathbb{S}, S \subseteq C \tag{3.36}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C \tag{3.37}$$

Here, $\mathbb{S}$ is the set of all infeasible subsets. As noted before, computing the entire set $\mathbb{S}$ is intractable. We therefor limit computations to $|S| \leq 3$.

3.6 Experimental Results

3.6.1 Data Sets

Due to the lack of publicly available benchmark data for the CDP, two data sets are created. Both sets, as well as code to generate new instances, are available online at (Kinable and Wauters, 2013). A summary of the instances is provided in Table 3.4. Data Set A contains instances with 10 to 20 *customers*, and 2 to 5 *vehicles*, whereas Data Set B contains instances with up to 50 *customers* and 20 *vehicles*. The *demands* of customers are always divisible by 5 and are selected at random from the interval $[10; 75]$. In a similar fashion, the *capacities* of the vehicles are selected between $[10; 25]$. It is unrealistic to have a unique capacity for each vehicle, and therefore the number of different vehicles is bounded by the number of *vehicle classes*. The *processing time* of a vehicle $k \in K$ is set proportional to its capacity, i.e., $p_k = l_k$ for all $k \in K$. The width of the time window of a customer $i \in C$ is computed by $\lambda_i q_i$, where λ_i is a scalar uniformly selected from the interval $[1.1, 2.1]$.
Each instance has up to 4 *production stations*. The locations of the start and end depot, construction sites, and production depots, lay in the euclidean plane. The travel time between two locations equals the euclidean distance, rounded upwards. For simplicity, the start and end depot are the same. The distance

between each station and the depot is uniformly selected from the interval $[1-30]$. Furthermore, for each customer, there exists a production station within a distance of $[1-25]$. Finally, the time lag γ is fixed to 5 for all deliveries.

3.6.2 Experiments

A number of experiments are conducted to assess the quality of the algorithms presented in this work. To solve the MIP models, IBMś CPLEX 12.5.1 has been used (default parameters), whereas IBMś CP Optimizer 12.5.1 has been used to solve the CP problem (default parameters). The results of the Steepest Descent heuristic (*SD-Heuristic*, Section 3.4.1), the fix-and-optimize MIP heuristic (*MIP fix & opt*, Section 3.4.2), the exact MIP algorithm (*MIP*, Section 3.3.1) and the CP algorithm (*CP*, Section 3.3.2) on Data Set A are provided in Table 3.7 in Section 3.9, page 83. The first column of Table 3.7 provides the instance names, following a "$W\_x\_y\_z$" naming scheme, where W reflects the data set, x the number of vehicles, y the number of customers and z the number of stations. The second column provides an upper bound on the optimal solution values; only the best bound obtained from the two procedures discussed in section 3.5 is shown. Per solution method, the table shows resp. the objective value, the gap between the objective value and the bound, and the computation time in milliseconds. Each solution method was allotted 5 minutes computation time per instance. Furthermore, the exact MIP, CP methods, as well as the MIP fix-and-optimize heuristic are initialized by the results obtained from the SD-heuristic; hence, their solutions are at least as good as the solution of the SD-heuristic. Finally, Table 3.5 displays the average gap obtained by all methods, as well as the average computation time.

The differences in results for the various methods are relatively small when compared on Data Set A (Table 3.7, Figure 3.2). Clearly, from an objective point of view, CP outperforms all methods (avg. gap 4.2%), followed by the fix-and-optimize heuristic (avg. gap 7.0%) the exact MIP procedure (avg. gap 7.3%) and the SD-heuristic (avg. gap 9.1%). Although the SD-heuristic performs the least in terms of objective value, it usually requires less than 50ms per instance, whereas the other methods require significantly more time. CP solved 40 out of 64 instances to optimality, the exact MIP method 37, the fix-and-optimize MIP heuristic 35 and the SD-heuristic 30.

The results for Data Set B are shown in Table 3.8 in 3.9. The results for the two approaches based on the MIP formulation were omitted from the graphs as these methods were unable to deal with the larger instance sizes. The time limit for the remaining methods is set to 10 minutes per instance. The average gaps obtained are resp. 12.1% for CP, and 16.3% for the SD-heuristic. Out of 128 instances, CP solved 55 instances to optimality, versus 40 instances for

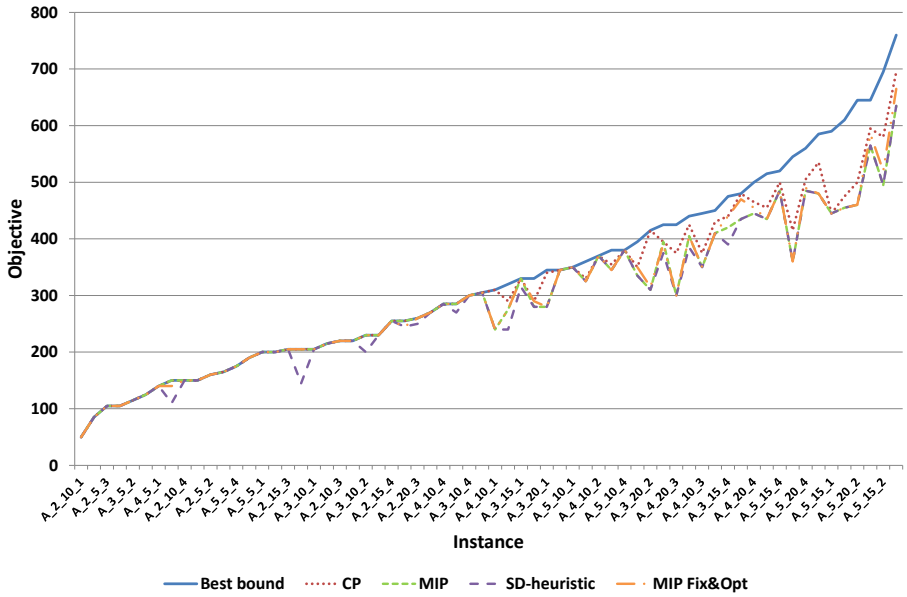


Figure 3.2: Comparison of the different methods on Data Set A.

the SD-heuristic. Compared to Data Set A, the average gaps have increased significantly, especially for instances with a high ratio between the number of customers and the number of vehicles (Figure 3.3, page 76). Computing strong bounds for these instances via the approaches elaborated in Section 3.5 is difficult and very time consuming as enumerating all infeasible subsets of sizes 2 and 3 is computationally expensive for large numbers of customers.
Often, CP finds good solutions in relatively little time, but fails to prove optimality. This is mainly due to the fact that most interval variables in the CP model are declared optional, diminishing the potential amount of constraint propagation. Typically, the smaller the initial domains of the variables (e.g., small time windows for the customers), the faster CP is capable of solving an instance.
Figure 3.4 (page 76) plots the influence of the time lag on several instances. As is visible from the monotonically increasing lines, increasing the time lag improves the overall flexibility of the schedule, thereby enabling better quality solutions.

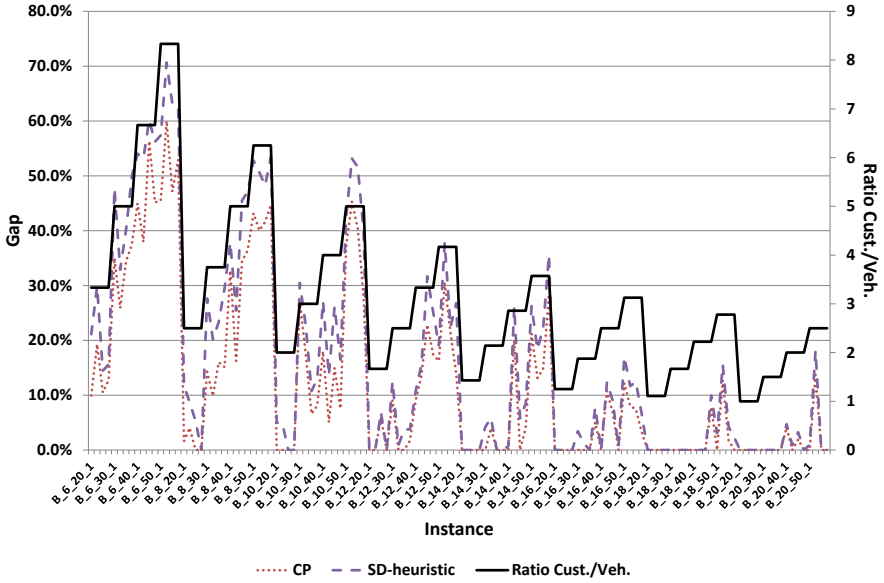


Figure 3.3: Comparison of the different methods on Data Set B.

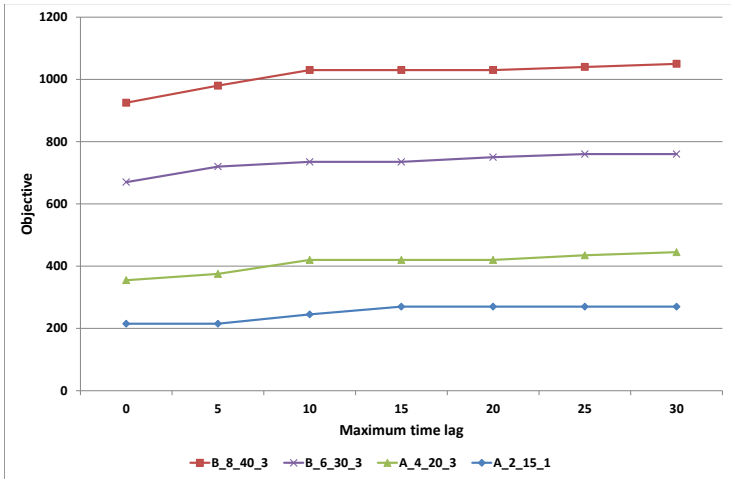


Figure 3.4: Influence of the time lag γ. Solutions are obtained via the CP model.

3.7 Conclusion

Although several variations of CDP exist in the literature, little attempts have been made to compare the various approaches mutually. In practice, a fair comparison is hindered by differences in problem definitions, as well as by the lack of publicly available benchmark data. This work introduces a more general version of CDP, bearing a strong resemblance with two well-studied scheduling and routing problems: the Parallel Machine Scheduling Problem and the Capacitated Vehicle Routing Problem. The problem is shown to be NP-Hard by reduction to the Hamiltonian Cycle problem. New exact, heuristic and hybrid solution approaches are proposed. Computational experiments are conducted on a library of CDP instances, and compared against bounds on the optimal solutions. The MIP model appears ineffective in solving large problem instances, but can be used to compute bounds instead. The CP model, on the other hand, is highly effective in finding high quality solutions in relatively little time, or to improve existing schedules. When computation time is a limiting factor, the Steepest-Descent heuristic is a viable choice, as it often yields good solutions in less than a second. Finally, when compared to the traditional MIP approach, the fix-and-optimize MIP heuristic produces, on average, better results in less time.

The current work primarily focuses on a static scheduling problem where the set of customers and their demands are known beforehand. Future work could therefore be aimed at an online version of the problem with a rolling scheduling horizon. A similar approach to the fix-and-optimize MIP heuristic may be used to fix part of the schedule which is currently being executed, while reoptimizing the delivery schedule towards the end of the time horizon. On a similar note, different objective functions may be investigated. For example, deliveries for customers which were not serviced by the concrete distributor due to a lack of available vehicle capacity may be postponed to a later point in time, while being treated with higher priority.

Algorithm 2: The best fit construction heuristic.

input : A permutation of customers π
output : A feasible CDP solution $\mathscr{S}$

```
1  Start with empty solution 𝒮 ;
2  foreach i ∈ π do /* For each customer                                  */
3      found ← TRUE ;
4      while found and ¬satisfied(𝒮,i) do
5          found ← FALSE ;
6          minStart ← MAXVALUE ;
7          minWaste ← MAXVALUE ;
8          bestK ← −1 ;
9          foreach k ∈ K do /* For each vehicle                           */
10             start ← earliestStartTimeOfVisit(𝒮,k,i) ;
11             waste ← wasteOfVisit(𝒮,k,i) ;
12             prevEnd ← endOfPreviousVisit(𝒮,i,start) ;
13             if start ≤ prevEnd + γ and start + p_k ≤ b_i then
14                 found ← TRUE ;
15                 newBest ← FALSE ;
16                 if start < minStart then
17                     newBest ← TRUE ;
18                 else if start == minStart and waste < minWaste then
19                     newBest ← TRUE ;
20                 else if start == minStart and waste == minWaste and
                   l_k > q_bestK then
21                     newBest ← TRUE ;
22                 if newBest then
23                     minStart ← start ;
24                     minWaste ← waste ;
25                     bestK ← k ;
26         if found then
27             scheduleVisit(𝒮,i,bestK,minStart) ;
28         else
29             found ← attemptChaining(𝒮,i) ;
30 clean (𝒮) ;
```

	Set A	**Set B**
Instances	64	128
Customers	10-20	20-50
Demands	10-75	10-75
Time Windows	$q_i \times [1.1, 2.1]$	$q_i \times [1.1, 2.1]$
Time lags	5	5
Vehicles	2-5	6-20
Capacity	10-25	10-25
Vehicle classes	2-3	3
Processing time	$p_k = l_k$	$p_k = l_k$
Stations	1-4	1-4
Cust.-Station	1-30	1-30
Depot-Station	1-25	1-25

Table 3.4: Data sets

	Data Set A		Data Set B	
	Gap(%)	Time(ms)	Gap(%)	Time(ms)
CP	4.2	197012	12.1	357313
MIP	7.3	134058	-	-
SD-Heuristic	9.1	23	16.3	1331
MIP Fix & Opt	7.0	100765	-	-

Table 3.5: Summary (averages).

3.8 Literature Summary Notation

Solution method:	
• Mixed Integer Programming *(m)*	The paper proposes a MIP model.
• Constraint Programming *(c)*	The paper proposes a CP model.
• Heuristic *(he)*	The paper proposes a heuristic approach.
• Hybrid *(hy)*	The paper proposes an hybrid approach, involving both exact approaches and local search techniques.
Time windows and limits:	
• Soft delivery time windows *(sd)*	Time windows may be violated, or only a preferred start time of the deliveries is provided.
• Hard delivery time windows *(hd)*	Deliveries must be made within a defined time window.
• Hard delivery ready time *(hs)*	Deliveries may not commence before a predefined time.
• Vehicle usage time *(sv)*	Vehicles may only be used for a certain amount of time, or may not be available during certain periods, e.g. due to maintenance.
• Concrete Perish Time *(p)*	The amount of time concrete may reside in the truck is limited. Consequently, customers may not be reachable from distant depots, or deliveries are aborted before the truck is empty as the truck's processing time exceeds the time limit.
Start, end location of vehicles:	
• Central depot *(c)*	All vehicles start from and/or return to a central depot.
• Production center *(p)*	All vehicles start from and/or return to a (specific) production center.
• Mixed *(m)*	A vehicle may start from and/or end at a production center or at a depot.
Production depots:	
• Homogeneous *(ho)*	All production depots are identical.
• Heterogeneous *(hs)*	Differences in production depots, e.g. they produce different types of concrete, or cannot service all customers.

• Scheduling *(s)*	Vehicle reloads must be scheduled, e.g. a vehicle may have to wait while another vehicle is being loaded.

Loading and Unloading:

• Fixed rate *(f)*	(un)loading takes a constant amount of time.
• Dynamic rate *(d)*	(un)loading time depends on the specific customer, vehicle, type of concrete, amount of concrete in the truck and/or the production station.

Fleet:

• Homogeneous *(ho)*	All vehicles are identical.
• Heterogeneous *(he)*	Vehicles differ in capacity or equipment carried.

Instrumentation:

- For some deliveries, specialized equipment must be present. Either the delivery truck has this equipment, or an additional vehicle having this equipment must be scheduled.

Deliveries, restrictions:

• Synchronization *(s)*	Deliveries for the same customer must be synchronized as they may not overlap in time, and/or have to take a minimum or a maximum time lag into consideration.
• Revisits *(r)*	A single truck may perform multiple deliveries for the same customer, not necessary in a consecutive order.
• Vehicle requirements *(v)*	Not all vehicles can perform all deliveries, e.g. because a vehicle cannot transport the type of concrete required by the customer, or a vehicle may be too large for the construction site.
• Reload *(l)*	Vehicles must reload after each delivery.
• Shared delivery *(sh)*	A vehicle may split its content over multiple customers without reloading in between deliveries.
• Split delivery *(sp)*	Customer may require multiple deliveries by different vehicles.
• Single source *(ss)*	For some customers, all concrete delivered must originate from the same production site.

Objectives (incl. weighted versions and composite objectives):

• Minimize vehicle usage *(u)*	The frequency a vehicle is used, or the time that a vehicle is used, incl. travel time, loading and unloading.
• Minimize wastage *(w)*	Amount of concrete delivered to a customer that surpasses the requested amount.
• Minimize delay *(d)*	Deviation from soft time window, deviation from requested start time, or vehicle overtime.
• Minimize outsourcing *(o)*	When the schedule fails accommodate deliveries for a particular customer, the deliveries may be outsourced.
• Minimize operating costs *(op)*	Operating costs incurred at the production site or the construction sites, e.g. the (weighted) time difference between the start and end time of the production of resp. the first and the last batch of concrete.
• Maximize utilization balance *(b)*	Ensure that vehicles are employed equally often.
• Minimize travel time or distance *(t)*	
• Minimize number of vehicles *(v_s)* used in the solution	
• Minimize number of vehicles *(v_c)* used per customer	
• Maximize number of satisfied customers *(s)*	

3.9 Computational Experiments

Table 3.7: Computational results Data Set A

		CP			MIP			SD-heuristic			MIP Fix-Opt		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
A_2_5_1	85	85	0%	414	85	0%	1055	85	0%	9	85	0%	375
A_2_5_2	160	160	0%	53	160	0%	139	160	0%	5	160	0%	44
A_2_5_3	105	105	0%	37	105	0%	39	105	0%	3	105	0%	33
A_2_5_4	105	105	0%	300000	105	0%	107	105	0%	10	105	0%	531
A_2_10_1	50	50	0%	212	50	0%	125	50	0%	24	50	0%	268
A_2_10_2	150	150	0%	300000	150	0%	4587	110	27%	21	140	7%	15227
A_2_10_3	220	220	0%	255	220	0%	378	220	0%	52	220	0%	261
A_2_10_4	150	150	0%	300000	150	0%	1443	150	0%	50	150	0%	1847
A_2_15_1	215	215	0%	431	215	0%	618	215	0%	89	215	0%	1326
A_2_15_2	320	290	9%	300000	275	14%	300000	240	25%	15	275	14%	90674
A_2_15_3	205	205	0%	300000	205	0%	2167	205	0%	12	205	0%	139759
A_2_15_4	255	255	0%	300000	255	0%	4902	255	0%	11	255	0%	49266
A_2_20_1	255	255	0%	300000	255	0%	10256	245	4%	52	245	4%	294900
A_2_20_2	270	270	0%	300000	270	0%	7624	270	0%	42	270	0%	201615
A_2_20_3	260	260	0%	300000	260	0%	10135	250	4%	27	260	0%	155839
A_2_20_4	380	355	7%	300000	345	9%	300000	345	9%	39	345	9%	162334
A_3_5_1	205	205	0%	175	205	0%	776	145	29%	1	205	0%	102
A_3_5_2	115	115	0%	187	115	0%	212	115	0%	0	115	0%	24
A_3_5_3	125	125	0%	16	125	0%	98	125	0%	0	125	0%	5
A_3_5_4	190	190	0%	18	190	0%	107	190	0%	1	190	0%	8
A_3_10_1	205	205	0%	353	205	0%	274	205	0%	3	205	0%	2230
A_3_10_2	230	230	0%	2637	230	0%	4039	200	13%	2	230	0%	23278
A_3_10_3	305	305	0%	300000	305	0%	6025	305	0%	3	305	0%	53302
A_3_10_4	300	300	0%	300000	300	0%	543	300	0%	5	300	0%	394
A_3_15_1	330	330	0%	300000	330	0%	11191	315	5%	44	330	0%	102587
A_3_15_2	425	395	7%	300000	395	7%	300000	375	12%	14	390	8%	4239
A_3_15_3	330	290	12%	300000	280	15%	300000	280	15%	9	290	12%	52873
A_3_15_4	475	440	7%	300000	420	12%	300000	390	18%	16	440	7%	254550
A_3_20_1	345	340	1%	300000	280	19%	300000	280	19%	32	280	19%	253366
A_3_20_2	415	415	0%	300000	310	25%	300000	310	25%	23	315	24%	255852
A_3_20_3	360	330	8%	300000	325	10%	300000	325	10%	56	325	10%	243884

Table 3.7: Computational results Data Set A

		CP			MIP			SD-heuristic			MIP Fix-Opt		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
A_3_20_4	480	480	0%	300000	435	9%	300000	435	9%	35	470	2%	149430
A_4_5_1	140	140	0%	15	140	0%	79	140	0%	0	140	0%	5
A_4_5_2	150	150	0%	22	150	0%	296	150	0%	0	150	0%	16
A_4_5_3	165	165	0%	16	165	0%	95	165	0%	0	165	0%	7
A_4_5_4	230	230	0%	21	230	0%	265	230	0%	0	230	0%	10
A_4_10_1	310	310	0%	300000	240	23%	300000	240	23%	5	240	23%	300000
A_4_10_2	370	370	0%	300000	370	0%	51041	370	0%	1	370	0%	312
A_4_10_3	445	375	16%	300000	350	21%	300000	350	21%	3	350	21%	300000
A_4_10_4	285	285	0%	300000	285	0%	657	285	0%	2	285	0%	40
A_4_15_1	545	415	24%	300000	360	34%	300000	360	34%	17	360	34%	296608
A_4_15_2	610	475	22%	300000	455	25%	300000	455	25%	31	455	25%	192264
A_4_15_3	450	430	4%	300000	410	9%	300000	410	9%	20	410	9%	29357
A_4_15_4	515	455	12%	300000	435	16%	300000	435	16%	29	435	16%	282456
A_4_20_1	585	535	9%	300000	480	18%	300000	480	18%	57	480	18%	196031
A_4_20_2	440	425	3%	300000	405	8%	300000	385	13%	58	405	8%	147700
A_4_20_3	425	375	12%	300000	300	29%	300000	300	29%	89	300	29%	300000
A_4_20_4	500	465	7%	300000	445	11%	300000	445	11%	39	455	9%	215212
A_5_5_1	200	200	0%	17	200	0%	113	200	0%	0	200	0%	7
A_5_5_2	200	200	0%	22	200	0%	187	200	0%	0	200	0%	11
A_5_5_3	220	220	0%	24	220	0%	234	220	0%	0	220	0%	14
A_5_5_4	175	175	0%	300000	175	0%	390	175	0%	0	175	0%	49
A_5_10_1	350	350	0%	62	350	0%	1593	350	0%	3	350	0%	71
A_5_10_2	345	345	0%	44	345	0%	835	345	0%	1	345	0%	36
A_5_10_3	285	285	0%	300000	285	0%	22468	270	5%	3	285	0%	158
A_5_10_4	380	380	0%	56	380	0%	1300	380	0%	2	380	0%	41
A_5_15_1	590	445	25%	300000	445	25%	300000	445	25%	19	445	25%	300000
A_5_15_2	695	580	17%	300000	495	29%	300000	495	29%	15	520	25%	241135
A_5_15_3	395	350	11%	300000	335	15%	300000	335	15%	20	350	11%	104745
A_5_15_4	520	500	4%	300000	485	7%	300000	485	7%	37	485	7%	168244
A_5_20_1	760	695	9%	300000	635	16%	300000	635	16%	49	665	13%	253379
A_5_20_2	645	500	22%	300000	460	29%	300000	460	29%	67	460	29%	298333

Table 3.7: Computational results Data Set A

		CP			MIP			SD-heuristic			MIP Fix-Opt		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
A_5_20_3	645	595	8%	300000	565	12%	300000	565	12%	63	585	9%	136384
A_5_20_4	560	505	10%	300000	485	13%	300000	485	13%	121	490	13%	172364

Table 3.8: Computational results Data Set B

		CP			SD-heuristic		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time
B_6_20_1	805	725	9.9%	600000	635	21.1%	431
B_6_20_2	855	690	19.3%	600000	600	29.8%	103
B_6_20_3	760	680	10.5%	600000	650	14.5%	48
B_6_20_4	705	615	12.8%	600000	595	15.6%	52
B_6_30_1	1300	845	35.0%	600000	680	47.7%	281
B_6_30_2	1140	845	25.9%	600000	765	32.9%	160
B_6_30_3	1060	695	34.4%	600000	635	40.1%	262
B_6_30_4	1000	625	37.5%	600000	500	50.0%	354
B_6_40_1	1545	850	45.0%	600000	710	54.0%	1359
B_6_40_2	1635	1015	37.9%	600000	765	53.2%	534
B_6_40_3	1570	685	56.4%	600000	625	60.2%	2266
B_6_40_4	1450	795	45.2%	600000	635	56.2%	850
B_6_50_1	1890	1030	45.5%	600000	805	57.4%	1869
B_6_50_2	2250	900	60.0%	600000	660	70.7%	1687
B_6_50_3	1740	920	47.1%	600000	640	63.2%	1767
B_6_50_4	2080	980	52.9%	600000	775	62.7%	2551
B_8_20_1	935	920	1.6%	600000	825	11.8%	44
B_8_20_2	865	830	4.0%	600000	790	8.7%	55
B_8_20_3	655	655	0.0%	77476	620	5.3%	36
B_8_20_4	820	820	0.0%	312	820	0.0%	80
B_8_30_1	1085	930	14.3%	600000	785	27.6%	250
B_8_30_2	1115	1005	9.9%	600000	890	20.2%	280
B_8_30_3	1155	970	16.0%	600000	885	23.4%	555
B_8_30_4	1320	1120	15.2%	600000	930	29.5%	410
B_8_40_1	1665	1120	32.7%	600000	1035	37.8%	1351
B_8_40_2	1415	1185	16.3%	600000	1055	25.4%	773
B_8_40_3	1495	980	34.4%	600000	815	45.5%	1087
B_8_40_4	1730	1100	36.4%	600000	915	47.1%	1024
B_8_50_1	1980	1125	43.2%	600000	935	52.8%	5810
B_8_50_2	1935	1160	40.1%	600000	955	50.6%	4615
B_8_50_3	1960	1145	41.6%	600000	1015	48.2%	3623

Table 3.8: Computational results Data Set B

		CP			SD-heuristic		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time
B_8_50_4	1835	1015	44.7%	600000	855	53.4%	4024
B_10_20_1	805	805	0.0%	24814	765	5.0%	36
B_10_20_2	825	825	0.0%	415312	785	4.8%	28
B_10_20_3	730	730	0.0%	361	730	0.0%	30
B_10_20_4	765	765	0.0%	383	765	0.0%	29
B_10_30_1	1215	885	27.2%	600000	845	30.5%	459
B_10_30_2	1355	1115	17.7%	600000	1045	22.9%	447
B_10_30_3	1210	1130	6.6%	600000	1080	10.7%	327
B_10_30_4	1235	1135	8.1%	600000	1075	13.0%	383
B_10_40_1	1475	1210	18.0%	600000	1075	27.1%	1086
B_10_40_2	1580	1500	5.1%	600000	1365	13.6%	1295
B_10_40_3	1605	1365	15.0%	600000	1185	26.2%	1235
B_10_40_4	1455	1345	7.6%	600000	1215	16.5%	1353
B_10_50_1	2265	1420	37.3%	600000	1300	42.6%	3394
B_10_50_2	1900	1040	45.3%	600000	890	53.2%	5223
B_10_50_3	2005	1195	40.4%	600000	970	51.6%	3597
B_10_50_4	1925	1390	27.8%	600000	1160	39.7%	4719
B_12_20_1	770	770	0.0%	260	770	0.0%	15
B_12_20_2	770	770	0.0%	502	770	0.0%	33
B_12_20_3	945	890	5.8%	600000	880	6.9%	64
B_12_20_4	850	850	0.0%	576	850	0.0%	71
B_12_30_1	1320	1200	9.1%	600000	1155	12.5%	324
B_12_30_2	1185	1185	0.0%	913	1175	0.8%	168
B_12_30_3	950	950	0.0%	46560	915	3.7%	209
B_12_30_4	1185	1165	1.7%	600000	1140	3.8%	327
B_12_40_1	1475	1340	9.2%	600000	1315	10.8%	481
B_12_40_2	1510	1305	13.6%	600000	1275	15.6%	903
B_12_40_3	1640	1265	22.9%	600000	1120	31.7%	1713
B_12_40_4	1550	1280	17.4%	600000	1160	25.2%	2258
B_12_50_1	1755	1470	16.2%	600000	1420	19.1%	3408
B_12_50_2	2000	1390	30.5%	600000	1245	37.8%	7886

Table 3.8: Computational results Data Set B

		CP			SD-heuristic		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time
B_12_50_3	1825	1435	21.4%	600000	1415	22.5%	6029
B_12_50_4	1940	1670	13.9%	600000	1420	26.8%	4249
B_14_20_1	830	830	0.0%	666	830	0.0%	15
B_14_20_2	695	695	0.0%	536	695	0.0%	15
B_14_20_3	840	840	0.0%	633	840	0.0%	40
B_14_20_4	755	755	0.0%	570	755	0.0%	17
B_14_30_1	1190	1190	0.0%	3690	1140	4.2%	275
B_14_30_2	1370	1315	4.0%	600000	1290	5.8%	261
B_14_30_3	1005	1005	0.0%	551	1005	0.0%	109
B_14_30_4	1205	1205	0.0%	1579	1205	0.0%	207
B_14_40_1	1395	1395	0.0%	4612	1385	0.7%	355
B_14_40_2	1725	1380	20.0%	600000	1275	26.1%	1929
B_14_40_3	1550	1550	0.0%	64809	1460	5.8%	433
B_14_40_4	1705	1635	4.1%	600000	1560	8.5%	1289
B_14_50_1	2285	1800	21.2%	600000	1685	26.3%	3747
B_14_50_2	2015	1750	13.2%	600000	1645	18.4%	8700
B_14_50_3	2095	1785	14.8%	600000	1630	22.2%	5462
B_14_50_4	2095	1500	28.4%	600000	1350	35.6%	9086
B_16_20_1	905	905	0.0%	882	905	0.0%	17
B_16_20_2	805	805	0.0%	748	805	0.0%	24
B_16_20_3	915	915	0.0%	908	915	0.0%	21
B_16_20_4	875	875	0.0%	812	875	0.0%	42
B_16_30_1	1305	1305	0.0%	120612	1260	3.4%	317
B_16_30_2	1175	1175	0.0%	6503	1160	1.3%	534
B_16_30_3	1105	1105	0.0%	1363	1105	0.0%	114
B_16_30_4	1090	1035	5.0%	600000	1005	7.8%	569
B_16_40_1	1340	1340	0.0%	1132	1340	0.0%	138
B_16_40_2	1580	1410	10.8%	600000	1385	12.3%	1419
B_16_40_3	1600	1490	6.9%	600000	1455	9.1%	2024
B_16_40_4	1615	1615	0.0%	3596	1605	0.6%	1518
B_16_50_1	2090	1835	12.2%	600000	1735	17.0%	2086

Table 3.8: Computational results Data Set B

		CP			SD-heuristic		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time
B_16_50_2	1930	1775	8.0%	600000	1705	11.7%	1169
B_16_50_3	2010	1860	7.5%	600000	1760	12.4%	2163
B_16_50_4	1980	1915	3.3%	600000	1850	6.6%	2831
B_18_20_1	820	820	0.0%	935	820	0.0%	24
B_18_20_2	740	740	0.0%	799	740	0.0%	16
B_18_20_3	775	775	0.0%	704	775	0.0%	15
B_18_20_4	840	840	0.0%	900	840	0.0%	17
B_18_30_1	1080	1080	0.0%	1727	1080	0.0%	159
B_18_30_2	1205	1205	0.0%	2009	1205	0.0%	66
B_18_30_3	1155	1155	0.0%	1683	1155	0.0%	77
B_18_30_4	1125	1125	0.0%	1784	1125	0.0%	60
B_18_40_1	1670	1670	0.0%	4303	1670	0.0%	633
B_18_40_2	1635	1635	0.0%	2409	1635	0.0%	363
B_18_40_3	1610	1610	0.0%	3895	1610	0.0%	539
B_18_40_4	1655	1525	7.9%	600000	1490	10.0%	1493
B_18_50_1	1795	1795	0.0%	586397	1740	3.1%	3333
B_18_50_2	1930	1685	12.7%	600000	1630	15.5%	9162
B_18_50_3	2005	1970	1.7%	600000	1920	4.2%	2101
B_18_50_4	1795	1795	0.0%	10385	1755	2.2%	1498
B_20_20_1	875	875	0.0%	593	875	0.0%	17
B_20_20_2	770	770	0.0%	452	770	0.0%	19
B_20_20_3	980	980	0.0%	1296	980	0.0%	19
B_20_20_4	765	765	0.0%	916	765	0.0%	16
B_20_30_1	1250	1250	0.0%	2247	1250	0.0%	71
B_20_30_2	1325	1325	0.0%	2627	1325	0.0%	178
B_20_30_3	1205	1205	0.0%	2337	1205	0.0%	91
B_20_30_4	1245	1245	0.0%	2330	1245	0.0%	84
B_20_40_1	1695	1625	4.1%	600000	1615	4.7%	1535
B_20_40_2	1725	1725	0.0%	25317	1710	0.9%	765
B_20_40_3	1540	1510	1.9%	600000	1490	3.2%	499
B_20_40_4	1530	1530	0.0%	4043	1530	0.0%	182

Table 3.8: Computational results Data Set B

		CP			SD-heuristic		
Instance	UB	Obj	Gap	Time	Obj	Gap	Time
B_20_50_1	2075	2075	0.0%	309753	2055	1.0%	1896
B_20_50_2	1825	1575	13.7%	600000	1495	18.1%	5757
B_20_50_3	1825	1825	0.0%	7912	1825	0.0%	1251
B_20_50_4	1890	1890	0.0%	2895	1890	0.0%	1159

Chapter 4

A Logic Based Benders Approach to the Concrete Delivery Problem

Abstract

The Concrete Delivery Problem (CDP) is a complex, real world optimization problem involving the allocation and distribution of concrete to construction sites. The key scheduling challenge for the CDP is the need for successive deliveries to a construction site to be sufficiently close in time. Although good results were obtained in terms of primal solutions for the CDP as introduced in the previous chapter, it remained difficult to close the optimality gap for many of the instances. In an attempt to reduce this optimality gap, this chapter presents an exact Logic Based Benders decomposition, thereby decomposing the CDP into a master problem and a subproblem. Based on a number of problem characteristics such as the availability of vehicles, geographical orientation of the customers and production centers, as well as the customers' demand for concrete, the master problem allocates concrete to customers. Next, the subproblem attempts to construct a feasible schedule, meeting all the routing and scheduling constraints. Infeasibilities in the schedule are communicated back to the master problem via a number of combinatorial inequalities (Benders cuts). The master problem is solved through a Mixed Integer Programming

The content of this chapter is based on joint work with M. Trick, see Kinable and Trick (2014).

approach, whereas the subproblem is solved via a Constraint Programming model and a dedicated scheduling heuristic. Experiments are conducted on a large number of problem instances, and compared against other exact methods presented in related literature. The resulting algorithm is capable of solving a number of previously unsolved benchmark instances to optimality and can improve the bounds for many other instances.

4.1 Introduction

In this chapter, a logic based Benders decomposition for the CDP is presented. The CDP, introduced in Chapter 3, comprises the allocation and distribution of concrete to customers, under a number of routing and scheduling constraints, while maximizing the amount of concrete delivered. Concrete is transported from production centers to the customer's construction sites by a set of heterogeneous vehicles. Often, multiple deliveries for the same customer are required as the customer's demand exceeds the capacity of a single truck. Consequently, delivery schedules for different trucks need to be synchronized as deliveries for the same customer may not overlap in time. Furthermore, successive deliveries must not differ in time too much since the concrete from an early delivery must still be liquid when a second arrives. For a detailed description of the CDP and a review of related works, refer to Chapter 3.

The logic based Benders procedure presented in this chapter decomposes the CDP into a master problem and a subproblem. The master problem (MP) allocates concrete to customers, while taking a number of resource restrictions into consideration. The subproblem (SP) attempts to find a feasible delivery schedule for the concrete trucks. A feasibility cut is generated and added to the master problem whenever no such schedule exists. The master problem and subproblem are solved iteratively, until a provable optimal (and feasible) schedule is obtained.

In the previous chapter several solution approaches for the CDP were investigated, including two exact approaches based on Mixed Integer and Constraint Programming, as well as a number of heuristic approaches. The best performance was obtained with a hybrid approach, using a dedicated scheduling heuristic, and a CP model to improve the heuristic solutions. Although good results were reported, the approach provided little insight as to the quality of the solutions. Moreover, alternative approaches to compute bounds on the optimal solution, including a Linear Programming approach could not close the optimality gap for most instances. The approach presented in this chapter addresses these issues, as bounds on the optimal solution are available through the master problem.

The CDP bears strong resemblance to a number of routing and scheduling

Parameter	Description
P	Set of concrete production sites
C	Set of construction sites, also denoted as customers. $\|C\| = n$
V	$V = C \cup \{0\} \cup \{n+1\}$
$0, n+1$	resp. the start and end depots of the trucks.
K	Set of trucks
q_i	Requested amount of concrete by customer $i \in C$
l_k	Capacity of truck $k \in K$
p_k	Time required to empty truck $k \in K$
a_i, b_i	Time window during which the concrete for customer i may be delivered.
t_{ij}	Time to travel from i to j, $i, j \in V \cup P$
γ	Maximum time lag between consecutive deliveries.
$m(i)$	Lower bound on the number of deliveries required to satisfy the demand of customer $i \in C$, i.e., $m(i) = \lceil \frac{q_i}{\max_{k \in K}(l_k)} \rceil$.
$n(i)$	Upper bound on the number of deliveries required to satisfy the demand of customer $i \in C$, i.e., $n(i) = \lceil \frac{q_i}{\min_{k \in K}(l_k)} \rceil$.

Table 4.1: Parameters defining the CDP

problems, including the Pickup-and-Delivery problem with Time-Windows and Split Deliveries and the Parallel Machine Scheduling Problem with Time Lags. Although the Benders decomposition in this work is discussed in the context of CDP, we must note that the techniques presented are not uniquely confined to this application. Moreover, Benders decompositions have recently been applied to a number of related assignment, scheduling and routing problems. Applications include Round Robin Tournament Scheduling (Rasmussen and Trick, 2007), Tollbooth Allocation (Bai and Rubin, 2009), Parallel Machine Scheduling (Tran and Beck, 2012), Lock Scheduling (Verstichel et al., 2013) and Strip Packing (Côté et al., 2013).

The remainder of this chapter is structured as follows. First, Section 4.2 presents the Benders decomposition, defining the master and subproblem in more detail, as well as their interaction. Experiments are conducted in Section 4.3, thereby comparing the Benders decomposition against some of the methods introduced in the previous chapter. Finally, Section 4.4 offers the conclusions.

4.2 A logic-based Benders decomposition

To solve the CDP defined in the previous chapter, a logic-based Benders decomposition is developed. The problem is decomposed in a master problem and a subproblem. The master problem, guided by the objective function, decides which customers are being serviced. To aid in this decision, a number of high-level problem characteristics are captured in the master problem, such as the availability of vehicles, their capacities, processing times and travel times between the customers and production centers. For a given subset of customers $\overline{C} \subseteq C$ selected by the master problem, the subproblem attempts to find a feasible delivery schedule in which the demand of all customers in $\overline{C}$ is satisfied, and all scheduling and routing constraints are met. A *feasibility* cut is generated and added to the master problem, effectively forcing the master problem to change the set $\overline{C}$, whenever no such schedule exists. When, on the other hand, a feasible solution to the subproblem exists, a provable optimal solution to the CDP problem is obtained. An overview of the solution procedure is presented in Algorithm 4.
When compared to the MIP or CP approaches presented in Chapter 3, this decomposition approach decouples the allocation of concrete to customers from the actual routing and scheduling problem. As a consequence, many of the conditional constraints (big-M constraints) can be omitted or strengthened.

Algorithm 4: Combinatorial Benders Decomposition of CDP

Output: An optimal Concrete Delivery Schedule

```
1  repeat ← true ;
2  while repeat do
3      Solve [MP];
4      get solution (y_i), i ∈ C;
5      repeat ← false ;
6      Solve [SP] for y_i, i ∈ C;
7      if [SP] is infeasible then
8          repeat ← true ;
9          add feasibility cut(s) to [MP] ;
10     else
11         get solution (ȳ, x, C);
12 return Optimal schedule (ȳ, x, C)
```

4.2.1 Master Problem

The master problem is defined through the following MIP model.

$$MP: \ max \sum_{i \in C} q_i y_i \tag{4.1}$$

$$\sum_{k \in K} l_k z_{ki} \geq q_i y_i \qquad \forall i \in C \tag{4.2}$$

$$\sum_{i \in C} z_{ki} \leq \Delta_k \qquad \forall k \in K \tag{4.3}$$

$$\sum_{i \in S} y_i \leq |S| - 1 \qquad \forall S \in \mathbb{S}, S \subseteq C \tag{4.4}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C \tag{4.5}$$

$$0 \leq z_{ki} \leq \Delta_{ki} \qquad \forall i \in C, k \in K \tag{4.6}$$

Here, boolean variables y_i, $i \in C$, denote whether customer i is serviced and integer variables z_{ki} record the number of deliveries from vehicle $k \in K$ to customer $i \in C$. The auxiliary z_{ki} variables are used to produce stronger limits on the y_i variables; they are not used in the subproblem described in Section 4.2.2.

The first constraint, (4.2), links the variables y_i and z_{ki}, $i \in C, k \in K$: a customer is satisfied if a sufficient amount of concrete is delivered. Constraints (4.3), (4.6) restrict the number of deliveries made by a single vehicle through the bounds Δ_{ki}, Δ_k, for all $k \in K$, $i \in C$. Δ_{ki}, Δ_k, are resp. bounds on the maximum number of deliveries vehicle k can make for customer i, and bounds on the total number of deliveries a vehicle k can make. Finally, Constraints (4.4) are the feasibility cuts obtained through the subproblem, prohibiting certain combinations of customers.

Δ_{ki} is calculated via Algorithm 5, whereas Δ_k is calculated via the recursive Algorithm 6. The latter algorithm utilizes a sorted array of customers; a customer $i \in C$ precedes a customer $j \in C$ in the array if $b_i < b_j \vee (b_i = b_j \wedge a_i \leq a_j)$. Computing a bound on the maximum number of deliveries a vehicle can make, is achieved by calculating a route from the starting depot 0 to the ending depot $n + 1$ through a number of customers. At each customer, the vehicle makes as many deliveries as possible. The exact number of deliveries it can make for a given customer is limited by: (1) the demand of the customer, (2) the available time to make the deliveries. In turn, the available time to perform the deliveries is limited by the time window of the customer, the processing time of the vehicle, and the time required to reload the vehicle. Furthermore,

whenever the vehicle completes its last delivery for a customer i at time t^i_{compl}, deliveries for the next customer j cannot commence before $t^i_{compl} + c_{ij}$. The recursive algorithm outlined in procedure 6 iterates over all possible subsets of customers in an efficient manner. At each iteration, the algorithm tracks the total number of deliveries made, the last location $i \in V$ visited, an index to the customer it will visit next, and the time it departed from location $i \in V$.

Algorithm 5: Calculating an upper bound on the number of deliveries vehicle $k \in K$ can make for customer $i \in C$

Input: Vehicle $k \in K$, Customer $i \in C$

```
concreteDelivered ← 0 ;
timeConsumed ← 0 ;
Δ_ki ← 0 ;
while a_i + timeConsumed + p_k ≤ b_i ∧ concreteDelivered < q_i do
    Δ_ki ← Δ_ki + 1 ;
    concreteDelivered ← concreteDelivered + l_k ;
    timeConsumed ← timeConsumed + p_k + c_{i,i} ;
return Δ_ki
```

4.2.2 Subproblem

Let $\overline{y_i}$, $i \in C$, be the optimal selection of customers obtained from problem MP, i.e., $\overline{C} = \{i \in C : \overline{y_i} = 1\}$. To assess the feasibility of this selection, a satisfiability subproblem (SP) is solved. Whenever no feasible solution to the subproblem exists, a cut is added to the Master problem:

$$\sum_{i \in \widehat{C}} y_i \leq |\widehat{C}| - 1$$

,where $\widehat{C} \subseteq \overline{C}$. The weakest cuts are obtained for $\widehat{C} = \overline{C}$. By reducing the size of $\widehat{C}$, stronger cuts may be obtained. The strongest cuts are based on Minimum Infeasible Subsets. In this context, a MIS is a subset of customers that cannot be accommodated in the same schedule; removing any of these customers from the set would however result in a subset of compatible customers. Note however that calculating a complete set of MIS is a difficult problem on its own.
The next paragraph outlines an exact procedure to establish the feasibility of a set $\overline{C}$. As this procedure is computationally expensive, we first try to solve the subproblem through the SD-heuristic (Section 3.4). Furthermore, instead of solving the subproblem for the entire set $\overline{C}$ at once, we first solve the problem for a smaller set $\widehat{C} \subset \overline{C}$. If this smaller subproblem turns out to be feasible, we

Algorithm 6: Calculating an upper bound on the number of deliveries vehicle $k \in K$ can make

Input: Vehicle $k \in K$, Array of customers sortedCustomers[], sorted.
Output: Δ_k

```
1  return maxDeliveries(k, 0, 0, 0, 0) ;
2  Function int maxDeliveries(k ∈ K, Δ_k, index, i ∈ V, complTime)
3      if index = |C| then
4          return Δ_k;
5      j ← sortedCustomers[index] ;
       /* Determine how many deliveries vehicle k can make for
          customer j                                              */
6      concreteDelivered ← 0 ;
7      startTime ← max(a_j,complTime +c_ij) ;    /* Start time 1st delivery
       for j ∈ C  */
8      timeConsumed ← 0 ;
9      deliveries ← 0 ;
10     while startTime + timeConsumed + p_k ≤ b_j ∧ concreteDelivered < q_j do
11         deliveries ← deliveries + 1 ;
12         concreteDelivered ← concreteDelivered + l_k ;
13         timeConsumed ← timeConsumed + p_k + c_jj ;
14     complTimeNew ← startTime + timeConsumed − c_jj;
15     if deliveries > 0 then
16         return max(maxDeliveries(k, Δ_k+deliveries, index +1, j,
           complTimeNew),
17                    maxDeliveries(k, Δ_k, index +1, i, complTime));
18     else
19         return maxDeliveries(k, Δ_k, index +1, i, complTime);
```

repeatedly add customers from $\overline{C}$ to $\widehat{C}$ and resolve the resulting subproblem. A modified version of the CP model (Algorithm 8, Section 5.3) may be used to establish the feasibility of a set $\overline{C}$ of customers:

Algorithm 7: CP subproblem

`Variable definitions:`

$s = \{0, 0, 0, oblig.\}$
$t = \{0, \infty, 0, oblig.\}$
$d^i_j = \{r_i, d_i, 0, oblig.\}$ $\quad \forall i \in \overline{C}, j \in \{1, \ldots, m(i)\}$
$d^i_j = \{r_i, d_i, 0, opt.\}$ $\quad \forall i \in \overline{C}, j \in \{m(i)+1, \ldots, n(i)\}$
$d^i_{j,k} = \{r_i, d_i, p_k, opt.\}$ $\quad \forall i \in \overline{C}, j \in \{1, \ldots, n(i)\}, k \in K$

`Constraints:`

forall $i \in \overline{C}$
  forall $j \in \{1, \ldots, n(i)\}$
    alternative(d^i_j, $\bigcup_{k \in K} d^i_{jk}$)
  $\sum_{k \in K} \sum_{j \in \{1,\ldots,n(i)\}} l_k \cdot$ presenceOf(d^i_{jk})$\geq q_i$
  forall $j \in \{1, \ldots, n(i)-1\}$
    endBeforeStart(d^i_j, d^i_{j+1})
    startBeforeEnd(d^i_{j+1}, d^i_j, $-\gamma$)
  forall $j \in \{m(i), \ldots, n(i)-1\}$
    $(\sum_{l \in \{1..j\}, k \in K} l_k \cdot$presenceOf($d^i_{lk}$)$< q_i) \rightarrow$ presenceOf($d^i_{j+1,k}$)
    presenceOf(d^i_{j+1})$\rightarrow$presenceOf(d^i_j)
forall $k \in K$
  noOverlapSequence($\bigcup_{i \in \overline{C}, j \in D_i} d^i_{jk} \cup s \cup t$)
  first(s, $\bigcup_{i \in \overline{C}, j \in D_i} d^i_{jk} \cup t$)
  last(t, $\bigcup_{i \in \overline{C}, j \in D_i} d^i_{jk} \cup s$)

The CP model utilizes a number of interval variables. For each interval variable, four parameters $\{a, b, d, o\}$ are specified, where a,b indicate resp. the earliest start time and latest completion time of the interval, and d is the minimum length of the interval. The last parameter o dictates whether an interval must (obligatory) or may (optional) be scheduled. For the definitions of the constraints refer to Table 3.3 in the previous chapter.
Variables d^i_j, $i \in \overline{C}, j \in \{1, \ldots, n(i)\}$, represent deliveries made for customer i. A delivery j for customer i is made if the corresponding interval variable d^i_j is present; otherwise it is absent. Variables $d^i_{j,k}$, $i \in \overline{C}, j \in \{1, \ldots, n(i)\}, k \in K$, link deliveries and the vehicles performing these deliveries. Clearly, each delivery

$j \in \{1, \ldots, n(i)\}$ for a customer $i \in \overline{C}$ can only be made by a single vehicle (Line (8)), and the amount of concrete delivered for a customer should cover its demand (Line (9)). Deliveries for the same customer may not overlap (Line (11)) and must respect a maximum time lag γ (Line (12)). Similarly, deliveries made by a single vehicle cannot overlap in time and must comply with travel times (Line (17)). Trucks must start their trip at the starting depot, represented by variable s, and must return to some ending depot identified by variable t after the deliveries are completed (Lines (18), (19)). Finally, Line (15) ensures that deliveries are made in order, and Line (14) tightens the link between the d^i_j and d^i_{jk} variables.

4.2.3 Generating an initial set of cuts

Before invoking the Benders procedure, first a number of initial cuts are computed and added to thet set $\mathbb{S}$ in the master problem. These cuts are generated by enumerating all Minimum Infeasible Subsets consisting of up to three customers. In addition, the best-fit heuristic (Algorithm 2 in Chapter 3) may be used to compute an additional set of cuts. The constructive heuristic is initialized with an ordered set of customers. The heuristic schedules deliveries for these customers in an iterative fashion, where the time of a delivery and the vehicle performing the delivery are determined via a number of heuristic criteria. If at some point, the heuristic fails to schedule the next delivery for a customer due to the lack of an available vehicle, the heuristic would normally remove this customer from the schedule and continue with the next customer. Instead of simply removing the customer from the schedule, an additional check is performed. Given the subset of customers $\widehat{C}$ which have received concrete in the partially completed heuristic schedule, an exact algorithm, e.g., the CP model from Section 4.2.2, is used to determine whether there exists a feasible solution where each of the customers in $\widehat{C}$ is satisfied. If no such schedule exist, a cut is generated for the customers in $\widehat{C}$ and the heuristic removes the customer it could not satisfy from the schedule. If, on the other hand, the exact approach is capable of finding a feasible schedule satisfying all customers in $\widehat{C}$, the heuristic continues from the schedule generated by the exact method.
Naturally, the approach outlined in the previous paragraph may be repeated for several orderings of the customers.

4.3 Computational Experiments

To assess the performance of the Benders procedure, a number of experiments are conducted on the data sets introduced in Section 3.6.1 (available online

(Kinable and Wauters, 2013)). The results of these experiments are reported in Tables 4.2, and 4.3. In these tables, the first column provides the instance name, following a "$W\_x\_y\_z$" naming convention, where W identifies the data set, x is the number of vehicles, y is the number of customers and z is the number of concrete production stations. For each instance, we computed an initial feasible solution using the CP procedure from Section 3.3.2. These solutions, reported in the second column, are used to warm-start the Benders procedure. The next 5 columns provide data on our Benders procedure:

- obj: The objective of the best (feasible) solution obtained through the Benders procedure.
- iCuts: The number of cuts added initially (Section 4.2.3)
- cuts: The number of cuts added during the Benders procedure (Section 4.2.2)
- c-time: The Time required to obtain the initial master problem, in seconds. This time is limited to 5 minutes, excluding the generation of the Minimum Infeasible subsets.
- s-time: The Time required to solve the Benders problem. For data set A, this time is limited to 10 minutes, and for data set B 15 minutes.

In Section 3.6 bounds on the optimal solutions are reported. These bounds are computed through four different procedures, but for each instance only the strongest bound is provided. The four different procedures are:

- Optimal MIP solution (when available)
- Optimal CP solution (when available)
- LP relaxation, strengthened with cuts from all Minimum Infeasible Subsets (MIS) of size 2 (Section 3.5).
- Solution to the MIP problem consisting of the objective function (4.1) and all cuts generated from MIS of size k (Constraint (4.4)), where $k = 3$ for data set A (Section 3.5).

The strongest of these bounds is reported in column 'bound*'. Column 'LP' gives the objective value of the LP relaxation of the MIP model (Equations (3.21)-(3.34) in Chapter 3), strengthened with the default cuts added by Cplex 12.5 and all MIS of up to size 3. The gaps are computed with respect to the objective in column *obj* [1]. Finally, the last two columns in the table provide

[1]Note: the computations in this chapter are performed on a different machine than the computations from Chapter 3; to ensure consistency, all computations in a chapter are performed on the same machine. However, as a consequence, the reported gaps in this chapter may be different from the gaps in the previous chapter.

the bounds obtained through the Benders procedure.
When comparing the bounds attained through our Benders procedure with the LP-bounds, we can observe that the LP based bounds are significantly weaker. The average gap between the LP bounds and the best primal solutions amounts to 9.26%, whereas the Benders procedure produces an average gap of 1.81%. This gap is also smaller than the average gap (3.62%) obtained from the bounding procedures from Chapter 3. In fact, none of the bounds computed through the Benders procedure are weaker than the bounds reported in column bound*.
In summary, we reduced the average gap for the instances in data set A from 3.72% to 1.81%, 19 instances had their bounds improved, and 11 new optimal solutions were found. For almost 65% of the instances, the optimal solutions were already obtained at the first iteration of the master problem, i.e., no additional cuts had to be generated.
The instances in data set B are significantly harder to solve than the instances in data set A. Table 4.3 shows the results obtained for the instances up to 10 vehicles; no improvements could be made to the remaining instances. For the instances in Table 4.3, the average gap induced by the Benders bounds is 10.08%, opposed to 11.32% from the bounding procedures discussed in Chapter 3. Furthermore, optimality was attained for two additional instances; 21 instances had their bounds improved.
Future attempts to improve the Benders decomposition approach should be targeted at improving the runtime of the subproblem, as this procedure takes the largest amount of time. Especially for the larger instances, solving the subproblem is challenging.

4.4 Conclusion

In this chapter, a Logic Based Benders decomposition which decouples the CDP into a master problem and a subproblem is presented. The master problem allocates concrete to customers, whereas the subproblem handles the routing and scheduling of the concrete delivery trucks. By decomposing the problem, part of the complexity is shifted to the subproblem. Furthermore, dedicated procedures may be used to solve these problems. Here, we solve the master problem through MIP, whereas the subproblem is solved through a dedicated scheduling heuristic and a CP model discussed in Chapter 3. Because the subproblem does not have to deal with the allocation of concrete as this is being handled by the master problem, we simplified and strengthened the latter CP model, thereby significantly improving its performance.
Computing bounds for CDP is a non-trivial task. Linear Programming-based bounds are generally very weak (see Section 3.6), as the problem has a large

number of conditional constraints. Furthermore, exact approaches based on CP often provide little insight as to the quality of the solution. Our Benders decomposition may however provide a viable alternative. Extensive computational tests show that the bounds computed through the Benders decomposition are consistently stronger than the bounds from Section 3.6, which where obtained by aggregating the bounds of four different procedures. By improving the bounds for a large number of instances, and simultaneously improving several primal solutions, we were able to solve a number of previously unsolved benchmark instances to optimality. To further enhance the performance of this Benders procedure, one would have to find a way to speed up the subproblem. One possible direction would be to decouple the subproblem even further, for example by fixing more variables in the master problem. Or, one could try to replace the current subproblem by a relaxation of the exact subproblem, which is easier to solve. Using this relaxation, it could still be possible to add a number of cuts to the master problem, thereby refining the master problem, with significantly less computational effort. Finally, one could also look into alternative decomposition approaches for CDP. One possible example is provided in Appendix A.

Table 4.2: Computational results Data Set A

		Benders decomposition					LP		Bound*		Benders	
Instance	iObj	obj	iCuts	cuts	c-time	s-time	bound	gap	bound	gap	bound	gap
A_2_5_1	85	85	28	0	1	0	85	0%	85	0%	85	0%
A_2_5_2	160	160	13	0	0	0	160	0%	160	0%	160	0%
A_2_5_3	105	105	26	0	0	0	105	0%	105	0%	105	0%
A_2_5_4	105	105	3	0	300	0	105	0%	105	0%	105	0%
A_2_10_1	50	50	142	0	3	0	50	0%	50	0%	50	0%
A_2_10_2	150	150	215	0	9	0	150	0%	150	0%	150	0%
A_2_10_3	220	220	154	0	3	0	230	4%	220	0%	220	0%
A_2_10_4	150	150	179	0	9	0	165	9%	150	0%	150	0%
A_2_15_1	215	215	567	0	10	0	225	4%	215	0%	215	0%
A_2_15_2	290	290	373	2	25	2	320	9%	320	9%	**290**	**0%**
A_2_15_3	205	205	502	0	77	0	215	5%	205	0%	205	0%
A_2_15_4	255	255	348	0	11	0	300	15%	255	0%	255	0%
A_2_20_1	255	255	549	0	38	0	260	2%	255	0%	255	0%
A_2_20_2	270	270	575	2	33	0	285	5%	270	0%	270	0%
A_2_20_3	260	260	651	0	15	0	280	7%	260	0%	260	0%
A_2_20_4	355	355	36	5	360	1	490	28%	380	7%	**355**	**0%**
A_3_5_1	205	205	0	0	3	0	205	0%	205	0%	205	0%
A_3_5_2	115	115	9	0	1	0	115	0%	115	0%	115	0%
A_3_5_3	125	125	0	0	0	0	125	0%	125	0%	125	0%
A_3_5_4	190	190	0	0	0	0	190	0%	190	0%	190	0%
A_3_10_1	205	205	100	0	5	0	205	0%	205	0%	205	0%
A_3_10_2	230	230	80	0	14	0	230	0%	230	0%	230	0%
A_3_10_3	305	305	114	0	23	0	305	0%	305	0%	305	0%
A_3_10_4	300	300	115	0	5	0	300	0%	300	0%	300	0%
A_3_15_1	330	330	485	0	35	0	330	0%	330	0%	330	0%
A_3_15_2	395	395	268	3	18	1	530	25%	425	7%	**395**	**0%**
A_3_15_3	290	290	378	23	49	7	430	33%	330	12%	**290**	**0%**

Table 4.2: Computational results Data Set A

		Benders decomposition					LP		Bound*		Benders	
Instance	iObj	obj	iCuts	cuts	c-time	s-time	bound	gap	bound	gap	bound	gap
A_3_15_4	440	440	10	15	306	49	550	20%	475	7%	**440**	**0%**
A_3_20_1	340	340	667	1	350	157	410	17%	345	1%	**340**	**0%**
A_3_20_2	415	415	28	0	516	0	510	19%	415	0%	415	0%
A_3_20_3	355	**360**	48	0	324	0	425	15%	360	0%	360	0%
A_3_20_4	480	480	31	0	319	0	590	19%	480	0%	480	0%
A_4_5_1	140	140	0	0	0	0	140	0%	140	0%	140	0%
A_4_5_2	150	150	0	0	0	0	150	0%	150	0%	150	0%
A_4_5_3	165	165	0	0	0	0	165	0%	165	0%	165	0%
A_4_5_4	230	230	0	0	0	0	230	0%	230	0%	230	0%
A_4_10_1	310	310	114	0	46	0	310	0%	310	0%	310	0%
A_4_10_2	370	370	48	0	4	0	390	5%	370	0%	370	0%
A_4_10_3	375	375	85	1	42	2	470	20%	445	16%	**375**	**0%**
A_4_10_4	285	285	23	0	1	0	285	0%	285	0%	285	0%
A_4_15_1	415	415	7	2	307	600	570	27%	545	24%	545	24%
A_4_15_2	475	475	4	18	312	600	650	27%	610	22%	**520**	**9%**
A_4_15_3	430	430	368	0	99	0	495	13%	450	4%	**430**	**0%**
A_4_15_4	490	490	5	0	304	600	515	5%	515	5%	515	5%
A_4_20_1	525	525	3	8	302	600	660	20%	585	10%	**575**	**9%**
A_4_20_2	425	425	497	4	38	1	490	13%	440	3%	**425**	**0%**
A_4_20_3	375	375	40	12	828	600	530	29%	425	12%	**405**	**7%**
A_4_20_4	465	465	18	17	1801	3	590	21%	500	7%	**465**	**0%**
A_5_5_1	200	200	0	0	0	0	200	0%	200	0%	200	0%
A_5_5_2	200	200	0	0	0	0	200	0%	200	0%	200	0%
A_5_5_3	220	220	0	0	0	0	220	0%	220	0%	220	0%
A_5_5_4	175	175	9	0	2	0	175	0%	175	0%	175	0%
A_5_10_1	350	350	0	0	0	0	350	0%	350	0%	350	0%
A_5_10_2	345	345	0	0	0	0	345	0%	345	0%	345	0%

Table 4.2: Computational results Data Set A

		Benders decomposition					LP		Bound*		Benders	
Instance	iObj	obj	iCuts	cuts	c-time	s-time	bound	gap	bound	gap	bound	gap
A_5_10_3	285	285	25	0	5	0	300	5%	285	0%	285	0%
A_5_10_4	380	380	0	0	0	0	380	0%	380	0%	380	0%
A_5_15_1	455	455	2	4	308	600	590	23%	590	23%	**510**	**11%**
A_5_15_2	580	580	0	0	301	600	695	17%	695	17%	695	17%
A_5_15_3	350	350	2	3	576	600	435	20%	395	11%	**385**	**9%**
A_5_15_4	500	500	3	0	1218	600	600	17%	520	4%	520	4%
A_5_20_1	700	705	252	61	245	536	900	22%	760	7%	**705**	**0%**
A_5_20_2	555	555	15	4	388	600	810	31%	645	14%	**630**	**12%**
A_5_20_3	595	595	8	16	302	600	695	14%	645	8%	**615**	**3%**
A_5_20_4	520	520	13	2	355	600	705	26%	560	7%	**557.5**	**7%**
AVG								9.26%		3.72%		1.81%
Optimal								28		41		52

Table 4.3: Computational results Data Set B

		Benders decomposition					LP		Bound*		Benders	
Instance	iObj	obj	iCuts	cuts	c-time	s-time	bound	gap	bound	gap	bound	gap
B_6_20_1	725	725	0	102	301	356	805	10%	805	10%	**725**	**0%**
B_6_20_2	700	700	0	31	300	900	855	18%	855	18%	**830**	**16%**
B_6_20_3	675	675	0	0	300	900	760	11%	760	11%	760	11%
B_6_20_4	615	615	0	29	300	900	705	13%	705	13%	**680**	**10%**
B_6_30_1	860	860	0	1	300	900	1300	34%	1300	34%	**1290**	**33%**
B_6_30_2	850	850	0	34	300	900	1140	25%	1140	25%	**1105**	**23%**
B_6_30_3	725	725	0	120	301	900	1060	32%	1060	32%	**1035**	**30%**
B_6_30_4	625	625	2	1	302	900	1000	38%	1000	38%	**913.33**	**32%**
B_6_40_1	835	835	0	23	302	900	1545	46%	1545	46%	**1431.11**	**42%**
B_6_40_2	1045	1045	0	33	301	900	1635	36%	1635	36%	**1473.75**	**29%**
B_6_40_3	735	735	11	79	315	900	1570	53%	1570	53%	**1191.67**	**38%**
B_6_40_4	750	750	2	9	305	900	1450	48%	1450	48%	**1095**	**32%**
B_6_50_1	1010	1010	0	22	305	900	1890	47%	1890	47%	**1490**	**32%**
B_6_50_2	955	955	2	25	307	900	2250	58%	2250	58%	**1495**	**36%**
B_6_50_3	920	920	1	16	311	900	1740	47%	1740	47%	**1295**	**29%**
B_6_50_4	1000	1000	1	0	385	900	2080	52%	2080	52%	**1395**	**28%**
B_8_20_1	920	920	0	1	300	897	935	2%	935	2%	**920**	**0%**
B_8_20_2	850	850	0	0	300	900	865	2%	865	2%	865	2%
B_8_20_3	655	655	0	0	293	0	655	0%	655	0%	655	0%
B_8_20_4	820	820	0	0	69	0	820	0%	820	0%	820	0%
B_8_30_1	920	920	0	0	300	900	1085	15%	1085	15%	1085	15%
B_8_30_2	1005	1005	0	0	300	900	1115	10%	1115	10%	1115	10%
B_8_30_3	975	975	0	4	300	900	1155	16%	1155	16%	**1140**	**14%**
B_8_30_4	1110	1110	0	0	300	900	1320	16%	1320	16%	1320	16%
B_8_40_1	1180	1180	0	1	301	900	1665	29%	1665	29%	**1655**	**29%**
B_8_40_2	1190	1190	0	6	300	900	1415	16%	1415	16%	1415	16%
B_8_40_3	995	995	0	0	300	900	1495	33%	1495	33%	1495	33%

Table 4.3: Computational results Data Set B

		Benders decomposition					LP		Bound*		Benders	
Instance	iObj	obj	iCuts	cuts	c-time	s-time	bound	gap	bound	gap	bound	gap
B_8_40_4	1105	1105	0	0	301	900	1730	36%	1730	36%	1730	36%
B_8_50_1	1130	1130	0	11	323	901	1980	43%	1980	43%	**1925**	**41%**
B_8_50_2	1150	1150	0	0	304	900	1935	41%	1935	41%	1935	41%
B_8_50_3	1210	1210	0	0	302	900	1960	38%	1960	38%	1960	38%
B_8_50_4	1105	1105	0	0	303	900	1835	40%	1835	40%	**1740**	**36%**
B_10_20_1	805	805	0	0	116	0	805	0%	805	0%	805	0%
B_10_20_2	825	825	0	0	237	0	825	0%	825	0%	825	0%
B_10_20_3	730	730	0	0	4	0	730	0%	730	0%	730	0%
B_10_20_4	765	765	0	0	1	0	765	0%	765	0%	765	0%
B_10_30_1	910	910	0	0	300	900	1215	25%	1215	25%	1215	25%
B_10_30_2	1170	1170	0	0	300	900	1355	14%	1355	14%	1355	14%
B_10_30_3	1135	1135	0	0	300	900	1210	6%	1210	6%	1210	6%
B_10_30_4	1165	1165	0	0	300	900	1235	6%	1235	6%	1235	6%
B_10_40_1	1210	1210	0	0	301	900	1475	18%	1475	18%	1475	18%
B_10_40_2	1485	1485	0	0	301	900	1580	6%	1580	6%	1580	6%
B_10_40_3	1375	1375	0	0	302	900	1605	14%	1605	14%	1605	14%
B_10_40_4	1365	1365	0	0	301	900	1455	6%	1455	6%	1455	6%
B_10_50_1	1425	1425	0	0	308	900	2265	37%	2265	37%	2265	37%
B_10_50_2	1010	1010	0	0	302	900	1900	47%	1900	47%	**1745**	**42%**
B_10_50_3	1260	1260	0	0	302	900	2005	37%	2005	37%	2005	37%
B_10_50_4	1455	1455	0	0	303	901	1925	24%	1925	24%	1925	24%
AVG								23.83%		23.83%		20.51%
Optimal								6		6		8

Chapter 5

Integrating CP, LP and Decision Diagrams for the Time-Dependent TSP

Abstract

This chapter presents an integrated method to solve the Time-Dependent TSP, a variation on the well-known TSP where travel times between two cities depend on the order the cities are visited. The method proposed relies on a generic Constraint Programming (CP) approach. To strengthen the CP model, both continuous and discrete relaxations of the problem are added. The discrete relaxation is obtained via a Multivalued Decision Diagram (MDD). Through the MDD, bounds on the optimum solution can be computed, enabling efficient pruning of the search space, which improves the domain propagation for the CP model. This chapter furthermore shows how MDDs can be used to consolidate the CP model by integrating structural information from other problem relaxations, such as a Linear Programming relaxations through the use of additive bounding. Experiments indicate that the integrated method outperforms traditional methods based on pure CP or Mixed Integer Programming. The proposed method is sufficiently generic to render it applicable to a variety of related sequencing problems.

The content of this chapter is based on joint work with A. Cire and W-J. Van Hoeve, Tepper School of Business, Carnegie Mellon University.

5.1 Introduction

The Time-Dependent Traveling Salesman Problem (TD-TSP) is a generalization of the Traveling Salesman Problem (TSP) where the travel time between cities depends on the order in which the cities are visited (sequence-dependent) or varies over time (time-dependent). The sequence-dependent variant has applications in the domain of machine scheduling (Picard and Queyranne (1978); Bigras et al. (2008)), whereas the time-dependent case naturally extends to routing problems taking common travel patterns such as congestions or delays during rush hours into consideration (e.g., Cordeau et al. (2012)). In this work the focus is on the sequence-dependent variant; unless stated otherwise, any future references to the TD-TSP implicitly assume the sequence-dependent variant.

The TD-TSP problem involves finding a Hamiltonian path of minimal cost through a sequence of $n+1$ cities, where the travel time between two cities i, j is given by $\delta_{i,j,t}$, where t is the position of city i in the sequence. In contrast to the TSP problem, for which instances of up to 100,000 nodes are solved to optimality, only recently attempts were made to solve TD-TSP instances containing up to 100 cities (Abeledo et al., 2013). In the past, several solution approaches for the TD-TSP problem have been proposed, including a number of approaches relying on Mixed Integer Programming (MIP). For an overview, refer to Picard and Queyranne (1978); Gouveia and Voss (1995); Miranda-Bront et al. (2010); Abeledo et al. (2013). A disadvantage of these MIP models is that it is notoriously difficult to incorporate additional constraints such as precedence relations or time windows, without deteriorating the quality of the models, due to the use of conditional (big-M) constraints. In this work, we study a Constraint Programming (CP) based approach for the TD-TSP problem. CP is well-known for solving complex scheduling problems, including problems with a large number of precedence relations, and has a number of advantages over MIP in terms of expressiveness and search heuristics. However, in general pure CP is not very suitable for solving sequencing problems like the TSP. This is an immediate consequence of the fact that the problem structure, expressed in terms of global constraints, is entirely decoupled from the objective function; due to the lack of effective bounding procedures, the CP search recedes to an inefficient exhaustive enumeration of the search space. To counteract these issues, Multivalued Decision Diagrams (MDDs) have been incorporated in the TD-TSP CP model through the use of a dedicated global constraint, which has recently been proven highly effective for other sequencing problems (Ciré and van Hoeve, 2013). MDDs provide a compact encoding of the solution space through which both lower and upper bounds on the optimum solution can be computed. Consequently, MDDs enable computing bounds on the objective value at each node of the CP search tree, thereby providing a means to effectively

prune the search tree and to guide the CP search. Moreover, the structural information encoded in the MDDs is used to perform domain propagation.
For many optimization problems, different relaxations are available based on various combinatorial structures of the underlying problem. Fischetti and Toth (1989) proposed an additive bounding procedure which combines these different relaxations to obtain a valid lower bound for the problem at hand. The resulting bound is generally stronger than the strongest bound obtainable from the individual relaxations. In this work, we will demonstrate how additive bounding can be used to include bounds obtained through for example Linear Programming relaxations into the MDDs. The resulting approach is interesting because it opens up opportunities to integrate well-studied relaxations for sequencing problems such as LP relaxations, the Held-Karp relaxation (Held and Karp, 1970), Assignment Relaxation, Lagrangian relaxations into the general framework of CP.

The remainder of this chapter is structured as follows. First section 5.2 gives a formal description of the TD-TSP. Next, a number of mathematical models, including two MIP models and a CP model, are provided in Section 5.3. Section 5.4 introduces MDDs and discusses how they are integrated in a CP model for the TD-TSP. Next, Section 5.5 focuses on additive bounding, and how to consolidate the bounds into the MDD. Computational experiments are conducted in Section 5.6, thereby comparing the MIP and CP approaches. Finally, Section 5.9 reflects on some of the limitations and issues encountered in the current implementation of the MDDs and the CP model. Section 5.10 offers the conclusion.

5.2 The TD-TSP problem description

Similar to related works, the TD-TSP in this chapter is treated as a scheduling problem, instead of a routing problem. Let $J = \{0\} \cup \{1, \ldots, n-1\} \cup \{n\}$ be a set of $n+1$ jobs that need to be scheduled on a single machine, where job 0 must be the first job, and job n must be the last job in the sequence. Between each pair of jobs $i, j \in J$, a sequence dependent setup time $\delta_{(i,j,t)}$, $0 \leq t < n$, is specified. In contrast to traditional scheduling problems, the setup time does not only depend on the jobs i, j but also on the position t when the job is performed. The TD-TSP problem may be modeled on a Time-Space network as shown in Figure 5.1, where each node (i, t) represents a job $i \in J$ performed at position t. Note that in the TD-TSP, in contrast to the traditional TSP problem, the arc costs on arcs $((i, t_1), (j, t_1 + 1))$ and $((i, t_2), (j, t_2 + 1))$ are not necessarily the same.

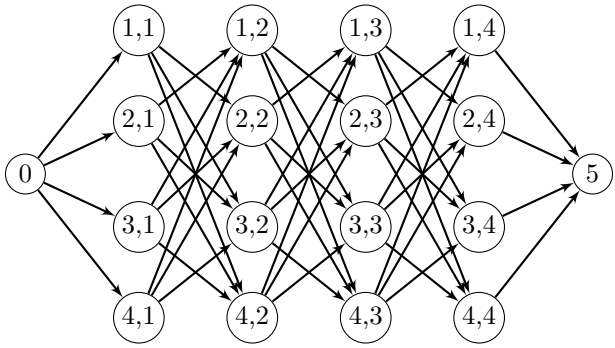


Figure 5.1: Time Space Network (source: Picard and Queyranne (1978))

5.3 Mathematical Models

This Section formally defines the TD-TSP problem through a number of mathematical models. First a CP model is presented, followed by a number of MIP models.

5.3.1 Constraint Programming Model

The TD-TSP problem can be straightforwardly defined through the following CP model:

Algorithm 8: CP model for TD-TSP

`Variable definitions:`

1 $\pi_0 = 0$
2 $\pi_j \in \{1, \ldots, n-1\} \qquad \forall j = \{1, \ldots, n-1\}$
3 $\pi_n = n$
4 $obj \in \{0, \ldots, \infty\}$

`Objective:`

5 **Min** obj

`Constraints:`

6 $obj = \sum_{i=0}^{n-1} \delta_{(\pi_i, \pi_{i+1}, i)}$
7 AllDiff$(\pi_0, \pi_1, \ldots, \pi_n)$

The permutation variables π_i, $i = 0, \ldots, n$ represent the jobs performed at each position i in the sequence. The AllDiff constraint ensures that each job is performed exactly once. Note that $\delta_{(\pi_i, \pi_{i+1}, i)}$ uses variables as subscripts. These are efficiently handled by 'element' constraints in CP.

5.3.2 Mixed Integer Programming models

The following MIP model, first presented by Picard and Queyranne (1978), is a network flow model on the Time-Space network depicted in Figure 5.1.

$$MIP1: min \sum_{j=1}^{n} \delta_{(0,j,0)} x_{0,j}^{0} + \sum_{t=1}^{n-2} \sum_{i=1}^{n-1} \sum_{\substack{j=1 \\ j \neq i}}^{n-1} \delta_{(i,j,t)} x_{ij}^{t} + \tag{5.1}$$

$$\sum_{i=1}^{n} \delta_{(i,n,n-1)} x_{i,n}^{n-1}$$

$$\text{s.t.} \sum_{j=1}^{n-1} x_{0j}^{0} = 1 \tag{5.2}$$

$$x_{0j}^{0} = \sum_{\substack{k=1 \\ k \neq j}}^{n} x_{jk}^{1} \qquad \forall j = 1, \ldots, n-1 \tag{5.3}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n-1} x_{ij}^{t} = \sum_{\substack{k=1 \\ k \neq j}}^{n-1} x_{jk}^{t+1} \qquad \forall t = 1, \ldots, n-3, \;\; j = 1, \ldots, n-1 \tag{5.4}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n-1} x_{ij}^{n-2} = x_{j,n}^{n-1} \qquad \forall j = 1, \ldots, n-1 \tag{5.5}$$

$$x_{0j}^{0} + \sum_{t=1}^{n-1} \sum_{\substack{i=1 \\ i \neq j}}^{n-1} x_{ij}^{t} = 1 \qquad \forall j = 1, \ldots, n-1 \tag{5.6}$$

$$x_{0j}^{0} \in \{0,1\} \qquad \forall j = 1, \ldots, n-1, \tag{5.7}$$

$$x_{i,n}^{n-1} \in \{0,1\} \qquad \forall i = 1, \ldots, n-1, \tag{5.8}$$

$$x_{ij}^{t} \in \{0,1\} \qquad \forall i, j = 1, \ldots, n-1, i \neq j, \;\; t = 1, \ldots, n-2 \tag{5.9}$$

In this model binary variables x_{ij}^{t} represent whether job $j \in J$ is performed immediately after job $i \in J$, which is performed at time t. Constraint (5.2) ensures that flow leaves the source node. Next, Constraints (5.3)-(5.5) implement flow preservation. Finally, Constraints (5.6) ensure that each job is only

performed once.
Several possible approaches to strengthen this model are presented in Gouveia and Voss (1995) and Abeledo et al. (2013). Most notably, the Time-Dependent TSP can be mapped to the traditional time-independent TSP through the following relation:

$$x_{ij} = \sum_{t=1}^{n} (x_{ij}^t + x_{ji}^t) \qquad \forall i, j = 0, \ldots, n \tag{5.10}$$

Consequently, all valid inequalities known for the TSP can also be used to strengthen this TD-TSP model.
A second MIP model, shown to be stronger than the aforementioned model (Gouveia and Voss, 1995), is as follows:

$$\begin{aligned} MIP2: min \ & \sum_{j=1}^{n-1} \delta_{(0,j,0)} y_{j,1} + \sum_{i=1}^{n-1} \sum_{\substack{j=1 \\ j \neq i}}^{n-1} \sum_{t=1}^{n-2} \delta_{(i,j,t)} x_{ij}^t + \\ & \sum_{i=1}^{n-1} \delta_{(i,n,n-1)} y_{i,n-1} \end{aligned} \tag{5.11}$$

$$\text{s.t.} \ \sum_{t=1}^{n-1} y_{it} = 1 \qquad \forall i = 1, \ldots, n-1 \tag{5.12}$$

$$y_{it} = \sum_{\substack{j=1 \\ j \neq i}}^{n-1} x_{ij}^t \qquad \forall i = 1, \ldots, n-1, \ \ t = 1, \ldots, n-2 \tag{5.13}$$

$$y_{jt} = \sum_{\substack{i=1 \\ i \neq j}}^{n-1} x_{ij}^{t-1} \qquad \forall j = 1, \ldots, n-1, \ \ t = 2, \ldots, n-1 \tag{5.14}$$

$$y_{it} \in \{0, 1\} \qquad \forall i = 1, \ldots, n-1, \ \ t = 1, \ldots, n-1 \tag{5.15}$$

$$x_{ij}^t \in \{0, 1\} \qquad \forall i, j = 1, \ldots, n-1, i \neq j, \ \ t = 1, \ldots, n-2 \tag{5.16}$$

In this model, the binary variables x_{ij}^t are identical to the ones in model MIP1. In addition, there are binary variables y_{it} indicating whether job $i \in J$ is performed at time t. Constraints (5.12) ensure that all jobs are executed. Constraints (5.13), (5.14) link the variables.

5.3.3 Column Generation Model

Another model for the TD-TSP, frequently occurring in literature (Picard and Queyranne, 1978; Abeledo et al., 2013; Bigras et al., 2008), is as follows:

$$MIP3 : min \sum_{p} \delta_p z_p \tag{5.17}$$

$$\sum_{p} a_i^p z_p = 1 \qquad \forall i = 1, \ldots, n \tag{5.18}$$

$$z_p \in \{0, 1\} \qquad \forall p \in P \tag{5.19}$$

Set P is the set of all *feasible* source-sink (s-t) paths on the Time-Space network (Figure 5.1), i.e., all paths representing a valid sequence of jobs. Binary variables z_p indicate whether path $p \in P$ is selected. Parameters δ_p record the cost (total travel time) incurred on path $p \in P$, and parameter a_i^p denotes whether a particular job i is present on path p. Clearly, in an optimal solution to MIP3, only one z_p variable equals one, whereas all the other variables equal zero. Due to the exponentially large set of paths P, this problem is typically solved through column generation. Abeledo et al. (2013); Bigras et al. (2008) consider a relaxed version of this model in which set P may also contain paths visiting the same job multiple times. As a consequence, the pricing subproblem consists of finding the shortest (s-t) path, which can be efficiently accomplished by a dynamic program in polynomial time. To strengthen the model, only paths without k-cycles, for $k = 2$, are generated. Furthermore a number of valid inequalities are added to the master problem.

5.4 Reinforcing the CP model

Although the CP model presented in Section 5.3.1 offers a valid formulation for the TD-TSP, the model is too weak to solve any reasonably sized problem instances. This can be attributed to a number of causes:

1. Each constraint in CP is comparable to a black box, implementing its own domain filtering mechanism. Communication between constraints is solely achieved through the variable domains. All structural relationships among variables are projected onto the variable domains. Since variable domains are merely represented as a set of numbers, a lot of structural information is lost and communication between constraints will be very limited.

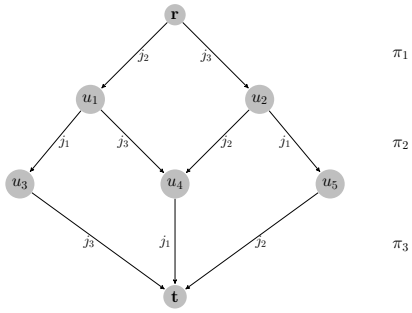


Figure 5.2: Example of an MDD (Source: Ciré and van Hoeve (2013)).

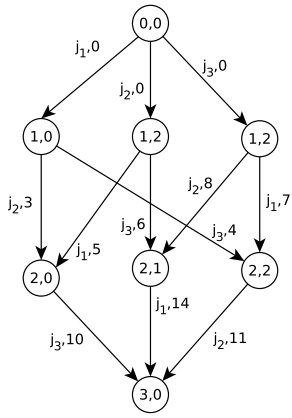


Figure 5.3: Calculating sum of setup times on an MDD. The π_i, $i = 1, 2, 3$ variables are the permutation variables associated with each layer. Each arc is associated with a job (j_1, j_2, j_3). In addition, the cost of traversing an arc is stated for each arc.

2. The AllDiff constraint only ensures that each job is scheduled exactly once, but does not take the objective function into consideration, limiting the effectiveness of the domain filtering.

3. The CP solver has little to no means to derive strong bounds on the optimum solution, prohibiting effective pruning of the CP search tree.

To alleviate these issues, Ciré and van Hoeve (2013) have proposed a method to strengthen CP models for sequencing problems with Decision Diagrams, thereby significantly improving the solver's performance on a variety of sequencing problems. Extending on this work, Section 5.4.1 discusses Decision Diagrams in the context of TD-TSP. Next, Section 5.4.2 presents an extended CP model, strengthened by the MDDs.

5.4.1 Decision Diagrams for the TD-TSP

A Multivalued Decision Diagram (MDD) is a directed acyclic graph in which the nodes are partitioned into $n + 1$ layers $L = L_0, \ldots, L_n$. An example of an MDD is given in Figure 5.2. Each arc $a \in \mathcal{A}$ in the graph is associated with a

particular job $j \in J$, denoted $val(a)$. A path from the root node s in layer L_0 to the sink node t in layer L_n encodes a sequence of jobs. The number of nodes in a layer $|L_i|$ denotes the width of the layer; the width of the MDD is the maximum width among all layers. An MDD is *exact* if each $s-t$ path corresponds to a feasible sequence, and all feasible paths are encoded in the graph. An MDD is *relaxed* if, next to all feasible paths, also a number of infeasible paths are present, i.e., a relaxed MDD encodes a superset of all feasible orderings.
In an exact MDD, the optimal ordering with respect to the objective that minimizes the total sum of time-dependent setup times (Equation (5.1)), can be found in polynomial time in the size of the MDD (Ciré and van Hoeve, 2013). Let $val(a) \in J$ be the job associated with a particular arc $a = (u, v)$. Furthermore, $\ell(a) \in L$ is the layer containing the tail of arc a, $in(u)$ the set of arcs incoming in node u, $out(u)$ the set of arcs leaving node u. Note that there is a one-to-one correspondence between the position of a job in the sequence, and the layers: an arc $a = (u, v)$ in an (s, t) path corresponds to a job $val(a)$ at position $\ell(a)$ in the sequence. Recall that $\delta_{i,j,t}$ represents the time required to travel from i to j when city i occurs at position t in the sequence. A cost c_a, associated with each arc $a = (u, v)$, is defined by the following recursive relation:

$$c_a = \begin{cases} 0 & \text{if } l(a) = L_0 \\ \min\limits_{a' \in in(u)} \{c_{a'} + \delta_{(val(a'),val(a),l(a'))}\} & \text{otherwise} \end{cases} \tag{5.20}$$

The minimum sum of time-dependent setup times is given by:

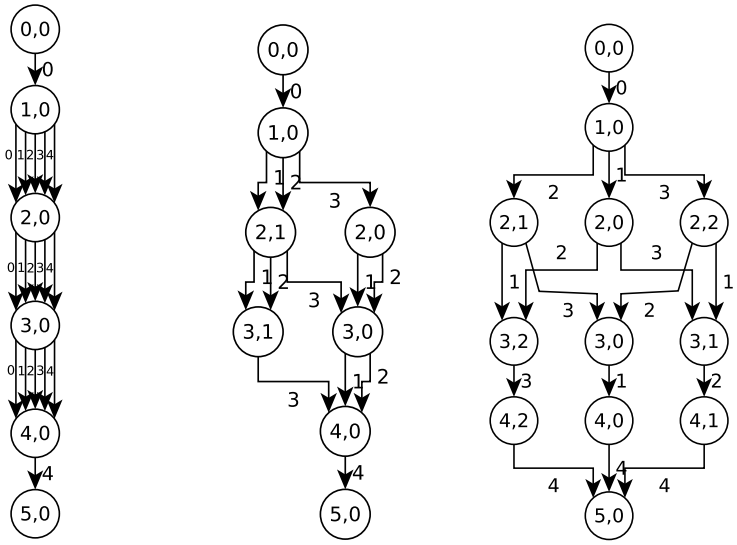
$$\min_{a \in in(t)} c_a \tag{5.21}$$

The optimal path corresponding with this objective cost can be recovered by traversing the tree from t to s, thereby following back the arcs that were selected in Equation (5.20).

Example Let the time-dependent setup times $\delta_{(i,j,t)}$, $i, j \in J$, $t = 0, 1$ for a set of jobs $J = \{j_1, j_2, j_3\}$ be given by the following relation: $\delta_{(i,j,t)} = \delta_{(i,j)} + t + 1$, where $\delta_{(i,j)}$ are the setup times defined as:

	j_1	j_2	j_3
j_1	-	2	3
j_2	4	-	5
j_3	6	7	-

The exact MDD in Figure 5.3 specifies for each arc $a = (u, v)$ the $val(a)$ and c_a, where c_a is calculated via Equation (5.20). For example, the cost of arc $((1, 0), (2, 0))$ is given by $\delta_{(1,2,0)} = 3$, and the cost of arc $((2, 0), (3, 0))$ is given



(a) Relaxed MDD (width=1)

(b) Relaxed MDD (width=2)

(c) Exact MDD (width=3)

Figure 5.4: MDDs of different width

by $\min\{\delta_{(2,3,1)} + 3, \delta_{(1,3,1)} + 5\} = 10$. The optimal path equals $\pi = 1, 2, 3$ and has objective value 10.

Typically, an MDD provides a compact representation of all feasible orderings of jobs. However, for many instances, an exact MDD is often too large to process efficiently, or to fit into memory. To limit the size of the MDD, a restriction on the width of the MDD is enforced. Figure 5.4b gives an example of an MDD with limited width. Note that for this example, there does not exist an exact MDD when the width is limited to 1 or 2. The procedure to calculate the minimum sum of time-dependent setup times in an MDD as described in the previous paragraph remains valid for relaxed MDDs. However, instead of the optimum solution, a lower bound on the optimum solution is obtained. The width of the MDD can be used to control the quality of the bound.

Constructing MDDs

Following the approaches outlined by Ciré and van Hoeve (2013), an MDD for the TD-TSP can be constructed via an iterative procedure. This procedure first

generates a relaxed MDD of width one, as depicted in 5.4a. Next, the MDD is strengthened by applying two elementary operations: filtering and refinement. These operations were first introduced by Andersen et al. (2007).
In the *filtering* process, infeasible arcs are identified and removed from the MDD. Informally stated, an arc can be eliminated under the following conditions:

- Filtering invalid permutations. An arc can be removed if all paths through this arc are infeasible. For example in Figure 5.4a, all arcs with $val(a) = 0$ below node $(1, 0)$ may be removed as any path starting from the root node already contains job 0, and no job may occur more than once in a feasible permutation.
- Objective based filtering. Given is an upper bound on the optimal solution, for example a feasible but not necessarily optimal sequence with objective value z. An arc can be removed if all (s-t) paths through this arc have a cost higher than z.

The filtering conditions based on invalid permutations can be formalized as follows (Ciré and van Hoeve, 2013). Define for each node v in the MDD the set $All_v^{\downarrow}$ as the set of arc labels that appear in every path from the root node r to node v. Similarly, define $Some_v^{\downarrow}$ as the union the arc labels encountered in the paths from the root node r to node v. Mathematically, $All_v^{\downarrow}$ and $Some_v^{\downarrow}$ are defined as:

$$All_{root}^{\downarrow} = Some_{root}^{\downarrow} = \emptyset \tag{5.22}$$

$$All_v^{\downarrow} = \bigcap_{a=(u,v)\in in(v)} (All_u^{\downarrow} \cup \{val(a)\}) \tag{5.23}$$

$$Some_v^{\downarrow} = \bigcup_{a=(u,v)\in in(v)} (Some_u^{\downarrow} \cup \{val(a)\}) \tag{5.24}$$

Analogous, $All_v^{\uparrow}$ and $Some_v^{\uparrow}$ which are computed from the sink node s of the MDD are defined as:

$$All_{sink}^{\uparrow} = Some_{sink}^{\uparrow} = \emptyset \tag{5.25}$$

$$All_u^{\uparrow} = \bigcap_{a=(u,v)\in out(v)} (All_v^{\uparrow} \cup \{val(a)\}) \tag{5.26}$$

$$Some_u^{\uparrow} = \bigcup_{a=(u,v)\in out(v)} (Some_v^{\uparrow} \cup \{val(a)\}) \tag{5.27}$$

An arc $a = (u, v)$ is removed (filtered) from the MDD if any of the following conditions hold:

$$val(a) \in All_u^{\downarrow} \tag{5.28}$$

$$|Some_u^{\downarrow}| = \ell(a) \wedge val(a) \in Some_u^{\downarrow} \tag{5.29}$$

$$val(a) \in All_v^{\uparrow} \tag{5.30}$$

$$|Some_v^{\uparrow}| = n - \ell(a) \wedge val(a) \in Some_v^{\uparrow} \tag{5.31}$$

$$|Some_u^{\downarrow} \cup \{val(a)\} \cup Some_v^{\uparrow}| < n \tag{5.32}$$

For a proof of correctness of these filtering conditions, refer to Ciré and van Hoeve (2013). Finally, the filtering condition based on the objective function is stated mathematically as follows. For a given arc $a = (u, v)$, let $c_a^{\downarrow}$ be the cost of the cheapest partial ordering represented by a path from the root node to node v through arc a, i.e.,:

$$c_a^{\downarrow} = \begin{cases} 0 & \text{if } a \in out(root) \\ \min\limits_{\substack{a' \in in(u) \\ val(a) \neq val(a')}} \{c_{a'}^{\downarrow} + \delta_{(val(a'), val(a), l(a'))}\} & \text{otherwise} \end{cases} \tag{5.33}$$

Analogous, define $c_a^{\uparrow}$ as:

$$c_a^{\uparrow} = \begin{cases} 0 & \text{if } a \in in(t) \\ \min\limits_{\substack{a' \in out(v) \\ val(a) \neq val(a')}} \{c_{a'}^{\uparrow} + \delta_{(val(a), val(a'), l(a))}\} & \text{otherwise} \end{cases} \tag{5.34}$$

An arc $a = (u, v)$ is removed if the following condition holds:

$$c_a^{\downarrow} + c_a^{\uparrow} > z \tag{5.35}$$

where z is a valid upper bound on the objective value. In this work, z is the cost of the best incumbent integer solution.
The bound derived from the MDD can be directly communicated to the CP solver through the following constraint:

$$obj \geq \min_{a \in in(t)} c_a^{\downarrow} \tag{5.36}$$

where *obj* is a variable that takes the value of the objective function.

Recall that a *relaxed* MDD encodes a superset of all feasible orderings because an exact representation of the solution space is often unmanageable. Intuitively,

a relaxed MDD can be made exact by splitting nodes; compare for example the relaxed MDD in Figure 5.4a and the exact MDD in Figure 5.4c. A node can be split if the resulting nodes are non-equivalent. More precisely, a node u can be split if there is a job $j \in J$ that appears in some paths from the root node to node u, but not in all paths, i.e., $j \in Some_u^{\downarrow} \setminus All_u^{\downarrow}$. For example, node $(2, 1)$ in Figure 5.4b could be split into two nodes as job 1 meets the aforementioned criteria. *Refinement* consists of identifying such nodes and splitting them into two or more non-equivalent nodes. Note that this process must always obey the maximum allowed width of the MDD. In practice, there are often more candidate nodes to refine than the width of the MDD permits. Hence, a refinement order needs to be determined, thereby prioritizing certain nodes in the refinement process.

A node u in the MDD is *exact* if there does not exist a job $j \in J$ s.t. $j \in Some_u^{\downarrow} \setminus All_u^{\downarrow}$ or $Some_u^{\downarrow} = All_u^{\downarrow}$. Furthermore, a layer L_i in the MDD is exact if all nodes in the layer are exact nodes, and if all layers $L_0, \ldots, L_{i-1}$ are also exact layers. Finally, as shown by Ciré and van Hoeve (2013), a job $j \in J$ has exactly one position in all orderings identified by the MDD iff there does not exist a node u in the MDD s.t. $j \in Some_u^{\downarrow} \setminus All_u^{\downarrow}$. The latter notion is used to determine the order in which the nodes in the MDD are refined. For a pre-defined ordered set of jobs $S = j_1, j_2, \ldots, j_n$, first refine all nodes u in the MDD for which $j_1 \in Some_u^{\downarrow} \setminus All_u^{\downarrow}$, then continue with the nodes for which $j_2 \in Some_u^{\downarrow} \setminus All_u^{\downarrow}$, and so on.

Refinement is performed layer-by-layer in a top-down fashion. At each layer, and for each job j in the ordered set of jobs S, the nodes in the set $\{u : j \in Some_u^{\downarrow} \setminus All_u^{\downarrow}\}$ are refined until the maximum width of the MDD is reached, or until there are no more nodes to refine. In this work, S is greedily determined as follows. Initially, let S be an empty set. For each $j \in J \setminus S$, determine the average setup time to all other jobs $k \in J \setminus S \setminus \{j\}$ over all possible positions in the sequence. The job j with the largest average setup time is selected and added to set S. This procedure is repeated until all jobs are ordered (i.e., $|S| = |J|$). This procedure ensures that jobs with large setup times to other jobs have a higher priority to be represented as exact activities, which intuitively should lead to stronger bounds in the MDD.

5.4.2 CP model with MDD

The CP model from Section 5.3 is modified as follows to accommodate for the MDD:

`Variable definitions:`

1 $\pi_0 = 0$
2 $\pi_i = 1, \ldots, n-1 \qquad \forall i = 1, \ldots, n-1$
3 $\pi_n = n$
4 $act_i = \{r_i = 0, d_i = \infty, l_i = 0\} \qquad \forall i = 1, \ldots, n-1$
5 $obj = 0, \ldots, \infty$

`Objective:`

6 **Min** obj

`Constraints:`

7 AllDiff($\pi_0, \pi_1, \ldots, \pi_n$)
8 MDDConstr($\pi_0, \ldots, \pi_n$,$act_0, \ldots, act_n$, width, obj,obj_func,dist)

Similar to the CP model from Section 5.3, this model has a set of permutation variables π_i, $i = 0, \ldots, n$. In addition a set of activities, representing the jobs, are specified [1]. This allows for a richer model in which also durations of the jobs, as well as release times and deadlines can be specified. In the context of the TD-TSP, a job $i \in J$ does not have any restrictions in terms of the release times (r_i), deadlines (d_i), or duration (l_i). To incorporate the MDD discussed in the previous section into the CP model, a new global constraint, MDDConstr(...), is implemented. As input, the constraint takes the permutation and activity variables, the maximum allowed width of the MDD, an objective variable, an objective function and the Time-Dependent distance matrix. Here the objective function is the sum of setup times; the model does not explicitly state an objective function as this is taken care of by the MDD. In addition, the MDD will ensure that the activities, represented as intervals, do not overlap in time, and that the permutation variables correspond with the activities whenever they are scheduled. Finally, note that the MDDConstr propagator does not guarantee domain consistency w.r.t. the AllDiff constraint, so the domain propagator of the AllDiff constraint may filter additional values. Hence it is computationally advantageous for the CP solver to have the AllDiff constraint in the model.
Whenever the propagator of the MDD constraint is invoked, first, the domains of the π_i, $i = 0, \ldots, n$ variables and the MDD are synchronized, thereby possibly removing arcs from the MDD. For example, if a job $j \in J$ is fixed to position i in the sequence, i.e., $\pi_i = j$, then all arcs $a \in \mathcal{A}$ with $\ell(a) \neq i$ and $val(a) = j$ can be removed from the MDD. Similarly, all arcs $a \in \mathcal{A} : \ell(a) = i, val(a) \neq j$ may be removed from the MDD. Finally, an arc $a = (u, v)$ may be removed if either node u has no incoming, or node v has no outgoing arcs. After the MDD propagator has synchronized the domains of the decision variables with the MDD, the MDD is filtered and refined as discussed in the previous section.

[1] Note: the activities in this model are modeled as interval variables, see Section 3.3.2 for details.

Finally, the domains of the variables are re-evaluated, thereby filtering values which are deemed infeasible by the MDD.

5.5 Strengthening MDD propagation through additive bounding

For most optimization problems, several relaxations exist based on different underlying (combinatorial) problem structures. For instance, for the well-known TSP, common relaxations are the Held-Karp relaxation (Held and Karp, 1970), Assignment relaxation, or Linear Program (LP) relaxation. To obtain a valid bound on the optimum solution, one could simply compute a bound from each relaxation, and return the strongest resulting bound. The disadvantage of such an approach is that the structural information of only one relaxation is used. To resolve this issue Fischetti and Toth (1989) proposed an additive bounding procedure, thereby aggregating the information from different relaxations to obtain a valid bound for the given problem, which is at least as strong as the strongest bound obtained from the individual relaxations. This Section first recapitulates on the additive bounding procedure proposed in Fischetti and Toth (1989). Next, we show how the additive bounding procedure can be used to incorporate information from various relaxations into the structure of an MDD. The resulting approach provides a number of merits:

- Integrating information from different problem relaxations into the structure of the MDD enables additional filtering conditions.
- The MDD provides an *interface* to incorporate structural information from various problem relaxations into the CP model.
- MDDs provide a discrete relaxation of the solution space; Linear Programming based relaxations on the other hand produce a continuous relaxation. Projecting structural information from a continuous relaxation onto a discrete structure potentially improves the overall strength of the relaxation, because it resembles the solution space of the original problem better; after all, all feasible solution to a sequencing problem are discrete.

5.5.1 Additive bounding

The additive bounding procedure proposed by Fischetti and Toth (1989) offers a means to compute a valid lower bound for a problem P by combining

multiple bounding procedures. Given a set of r (lower) bounding procedures $L^1, L^2, \ldots, L^r$ for a given problem P, where P is stated as:

$$P: \ min\ cx \tag{5.37}$$

$$x \in F(P) \tag{5.38}$$

where c, x are resp. a row vector of costs, and a column vector of n variables, and $F(P) \subset \{x \in \mathbb{R}^n : x \geq 0\}$. Suppose that for a given cost vector $\overline{c}$, lower bound procedure $L^k(\overline{c})$, $k = 1, \ldots, r$, produces a valid lower bound μ^k, as well as a residual cost row vector c^k s.t.:

$$c^k \geq 0 \tag{5.39}$$

$$\mu^k + c^k x \leq \overline{c}x \tag{5.40}$$

The additive bounding procedure outlined by Fischetti and Toth (1989) starts by calculating bound μ^1 and residual cost vector c^1 through bounding procedure $L^1(\overline{c})$, where $\overline{c}$ is the original cost vector in the problem instance. Next, $L^k(c^{k-1})$ for $k = 2, \ldots, r$ can be solved recursively, thereby obtaining bounds $\mu^1, \ldots, \mu^r$. Fischetti and Toth (1989) prove that $\mu = \mu^1 + \cdots + \mu^r$ is a valid lower bound for problem P.

5.5.2 Projecting information from the additive bounding procedure onto the MDD

After solving the additive bounding procedure, one obtains a bound μ and a final residual cost vector c^r. In case of the TD-TSP, the cost vector $\overline{c}$ corresponds with a cost per time-dependent arc, i.e., $\overline{c} = \delta(i, j, t)$. The residual cost vector c^r defines a residual cost $(\delta^r(i, j, t))$ per time-dependent arc (i, j, t). For a given arc $a = (u, v)$ in the MDD, let $SR_a^{\downarrow}$ be the sum of reduced costs incurred by the least cost partial ordering represented by a path from the root node to node v, through arc a, in the MDD. In this context, the least cost partial ordering is computed in terms of reduced costs, not in terms of the original arc costs. Mathematically stated, we have:

$$SR_a^{\downarrow} = \begin{cases} 0 & \text{if } a \in out(root) \\ \min\limits_{\substack{a' \in in(u) \\ val(a) \neq val(a')}} \{SR_{a'}^{\downarrow} + \delta^r_{(val(a'), val(a), l(a'))}\} & \text{otherwise} \end{cases} \tag{5.41}$$

Analogous, $SR_a^{\uparrow}$ is calculated as:

$$SR_a^{\uparrow} = \begin{cases} 0 & \text{if } a \in in(t) \\ \min\limits_{\substack{a' \in out(v) \\ val(a) \neq val(a')}} \{SR_{a'}^{\uparrow} + \delta^r_{(val(a),val(a'),l(a))}\} & \text{otherwise} \end{cases} \tag{5.42}$$

Based on the assumption in equation (5.40), we can now state the following filtering condition for the MDD. An arc $a = (u, v)$ must be eliminated from the MDD if:

$$SR_a^{\downarrow} + SR_a^{\uparrow} + \mu > z \tag{5.43}$$

where z is a valid upper bound on the objective value. Note that the left hand side of this equation may be rounded up if the original distance matrix contains integer values only.
The bound derived from the MDD can be directly communicated to the CP solver through the following constraint:

$$obj \geq \min_{a \in in(t)} SR_a^{\downarrow} + \mu \tag{5.44}$$

where *obj* is a variable that takes the value of the objective function.

Finally note that the additive bounding procedure can be executed as a preprocessing step because the bounds are independent of the MDD or the CP model. Consequently, apart from executing the additive bounding procedure, the computational overhead is very limited: the MDD must only keep track of the $SR_a^{\downarrow}$ and $SR_a^{\uparrow}$ values.
In this work we only consider a single bounding procedure, namely the LP relaxation obtained from the first MIP model (equations (5.1)-(5.9)) in Section 5.3. Additional bounding procedures may be incorporated in future work.

5.6 Computational Experiments

Analogous to Abeledo et al. (2013), we conducted experiments on a number of TD-TSP instances, derived from well-known TSPLib instances. The time-dependent setup times $\delta_{(i,j,t)}$ are defined as $(n-t)\dot{\delta}_{(i,j)}$, where $\delta_{(i,j)}$ is the distance between cities i and j as specified in the TSPLib instance. Furthermore, the source node is always the first city defined in the TSPLib file. The "dTSP" instances have been obtained through personal communication with one of the authors of Abeledo et al. (2013). Experiments in this work are performed on a system with an Intel Xeon CPU E5420 2.50GHz, having 4Gb RAM, using CPLEX and CP Optimizer version 12.4.

To establish a baseline, we first compare the Mixed Integer Programming based models, i.e. the two MIP models from Sections 5.3.2-5.3.3, based on the models from resp. Picard and Queyranne (1978); Gouveia and Voss (1995); Abeledo et al. (2013). Note that, in contrast to the generic methods described in this chapter, the method presented by Abeledo et al. (2013) is a dedicated solution approach; we add a comparison to their results merely for reference purposes. The results are reported in Table 5.1. The column *Inst* gives the instance name of the TSPLib instance used to create the TD-TSP instance. The number in each name corresponds with the number of jobs in each instance. Column *bBound* gives the strongest known lower bound for each instance. Numbers in bold are optimal solutions. Per method, the best objective, the optimality gap (gap between the objective and the bBound) are reported. Each method is allotted a runtime of at most 30 minutes per instance. The actual runtime in seconds is given in column $t(s)$. In addition, for the two compact MIP formulations, Picard and Queyranne (1978); Gouveia and Voss (1995), we also report the strongest bound derived after a runtime of 30 minutes, and the gap (*lpGap*) between this bound and the objective value. The results in the last three columns in Table 5.1 have been derived from (Abeledo et al., 2013); these computations are performed on a different machine and the authors imposed a maximum runtime of 48 hours per instance.

While comparing the first two generic MIP approaches, no clear winner can be indicated; neither of the models dominates the other model in terms of objective or bounds. The third method, the dedicated column generation approach, produces the best results, thereby solving the majority of instances (35/38) to optimality and obtaining tight gaps (average 1.64%) for the remaining instances, albeit with a longer runtime.

Table 5.2 compares two CP approaches. The first approach is based on the pure CP model from Section 5.3.1, whereas the second approach (CP^{ab}_{MDD}) utilizes the CP model from Section 5.4.2, strengthened with the MDD and the additive bounding procedure outlined in Section 5.5. The pure CP model (Section 5.3.1) could not be solved directly through CP Optimizer as the 3-dimensional element constraints in the objective function required too much memory. Therefore, the objective function $\min \sum_{i \in n-1} \delta_{(\pi_i, \pi_{i+1}, i)}$ was replaced by its equivalent $\min \sum_{i \in n-1} (n-i) \delta_{(\pi_i, \pi_{i+1})}$. Furthermore, CP Optimizer has no support for multi-dimensional element constraints so, in the implementation, the distance matrix is represented as a 1-dimensional array; n auxiliary variables are used to index the array. To run the CP model, the default search options were used (8 workers). For the CP^{ab}_{MDD} approach, Depth-First Search was used, as well as extended inference, 1 worker and an MDD width of 128. In addition, for all the MDD based approaches in this section, a search phase is defined, thereby ensuring that CP optimizer only branches on the job ordering, and not on, for example, the start times of the interval variables. Finally, note that although

it is possible to compute a lower bound on each node of the CP search tree through the MDD, CP Optimizer does not allow us to iterate over the remaining open nodes after the time limit is reached; hence, it is not possible to obtain a valid lower bound once the CP search terminates.

The pure CP approach (Table 5.2) is unable to solve any instances to optimality. Even though it finds feasible solutions for all instances, the optimality gap is very large (average gap 45.02%). In contrast, the CP^{ab}_{MDD} approach performs very well. Through this approach, 12 instances could be solved to optimality, and an average optimality gap of 6.49% was realized. This is significantly better than the two generic MIP approaches compared in Table 5.1: Picard and Queyranne (1978) and Gouveia and Voss (1995) obtained average optimality gaps of resp. 33.37% and 31.64%, while only solving 6 instances to optimality. Furthermore, the largest instance in our data set that could be solved to optimality within the alloted time limit (1800 seconds, see column t(s) for runtime in seconds) through MIP contains 29 vertices (bayg29), versus 14 for the pure CP approach (gr17), and 42 (swiss42) in CP^{ab}_{MDD}.

In Figure 5.5 the influence of the size of the MDD is evaluated. These experiments are conducted without additive bounding. When increasing the size of the MDD, the number of fails decrease (Fig 5.5a), i.e., the number of branches in the CP search tree that did not yield a solution. However, when the size is chosen too large, the computation time increases (Fig 5.5b). Overall, an MDD width of 64 or 128 yields the best results. Unless stated otherwise, the remaining experiments will use an MDD width of 128.

The remaining three subsections will focus resp. on the impact of the refinement order, different branching strategies, and the impact of the additive bounding procedure.

Table 5.1: MIP comparison

		Picard and Queyranne (1978)					Gouveia and Voss (1995)					Abeledo et al. (2013)		
inst	bBound	obj	bound	lpGap	gap	t(s)	obj	bound	lpGap	gap	t(s)	obj	t(s)	gap
burma14	**20315**	20315	20315	0.00%	0.00%	0	20315	20315	0.00%	0.00%	1	0	0	-
gr17	**12994**	12994	12994	0.00%	0.00%	2	12994	12994	0.00%	0.00%	1	12994	2	0.00%
gr21	**24345**	24345	24345	0.00%	0.00%	4	24345	24345	0.00%	0.00%	3	24345	4	0.00%
gr24	**13795**	13795	13795	0.00%	0.00%	22	13795	13795	0.00%	0.00%	26	13795	5	0.00%
bays29	**26862**	26862	26862	0.00%	0.00%	544	26862	26862	0.00%	0.00%	140	26862	8	0.00%
bayg29	**22230**	22230	22230	0.00%	0.00%	538	22230	22230	0.00%	0.00%	255	22230	15	0.00%
dTSP40.0	**10311**	11506	8794.91	23.56%	10.39%	1800	11446	8869.76	22.51%	9.92%	1801	10311	670	0.00%
dTSP40.1	**9807**	11249	7973.14	29.12%	12.82%	1801	10761	8446.45	21.51%	8.87%	1801	9807	82	0.00%
dTSP40.2	**9525**	11275	8158.27	27.64%	15.52%	1800	9732	8472.79	12.94%	2.13%	1801	9525	75	0.00%
dTSP40.3	**9156**	10028	7596.29	24.25%	8.70%	1801	11614	7676.44	33.90%	21.16%	1801	9156	36	0.00%
dTSP40.4	**9079**	10542	8232.9	21.90%	13.88%	1801	10004	8282.54	17.21%	9.25%	1802	9079	31	0.00%
dTSP50.0	**12680**	14761	11064	25.05%	14.10%	1800	14825	11166.5	24.68%	14.47%	1885	12680	61	0.00%
dantzig42	**12528**	14463	10418.9	27.96%	13.38%	1800	13029	11437.6	12.21%	3.85%	1801	12528	28	0.00%
swiss42	**22327**	23302	19567.6	16.03%	4.18%	1800	22595	19955.8	11.68%	1.19%	1801	22327	20	0.00%
att48	**209320**	230184	185955	19.21%	9.06%	1800	217293	192410	11.45%	3.67%	1801	209320	103	0.00%
gr48	**102378**	118346	84188.5	28.86%	13.49%	1800	118006	87043.4	26.24%	13.24%	1800	102378	76	0.00%
hk48	**247926**	277865	222140	20.05%	10.77%	1800	276782	224076	19.04%	10.43%	1801	247926	124	0.00%
dTSP50.1	**12853**	15415	10654	30.89%	16.62%	1801	13559	10868.7	19.84%	5.21%	1800	12853	275	0.00%
dTSP50.2	**12357**	16206	9952.66	38.59%	23.75%	1800	14680	10409.9	29.09%	15.82%	1801	12357	78	0.00%
dTSP50.3	**12722**	16826	10587.7	37.08%	24.39%	1800	13968	10458.5	25.13%	8.92%	1800	12722	138	0.00%
dTSP50.4	**13289**	17114	10839.1	36.67%	22.35%	1800	15198	10753.5	29.24%	12.56%	1800	13289	819	0.00%
eil51	**10178**	11839	8688.4	26.61%	14.03%	1800	10932	9070.56	17.03%	6.90%	1800	10178	33	0.00%
berlin52	**143721**	157538	117106	25.66%	8.77%	1800	197185	116103	41.12%	27.11%	1800	143721	104	0.00%
brazil58	**512361**	1530462	413056	73.01%	66.52%	1800	1154073	409773	64.49%	55.60%	1800	512361	633	0.00%
st70	**20557**	99896	15822	84.16%	79.42%	1801	117241	15684.6	86.62%	82.47%	1800	20557	2895	0.00%
eil76	**17976**	73250	15602.5	78.70%	75.46%	1800	73250	15634.8	78.66%	75.46%	1801	17976	233	0.00%
pr76	**3455242**	4399426	2543800	42.18%	21.46%	1801	4399426	2496380	43.26%	21.46%	1800	3455242	58045	0.00%
gr96	**2097170**	3246565	1604300	50.58%	35.40%	1801	3246565	1586110	51.14%	35.40%	1800	2097170	160250	0.00%
rat99	49545.9	96016	49545.9	48.40%	48.40%	1803	96016	49480.1	48.47%	48.40%	1801	58288	172800	15.00%
rd100	**340047**	2613109	268584	89.72%	86.99%	1802	2613109	266803	89.79%	86.99%	1801	340047	18582	0.00%
kroA100	**983128**	7840954	707135	90.98%	87.46%	1805	9747612	696035	92.86%	89.91%	1800	983128	106336	0.00%
kroB100	**986008**	7292215	698352	90.42%	86.48%	1806	7840298	677493	91.36%	87.42%	1820	986008	7684	0.00%
kroC100	**961324**	9178983	693642	92.44%	89.53%	1802	9178983	681445	92.58%	89.53%	1801	961324	39559	0.00%
kroD100	680265	8843998	680265	92.31%	92.31%	1802	8843998	674545	92.37%	92.31%	1801	976965	172800	30.37%
kroE100	**971266**	9852780	691880	92.98%	90.14%	1802	9852780	684834	93.05%	90.14%	1801	971266	117965	0.00%
eil101	23326.7	114826	23326.7	79.69%	79.69%	1802	114826	23136.4	79.85%	79.69%	1801	27519	172800	15.23%
lin105	**603910**	1671784	430828	74.23%	63.88%	1804	1671784	429515	74.31%	63.88%	1803	603910	6317	0.00%
pr107	**2026626**	2846848	1716160	39.72%	28.81%	1803	2846848	1716160	39.72%	28.81%	1801	2026626	9947	0.00%

Table 5.1: MIP comparison

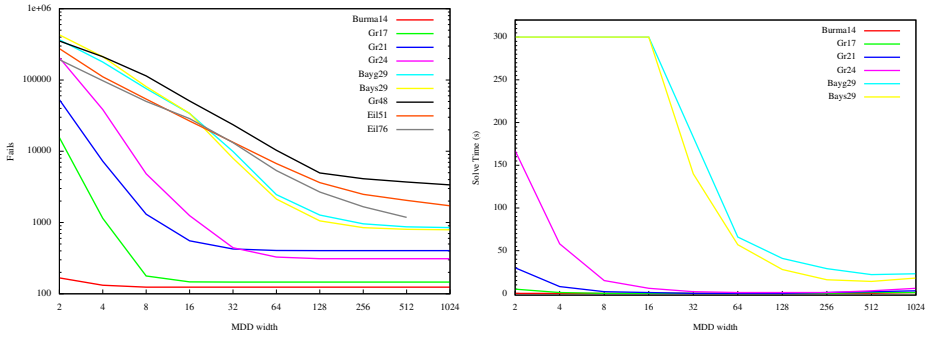
		Picard and Queyranne (1978)					Gouveia and Voss (1995)					Abeledo et al. (2013)		
inst	bBound	obj	bound	lpGap	gap	t(s)	obj	bound	lpGap	gap	t(s)	obj	t(s)	gap
Solved					6					6				35
avg				41.54%	33.37%				39.30%	31.64%				1.64%

Table 5.2: CP comparison

		Pure CP			CP^{ab}_{MDD}		
inst	bBound	obj	t(s)	gap	obj	t(s)	gap
burma14	**20315**	20315	1800	0.00%	20315	0	0.00%
gr17	**12994**	12994	1800	0.00%	12994	0	0.00%
gr21	**24345**	24498	1800	0.62%	24345	1	0.00%
gr24	**13795**	13863	1800	0.49%	13795	1	0.00%
bays29	**26862**	28690	1800	6.37%	26862	3	0.00%
bayg29	**22230**	23160	1800	4.02%	22230	3	0.00%
dTSP40.0	**10311**	14604	1800	29.40%	10311	1800	0.00%
dTSP40.1	**9807**	13983	1800	29.86%	9807	1538	0.00%
dTSP40.2	**9525**	12599	1800	24.40%	9525	914	0.00%
dTSP40.3	**9156**	13032	1800	29.74%	9156	1113	0.00%
dTSP40.4	**9079**	13475	1800	32.62%	9079	16	0.00%
dTSP50.0	**12680**	21277	1800	40.41%	13546	1800	6.39%
dantzig42	**12528**	16798	1800	25.42%	12624	1800	0.76%
swiss42	**22327**	32460	1800	31.22%	22327	105	0.00%
att48	**209320**	289997	1800	27.82%	209320	1798	0.00%
gr48	**102378**	179381	1800	42.93%	109569	1800	6.56%
hk48	**247926**	377511	1800	34.33%	286087	1800	13.34%
dTSP50.1	**12853**	20007	1800	35.76%	13231	1800	2.86%
dTSP50.2	**12357**	22013	1800	43.86%	12465	1800	0.87%
dTSP50.3	**12722**	20628	1800	38.33%	13085	1800	2.77%
dTSP50.4	**13289**	22820	1800	41.77%	14040	1800	5.35%
eil51	**10178**	16006	1800	36.41%	10271	1800	0.91%
berlin52	**143721**	219682	1800	34.58%	145824	1800	1.44%
brazil58	**512361**	1003840	1800	48.96%	615000	1800	16.69%
st70	**20557**	50666	1800	59.43%	21805	1801	5.72%
eil76	**17976**	46122	1800	61.03%	18969	1800	5.23%
pr76	**3455242**	10789626	1800	67.98%	3619821	1800	4.55%
gr96	**2097170**	9486169	1800	77.89%	2506860	1801	16.34%
rat99	49545.9	213637	1800	76.81%	62751	1800	21.04%
rd100	**340047**	1777195	1800	80.87%	386357	1800	11.99%
kroA100	**983128**	4718805	1800	79.17%	1143792	1800	14.05%
kroB100	**986008**	4927266	1800	79.99%	1178211	1800	16.31%
kroC100	**961324**	4620418	1800	79.19%	996577	1801	3.54%
kroD100	680265	4768584	1800	85.73%	1105147	1800	38.45%
kroE100	**971266**	4575812	1800	78.77%	1072258	1800	9.42%
eil101	23326.7	95771	1800	75.64%	30751	1800	24.14%
lin105	**603910**	3615063	1800	83.29%	672456	1800	10.19%
pr107	**2026626**	14155245	1800	85.68%	2197074	1801	7.76%
Solved				2			12
avg				45.02%			6.49%

5.6.1 Impact of additive bounding

This section investigates the impact of the additive bounding procedure. Recall from Section 5.5 that the LP relaxation of the MIP model (Constraints (5.1)-(5.9)) in Section 5.3 is used to compute a valid lower bound for the additive bounding procedure. Table 5.3 compares the bounds obtained after solving the root node of the CP search tree for different solution procedures. Column *LP*



(a) Number of fails, i.e., number of backtracks.

(b) Time to solve instances. Only instances that could be solved within 5 min are included.

Figure 5.5: Influence of the MDD width.

gives the lower bound obtained by solving the LP relaxation of the MIP model (Constraints (5.1)-(5.9)) in Section 5.3.2. Next, column CP_{MDD} provides the bounds obtained by solving the root node of the CP search tree when solving the model from Section (5.4.2) without additive bounding. Finally, CP^{ab}_{MDD} provides the same bound, but this time the MDD is strengthened with the information from the additive bounding procedure. Columns δ_{LP} and δ_{MDD} give the percentage improvement of CP^{ab}_{MDD} w.r.t. resp. LP and CP_{MDD}.
As can be observed from Table 5.3, the LP bound is usually stronger than the CP_{MDD} bound. However, when comparing CP^{ab}_{MDD} to both LP and CP_{MDD} it becomes apparent that CP^{ab}_{MDD} is significantly stronger. The same observation holds under different MDD widths. As a consequence, one can conclude that the bounds obtained from the MDD and the LP relaxation indeed strengthen each other.

Table 5.3: Lower bound on the root node of the CP search tree

		MDD width: 128				MDD width: 512			
inst	LP	CP_{MDD}	CP^{ab}_{MDD}	δ_{LP}	δ_{MDD}	CP_{MDD}	CP^{ab}_{MDD}	δ_{LP}	δ_{MDD}
burma14	17189.2	18607	19644	12.50%	5.28%	20177	20314	15.38%	0.67%
gr17	10897.7	11733	12412	12.20%	5.47%	12960	12993	16.13%	0.25%
gr21	20378.5	13792	21779	6.43%	36.67%	17044	22649	10.02%	24.75%
gr24	11770.5	10533	12731	7.54%	17.26%	10845	12731	7.54%	14.81%
bays29	23163.1	20073	23985	3.43%	16.31%	20962	24305	4.70%	13.75%
bayg29	19319	16707	19991	3.36%	16.43%	17285	20238	4.54%	14.59%
dTSP40.0	7389.63	2981	8029	7.96%	62.87%	4069	8175	9.61%	50.23%
dTSP40.1	6739.45	3158	7022	4.02%	55.03%	3542	7196	6.34%	50.78%
dTSP40.2	6962.01	2649	7533	7.58%	64.83%	3024	7908	11.96%	61.76%
dTSP40.3	6926.59	4164	7214	3.98%	42.28%	4813	7410	6.52%	35.05%

Table 5.3: Lower bound on the root node of the CP search tree

		MDD width: 128				MDD width: 512			
inst	LP	CP_{MDD}	CP^{ab}_{MDD}	δ_{LP}	δ_{MDD}	CP_{MDD}	CP^{ab}_{MDD}	δ_{LP}	δ_{MDD}
dTSP40.4	7347.67	4506	7925	7.28%	43.14%	5259	8000	8.15%	34.26%
dTSP50.0	10465.7	5248	10633	1.57%	50.64%	5537	10646	1.69%	47.99%
dantzig42	10135.6	7088	10362	2.18%	31.60%	7412	10362	2.18%	28.47%
swiss42	18190.1	7509	19006	4.29%	60.49%	8117	19198	5.25%	57.72%
att48	175585	107684	184220	4.69%	41.55%	113132	185178	5.18%	38.91%
gr48	81228.1	43978	84242	3.58%	47.80%	48372	84863	4.28%	43.00%
hk48	207077	140272	213943	3.21%	34.43%	156728	215777	4.03%	27.37%
dTSP50.1	10197.1	3952	10531	3.17%	62.47%	4288	10782	5.42%	60.23%
dTSP50.2	9256.64	3921	9624	3.82%	59.26%	4103	9848	6.00%	58.34%
dTSP50.3	9574.44	4898	9903	3.32%	50.54%	5025	9948	3.76%	49.49%
dTSP50.4	9726.51	3496	10335	5.89%	66.17%	3890	10394	6.42%	62.57%
eil51	8676.25	4153	9029	3.91%	54.00%	4230	9106	4.72%	53.55%
berlin52	112350	45891	116426	3.50%	60.58%	49056	119694	6.14%	59.02%
brazil58	400508	262922	421502	4.98%	37.62%	292960	433534	7.62%	32.43%
st70	15469.1	5312	15961	3.08%	66.72%	5518	16224	4.65%	65.99%
eil76	15591	8585	15791	1.27%	45.63%	8787	15832	1.52%	44.50%
pr76	2496050	2075295	2675408	6.70%	22.43%	2116910	2679143	6.83%	20.99%
gr96	1586060	578690	1614062	1.73%	64.15%	597453	1626302	2.47%	63.26%
rat99	49480.1	29108	50502	2.02%	42.36%	29266	50577	2.17%	42.14%
rd100	266791	68053	272846	2.22%	75.06%	70660	273693	2.52%	74.18%
kroA100	693870	208424	712451	2.61%	70.75%	223039	719992	3.63%	69.02%
kroB100	676886	222370	683928	1.03%	67.49%	237975	698269	3.06%	65.92%
kroC100	679173	260244	690103	1.58%	62.29%	264092	692536	1.93%	61.87%
kroD100	670646	219547	684981	2.09%	67.95%	241669	697031	3.79%	65.33%
kroE100	684831	243637	710019	3.55%	65.69%	255939	720644	4.97%	64.48%
eil101	23105.1	9177	23308	0.87%	60.63%	9392	23312	0.89%	59.71%
lin105	429496	209950	446820	3.88%	53.01%	233126	451150	4.80%	48.33%
pr107	1716160	1254562	1746088	1.71%	28.15%	1262115	1748638	1.86%	27.82%
avg				4.18%	47.76%			5.49%	44.57%

Table 5.4 compares three different procedures:

1. CP with MDD (Section 5.4.2).
2. CP with MDD and additive bounding.
3. CP with MDD, additive bounding, but no arc filtering based on the additive bound, i.e., no arcs are filtered based on the condition stated in equation 5.43.

From the results it can be observed that strengthening the MDD through the additive bounding procedure significantly improves the results. Furthermore, whenever arc filtering based on the additive bound is disabled, a clear performance decrease is observed. The difference in performance between the third approach (CP^{ab}_{MDD}, no arc filtering) and the first approach (CP_{MDD}) can be fully attributed to the presence of constraint (5.44) in the third approach.

Currently, only a single LP is used for the additive bounding procedure. Straightforwardly, the bound derived through this procedure can be improved by adding additional bounding procedures. Furthermore, valid inequalities can be incorporated to strengthen LP relaxation. Extensive overviews of valid inequalities for the TD-TSP are authored by Gouveia and Voss (1995); Abeledo et al. (2013).

Table 5.4: Analyzing the impact of the additive bounding procedure.

		CP_{MDD}					CP^{ab}_{MDD}					CP^{ab}_{MDD}, no arc filtering				
inst	bestBound	obj	gap	fails	ttb(s)	t(s)	obj	gap	fails	ttb(s)	t(s)	obj	gap	fails	ttb(s)	t(s)
burma14	20315	20315	0.00%	25	0	0	20315	0.00%	25	0	0	20315	0	25	0	0
gr17	12994	12994	0.00%	20	0	0	12994	0.00%	20	0	0	12994	0.00%	20	0	0
gr21	24345	24345	0.00%	52	1	1	24345	0.00%	52	0	1	24345	0.00%	52	1	1
gr24	13795	13795	0.00%	36	1	1	13795	0.00%	36	1	1	13795	0.00%	36	1	1
bays29	26862	26862	0.00%	130	2	14	26862	0.00%	72	2	3	26862	0.00%	129	2	13
bayg29	22230	22230	0.00%	185	2	20	22230	0.00%	93	2	3	22230	0.00%	183	2	18
dTSP40.0	10311	10446	1.29%	7618	970	1800	10311	0.00%	5687	808	1800	10446	1.29%	7825	914	1800
dTSP40.1	9807	9807	0.00%	6479	211	1800	9807	0.00%	4180	106	1538	9807	0.00%	6768	199	1800
dTSP40.2	9525	9872	3.51%	6081	463	1800	9525	0.00%	2649	370	914	9872	3.51%	6610	422	1800
dTSP40.3	9156	9491	3.53%	5084	209	1800	9156	0.00%	2593	787	1113	9491	3.53%	5355	197	1800
dTSP40.4	9079	9079	0.00%	1196	13	519	9079	0.00%	88	9	16	9079	0.00%	975	12	391
dTSP50.0	12680	13636	7.01%	7520	1371	1800	13546	6.39%	6486	760	1800	13636	7.01%	7974	1339	1800
dantzig42	12528	12658	1.03%	7285	281	1800	12624	0.76%	5981	50	1800	12658	1.03%	7964	254	1800
swiss42	22327	22327	0.00%	4643	25	1800	22327	0.00%	316	11	105	22327	0.00%	4690	23	1800
att48	209320	222760	6.03%	9843	1102	1800	209320	0.00%	5256	740	1798	222760	6.03%	11300	915	1800
gr48	102378	110615	7.45%	6997	545	1800	109569	6.56%	7016	1577	1800	110615	7.45%	7393	504	1800
hk48	247926	295716	16.16%	7841	668	1800	286087	13.34%	7165	1787	1800	295716	16.16%	8590	604	1800
dTSP50.1	12853	14025	8.36%	5506	945	1800	13231	2.86%	5284	1467	1800	14025	8.36%	5942	848	1800
dTSP50.2	12357	13001	4.95%	4944	184	1800	12465	0.87%	4510	692	1800	13001	4.95%	5372	165	1800
dTSP50.3	12722	13587	6.37%	4960	682	1800	13085	2.77%	4211	722	1800	13587	6.37%	5173	650	1800
dTSP50.4	13289	14042	5.36%	6889	44	1800	14040	5.35%	6499	1623	1800	14042	5.36%	7730	41	1800
eil51	10178	10347	1.63%	6011	161	1800	10271	0.91%	3972	62	1800	10347	1.63%	6327	106	1800
berlin52	143721	145824	1.44%	5353	35	1800	145824	1.44%	4959	22	1800	145824	1.44%	5946	32	1800
brazil58	512361	615740	16.79%	7037	1556	1800	615000	16.69%	9558	1512	1800	615740	16.79%	7505	1451	1800
st70	20557	22044	6.75%	4238	389	1800	21805	5.72%	3302	1149	1801	22044	6.75%	4351	368	1800
eil76	17976	18969	5.23%	3335	211	1800	18969	5.23%	3358	179	1800	18969	5.23%	3513	201	1800
pr76	3455242	3619821	4.55%	4793	130	1800	3619821	4.55%	4761	124	1800	3619821	4.55%	4947	129	1800
gr96	2097170	2506860	16.34%	2687	577	1801	2506860	16.34%	2664	559	1801	2506860	16.34%	2875	559	1801
rat99	49545.9	62751	21.04%	3405	354	1800	62751	21.04%	3182	375	1800	62751	21.04%	3454	376	1800
rd100	340047	387808	12.32%	2774	1375	1800	386357	11.99%	2712	1750	1800	387808	12.32%	2923	1330	1800
kroA100	983128	1143792	14.05%	3386	1592	1800	1143792	14.05%	3415	1530	1800	1143792	14.05%	3447	1564	1800
kroB100	986008	1178211	16.31%	3675	357	1800	1178211	16.31%	3770	375	1800	1178211	16.31%	3813	378	1800
kroC100	961324	1000359	3.90%	3846	1697	1800	996577	3.54%	3731	1088	1801	1000359	3.90%	4034	1637	1800
kroD100	680265	1105147	38.45%	3870	1541	1800	1105147	38.45%	3806	1523	1800	1105147	38.45%	3905	1533	1800
kroE100	971266	1072258	9.42%	3342	1592	1800	1072258	9.42%	3337	1398	1800	1072258	9.42%	3429	1548	1800
eil101	23326.7	30751	24.14%	3115	1298	1800	30751	24.14%	3233	917	1800	30751	24.14%	3209	1275	1800
lin105	603910	672456	10.19%	3668	493	1800	672456	10.19%	3331	492	1800	672456	10.19%	3761	506	1800

Table 5.4: Analyzing the impact of the additive bounding procedure.

		CP_{MDD}					CP^{ab}_{MDD}					CP^{ab}_{MDD}, no arc filtering				
inst	bestBound	obj	gap	fails	ttb(s)	t(s)	obj	gap	fails	ttb(s)	t(s)	obj	gap	fails	ttb(s)	t(s)
pr107	2026626	2210837	8.33%	3330	430	1801	2197074	7.76%	3511	1689	1801	2210837	8.33%	3431	460	1800
avg			7.42%					6.49%					7.42%			

5.7 Impact of the refinement order

Section 5.4.1 describes how nodes in the MDD are selected for refinement, and in which order, based on an ordered set of jobs S. Section 5.4.1 discusses a greedy strategy to calculate S. Here we compare this greedy strategy against a purely random strategy, thereby measuring the impact of the refinement order (see table 5.5). Table 5.5 shows the results obtained after solving the root node of the CP search tree for both the greedy and the random strategy. For the random strategy, the root node of the CP search tree is solved 200 times with different random orderings of S. For each instance the average, min, and max bound are reported, as well as the standard deviation (see table 5.5).
All instances where the greedy strategy produces a stronger bound than the average bound obtained through random ordering are marked in bold. As can be observed from the results, for 4 out of 38 instances, the random refinement order produces on average a better bound (column avg LB). For the majority of instances, the best bound (max LB) observed while using a random ordering is better than the bound obtained using the greedy procedure, indicating that there is still room to improve the refinement ordering. It must however be noted that fixing the refinement order to the best random ordering (i.e., the random ordering that yielded the best bound at the root node) did not necessarily result in a faster solve time of the instance. This suggests that one might want to look into possibilities to change the refinement order dynamically.

Table 5.5: Refinement Order

	Greedy Ref.	Random Ref. (200 iterations)			
inst	LB	min LB	max LB	avg LB	std. dev.
burma14	**19644**	17202	20314	19286.6	478.82
gr17	**12412**	10931	12893	11985.6	397.19
gr21	21779	20802	23257	**22135.2**	434
gr24	**12731**	11771	13150	12383.8	328.75
bays29	23985	23187	25124	**24155.5**	373.62
bayg29	19991	19433	20753	**20043**	249.93
dTSP40.0	**8029**	7408	8762	7982.21	258.44
dTSP40.1	7022	6751	7872	**7312.9**	209.43
dTSP40.2	**7533**	7097	8087	7532.53	215.8
dTSP40.3	7214	7011	7836	**7385.16**	144.82
dTSP40.4	**7925**	7393	8095	7643.05	128.86
dTSP50.0	10633	10466	11212	**10726**	147.25
dantzig42	10362	10205	11024	**10541**	163.32
swiss42	19006	18474	20103	**19144.9**	310.43
att48	**184220**	175810	187145	180439	2487.04
gr48	84242	81278	87573	**84273.9**	1326.76
hk48	**213943**	207533	221770	213852	2755.86
dTSP50.1	10531	10241	11105	**10593.4**	177.55
dTSP50.2	9624	9321	10332	**9753.01**	183.74
dTSP50.3	9903	9672	10851	**10128.2**	207.35
dTSP50.4	**10335**	9783	10872	10195.8	191.43
eil51	**9029**	8760	9156	8914.5	78.54

Table 5.5: Refinement Order

	Greedy Ref.	Random Ref. (200 iterations)			
inst	LB	min LB	max LB	avg LB	std. dev.
berlin52	116426	112351	124623	**118184**	2502.75
brazil58	**421502**	401001	438189	415800	9211.83
st70	15961	15583	16655	**15984.7**	195.9
eil76	15791	15643	16288	**15876.4**	118.71
pr76	**2675408**	2512385	2680058	2565340	33078.9
gr96	1614062	1595063	1700845	**1633590**	18826.3
rat99	**50502**	49525	50843	50085.3	257.33
rd100	**272846**	267259	279549	271868	2306.96
kroA100	712451	701630	746095	**719162**	8865.75
kroB100	683928	683500	728790	**703220**	9386.01
kroC100	690103	683819	731426	**705061**	8977.44
kroD100	684981	672923	720513	**693235**	9297.49
kroE100	**710019**	686070	729501	702325	8752.22
eil101	23308	23161	23989	**23453.1**	162.22
lin105	**446820**	431994	452328	441044	4068.64
pr107	**1746088**	1716998	1769142	1744260	9226.86
best	17			21	

5.8 Branching rules

In the foregoing experiments, branching is performed on the permutation variables π_j, $j = 0, \ldots, n$. The exact variables to branch on, as well as the values to assign to them are fully determined by CP Optimizer. To the best of the author's knowledge, no information is available on how CP Optimizer selects these variables or assigns the values. In this Section, a number of experiments are conducted with custom branching rules; a comparison of the different methods is provided in tables 5.6, 5.7.

Branching on cheapest extension - Given a partial path represented by fixed variables $[\pi_0, \pi_1, \ldots, \pi_i]$, $i < n$, we can branch on the first unfixed variable π_{i+1} by selecting the value j in the domain of π_{i+1} such that the total cost of the permutation $\pi_0, \ldots, \pi_{i+1}$ is minimal among all choices of value j. This information can be obtained directly from the MDD. Two branches are created: (1) $\pi_{i+1} = j$, (2) $\pi_{i+1} \neq j$.

Branch on expected path cost - Given a partial path represented by fixed variables, $[\pi_0, \pi_1, \ldots, \pi_i]$, $i < n$, s.t. π_{i+1} is the first unfixed variable. Two branches are created: (1) $\pi_{i+1} = j$, (2) $\pi_{i+1} \neq j$, where j is selected in such a way that the total *expected* cost for completing the partial path is minimized. The expected cost of the complete path is obtained by calculating the exact cost of the partial path $[\pi_0, \pi_1, \ldots, \pi_i, j]$ plus a lower bound on the cost to complete this path. This lower bound equals $c_a^{\uparrow}$ (Equation (5.34)), where a is the last arc in the path represented by the permutation $[\pi_0, \pi_1, \ldots, \pi_i, j]$ in the MDD.

Branch on job repetition: Let $\pi = \pi_0, \pi_1, \ldots, \pi_n$ be the permutation associated with the shortest s-t path in the MDD. If π is a feasible permutation, i.e., there are no repeated jobs, then by definition this permutation is an optimal solution for the current node in the search tree and all variables can be fixed accordingly. However, when π represents an infeasible sequence, branching is required. Let $j \in J$ be the job that occurs most often in π and let t be the first position job j occurs in π. Two branches are created: (1) $\pi_t = j$, (2) $\pi_t \neq j$. In case there are two activities $i, j \in J$ which are repeated equally often in π, ties are broken by selecting the activity which occurs first in the sequence. The rational behind this branching strategy is that the branching is guided by the shortest path, while fixing positions of frequently re-occurring jobs first.
Branch on feasible path - Given a partial path represented by fixed variables, $[\pi_0, \pi_1, \ldots, \pi_i]$, $i < n$, s.t. π_{i+1} is the first unfixed variable. Use a greedy heuristic on the MDD to find a *feasible*, but not necessarily shortest, s-t path, i.e., the path may not have repeated jobs. Let π be the permutation corresponding to this path. Two branches are created, thereby (1) assigning variable π_{i+1} its corresponding value in permutation π, or (2) removing that value from the domain of π_{i+1}.

When comparing the data in Tables 5.6, 5.7, the best results are obtained by branching on the cheapest extension, thereby obtaining an average optimality gap of 6.48% and equally good or better results than the other branching rules for 31 out of the 38 instances. This suggests that the MDD indeed can be used to guide the search towards good solutions. The default branching scheme of CP optimizer, which does not depend on the MDD, obtains slightly worse results: an average optimality of 6.49% and equal or better results on 30 out of 38 instances. One possible explanation for the good results of the default search could be related to implementation efficiency: the number of fails reported per instance for the default strategy is significantly larger than the number of fails in the corresponding columns for any of the other methods. This could indicate that their strategy for selecting and assigning variables and values for the branching strategy involves less computational overhead, rendering it possible to search a much larger area of the search space.
The branching rule based on the expected path cost obtains significantly worse results than the rule which branches on the cheapest extension. It is currently unclear what causes this behavior because the expected cost branching rule intuitively seems to use more accurate information when deciding on which variable-value pair to branch than the cheapest extension rule. Similarly, the remaining two branching rules do not perform well in practice either.
Some alternative branching strategies which may be considered in future work are:
Branch on MDD node Given a node u in a layer l of the MDD, two branches

can be created (Bergman et al., 2013): (1) all nodes, except node u, are removed from layer l, thereby forcing every path in the MDD through node u (2) node u is removed from layer l.

Counting based branching Use the MDD to derive statistical information about the variables and possible value assignments. For example, count the number of different paths in the MDD that would assign a job $j \in J$ to a position i in the sequence and branch on the job that has the highest certainty to be assigned to a particular position. Counting based search strategies for CP have been applied previously, see for instance (Pesant et al., 2012).

Table 5.6: Branching rule comparison

	Pure CP (no MDD)					CP^{AB}_{MDD} (Default search)					CP^{AB}_{MDD} (Cheapest Extension)				
inst	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap
burma14	**20315**	25288k	14	1800	0.00%	**20315**	25	0	0	0.00%	**20315**	25	0	0	0.00%
gr17	**12994**	17399k	44	1800	0.00%	**12994**	20	0	0	0.00%	**12994**	18	0	0	0.00%
gr21	24498	11226k	902	1800	0.62%	**24345**	52	0	1	0.00%	**24345**	51	1	1	0.00%
gr24	13863	8890k	1058	1800	0.49%	**13795**	36	1	1	0.00%	**13795**	36	1	1	0.00%
bays29	28690	5756k	1669	1800	6.37%	**26862**	72	2	3	0.00%	**26862**	66	3	4	0.00%
bayg29	23160	5856k	1209	1800	4.02%	**22230**	93	2	3	0.00%	**22230**	93	3	4	0.00%
dTSP40.0	14604	2897k	199	1800	29.40%	**10311**	5687	808	1800	0.00%	**10311**	5349	683	1800	0.00%
dTSP40.1	13983	2952k	130	1800	29.86%	**9807**	4180	106	1538	0.00%	**9807**	4134	74	1633	0.00%
dTSP40.2	12599	3032k	58	1800	24.40%	**9525**	2649	370	914	0.00%	**9525**	2654	392	968	0.00%
dTSP40.3	13032	2997k	1792	1800	29.74%	**9156**	2593	787	1113	0.00%	**9156**	2493	779	1121	0.00%
dTSP40.4	13475	2954k	679	1800	32.62%	**9079**	88	9	16	0.00%	**9079**	88	13	19	0.00%
dTSP50.0	21277	1975k	1747	1800	40.41%	**13546**	6486	760	1800	6.39%	**13546**	6052	803	1800	6.39%
dantzig42	16798	2321k	143	1800	25.42%	**12624**	5981	50	1800	0.76%	**12624**	5636	66	1800	0.76%
swiss42	32460	2928k	1793	1800	31.22%	**22327**	316	11	105	0.00%	**22327**	318	15	112	0.00%
att48	289997	1609k	601	1800	27.82%	**209320**	5256	740	1798	0.00%	**209320**	5143	769	1800	0.00%
gr48	179381	1887k	644	1800	42.93%	**109569**	7016	1577	1800	6.56%	**109569**	6712	1649	1800	6.56%
hk48	377511	1839k	1639	1800	34.33%	286087	7165	1787	1800	13.34%	287503	6801	1744	1800	13.77%
dTSP50.1	20007	1839k	997	1800	35.76%	13231	5284	1467	1800	2.86%	13135	3994	297	1800	2.15%
dTSP50.2	22013	1924k	1648	1800	43.86%	**12465**	4510	692	1800	0.87%	**12465**	4307	749	1800	0.87%
dTSP50.3	20628	1869k	172	1800	38.33%	**13085**	4211	722	1800	2.77%	**13085**	3844	1219	1800	2.77%
dTSP50.4	22820	1894k	415	1800	41.77%	14040	6499	1623	1800	5.35%	**13916**	7644	1442	1800	4.51%
eil51	16006	2012k	1321	1800	36.41%	**10271**	3972	62	1800	0.91%	**10271**	4064	379	1801	0.91%
berlin52	219682	1499k	1050	1800	34.58%	**145824**	4959	22	1800	1.44%	**145824**	4752	32	1800	1.44%
brazil58	1003840	954k	379	1800	48.96%	615000	9558	1512	1800	16.69%	615000	8989	1608	1800	16.69%
st70	50666	834k	752	1800	59.43%	**21805**	3302	1149	1801	5.72%	22701	4555	111	1800	9.44%
eil76	46122	843k	1079	1800	61.03%	18969	3358	179	1800	5.23%	**18936**	3079	283	1800	5.07%
pr76	10789626	580k	248	1800	67.98%	**3619821**	4761	124	1800	4.55%	**3619821**	4424	165	1800	4.55%
gr96	9486169	294k	1738	1800	77.89%	2506860	2664	559	1801	16.34%	**2425524**	2442	477	1800	13.54%
rat99	213637	262k	1554	1800	76.81%	**62751**	3182	375	1800	21.04%	63500	3698	1523	1800	21.97%
rd100	1777195	330k	688	1800	80.87%	386357	2712	1750	1800	11.99%	**374274**	2873	877	1801	9.14%
kroA100	4718805	268k	1721	1800	79.17%	**1143792**	3415	1530	1800	14.05%	**1143792**	2920	1785	1800	14.05%
kroB100	4927266	313k	1781	1800	79.99%	**1178211**	3770	375	1800	16.31%	**1178211**	2992	574	1801	16.31%
kroC100	4620418	268k	1344	1800	79.19%	**996577**	3731	1088	1801	3.54%	1010087	2556	1509	1801	4.83%
kroD100	4768584	324k	1682	1800	85.73%	**1105147**	3806	1523	1800	38.45%	**1105147**	3184	1769	1801	38.45%
kroE100	4575812	274k	1622	1800	78.77%	**1072258**	3337	1398	1800	9.42%	**1072258**	2856	1596	1800	9.42%
eil101	95771	354k	1567	1800	75.64%	**30751**	3233	917	1800	24.14%	32555	3484	730	1800	28.35%
lin105	3615063	242k	1781	1800	83.29%	**672456**	3331	492	1800	10.19%	**672456**	2600	719	1800	10.19%

Table 5.6: Branching rule comparison

	Pure CP (no MDD)					CP^{AB}_{MDD} (Default search)					CP^{AB}_{MDD} (Cheapest Extension)				
inst	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap
pr107	14155245	175k	1746	1800	85.68%	2197074	3511	1689	1801	7.76%	**2117815**	2650	1336	1801	4.31%
avg					45.02%					6.49%					6.48%
best					2					30					31
solved					2					13					13

Table 5.7: Branching rule comparison

	CP^{AB}_{MDD} (Job Repetition)					CP^{AB}_{MDD} (Feasible Path)					CP^{AB}_{MDD} (Expected Cost)				
inst	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap
burma14	**20315**	4	0	0	0.00%	**20315**	4	0	0	0.00%	**20315**	3	0	0	0.00%
gr17	**12994**	5	0	0	0.00%	**12994**	5	0	0	0.00%	**12994**	1	0	0	0.00%
gr21	**24345**	9	1	1	0.00%	**24345**	10	1	1	0.00%	**24345**	5	1	1	0.00%
gr24	**13795**	13	1	1	0.00%	**13795**	11	1	1	0.00%	**13795**	7	1	1	0.00%
bays29	**26862**	28	5	5	0.00%	**26862**	44	6	7	0.00%	**26862**	25	5	5	0.00%
bayg29	**22230**	31	5	6	0.00%	**22230**	41	4	7	0.00%	**22230**	27	5	6	0.00%
dTSP40.0	10678	4507	1765	1800	3.44%	11886	5230	836	1800	13.25%	10363	4731	1353	1800	0.50%
dTSP40.1	10855	5475	1672	1800	9.65%	10788	6559	1132	1800	9.09%	10817	5467	1664	1800	9.34%
dTSP40.2	**9525**	2911	520	1259	0.00%	**9525**	4948	1558	1800	0.00%	**9525**	2890	512	1237	0.00%
dTSP40.3	**9156**	2343	864	1313	0.00%	**9156**	3874	1108	1445	0.00%	**9156**	2323	835	1268	0.00%
dTSP40.4	**9079**	88	23	30	0.00%	**9079**	174	38	54	0.00%	**9079**	78	23	30	0.00%
dTSP50.0	14506	6003	58	1800	12.59%	14522	4166	888	1800	12.68%	14506	5725	60	1800	12.59%
dantzig42	12845	5508	959	1800	2.47%	14334	5630	1659	1800	12.60%	12845	5496	956	1800	2.47%
swiss42	**22327**	2461	1170	1207	0.00%	23145	4949	832	1800	3.53%	**22327**	2435	1195	1233	0.00%
att48	**209320**	2309	39	1283	0.00%	211155	5655	1789	1800	0.87%	**209320**	2288	39	1289	0.00%
gr48	115845	5489	1330	1800	11.63%	112610	5214	1078	1800	9.09%	115845	5347	1359	1800	11.63%
hk48	**254330**	4479	572	1800	2.52%	281987	5071	832	1800	12.08%	**254330**	4494	568	1800	2.52%
dTSP50.1	15031	4348	1375	1800	14.49%	**12956**	4664	1336	1800	0.79%	15031	4178	1430	1800	14.49%
dTSP50.2	13381	4015	1783	1800	7.65%	14331	4328	47	1800	13.77%	13381	4001	1774	1800	7.65%
dTSP50.3	14101	4068	602	1800	9.78%	14378	4622	985	1800	11.52%	14101	4050	601	1800	9.78%
dTSP50.4	15432	3916	834	1800	13.89%	15577	3742	1333	1800	14.69%	15432	3897	834	1800	13.89%
eil51	10343	3131	713	1801	1.60%	10641	4525	1394	1800	4.35%	10343	3096	716	1801	1.60%
berlin52	151468	4140	1766	1800	5.11%	154578	4403	1571	1800	7.02%	151468	4158	1750	1800	5.11%
brazil58	**568936**	4165	1560	1800	9.94%	618713	4481	1569	1800	17.19%	**568936**	4166	1546	1800	9.94%
st70	29289	3425	1542	1800	29.81%	27548	2678	1398	1800	25.38%	29227	3419	1539	1800	29.66%
eil76	20619	2517	1633	1800	12.82%	20867	2209	1517	1800	13.85%	20619	2481	1630	1800	12.82%
pr76	3962057	4092	448	1800	12.79%	3912522	2114	815	1800	11.69%	3962057	4057	454	1800	12.79%
gr96	-			1800	0.00%	3229801	1793	1027	1800	35.07%	-			1800	0.00%
rat99	69554	1507	1620	1800	28.77%	78064	1201	1475	1801	36.53%	69095	1498	1625	1801	28.29%
rd100	458708	1889	1563	1801	25.87%	534474	1815	802	1800	36.38%	458708	1894	1549	1800	25.87%
kroA100	1346888	2420	1645	1800	27.01%	1544824	3486	695	1800	36.36%	1346888	2413	1641	1800	27.01%
kroB100	1604741	1740	1556	1801	38.56%	1548209	1584	567	1801	36.31%	1604741	1759	1537	1800	38.56%
kroC100	1462416	3709	614	1800	34.26%	1442089	1041	1773	1800	33.34%	1462416	3619	635	1800	34.26%
kroD100	1488613	2303	1344	1801	54.30%	1503006	1927	1636	1801	54.74%	1488613	2240	1381	1800	54.30%
kroE100	1309642	2270	1766	1801	25.84%	1297364	2613	1310	1800	25.14%	1315731	2170	1795	1800	26.18%
eil101	31663	2123	1427	1801	26.33%	34781	1109	1400	1801	32.93%	31572	2083	1439	1800	26.12%
lin105	793563	2408	1568	1801	23.90%	804979	1209	974	1801	24.98%	793563	2419	1560	1800	23.90%

Table 5.7: Branching rule comparison

	CP^{AB}_{MDD} (Job Repetition)					CP^{AB}_{MDD} (Feasible Path)					CP^{AB}_{MDD} (Expected Cost)				
inst	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap	obj	fails	ttb(s)	t(s)	gap
pr107	2643105	1007	1040	1800	23.32%	2442934	3896	1373	1800	17.04%	2643105	994	1028	1800	23.32%
avg					12.66%					14.80%					12.56%
best					13					10					13
solved					12					9					12

5.9 Implementation limitations

As stated before, the MDD technology is implemented as a special global constraint in IBM's ILOG CP Optimizer version 12.4. The current implementation is however hindered by a number of limitations imposed by CP Optimizer. Some of the main limitations are:

- Given a (heuristically obtained) feasible initial solution, it is not possible to warm-start the CP search when MDDs are used. For efficiency reasons, while dealing with MDDs, the CP search traverses the search nodes in a depth-first manner. CP Optimizer ignores warm-starts all together when depth-first search is enabled.
- During the traditional branch-and-bound search, primal heuristics can be used to find good feasible solutions faster. For example, it is possible to run a quick search on the MDD to find a feasible s-t path. It is however not possible to communicate solutions discovered by a primal heuristic back to CP Optimizer as CP Optimizer does not grant the user access to its internal solution pool.
- At each node of the CP search tree, a lower bound on the objective value can be calculated through the MDD. Whenever the CP search terminates before a proof of optimality is established, the weakest lower bound incurred over all unexplored search nodes is a valid bound on the optimal solution. Naturally this bound is often stronger than the bound obtained at the root node of the CP search tree. Currently CP Optimizer does not provide access to the individual search nodes so it is not possible to compute this bound.
- The MDD implementation is not thread safe; hence only 1 worker (thread) can be used in the CP search.

5.10 Conclusion

In this chapter, the TD-TSP problem is solved through an integrated Constraint Programming approach. Since CP models are often ineffective in solving sequencing problems like the TSP, MDDs are incorporated to strengthen the model. By integrating the MDDs, significantly more information about the problem structure and solution space are consolidated into the CP model. Furthermore, since bounds on the optimal solution can be computed through the MDDs, domain propagation is improved, the search tree can be pruned more

effectively, and computational effort required to prove optimality is reduced. Finally we show how structural information from different problem relaxations can be incorporated into the MDD through additive bounding.

Computational experiments clearly show that the integrated approach outperforms both traditional MIP and CP formulations for the TD-TSP problem, as it finds better solutions in less time. The additional filtering induced by the MDD propagator significantly reduces the size of the search tree. For a number of instances, the traditional CP approach could find the optimal solution, but failed to prove optimality in a reasonable amount of time. Due to the bounds calculated by the MDD propagator, the integrated approach could attest optimality for these instances in a matter of seconds.

A major drawback of current CP implementations, is the fact that CP does not provide any insight as to the quality of an incumbent solution. MDDs provide an outcome here, as lower bounds on the optimal solution are derived from the MDD at each node of the search tree. Additionally, MDDs can be used to formulate branching decisions, thereby guiding the search of the CP solver. Finally, we have shown how MDDs can be used to accommodate structural information from various problem relaxations into the CP search.

The approach presented in this work is generic in the sense that it hardly relies on problem-specific information. Therefore, future research could be aimed at expanding this work to related sequencing problems. Alternatively, one could attempt to include existing dedicated solution approaches for the TD-TSP into the current framework to improve its overall performance.

Chapter 6

The Balanced Traveling Salesman Problem

Abstract

Given a weighted graph $G(V, E)$, the 2-Balanced Traveling Salesman Problem (2-BTSP) asks for two perfect weighted matchings in G such that (1) the two matchings together form a Hamiltonian cycle in G and (2) the absolute difference in costs between the two matchings is minimized. The problem is shown to be NP-Hard, even when the graph is complete. Mixed Integer Programming models are presented to solve the 2-BTSP problem, as well as the more general k-BTSP problem for $2 \leq k \leq |V|$. One model is solved through branch-bound-cut, whereas the other model is solved through a branch-price-cut framework. Computational experiments are conducted on TSPLib instances for $k = 2$. For this particular data set, the branch-bound-cut achieves superior results over the branch-and-price approach, showing that decomposing a problem is not always beneficial. Future research should reveal whether these results are also valid for different instance classes, or for the general case where $2 \leq k \leq |V|$.

The content of this chapter is based on joint work with Bart Smeulders and Frits Spieksma, Faculty of Economics and Business, KU Leuven

6.1 Introduction

Given is a graph $G(V, E)$ where V is the set of vertices, E a set of weighted edges, and a set of k salesmen; we will assume that $|V|$ is a multiple of k. In the traditional Traveling Salesman Problem, the objective is to find a Hamiltonian cycle in a graph $G(V, E)$ while minimizing the total length of the cycle. In this work we study the problem of finding a Hamiltonian Cycle in graph $G(V, E)$ to be traversed by k salesmen as follows. The salesmen alternatingly traverse in a fixed order a single edge of the Hamiltonian Cycle. The distance traveled by a salesman equals the sum of distances of its edges. The objective is to minimize the difference between the maximum distance traveled by a salesman, and the minimum distance traveled by a salesman. Henceforth we will refer to this problem as the k-Balanced Traveling Salesman Problem (k-BTSP). Special cases of this problem arise when $k = 2$ or $k = |V|$. In the former case, the problem requires to find 2 edge-disjoint, weighted, perfect matchings in graph $G(V, E)$ such that the edges form a Hamiltonian cycle, while minimizing the absolute difference between the matchings' weights. In the latter case, the problem asks to find a Hamiltonian cycle in the graph while minimizing the difference between the longest and shortest edge in the tour. The problem for $k = |V|$ has been studied by Larusic and Punnen (2011). In this work we will primarily focus on the problem for $k = 2$, first studied by Delcour (2012), but models for the general case will be presented as well (Appendix B). An example of a 2-BTSP instance is given in Figure 6.1.

The BTSP is an example of a larger class of balancing problems where an equal or fair distribution of resources is pursued. The general mathematical form of these problems can be stated as:

$$\underset{S \in \mathcal{F}}{\text{minimize}} \{ \max_{s \in S} w(s) - \min_{s \in S} w(s) \} \tag{6.1}$$

where $\mathcal{F}$ is a family of feasible subsets (solutions) of some superset $\mathscr{S}$, and $w(s)$ a cost (weight) function which assesses the cost (weight) of element $s \in \mathscr{S}$. For instance, in the context of 2-BTSP, superset $\mathscr{S}$ encompasses all possible perfect matchings in a graph $G(V, E)$ and $\mathcal{F}$ comprises of all subsets $S \subseteq \mathscr{S}$ s.t. $|S| = 2$ and the matchings in S form a Hamiltonian cycle. The objective function in (6.1) minimizes the imbalance: the absolute difference in costs between the most expensive and least expensive element in the solution.

The general class of balancing optimization problems as defined by equation (6.1) has been characterized first by Martello et al. (1984). Many optimization problems identified in literature can be reduced to the aforementioned structure. A number of examples can be found in Martello et al. (1984); Camerini et al. (1986); Katoh and Iwano (1994); Cappanera and Scutellà (2005); Zeitlin (1981); Berežný and Lacko (2005).

The k-BTSP problem studied in this work is a direct extension of the work

of Larusic and Punnen (2011) which focuses on the special case of $|V|$-BTSP. Larusic and Punnen (2011) developed several heuristics, mainly relying on lower and upper bounding procedures, to solve $|V|$-BTSP. Larusic and Punnen (2011) argue that their algorithms can be used to solve an optimization problem originating in the maintenance of aircraft engines.
Another optimization problem related to BTSP is discussed by Bassetto and Mason (2011), who explore a periodic routing problem, asking to find two tours of minimal total length through a number of nodes (customers). Some customers must occur in both tours, others must occur in exactly one tour. The absolute difference between the number of customers in each tour is restricted from above, thereby obtaining two balanced tours. Unlike the balanced optimization problems discussed before, Bassetto and Mason (2011) do not consider reducing the imbalance between the two tours as part of their objective function; instead, the maximum allowed imbalance is part of the problem input and enforced through a constraint in the model.
The last problem, which structurally bears strong resemblance to the BTSP problem, is the Market-Split problem (Cornuéjols and Dawande, 1998; Aardal et al., 2000). The Market-Split problem attempts to minimize the total amount of slack (positive and negative) which has to be added to a set of diophantine equations to make the system feasible. The problem is often introduced with the example of a company having two sales divisions responsible for supplying retailers with products. The objective is to allocate each retailer to either one of the divisions such that each division controls a predefined fraction of the market for a given product; deviation of these predefined fractions should be minimized. A similar structure will be used to model the k-BTSP problem.
The remainder of this chapter is structured as follows. Section 6.2 analyzes the complexity of the k-BTSP problem, showing that the problem, including some special cases, is NP-Hard. Section 6.3 introduces several Mixed Integer Programming formulations for the 2-BTSP problem, including a column generation model. A branch-and-price framework to solve the column generation model is outlined in Section 6.4. Details with respect to the implementation of the models presented in the aforementioned sections are provided in Section 6.5. Experiments for 2-BTSP are reported in Section 6.6. Finally, Section B.1 extends the models presented in Section 6.3 for $k = 2$ to the general case. Section 6.7 offers the conclusion.

6.2 Complexity Analysis

In this section we analyze the computational complexity of the k-BTSP, $k \geq 2$. We show that deciding whether a feasible solution exists to an instance of k-BTSP is NP-complete, and we show that, even for a complete graph, it is

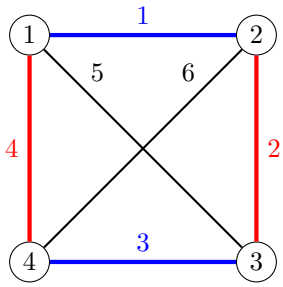


Figure 6.1: Example of a 2-BTSP solution. The blue salesman travels a distance of $1 + 3 = 4$, whereas the red salesman travels $2 + 4 = 6$. The objective of the solution equals $|4 - 6| = 2$

NP-hard to find an optimum solution.
Let us formally state the decision version of k-BTSP:

Input: an undirected, weighted graph $G = (V, E)$, an integer $k \geq 2$, with $|V|$ a multiple of k.

Goal: do there exist k disjoint edge sets $E_1, E_2, \ldots, E_k$ with $E_i \subset E$, $|E_i| = \frac{|V|}{k}$ for $i = 1, \ldots, k$ such that the edge sequence $e_1, e_2, \ldots, e_{|V|}$, with $e_i \in E_i$ (modulo k), forms a Hamiltonian Cycle (HC) in G.

It is not difficult to verify that for each fixed $k \geq 2$, the decision version of k-BTSP is at least as hard as deciding whether a graph whose number of nodes is a multiple of k admits a HC. Hence we can state the following theorem:

Theorem 2. *The decision version of k-BTSP is NP-complete for each $k \geq 2$.*

The optimization version of k-BTSP, simply referred to as k-BTSP, involves a cost $c_e \geq 0$ for each edge $e \in E$. Again, let us formally state the problem:

Input: an undirected, weighted graph $G = (V, E)$, with edge costs $c_e \geq 0$ for all $e \in E$, an integer $k \geq 2$, with $|V|$ a multiple of k.

Goal: find k disjoint edge sets $E_1, E_2, \ldots, E_k$ with $E_i \subset E$, $|E_i| = \frac{|V|}{k}$ for $i = 1, \ldots, k$ such that the edge sequence $e_1, e_2, \ldots, e_{|V|}$, with $e_i \in E_i$ (modulo k), forms a HC in G and such that $\max_i w(E_i) - \min_i w(E_i)$ is minimal, where $w(Q)$ is defined as $w(Q) = \sum_{e \in Q} c_e$ for some subset $Q \subseteq E$.

Even for complete graphs, k-BTSP is a difficult problem:

Theorem 3. *The k-BTSP is NP-hard for complete graphs, for* $k \geq 2$.

Proof. We prove the theorem for $k = 2$. Again we use a reduction from HC: given an undirected graph $H = (W, F)$, does H contain a HC? Here it is assumed wlog. that $|W|$ is even. We now build the *complete*, edge-weighted graph $G = (V, E)$ that corresponds to an instance of 2-BTSP. Let $V = W$, and for each edge $e \in F$, we introduce an edge $e \in E$, with edge cost $d_e = 0$. Consider now the edges not in F; we denote them by $\{e_1, e_2, \ldots, e_p\}$, with $p = |E \setminus F|$. Each of these edges is also present in E; we set $d_{e_j} = 2^j$, $j = 1, \ldots, p$. This completes the instance of 2-BTSP.
Observe that every edge $e \in E \setminus F$ has a unique cost, and that the largest edge cost in a set of edges that correspond to a solution of 2-BTSP, exceeds the sum of the costs of all other edges in that solution. We now argue that the existence of a HC is equivalent to the instance of 2-BTSP having an optimum solution with value 0. Clearly, if there is a HC then the two salesmen each have a matching with cost 0, leading to a difference of 0. On the other hand, suppose that the difference between the costs of the two matchings forming a HC equals 0. Consider the most expensive edge in this pair of matchings, and denote its cost by d_{max}. If $d_{max} > 0$, then, by choice of the edge-weights, the sum of the edge-weights in the other matching will be less than d_{max}. Hence, no solution with value 0 exists. It follows that $d_{max} = 0$, which implies that all edges in the two matchings that form a HC have cost 0, leading to a HC in the graph H. □

The complexity of 2-BTSP for complete graphs with edge costs in $\{1, 2\}$ remains open.

6.3 Formulations for 2-BTSP

In case of two salesmen, the problem involves selecting two edge-disjoint perfect matchings in an undirected, weighted graph. We will treat a slightly more general problem where the edges (and their costs) which may be used by a particular salesman are not necessarily the same. More precisely, we define a graph $G = (V, E_1 \cup E_2)$, where E_1 is the set of edges that may be used for the first matching, and E_2 the set of edges that may be used for the second matching. For identification purposes, the first (second) matching will be referred to as the blue (red) matching. Let M_1 (M_2) refer to the set of perfect matchings in (V, E_1) ((V, E_2)), and $M = M_1 \cup M_2$. Each edge e in M_1 (M_2) has a cost d_e^b (d_e^r). The cost of a matching $m \in M_1$ is defined as $c_m^b \equiv \sum_{e:e \in m} d_e^b$. We write '$e \in m$' to denote that edge e is in matching m. Analogous, the cost c_m^r of a

matching $m \in M_2$ is $\sum_{e:e\in m} d_e^b$. Note that this problem definition does not require that $E_1 \cap E_2 = \emptyset$. Furthermore, a single edge $e \in E_1 \cap E_2$ may have different weights in the red or the blue matching ($c_e^b = c_e^r$ does not necessarily hold). Finally, let $E = E_1 \cap E_2$ be the set of edges that can be traversed by both salesmen and define $\delta(S)$, $S \subseteq V$ as the set of edges having exactly one endpoint in S. Additionally, $\delta(v), v \in V$ is used as shorthand notation for $\delta(\{v\})$. The following two models, F0 and F1, define the 2-BTSP problem:

$$F0 : min \quad |g| \tag{6.2}$$

$$\text{s.t.} \quad \sum_{e\in\delta(i)\cap E_1} x_e^b = 1 \qquad \forall i \in V \tag{6.3}$$

$$\sum_{e\in\delta(i)\cap E_2} x_e^r = 1 \qquad \forall i \in V \tag{6.4}$$

$$x_e^b + x_e^r \leq 1 \qquad \forall e \in E_1 \cap E_2 \tag{6.5}$$

$$\sum_{e\in E_1} d_e^b x_e^b - \sum_{e\in E_2} d_e^r x_e^r = g \tag{6.6}$$

$$\sum_{e\in\delta(S)\cap E_1} x_e^b + \sum_{e\in\delta(S)\cap E_2} x_e^r \geq 2 \qquad \forall S \subset V, |S| \geq \emptyset \tag{6.7}$$

$$g \in \mathbb{R} \tag{6.8}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E_1 \cup E_2 \tag{6.9}$$

Model $F0$ uses binary variables x_e indicating whether edge $e \in E_1 \cup E_2$ is used in the solution. Constraints (6.3) and (6.4) ensure that each vertex is incident to exactly one red edge and one blue edge. Furthermore, each edge may only be assigned to a single salesman (Constraint (6.5)). Constraint (6.6) models the objective function, and Constraints (6.7) implement the subtour elimination constraints.

$$F1: min \quad |g| \tag{6.10}$$

$$\text{s.t.} \quad \sum_{m \in M_1} z_m^b = 1 \tag{6.11}$$

$$\sum_{m \in M_2} z_m^r = 1 \tag{6.12}$$

$$\sum_{m \in M:\ e \in m} (z_m^b + z_m^r) \leq 1 \qquad \forall e \in E \tag{6.13}$$

$$\sum_{m \in M_1} c_m^b z_m^b - \sum_{m \in M_2} c_m^r z_m^r = g \tag{6.14}$$

$$\sum_{e \in \delta(S)} \sum_{m \in M: e \in m} z_m^b + z_m^r \geq 2 \qquad \forall S \subset V, |S| \geq 3 \tag{6.15}$$

$$z_m^b \in \{0,1\} \qquad \forall m \in M_1 \tag{6.16}$$

$$z_m^r \in \{0,1\} \qquad \forall m \in M_2 \tag{6.17}$$

$$g \in \mathbb{R} \tag{6.18}$$

Model $F1$ explicitly selects a perfect matching for each salesman. A single matching needs to be assigned to each salesman (Constraints (6.11), (6.12)). An edge may only be traversed by a single salesman (Constraints (6.13)). Finally, Constraint (6.14) implements the objective function and Constraints (6.15) eliminate subtours.

Each of the above models is non-linear; linearizing them is however straightforward by replacing every occurrence of $|g|$ by $g^1 + g^2$, every occurrence of g by $g^1 - g^2$, and setting $g^1, g^2 \geq 0$. The continuous relaxations of the *linearized* models F0 resp. F1 will be referred to as LPF0 resp. LPF1.

Both models F0 and F1 contain an exponentially large set of subtour elimination constraints (Constraints (6.7) resp. (6.15)). When solving these models, initially only a subset of these constraints will be considered; additional constraints are separated following a cutting plane approach. More details with respect to the cutting plane procedure can be found in Section 6.5.

6.4 Column generation

While model F0 can be solved through a traditional branch-bound-cut (BBC) procedure, doing so for model F1 is not straightforward due to the possibly exponentially large set of variables. Therefore, model F1 is solved through a branch-price-cut (BPC) framework.

6.4.1 Pricing Problem

The dual of LPF1, the LP relation of model F1, is given as follows. Associate dual variables v, w, u, y, ζ with resp. constraints (6.11)-(6.15). The dual of LPF1 becomes:

$$F1_{dual} : max \quad v + w + \sum_{e \in E} u_e + 2 \sum_{\substack{S \subset V \\ |S| \geq 3}} \zeta_S \tag{6.19}$$

$$\text{s.t.} \quad v + \sum_{e \in m} u_e + c_m^b y + \sum_{\substack{S \subset V \\ |S| \geq 3}} |\delta(S) \cap m| \zeta_S \leq 0 \qquad \forall m \in M_1 \tag{6.20}$$

$$w + \sum_{e \in m} u_e - c_m^r y + \sum_{\substack{S \subset V \\ |S| \geq 3}} |\delta(S) \cap m| \zeta_S \leq 0 \qquad \forall m \in M_2 \tag{6.21}$$

$$-1 \leq y \leq 1 \tag{6.22}$$

$$u_e \leq 0 \qquad \forall e \in E \tag{6.23}$$

$$u_e = 0 \qquad \forall e \in (E_1 \cup E_2) \setminus E \tag{6.24}$$

$$\zeta_S \geq 0 \qquad \forall S \subset V, |S| \geq 3 \tag{6.25}$$

Solving the pricing problem for this problem amounts to computing a maximum weight perfect matching in graphs $G(V, E_1)$, $G(V, E_2)$ resp. with modified edge costs. For salesman $t = 1$ the modified edge costs δ_e, $e \in E_1$ are defined as

follows:

$$\delta_e = u_e + d_e^b y + \sum_{\substack{S \subset V \\ |S| \geq 3}} a_e(S) \qquad \forall e \in E_1$$

where $a_e(S) = 0$ if $e \notin \delta(S)$, ζ_S otherwise, for all $e \in E_t$, $t \in T$, $S \subset V$, $|S| \geq 3$. The pricing problem for salesman $t = 1$ then amounts to finding a perfect matching $m \in M_1$ s.t.:

$$\sum_{e \in E_1:\ e \in m} \delta_e > -v \tag{6.26}$$

Similarly, for salesman $t = 2$ we have:

$$\delta_e = u_e - d_e^b y + \sum_{\substack{S \subset V \\ |S| \geq 3}} a_e(S) \qquad \forall e \in E_2$$

resulting in the problem of finding a perfect matching $m \in M_2$ s.t. :

$$\sum_{e \in E_2:\ e \in m} \delta_e > -w \tag{6.27}$$

6.4.2 Branching

After solving LPF1 to optimality, the resulting z_m^r, z_m^b variables may be fractional. Hence a branching framework is required. If there is a fractional solution, then there is an edge $e = (i, j)$ such that either $0 < \sum_{m \in M_1:\ e \in m} z_m^b < 1$ or $0 < \sum_{m \in M_2:\ e \in m} z_m^r < 1$ holds, or both. Indeed, one easily verifies that integrality of the z_m^r, z_m^b variables implies that each edge is either part of the blue or red matching or not used at all. Suppose that $0 < \sum_{m \in M_1:\ e \in m} z_m^b < 1$ holds (analogous for $0 < \sum_{m \in M_2:\ e \in m} z_m^r < 1$). There exist two possibilities: in an optimal solution edge $e = (i, j)$ is part of the blue matching, or it is not. In the former case all edges incident to vertices i, j can be removed from E_1, except edge (i, j), and edge (i, j) may be removed from set E_2. In the latter case, edge e is removed from set E_1.

6.4.3 Initialization

Each node of the BPC tree must be initialized with a feasible subset of columns $M' \subseteq M$ which together satisfy the constraints (6.11)-(6.15) in LPF1, or one must prove that such a set of columns does not exist, rendering the node infeasible. Due to the special objective function of the BTSP, it is not obvious

whether the techniques described in Section 2.5.2 can be reused to generate a set of artificial columns with a high price as a starting solution because it is hard to guarantee that the artificial columns will price out if there exists a non-artificial solution. Furthermore, due to the subtour elimination constraints and the fact that the graphs may be incomplete, it may prove difficult to generate a set of artificial columns which collectively satisfy constraints (6.11)-(6.15). And even if this would be possible, then this would require to solve a separate subproblem at each node of the BPC tree to establish a feasible initial solution. Fortunately, all these issues can be avoided by using an approach very similar to the one used in Phase I of the Simplex algorithm (Chvátal, 1983). Instead of solving LPF1 directly, we will work with a model LPF1' which relaxes a number of the constraints in model LPF1. Indeed, by replacing hard constraints (6.11), (6.12), (6.15) by soft constraints, a model is obtained for which a trivial initial solution exists. Model LPF1' is defined as follows:

$$LPF1' : min \quad g^1 + g^2 + \lambda U \tag{6.28}$$

$$\text{s.t.} \quad \sum_{m \in M_1} z_m^b + \lambda = 1 \tag{6.29}$$

$$\sum_{m \in M_2} z_m^r + \lambda = 1 \tag{6.30}$$

$$\sum_{m \in M:\ e \in m} z_m^r + z_m^b \leq 1 \qquad \forall e \in E \tag{6.31}$$

$$\sum_{m \in M_1} c_m^b z_m^b - \sum_{m \in M_2} c_m^r z_m^r - g^1 + g^2 = 0 \tag{6.32}$$

$$\sum_{e \in \delta(S)} \sum_{m \in M: e \in m} z_m^b + z_m^r + 2\lambda \geq 2 \qquad \forall S \subset V, |S| \geq 3 \tag{6.33}$$

$$\lambda \leq 1 \tag{6.34}$$

$$z_m^b \geq 0 \qquad \forall m \in M_1 \tag{6.35}$$

$$z_m^r \geq 0 \qquad \forall m \in M_2 \tag{6.36}$$

$$g^1, g^2, \lambda \geq 0 \tag{6.37}$$

In this model, constraints (6.11), (6.12), (6.15) may be violated, but any violation is penalized in the objective, where $\lambda U > 0$ is the penalty incurred when $\lambda > 0$, with U being a fixed constant.
Associate dual variables v, w, u, y, ζ, μ with resp. constraints (6.29)-(6.34). The

dual of LPF1' becomes:

$$LPF1'_{dual}: max \quad v + w + \sum_{e \in E} u_e + \mu \tag{6.38}$$

$$\text{s.t.} \quad v + \sum_{e \in m} u_e + c^b_m y + \sum_{\substack{S \subset V \\ |S| \geq 3}} |\delta(S) \cap m| \zeta_S \leq 0 \qquad \forall m \in M_1 \tag{6.39}$$

$$w + \sum_{e \in m} u_e - c^r_m y + \sum_{\substack{S \subset V \\ |S| \geq 3}} |\delta(S) \cap m| \zeta_S \leq 0 \qquad \forall m \in M_2 \tag{6.40}$$

$$-1 \leq y \leq 1 \tag{6.41}$$

$$v + w + \mu + \sum_{\substack{S \subset V \\ |S| \geq 3}} 2\zeta_S \leq U \tag{6.42}$$

$$\mu \leq 0 \tag{6.43}$$

$$u_e \leq 0 \qquad \forall e \in E \tag{6.44}$$

$$u_e = 0 \qquad \forall e \in (E_1 \cup E_2) \setminus E \tag{6.45}$$

$$\zeta_S \geq 0 \qquad \forall S \subset V, |S| \geq 3 \tag{6.46}$$

From the construction of model LPF1' is is apparent that it always has a feasible solution having objective value U with $z^r_m = z^b_m = 0$ for all $m \in M$, and $\lambda = 1$. Furthermore, by comparing the dual formulations of models LPF1 and LPF1', it is clear that the pricing problem can be solved through the same algorithmic implementation. Finally observe that by setting $\lambda = 0$, LPF1' becomes identical to LPF1. In fact, LPF1' only yields a feasible solution to LPF1 iff $\lambda = 0$. To prove infeasibility of LPF1, we must guarantee that any solution in LPF1' with $\lambda > 0$ is more expensive in terms of the objective value than any solution to LPF1' with $\lambda = 0$. This can be achieved by setting U to a large value. However, it is well known that this reduces the stability of the column generation procedure and potentially introduces numerical issues while

implementing the model. Recall that a node in the BPC tree can be pruned if either of the following conditions holds:

1. the node is infeasible
2. the lower bound on the node exceeds the upper bound, i.e., the best incumbent integer solution.

Hence it suffices to ensure that there does not exist a solution to LPF1' with $\lambda > 0$ having an objective value less than W, where W is the value of the best incumbent integer solution or any other valid upper bound on the optimal objective value. The latter can be achieved by the following procedure:

Algorithm 9: Penalty update procedure for LPF1'

```
1 U=W
2 repeat
3 |   solve LPF1'. Let ω be the resulting objective value
4 |   if λ = 0 then
5 |   |   Feasible solution for LPF1 has been found with objective value ω
6 |   else                                              /* 0 < λ ≤ 1 */
7 |   |   U := U/λ                                      /* Increase penalty */
8 until λ = 0 ∨ ω ≥ W
```

When the procedure terminates, either a solution to LPF1' with $\lambda = 0$ is discovered, or a solution with $\lambda > 0$, $\omega \geq W$ is obtained in which case the node can be pruned.
Note that the same technique can be used at the root node of the BPC tree if no (heuristically obtained) feasible solution is available. In this case, W is initialized to a feasible upper bound on the objective value, e.g.,:

$$W = \max\{\max_{m \in M_1} c^b_m - \min_{m \in M_2} c^b_m, \max_{m \in M_2} c^r_m - \min_{m \in M_1} c^r_m\} \tag{6.47}$$

This bound is easy to calculate by computing both a min cost and a max cost perfect matching on graphs $G(V, E_1)$ and $G(V, E_2)$.
Finally note that the structure of LPF1' and its dual bares close resemblance to Du Merle's 3-piecewise stabilization method outlined in Section 2.4.2. Indeed, the introduction of the soft constraints in LPF1' adds a certain amount of stabilization to the model.

Example

An example of the procedure outlined in the previous paragraph is shown below. For each iteration of Algorithm 9, the resulting columns for LPF1' are reported.

- Iteration 1 of Algorithm 9:

Variable	Variable Value	Matching Cost
z_1^b	0.09346	1632
z_2^b	0.85981	1684
z_1^r	0.04673	1871
z_2^r	0.90654	1669

$|g| = 0.09346 \times 1632 + 0.85981 \times 1684 - 0.04673 \times 1871 - 0.90654 \times 1669 = 0$

$U = W = 86$

$\lambda = 0.04673$

$\omega = |g| + \lambda U = 0.04673 \times 86 = 4.01869$

$\omega < W$ so update $U \rightarrow U = \dfrac{86}{0.04673} = 1840.4$

- Iteration 2 of Algorithm 9:

Variable	Variable Value	Matching Cost
z_1^b	0	1632
z_2^b	1	1684
z_1^r	0	1871
z_2^r	1	1669

$U = 1840.4$

$|g| = |1684 - 1669| = 15$

$\lambda = 0$

$\omega = |g| + \lambda U = 15.0$

A feasible solution for LPF1 with value 15 has been found.

6.5 Some implementation details

The implementation of the BPC framework is fully deterministic and relies on reversible data structures. Instead of copying entire MIP models, graph structures, etc, each branch in the BPC tree introduces a number of reversible changes to the underlying data structures. When backtracking in the BPC tree, changes are automatically reverted. Since changes can be made locally, this is much cheaper and faster than copying and modifying entire data structures, especially if only small changes need to be made. Furthermore, this makes it much easier to implement different search strategies based on for example Depth First Search, Breadth First Search or Strong Branching techniques. In this work we use Depth First Search to minimize the overhead involved when backtracking.
When solving the pricing problem for the column generation procedure, a pricing problem is solved for every salesman. Strictly speaking it suffices to halt the pricing problem after the first negative reduced cost column is discovered, but in practice better results are obtained when multiple columns are returned simultaneously. Hence, computations for the pricing problem are carried out for every salesman.
While running the BBC procedure, a trade-off must be made between generating columns and generating cuts. In this work, we only generate subtour elimination constraints; no other cuts are separated. Three alternative approaches are considered:

1. Generate cuts each time the master problem is solved.
2. Generate cuts as soon as the pricing problem fails to identify columns with negative reduced cost or when the solution of the master problem equals the lower bound of the BPC node.
3. Generate cuts each time the master problem is solved and yields an integer solution.

Sorted in descending order, the above methods invoke the separation routines from often to least often. Clearly, invoking the separation routines often increases the computational overhead, but this may help in cutting off infeasible regions of the search space early in the process. In the next Section, a comparison of these three different approaches is included.
For the BBC approach, separation is only performed whenever an integer solution is encountered; computational experiments indicate that performing separation at every node of the BBC tree, including all fractional nodes, is computationally too expensive.

6.6 Computational Experiments for 2-BTSP

6.6.1 Instances

All instances used for the computational experiments are based on TSPLib instances. Recall that the number of vertices in each instance needs to be divisible by $k = 2$, the number of salesmen. If this is not the case for a particular TSPLib instance, vertices are removed, starting from the last vertex, until this condition holds. In the experiments, the edge sets E_1 and E_2 are identical. For a number of instances, a percentage p of the most expensive edges are removed, thereby obtaining an incomplete graph. To guarantee a certain amount of connectivity in the graph, an edge is only removed if its adjacent vertices have a degree larger than two. The exact number of edges removed equals $\lfloor \frac{p \times |E|}{100} \rfloor$, where $|E|$ equals the number of edges in the complete graph.

6.6.2 Experimental Results for 2-BTSP

Experiments are conducted on TSPLib instances, ranging from 14 to 318 nodes. In Table 6.2 the performance of model F0 (equations (6.2)-(6.9)), solved through BBC, is compared against model F1 which is solved through BPC (equations (6.10)-(6.18)). For each instance, an extensive number of statistics, summarized in Table 6.1, are reported. When solving the problem instances, a time limit of 600 seconds was enforced. Only the instances which were solved in less than 600 seconds (column t(ms)) are solved to optimality. Instances with "inf" in column *obj* are infeasible or no feasible solution could be found within the allotted time. The lower bound reported for the BPC method is calculated by taking the weakest bound over all open (unexplored) nodes; the lower bound reported for the BBC method is acquired directly from Cplex. All computations are performed with IBM ILOG Cplex 12.6.

When comparing the results reported in Table 6.2, it is apparent that the BBC approach significantly outperforms the BPC method. With BBC, all 45 instances were solved to optimality, whereas BPC could only solve 36 instances within the allotted time. For the hardest instances having 264 or more nodes, BPC could not find a single feasible solution. When comparing the instances which were solved by both BPC and BBC, BBC solved the instances significantly faster.
When studying the performance of the BPC approach in more detail, a number of observations can be made:

Name	TSPLib instance name (number indicates number of vertices)
red	Percentage of edges removed.
LB	Lower bound on optimal value
obj	Objective value of best solution
nodes	Number of nodes in the BPC tree which have been solved (closed)
t(ms)	instance solve time
t_{master}	time spent solving master and invoking separation routines.
$t_{pricing}$	time spent solving pricing problems
it	total number of column generation iterations performed (summed over all b&p nodes)
rest	total number of times the penalty coefficient is increased.
cols	total number of columns generated (summed over all b&p nodes)

Table 6.1: Column abbreviations in computational results

- The majority of time is spent on solving pricing problems. Currently, the pricing problems are implemented as MIP problems. These could be replaced by more efficient, dedicated implementations for max weighted matchings. However, in order to obtain better results than the BBC method, such implementation must be at least 100-1000 times faster.

- Although not apparent from the results in Table 6.2, the lack of strong bounds forms a major bottleneck for the BPC approach. Quite often, a branch in the BPC tree is fully explored. At each non-leave node, a lower bound of zero is encountered. Only at the leaves, non-zero integer solutions are discovered. Consequently it is very difficult to guide the search or to cut of branches due to the weak lower bounds. Note that, based on their experiments, Cornuéjols and Dawande (1998) drew the same conclusion, explaining why their traditional BBC approach performed poorly for the related Market-Split problem.

- A branching decision is made as soon as the master problem reaches a feasible solution equal to the lower bound of the BPC node. Typically, this often happens after only a few iterations. Hence, the number of iterations, as well as the number of generated columns per BPC node is very low.

- The penalty update scheme (Algorithm 9) works well: the number of penalty updates is significantly lower than the number of nodes in the BPC tree, meaning that for the majority of nodes no updates are required. This approach is computationally much cheaper than using a dedicated method to generate a feasible initial solution at each node of the tree (or to prove that such a solution does not exist).

- Some instances are determined to be infeasible without performing any column generation iterations (0 nodes or iterations). This is due to the fact that an initial upper bound on the optimal solution (see equation (6.47)) is calculated. If for a given salesman $t \in T$ and corresponding edge set E_t no perfect matching exists, than the instance is infeasible per definition.

As elaborated in Section 6.5, there are multiple moments during the column generation process when cuts can be generated. In Tables 6.2, 6.3, 6.4 we compare three different approaches for generating the cuts. In the BCP results in Table 6.2, cuts are generated each time the master problem is solved. In Table 6.3, cuts are generated when the column generation procedure ends (i.e., when there are no more negative reduced cost columns). Finally, in Table 6.4 cuts are generated only when the master problem yields an integer solution. Clearly, the best results are obtained when cuts are generated each time the master problem is solved, both in terms of solution speed and number of instances solved. Generating cuts only when the master problem yields an integer solution requires the least number of invocations of the separation algorithms, but the BPC tree becomes significantly larger as fewer solutions are being cut-off during the search process.
Although in the experiments reported in this section BCP obtains superior results over the BPC approach, a number of remarks are in place. The variety of instances used in the experiments is fairly limited. All but two instances in our benchmark set have an optimal objective of zero, or are simply infeasible. All instances are based on TSPLib instances and hence are defined in the Euclidean plane. It turns out to be difficult to generate feasible instances with a non-zero objective; simply reducing the number of arcs in an instance or assigning randomly selected edge weights does not suffice. It remains to be seen how BBC and BPC compare on problem instances with different characteristics. The poor performance of BPC for 2-BTSP could however be more fundamental in nature. First of all, splitting a problem into a master problem and a subproblem introduces a certain amount of overhead. In addition, by splitting the problem into two problems, the interaction or connectivity between the problems is weakened. A complex problem should only be split into multiple parts if the resulting parts are somehow easier or more efficient to solve. In case of 2-BTSP, splitting-off an easy matching problem does not seem to benefit the master problem sufficiently.

Table 6.2: Branch-Bound-Cut versus Branch-Price-Cut

		Branch-and-bound			Branch-price-cut								
Name	red(%)	LB	obj	t(ms)	LB	obj	nodes	t(ms)	t_{master}	$t_{pricing}$	it	rest	cols
burma14	50	0	0	373	0	0	186	3734	640	2574	490	22	347
burma14	60	22	22	89	22	22	71	2215	137	1937	211	11	194
burma14	70		inf	12		inf	1	54	1	49	3	0	2
burma14	75		inf	9		inf	1	17	1	14	3	0	2
burma14	80		inf	1		inf	1	28	2	24	5	1	4
ulysses16	50	0	0	57	0	0	98	1096	162	802	211	3	122
ulysses16	60	0	0	155	0	0	1458	19080	2262	15059	3692	142	2379
ulysses16	70	1	1	52	1	1	177	5126	353	4534	663	59	534
ulysses16	75		inf	8		inf	1	54	2	49	5	0	6
ulysses16	80		inf	6		inf	1	29	1	24	5	0	6
ulysses22	50	0	0	298	0	0	874	10800	1838	7729	2237	40	1413
ulysses22	60	0	0	391	0	0	121	2341	303	1874	349	8	241
ulysses22	70		inf	18		inf	1	200	3	193	7	0	10
ulysses22	75		inf	2		inf	0	2	0	0	0	0	0
ulysses22	80		inf	1		inf	0	2	0	0	0	0	0
berlin52	50	0	0	962	0	0	138	5900	1639	3749	330	4	255
berlin52	60	0	0	663	0	0	353	17838	3960	12744	1243	70	882
berlin52	70		inf	7		inf	1	247	2	239	4	1	4
berlin52	75		inf	6		inf	1	307	1	300	4	1	4
berlin52	80		inf	5		inf	0	3	0	0	0	0	0
eil76	50	0	0	2952	0	0	137	14993	4441	9585	345	0	270
eil76	60	0	0	1489	0	0	118	12399	3163	8534	265	0	248
eil76	70	0	0	1288	0	0	69	12025	3563	8090	206	0	215
eil76	75	0	0	313	0	0	123	9869	3514	5832	294	0	239
eil76	80	0	0	973	0	0	86	13686	4698	8643	326	0	320
gr96	50	0	0	13467	0	0	1898	119396	60006	41495	4723	56	2907
gr96	60	0	0	45129	0	0	1463	90051	42313	36204	3459	36	2077
gr96	70	0	0	8896	0	0	319	38973	14192	22697	810	0	609
gr96	75	0	0	11696	0	0	5126	229577	120008	82184	11975	210	7287
gr96	80	0	0	21621	0	0	613	42895	18917	21064	1307	0	970
gil262	50	0	0	171112		inf	174	600387	111474	456806	463	0	502
gil262	60	0	0	189502	0	0	328	538298	157528	353061	821	0	749
gil262	70	0	0	115102	0	0	261	459705	120343	323674	640	0	664
gil262	75	0	0	120470	0	0	315	598194	196824	386605	923	0	890
gil262	80	0	0	60457	0	0	272	560691	185758	364804	874	0	883
pr264	50	0	0	241130		inf	3	600039	5514	495817	16	0	26
pr264	60		inf	10264		inf	1	157514	1300	156112	7	0	12
pr264	70		inf	6338		inf	0	602209	0	0	0	0	0

Table 6.2: Branch-Bound-Cut versus Branch-Price-Cut

		Branch-and-bound			Branch-price-cut								
Name	red(%)	LB	obj	t(ms)	LB	obj	nodes	t(ms)	t_{master}	$t_{pricing}$	it	rest	cols
pr264	75		inf	28951		inf	1	370040	795	369137	5	0	8
pr264	80		inf	29202		inf	0	624055	0	0	0	0	0
lin318	50	0	0	189306		inf	25	600050	26119	553427	57	0	64
lin318	60	0	0	195716		inf	8	600027	12919	534859	31	0	46
lin318	70	0	0	444762		inf	12	600151	19547	544568	41	0	58
lin318	75	0	0	257885		inf	18	600042	22167	540819	56	0	76
lin318	80	0	0	25560		inf	10	600051	20546	384247	45	0	70
Total:				2196696				8764390	1166956	5760158	37151	664	25595
Opt:				45				36					

Table 6.3: Branch-Price-Cut (Cuts calculated when column generation terminates.)

		Branch-price-cut								
Name	red(%)	LB	obj	nodes	t(ms)	t_{master}	$t_{pricing}$	it	rest	cols
burma14	50	0	0	180	3681	155	2603	550	24	333
burma14	60	22	22	71	2205	78	1900	249	11	193
burma14	70		inf	1	62	0	58	5	0	4
burma14	75		inf	1	23	1	16	5	0	4
burma14	80		inf	1	36	2	29	5	1	4
ulysses16	50	0	0	98	1069	63	770	240	3	122
ulysses16	60	0	0	1027	14501	825	11454	3308	70	1737
ulysses16	70	1	1	177	5170	204	4606	759	59	534
ulysses16	75		inf	1	53	2	49	6	0	6
ulysses16	80		inf	1	28	1	25	6	0	6
ulysses22	50	0	0	721	8924	470	6438	2018	9	1051
ulysses22	60	0	0	195	2870	140	2175	545	3	281
ulysses22	70		inf	1	215	1	208	8	0	10
ulysses22	75		inf	0	3	0	0	0	0	0
ulysses22	80		inf	0	2	0	0	0	0	0
berlin52	50	0	0	106	4930	140	3439	321	0	197
berlin52	60	0	0	90	6877	162	5418	337	0	220
berlin52	70		inf	1	244	1	237	4	1	4
berlin52	75		inf	1	452	1	444	4	1	4
berlin52	80		inf	0	3	0	0	0	0	0
eil76	50	0	0	128	10416	301	7014	373	4	238
eil76	60	0	0	125	14722	457	9861	615	2	362
eil76	70	0	0	123	12876	329	9193	492	2	291
eil76	75	0	0	88	10165	155	7614	309	0	250
eil76	80	0	0	137	20092	437	15296	792	24	514
gr96	50	0	0	2438	137883	8487	49626	7343	108	3739
gr96	60	0	0	9068	410411	23563	132888	25512	31	12587
gr96	70	0	0	3034	171873	8426	59504	10409	155	5015
gr96	75	0	0	3473	157793	7036	49868	9832	0	4525
gr96	80	0	0	3286	167763	6348	66411	9898	73	4809
gil262	50		inf	123	600828	4142	520407	471	0	372
gil262	60	0	1	303	600702	8003	473659	1116	0	851
gil262	70	0	1	333	600192	7591	453585	1386	0	1010
gil262	75	0	0	253	537214	5028	438046	1042	0	858
gil262	80	0	0	276	422336	4236	321094	1023	0	734
pr264	50		inf	5	600047	551	517620	66	0	34
pr264	60		inf	1	288164	111	286794	17	0	18
pr264	70		inf	0	622053	0	0	0	0	0
pr264	75		inf	0	609757	0	0	0	0	0
pr264	80		inf	0	604529	0	0	0	0	0
lin318	50		inf	21	600043	926	565766	69	0	64
lin318	60		inf	7	600040	626	453164	65	0	46
lin318	70		inf	15	600283	1049	574250	130	0	96
lin318	75		inf	29	600019	709	571987	103	0	78
lin318	80		inf	26	600054	827	562291	139	0	110
Total:					9651603	91584	6185807	79572	581	41311
Opt:					33					

Table 6.4: Branch-Price-Cut (Cuts calculated when master problem yields integer solution.)

		Branch-price-cut								
Name	red(%)	LB	obj	nodes	t(ms)	t_{master}	$t_{pricing}$	it	rest	cols
burma14	50	0	0	779	9179	500	7287	1548	53	799
burma14	60	22	22	371	7042	341	6184	1043	89	741
burma14	70		inf	1	50	1	46	3	0	2
burma14	75		inf	1	17	1	13	3	0	2
burma14	80		inf	1	28	1	24	5	1	4
ulysses16	50	0	0	8985	81225	4991	66949	20698	1092	12127
ulysses16	60	0	0	94	1010	58	846	201	8	131
ulysses16	70	1	1	249	3725	184	3316	653	42	437
ulysses16	75		inf	5	95	6	81	14	0	13
ulysses16	80		inf	3	68	4	60	12	1	11
ulysses22	50	0	0	1479	10999	765	8423	2746	13	1439
ulysses22	60	0	0	1036	11550	749	9585	2106	29	1176
ulysses22	70		inf	1293	48255	2219	44439	3238	2	2755
ulysses22	75		inf	0	2	0	0	0	0	0
ulysses22	80		inf	0	2	0	0	0	0	0
berlin52	50	0	0	9403	97749	8559	56704	16899	114	8652
berlin52	60	0	0	133	4635	396	3828	273	0	211
berlin52	70		inf	1	248	1	241	4	1	4
berlin52	75		inf	1	285	0	279	4	1	4
berlin52	80		inf	0	4	0	0	0	0	0
eil76	50	0	0	176	10820	1399	8216	378	1	292
eil76	60	0	0	95	10221	1131	8493	249	1	211
eil76	70	0	0	8255	129048	14022	79932	15174	175	8581
eil76	75	0	0	126	8200	931	6754	305	3	249
eil76	80	0	0	791	23984	2584	18853	1720	67	1127
gr96	50	0	1	30984	600024	57896	230197	54741	249	27255
gr96	60	0	0	7541	133530	13579	56544	13086	0	5607
gr96	70	0	0	9941	190095	17626	111362	16852	0	8632
gr96	75	0	1	28344	600013	66951	379861	55651	490	29992
gr96	80	0	0	1850	50229	5174	36600	4801	429	2776
gil262	50	0	10	460	600243	82356	467455	974	0	845
gil262	60	0	0	393	436243	57485	345910	847	0	776
gil262	70	0	0	365	441617	69906	350732	806	0	747
gil262	75	0	0	656	425033	72862	325917	1375	8	1034
gil262	80	0	0	269	403318	64087	328919	706	0	713
pr264	50		inf	22	600052	4095	562508	55	0	66
pr264	60		inf	28	600026	3585	584608	67	0	78
pr264	70		inf	0	608043	0	0	0	0	0
pr264	75		inf	1	601923	509	334252	5	0	8
pr264	80		inf	0	602474	0	0	0	0	0
lin318	50		inf	26	600330	7220	556223	60	0	68
lin318	60		inf	19	600027	6001	574171	47	0	56
lin318	70		inf	12	600032	3373	571599	30	0	36
lin318	75		inf	8	601443	3738	147634	19	0	22
lin318	80		inf	11	600462	3537	269335	29	0	36
Total:					10353598	578823	6564380	217427	2869	117715
Opt:					32					

6.7 Conclusion

In this chapter a first attempt has been made to solve the 2-Balanced TSP, a special case of k-BTSP. In k-BTSP, k salesmen alternatingly traverse edges in a weighted graph until a tour (Hamiltonian cycle) is completed. The objective is to find a tour such that each salesman travels approximately the same distance. The k-BTSP belongs to a hard class of optimization problems where equity or fairness in the distribution of shared resources is aspired. We show that the problem is NP-Hard, even when the graph is complete. Two models are presented for the 2-BTSP; one can be solved through a traditional branch-bound-cut approach, whereas the other is embedded in a branch-price-cut framework. Experiments conducted for the special case where $k = 2$ indicate that the best results are obtained with the branch-bound-cut approach. Although column generation methods in general produce strong bounds, this does not seem to be the case for 2-BTSP. Due to the inability to prune the search tree effectively, each branch of the branch-and-price tree is almost always fully explored, thereby reducing the efficiency of the branch-and-price approach significantly.
The pricing problem in the branch-and-price approach amounts to finding a maximum weight matching in an arbitrary graph. In the current implementation, this problem is solved through Mixed Integer Programming. We anticipate that a performance increase can be realized when the pricing problem is solved through a dedicated matching algorithm. Furthermore, additional experiments should be conducted on a wider variety of problem instances, and for different values of k, to assess whether the results obtained for 2-BTSP are also valid for different instance classes.

Chapter 7

Conclusion

Many of today's real world optimization challenges do not involve merely a single problem; they often encompass a multitude of interconnected problems. Although many of these optimization problems can be formulated through Mixed Integer or Constraint Programming models, solving them to optimality is only possible for moderately small problem instances. The dependencies between the subproblems often yield an excessive number of conditional constraints, which deteriorate the quality of the model significantly. Furthermore, MIP and CP solvers often make poor branching decisions, due to the solvers' unawareness of the underlying problem structure. A major challenge lies in the design of decomposition procedures having the capacity to cope with the aforementioned problems efficiently. However, in practice it is often unclear how to decompose a problem in such a way that the resulting components can be solved more efficiently, without breaking the problem's structure, while, at the same time, rendering the decomposition sufficiently robust to handle changes to the problem definition. Moreover, desirably, a decomposition approach must preserve some notion of optimality, i.e., it must provide insight into the quality of a solution.

In this thesis, three decomposition approaches have been studied for four optimization problems: the School Bus Routing Problem, the Concrete Delivery Problem, the Time-Dependent TSP and the Balanced TSP. The first chapter of this thesis dealt with the SBRP for which a CG procedure has been developed. This procedure decouples the SBRP into a master problem and a subproblem: the subproblem generates bus schedules, whereas the master problem selects a set of compatible schedules and assigns them to the available buses. Real world applications of the SBRP are more involved than the problem considered in Chapter 2. Nevertheless, the CG framework proposed

in this chapter is sufficiently robust to incorporate a number of additional side constraints frequently encountered in real world applications. For instance, the pricing problem, currently solved through a Labeling algorithm, could easily accommodate time windows on the pickup times of students, or precedence relations. Similarly, the master problem could account for differences in vehicle capacities, or the availability of equipment on board the vehicles for students with special needs. Computational results on a large data set indicate that the proposed decomposition obtains significantly stronger lower and upper bounds for a large number of instances than alternative methods from related work. Furthermore, the branch-and-price framework used to solve the instances combines well with existing (heuristic) solution approaches. In fact, warm-starting the branch-and-price approach decreases computation times considerably.

The CDP was introduced in Chapter 3. The chapter provides an extensive literature comparison, proves that CDP is NP-hard, and presents a number of exact and heuristic approaches for CDP. Despite the fact that the various exact and heuristic algorithms are capable of obtaining good primal solutions, it remains challenging to establish the quality of the solutions due to the lack of strong bounding procedures. The latter is a direct consequence of the number of conditional constraints in CDP. To address this issue, Chapter 4 proposes a logic based Benders decomposition for CDP. Similar to CG procedures, the Benders decomposition decouples the problem into a master problem and a subproblem. Based on a number of problem characteristics such as travel distances to the customers, requested amount of concrete, available vehicles, their capacities and processing times, the master problem selects a number of customers to service. In turn, the subproblem attempts to establish a feasible delivery schedule for the selected customers. Cuts are added to the master problem when such a schedule does not exist. Computational experiments show that strong bounds can be acquired through this approach. For a number of instances, optimality could be attested; a reduction of the optimality gap was realized for many of the remaining instances.
Benders decompositions can be very powerful for solving large scale optimization problems. Nevertheless some important remarks should be made. A Benders procedure primarily functions as a bounding procedure. Whenever the Benders decomposition is terminated prematurely, for example due to a time limit, the procedure may only output a bound, and not necessarily a primal feasible solution. In practice this can often be alleviated by integrating a primal heuristic into the Benders procedure. Finally, in contrast to e.g., CG procedures which can be warm-started by strong primal solutions, it often proves difficult to derive a strong set of initial cuts for Benders procedures.

An integrated CP approach for the TD-TSP is developed in Chapter 5. MIP based approaches typically search over the solution space of a problem, or a relaxation thereof, in an attempt to find a globally optimal solution. In contrast, CP based techniques rely on a localized strategy where constraints are considered one-by-one, filtering inconsistent values from the variable domains until a fixed point is reached. Such an approach has the advantage that each constraint can be implemented independently through some specialized algorithm, thereby exploiting combinatorial substructures in the problem. Conversely, CP implementations are likely to miss out on certain global properties of a problem due to the fact that these properties cannot be captured through domain propagation. Similarly, communication between the constraints is restricted to the information that can be conveyed through the variable domains. When insufficient information is encoded into the variable domains, structural relations within the problem may be lost. An example of the latter problem is observed in Chapter 5.
Experiments in Chapter 5 showed that pure CP approaches for the TD-TSP perform poorly due to the fact that the constraints modeling the solution space cannot take the quality of a solution into consideration. By incorporating significantly more structural information into the CP model through the use of an MDD, considerable performance improvements are realized. MDDs model a relaxation of the solution space and, as such, are used to improve domain propagation or to guide the search. Furthermore, MDDs enable a much tighter coupling between the problem structure and its objective function. Consequently, bounds on the objective value can be derived from the MDD during the CP search, which makes efficient pruning of the search space possible. Finally, the same bounds may be used to provide insight into the quality of an incumbent solution whenever the search is terminated before optimality is established. To strengthen the bounds derived from the MDDs, Chapter 5 shows how structural information from other relaxations, such as for example Linear Program relaxations, can be projected onto the MDD via an additive bounding procedure.
The CP approach discussed in Chapter 5 is very robust because it hardly relies on any problem-specific knowledge. Consequently, changes in the problem formulation can be easily accounted for. Unlike several dedicated solution approaches for TD-TSP, the CP approach can easily accommodate time-windows, precedence relations, time lags or even resource constraints. In fact, adding additional constraints to the problem may very well increase domain propagation.

Chapter 6, and by extension Appendix B, compares two models for the k-BTSP: one relies on branch-bound-cut (BBC), whereas the other uses a branch-price-cut (BPC) approach. The experimental results indicate that for the 2-BTSP, BBC obtains superior results over BPC. Although for many optimization problems

CG based approaches produce strong bounds, as can for example be observed from Chapter 2, this is not the case for 2-BTSP. Often, entire branches of the search tree are explored, while only the leaf nodes yield non-zero lower bounds. Without stronger bounds it is not possible to prune the search tree efficiently. Future experiments for BTSP should reveal whether the current results for 2-BTSP also extend to different problem classes, and to different values of k. We do anticipate that there are problem instances, possibly for larger values of k where BBC encounters the same drawbacks as the BPC approach. This would be in line with the findings reported by Cornuéjols and Dawande (1998) who treat an optimization problem with similar characteristics.
Decompositions based on column generation work well if the original (hard) problem can be split into two easier subproblems, preferably in such a way that the pricing problem has a special structure, or can be solved by efficient dedicated algorithms. In case of the CG approach for the BTSP, the pricing problem is indeed a matching problem which can be solved efficiently in polynomial time by a dedicated algorithm. However, the master problem does not offer sufficient guidance to steer the algorithm quickly towards good solutions. This problem is partially inherent to the solution space of BTSP. For the standard TSP problem, good solutions in terms of the objective value usually are very similar, that is, near optimal solutions usually bear strong resemblance to the optimal solutions. Contrastingly, this does not hold for k-BTSP where a near optimal solution may be very different from any optimal solution. When good solutions are scattered scarcely over the search space, it becomes much harder to find them, both heuristically as well as with exact methods.

Decomposition approaches offer many advantages when it comes to solving complex, real world optimization challenges. Often, large problems can be split into smaller subproblems which can be solved efficiently through dedicated algorithms. Simultaneously, strong bounds may be derived from the underlying subproblems by exploiting their structural properties. Finally, smart decompositions may lead to reductions in development time when code can be reused for re-occurring subproblems. However, care must be taken while implementing decomposition techniques. By decoupling a problem into multiple components, structural relations between the components may be lost or severely weakened. Moreover, communication between the subproblems may cause computational overhead or could lack sufficient structural information. Lastly, it may be difficult to infer sufficient information from each of the subproblems to obtain a solution for the original problem when the relations between the components are weak.
An exiting, but very challenging road lays ahead in terms of future developments for optimization methodologies. Techniques from different areas will become more and more entangled into integrated optimization methods, thereby

combining the strengths from various disciplines. To some extent this can already be observed in today's mathematical solvers, as they no longer rely on a single algorithm, but employ a vast array of analytical and combinatorial techniques to solve optimization problems. Next generation solvers are likely to perform automated analysis of the problem structures, while decomposing them without human interference. Similarly, programming paradigms such as CP will play a crucial role when it comes to rapid development and prototyping of optimization strategies, especially due to their expressive power and solid computational performances.

Appendix A

Extension: A Column Generation Approach to the Concrete Delivery Problem

In Chapter 4 a Benders decomposition is presented for the Concrete Delivery Problem (CDP) which decomposes the problem into a master problem and a subproblem: the master problem assigns concrete to customers, whereas the subproblem establishes a feasible schedule satisfying all scheduling and routing constraints. An alternative decomposition can be realized through Column Generation (CG). As a possible extension to the current work, this appendix presents a reformulation of model CDP2 (Constraints (3.21)-(3.34)) in Chapter 3 which can be solved through CG. As a result, the problem is decomposed into a master problem and a separate subproblem per vehicle. The subproblems independently generate feasible schedules for each vehicle, while the master problem attempts to select a set of compatible schedules such that they collectively satisfy all scheduling constraints.

Parameter	**Description**
C	Set of customers
K	Set of heterogeneous vehicles
p_k	Processing time of truck $k \in K$
q_i	Requested amount of concrete by customer $i \in C$
$n(i)$	Upper bound on the number of deliveries required to satisfy the demand of customer $i \in C$
$[a_i, b_i]$	Interval during which concrete may be delivered for customer $i \in C$
γ	Maximum allowed time lag between two consecutive deliveries
$\mathcal{R}_k$	Set of all feasible delivery schedules for a vehicle $k \in K$
$\mathcal{R}$	$\bigcup_{k \in K} \mathcal{R}_k$
q_i^r	Total amount of concrete delivered to customer $i \in C$ in route $r \in \mathcal{R}_k$
d_{ij}^r	Binary parameter indicating whether delivery $j \in \{1, \ldots, n(i)\}$ for customer $i \in C$ is made in route $r \in \mathcal{R}_k$
c_{it}^r	Binary parameter indicating whether concrete for customer $i \in C$ is delivered at time $t \in [a_i, b_i]$ in route $r \in \mathcal{R}_k$
t_{ij}^r	Integer parameter indicating when delivery $j \in \{1, \ldots, n(i)\}$ for customer $i \in C$ is made in route $r \in \mathcal{R}_k$

Table A.1: Parameters used in the master problem reformulation for CDP

The master problem is as follows:

$$max. \sum_{i \in C} q_i y_i \tag{A.1}$$

$$\text{s.t.} \sum_{k \in K} \sum_{r \in \mathcal{R}_k} q_i^r z_{rk} \geq q_i y_i \qquad \forall i \in C \tag{A.2}$$

$$\sum_{r \in \mathcal{R}_k} z_{rk} = 1 \qquad \forall k \in K \tag{A.3}$$

$$\sum_{k \in K} \sum_{r \in \mathcal{R}_k} d_{ij}^r z_{rk} \leq 1 \qquad \forall i \in C, j \in \{1, \ldots, n(i)\} \tag{A.4}$$

$$\sum_{k \in K} \sum_{r \in \mathcal{R}_k} c_{it}^r z_{rk} \leq 1 \qquad \forall i \in C, t \in [a_i, b_i] \tag{A.5}$$

$$\sum_{k \in K} \sum_{r \in \mathcal{R}_k} t_{ij}^r z_{rk} + p_k + \gamma \geq \sum_{k \in K} \sum_{r \in \mathcal{R}_k} t_{ij+1}^r z_{rk} \qquad \forall i \in C, j \in \{1, \ldots, n(i)-1\} \tag{A.6}$$

$$\sum_{k \in K} \sum_{r \in \mathcal{R}_k} d_{iu}^r z_{rk} \geq \sum_{k \in K} \sum_{r \in \mathcal{R}_k} d_{iv}^r z_{rk} \qquad \forall i \in C, u, v \in \{1, \ldots, n(i)\}, u < v \tag{A.7}$$

$$z_{rk} \in \{0, 1\} \qquad \forall r \in \mathcal{R}_k, k \in K \tag{A.8}$$

$$y_i \in \{0, 1\} \qquad \forall i \in C \tag{A.9}$$

Two sets of binary variables are used in this model. The z_{rk} variables assign a route $r \in \mathcal{R}_k$ to a vehicle $k \in K$, selected from the set of all feasible routes $\mathcal{R}_k$ for this particular vehicle. The y_i, $i \in C$, variables indicate whether customer i receives its requested amount of concrete. The objective function (Equation (A.1)) maximizes the number of satisfied customers, weighted by their demand. A customer is satisfied if at least its demand in concrete is delivered (Constraints (A.2)). A valid delivery schedule must be assigned to each vehicle $k \in K$ (Constraints (A.3)). The schedules assigned to the vehicles need to be synchronized. A particular delivery $j \in \{1, \ldots, n(i)\}$ for a customer $i \in C$ may only be made once (Constraints (A.4)), deliveries for the same customer may not overlap in time (Constraints (A.5)), and consecutive deliveries for the same customer must adhere to the maximum allowed time lag (Constraints (A.6)). Finally, a delivery $v \in \{2, \ldots, n(i)\}$ for customer $i \in C$ cannot be made if a delivery $u \in \{1, \ldots, n(i)-1\}$, $u < v$ has not been made (Constraints (A.7)).
The linear relaxation of the model stated above can be solved through Column

Generation. The dual of the linearized model is as follows:

$$min. \sum_{k \in K} \beta_k +$$

$$\sum_{i \in C} \left[\sum_{j \in \{1,\ldots,n(i)\}} \delta_{ij} + \sum_{t \in [a_i, b_i]} \epsilon_{it} - (\sum_{k \in K} p_k + \gamma) \sum_{j \in \{1,\ldots,n(i)-1\}} \lambda_{ij} \right] \tag{A.10}$$

$$\text{s.t.} \sum_{i \in C} \left[q_i^r \alpha_i + \sum_{j \in 1,\ldots,n(i)} d_{ij}^r \delta_{ij} + \sum_{t \in [a_i, b_i]} c_{it}^r \epsilon_{it} \right] +$$

$$\sum_{i \in C} \left[\sum_{j \in 1,\ldots,n(i)-1} t_{ij}^r \lambda_{ij} - t_{ij+1}^r \lambda_{ij} \right] +$$

$$\sum_{i \in C} \left[\sum_{\substack{u,v \in \{1,\ldots,n\} \\ u<v}} d_{iu}^r \mu_{iuv} - d_{iv}^r \mu_{iuv} \right] \geq -\beta_k \qquad \forall k \in K, r \in \mathcal{R}_k \tag{A.11}$$

$$\alpha_i \leq -1 \qquad \forall i \in C \tag{A.12}$$

$$\beta_k \in \mathbb{R} \qquad \forall k \in K \tag{A.13}$$

$$\delta_{ij} \geq 0 \qquad \forall i \in C, j \in \{1, \ldots, n(i)\} \tag{A.14}$$

$$\epsilon_{it} \geq 0 \qquad \forall i \in C, t \in [a_i, b_i] \tag{A.15}$$

$$\lambda_{ij} \leq 0 \qquad \forall i \in C, j \in \{1, \ldots, n(i) - 1\} \tag{A.16}$$

$$\mu_{ij} \leq 0 \qquad \forall i \in C \tag{A.17}$$

Notice that in the dual formulation there exists a constraint per vehicle. Consequently, the pricing problem decouples into $|K|$ different subproblems, each of which can be solved independently on a time-space network. Future research should indicate whether the resulting procedure yields stronger bounds than, for example, the Benders' procedure from Chapter 4, or perhaps even better primal solutions.

Appendix B

Extension: Generalizing 2-BTSP

B.1 k-balanced TSP

Naturally, the models presented in Chapter 6 can be generalized from 2 salesmen to k salesmen, where $2 \leq k \leq |V|$. Although we do not intend to include computational results for the general case, for completeness sake we cover the mathematical models for k-BTSP in this appendix.
In contrast to 2-BTSP, the k-BTSP for $2 < k < |V|$ has to be defined on a *directed* graph $G(V, A)$, where V is the set of vertices, and A the set of weighted arcs. Let T be an *ordered* set of k salesmen, and A_t the set of arcs that may be used by salesman $t \in T$, with $A = \bigcup_{t \in T} A_t$. While solving the k-BTSP for $k > 2$, the problem no longer amounts to finding k *perfect* matchings which collectively form a Hamiltonian Cycle. Instead, the problem requires to find k matchings, each consisting of exactly $\frac{|V|}{k}$ arcs, which together form a Hamiltonian Cycle. It must be emphasized that the salesmen traverse arcs alternatingly, so the cycle is strictly traversed in the order $[t_1, t_2, \ldots, t_k, t_1, t_2, \ldots, t_k]$, $t_i \in T$. M_t for $t \in T$ is defined as the set of matchings in (V, A_t) consisting of exactly $\frac{|V|}{k}$ arcs, and $M = \bigcup_{t \in T} M_t$.
To simplify notation, two sets of parameters are introduced: u_{im}, for $i \in V$, $m \in M$ and v_{am} for $a \in A$, $m \in M$. u_{im} equals 1 if matching $m \in M$ is incident to vertex $i \in V$, 0 otherwise. Similarly, v_{am} equals 1 if arc $a \in A$ occurs in matching $m \in M$, 0 otherwise. Finally, $\delta^+(S)$ resp. $\delta^-(S)$ are used to denote the set of arcs having resp. either their tail or their head in $S \subseteq V$, and

$\delta(S) = \delta^+(S) \cup \delta^-(S)$.

B.1.1 MIP model with $|\mathbf{E}|\mathbf{k}$ variables

Model F0 (Equations (6.2)-(6.9)) for 2-BTSP, solvable through branch-bound-cut, can be extended to k salesmen as follows:

$$F2: min \quad w \tag{B.1}$$

$$\text{s.t.} \quad w \geq \sum_{a \in A} d_a^s x_{as} - d_a^t x_{at} \qquad \forall s, t \in T \tag{B.2}$$

$$\sum_{t \in T} \sum_{a \in \delta^+(i)} x_{at} = 1 \qquad \forall i \in V \tag{B.3}$$

$$\sum_{e \in \delta^+(i)} x_{at} = \sum_{a \in \delta^-(i)} x_{a(t+1)\%k} \qquad \forall i \in V, t \in T \tag{B.4}$$

$$\sum_{t \in T} \sum_{a \in \delta(S)} x_{at} \geq 2 \qquad \forall S \subset V, S \neq \emptyset \tag{B.5}$$

$$x_{at} \in \{0, 1\} \qquad \forall t \in T, a \in A_t \tag{B.6}$$

Here, binary variable x_{at} indicates whether salesman $t \in T$ travels along arc $a \in A$. It is assumed that $x_{at} = 0$ for all $t \in T, a \notin A_t$. The '%' sign in Equation (B.4) is the modulus sign.

B.1.2 Column generation model for $2 \leq \mathbf{k} < |\mathbf{V}|$

Model F1 (Equations (6.10)-(6.18)), solvable through branch-price-cut, is extended to the general case as follows:

$$F3: min \quad c^+ - c^- \tag{B.7}$$

$$\text{s.t.} \quad \sum_{m \in M} c_m z_m^t \leq c^+ \qquad \forall t \in T \tag{B.8}$$

$$\sum_{m \in M} c_m z_m^t \geq c^- \qquad \forall t \in T \tag{B.9}$$

$$\sum_{m \in M} z_m^t = 1 \qquad \forall t \in T \tag{B.10}$$

$$\sum_{m \in M} \sum_{t \in T} u_{im} z_m^t = 2 \qquad \forall i \in V \tag{B.11}$$

$$\sum_{m \in M} \sum_{t \in T} v_{am} z_m^t \leq 1 \qquad \forall a \in A \tag{B.12}$$

$$\sum_{m \in M} \sum_{a \in \delta^-(i)} v_{am} z_m^t = \sum_{m \in M} \sum_{a \in \delta^+(i)} v_{am} z_m^{(t+1)\%k} \qquad \forall i \in V, t \in T \tag{B.13}$$

$$\sum_{t \in T} \sum_{a \in \delta(S)} \sum_{m \in M} v_{am} z_m^t \geq 2 \qquad \forall S \subset V, S \neq \emptyset \tag{B.14}$$

$$z_m^t \in \{0,1\} \qquad \forall t \in T, m \in M_t \tag{B.15}$$

$$c^+, c^- \geq 0 \tag{B.16}$$

In the above model, z_m^t is a boolean variable indicating whether matching $m \in M$ is assigned to salesman $t \in T$. It is assumed that $z_m^t = 0$ for all $t \in T, m \in M \setminus M_t$. The objective function (Equation (B.7)) calculates the absolute difference between the most expensive and the least expensive matching assigned to the salesmen. Constraints (B.8), (B.9) are used to calculate the absolute value in the objective function. The remaining constraints model the problem. First, Constraints (B.10) ensure that exactly one matching is assigned to each salesman. Next, Constraints (B.11) and (B.12) require resp. that each city $i \in V$ is visited, and that each arc is used at most once. Each segment of the route must be covered by a different salesman, i.e if salesman $t \in T$ travels from i to j, then salesman $(t+1)$ must travel from j to k, $i, j, k \in V$. This behavior is implemented by Constraints (B.13). Subtour elimination is performed by Constraints (B.14). Recall the DFJ subtour elimination constraints present in

the first model (Constraints (B.5)):

$$\sum_{t \in T} \sum_{a \in \delta(S)} x_{at} \geq 2 \qquad \forall S \subset V, S \neq \emptyset \tag{B.17}$$

where x_{at} is a boolean variable indicating whether salesman $t \in T$ covers arc $a \in A$. Notice that the following relation holds due to Equations (B.12) and (B.15):

$$0 \leq x_{at} = \sum_{m \in M} v_{am} z_m^t \leq 1 \qquad \forall t \in T, a \in A \tag{B.18}$$

By substituting Equation (B.18) into Equation (B.17), Equation (B.14) is obtained.
Finally, notice that Constraints (B.11), and (B.12) may be omitted as they are implied by Constraints (B.10), (B.14); nevertheless they reinforce the model when only a subset of the subtour elimination constraints are present.

Dual model

Associate the following dual variables with constraints (B.8)-(B.14) respectively:

$$\alpha_t^+, \alpha_t^-, \beta_t \qquad \forall t \in T \qquad Eq : (B.8) - (B.10) \tag{B.19}$$

$$\gamma_i \qquad \forall i \in V \qquad Eq : (B.11) \tag{B.20}$$

$$\delta_a \qquad \forall a \in A \qquad Eq : (B.12) \tag{B.21}$$

$$\epsilon_{it} \qquad \forall i \in V, t \in T \qquad Eq : (B.13) \tag{B.22}$$

$$\zeta_S \qquad \forall S \subset V, S \neq \emptyset \qquad Eq : (B.14) \tag{B.23}$$

Following the same notation as before, let LPF3 be the linear program relaxation of model F3. The dual of LPF3 becomes:

$$LPF3_{dual}: max \quad \sum_{i\in V} 2\gamma_i - \sum_{a\in A}\delta_a + 2\sum_{\substack{S\subset V\\ S\neq\emptyset}}\zeta_S + \sum_{t\in T}\beta_t \tag{B.24}$$

$$\text{s.t.} \quad -c_m\alpha_t^+ + c_m\alpha_t^- + \beta_t - \sum_{a\in A} v_{am}\delta_a + \sum_{\substack{a\in\delta(S)\\ S\subset V\\ S\neq\emptyset}} v_{am}\zeta_S +$$

$$\sum_{i\in V}\left(u_{im}\gamma_i + \sum_{a\in\delta^-(i)} v_{am}\epsilon_{it} - \sum_{a\in\delta^+(i)} v_{am}\epsilon_{it}\right) \leq 0 \qquad \forall m\in M,\ t\in T \tag{B.25}$$

$$\sum_{t\in T}\alpha_t^+ \leq 1 \tag{B.26}$$

$$\sum_{t\in T}\alpha_t^- \geq 1 \tag{B.27}$$

$$\alpha_t^+, \alpha_t^- \geq 0 \qquad \forall t\in T \tag{B.28}$$

$$\delta_a, \zeta_S \geq 0 \qquad \forall a\in A,\ S\subset V, S\neq\emptyset \tag{B.29}$$

$$\beta_t, \gamma_i, \epsilon_{it} \in \mathbb{R} \qquad \forall t\in T, i\in V \tag{B.30}$$

Pricing Problem

Let $G(V, A)$ be the directed, weighted graph on which k-BTSP is defined, where the travel cost d_a^t of arc $a \in A$ depends on the salesman $t \in T$ traversing the arc. For a given salesman $t \in T$, define the modified arc cost as:

$$c_a^t = d_a^t\alpha_t^- - d_a^t\alpha_t^+ - \delta_a + \gamma_i + \gamma_j - \epsilon_{it} + \epsilon_{jt} + \sum_{\substack{S\subset V\\ S\neq\emptyset}} b_a(S) \qquad \forall t\in T, a=(i,j)\in A_t$$

where $b_a(S) = 0$ if $a \notin \delta(S)$, and $b_a(S) = \zeta_S$ otherwise, for all $a \in A$, $S \subset V, S \neq \emptyset$.

Recall that a matching $m \in M$ consists of a set of $\frac{|V|}{k}$ arcs, none of which share a start or end vertex. For a given salesman $t \in T$, the pricing problem now amounts to finding a matching $m \in M$ s.t.:

$$\sum_{a\in A} v_{am}c_a^t > -\beta_t \tag{B.31}$$

The MIP model of the pricing problem for a salesman $t \in T$:

$$PR: max \quad \sum_{a \in A} c_a^t x_a \tag{B.32}$$

$$\text{s.t.} \quad \sum_{a \in A} x_a = \frac{|V|}{k} \tag{B.33}$$

$$\sum_{a \in \delta(i)} x_a \leq 1 \qquad \forall i \in V \tag{B.34}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A_t \tag{B.35}$$

$$x_a = 0 \qquad \forall a \in A \setminus A_t \tag{B.36}$$

B.1.3 Special case $|\mathbf{V}| = \mathbf{k}$

When $k = |V|$, the pricing problem simplifies to selecting a single arc. The z_m^t variables can then be translated to assigning an arc to a salesman, which renders them identical to the x_{at} variables in the model presented in Section B.1.1. Therefore, one should not solve this special case through column generation but use the branch-bound-cut approach instead. Also note that, similar to the k-BTSP problem with $k = 2$, the problem of $|V| = k$ can be modeled on an *undirected* graph.

Bibliography

K. Aardal, R. E. Bixby, C. A. J. Hurkens, A. K. Lenstra, and J. W. Smeltink, "Market split and basis reduction: Towards a solution of the Cornuéjols-Dawande instances," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 192–202, 2000.

H. G. Abeledo, R. Fukasawa, A. A. Pessoa, and E. Uchoa, "The time dependent traveling salesman problem: Polyhedra and algorithm," *Math. Program. Comput.*, vol. 5, no. 1, pp. 27–55, 2013.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993, ch. 5.

M. Albareda Sambola, "Models and algorithms for location-routing and related problems," Ph.D. dissertation, Universitat Politècnica de Catalunya, June 2003.

H. B. Amor and J. Desrosiers, "A proximal trust-region algorithm for column generation stabilization," *Computers and Operations Research*, vol. 33, pp. 910–927, April 2006.

H. B. Amor, J. Desrosiers, and A. Frangioni, "On the choice of explicit stabilizing terms in column generation," *Discrete Applied Mathematics*, vol. 157, no. 6, pp. 1167 – 1184, 2009.

H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, "A constraint store based on multivalued decision diagrams," in *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, ser. CP'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 118–132.

L. Asbach, U. Dorndorf, and E. Pesch, "Analysis, modeling and solution of the concrete delivery problem," *European Journal of Operational Research*, vol. 193, no. 3, pp. 820 – 835, 2009.

L. Bai and P. A. Rubin, "Combinatorial Benders cuts for the minimum tollbooth problem." *Operations Research*, vol. 57, no. 6, pp. 1510–1522, 2009.

R. Baldacci, M. Dell'Amico, and J. S. González, "The capacitated *m*-ring-star problem," *Operations Research*, vol. 55, no. 6, pp. 1147–1162, 2007.

J. F. Bard and S. Rojanasoonthon, "A branch-and-price algorithm for parallel machine scheduling with time windows and job priorities," *Naval Research Logistics (NRL)*, vol. 53, no. 1, pp. 24–44, 2006.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, no. 3, pp. pp. 316–329, 1998.

T. Bassetto and F. Mason, "Heuristic algorithms for the 2-period balanced traveling salesman problem in euclidean graphs," *European Journal of Operational Research*, vol. 208, no. 3, pp. 253 – 262, 2011.

J. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, pp. 238 – 252, 1962.

Š. Berežný and V. Lacko, "The color-balanced spanning tree problem," *Kybernetika*, vol. 41, no. 4, pp. 539–546, 2005.

L. Berghman, R. Leus, and F. Spieksma, "Optimal solutions for a dock assignment problem with trailer transportation," *Annals of Operations Research*, pp. 1–23, 2011.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker, "Discrete optimization with decision diagrams," Carnegie Mellon University, Tech. Rep., 2013. [Online]. Available: http://www.andrew.cmu.edu/user/vanhoeve/papers/discrete_opt_with_DDs.pdf

M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. Lübbecke, E. Malaguti, and E. Traversi, "Automatic Dantzig–Wolfe reformulation of mixed integer programs," *Mathematical Programming*, pp. 1–34, 2014.

L.-P. Bigras, M. Gamache, and G. Savard, "The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times." *Discrete Optimization*, vol. 5, no. 4, pp. 685–699, 2008.

L. Bodin and L. Berman, "Routing and scheduling of school buses by computer," *Transportation Science*, vol. 13, no. 2, pp. 113–129, 1979.

R. Bowerman, B. Hall, and P. Calami, "A multiobjective optimization approach to urban school bus routing : Formulation and solution method," *Transportation Research: Part A, Policy and Practice*, vol. 29, pp. 107–123, 1995.

O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck, "Comparison of bundle and classical column generation," *Mathematical Programming*, vol. 113, pp. 299–344, January 2008.

P. M. Camerini, F. Maffioli, S. Martello, and P. Toth, "Most and least uniform spanning trees," *Discrete Applied Mathematics*, vol. 15, no. 2–3, pp. 181 – 197, 1986.

P. Cappanera and M. G. Scutellà, "Balanced paths in acyclic networks: Tractable cases and related approaches," *Netw.*, vol. 45, no. 2, pp. 104–111, Mar. 2005.

L. Chapleau, J.-A. Ferland, and J.-M. Rousseau, "Clustering for routing in densely populated areas," *European Journal of Operational Research*, vol. 20, no. 1, pp. 48–57, April 1985.

V. Chvátal, *Linear Programming*. W. H. Freeman, 1983, ch. 13.

A. A. Ciré and W. J. van Hoeve, "Multivalued decision diagrams for sequencing problems," *Operations Research*, vol. 61, no. 6, pp. 1411–1428, 2013.

G. Codato and M. Fischetti, "Combinatorial Benders' cuts for mixed-integer linear programming," *Operations Research*, vol. 54, no. 4, pp. 756–766, 2006.

M. Conforti, G. Cornuejols, and G. Zambelli, *Integer programming*, ser. Graduate Texts in Mathematics. Springer, 2014, to appear in Fall 2014.

J.-F. Cordeau, G. Ghiani, and E. Guerriero, "Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem," *Transportation Science*, vol. 48, no. 1, pp. 46–58, 2012.

G. Cornuéjols and M. Dawande, *A Class of Hard Small 0—1 Programs*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, vol. 1412, pp. 284–293.

J.-F. Côté, M. Dell'Amico, and M. Iori, "Combinatorial Benders' cuts for the strip packing problem," Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation and Department of Computer Science and Operations Research, Université de Mon (CIRELT), Tech. Rep. CIRRELT-2013-27, April 2013.

E. Delcour, "Two salesmen and a bike," Master's thesis, Faculty of Economics, KU Leuven, 2012.

M. Dell'Amico, G. Righini, and M. Salani, "A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection," *Transportation Science*, vol. 40, pp. 235–247, May 2006.

G. Desaulniers, J. Desrosiers, and M. Solomon, Eds., *A Primer in Column Generation*. Springer US, 2005.

J. Desrosiers, J. Ferland, J.-M. Rousseau, G. Lapalme, and L. Chapleau, "TRANSCOL - a multiperiod school bus routing and scheduling system," *Delivery of Urban Services - TIMS Studies in the Management Sciences 22*, pp. 47–71, 1986.

G. Dulac, J. A. Ferland, and P. A. Forgues, "School bus routes generator in urban surroundings," *Computers and Operations Research*, vol. 7, no. 3, pp. 199 – 213, 1980.

M. Durbin and K. Hoffman, "OR Practice - The Dance of the Thirty-Ton Trucks: Dispatching and Scheduling in a Dynamic Environment." *Operations Research*, vol. 56, no. 1, pp. 3–19, 2008.

M. Fischetti and P. Toth, "An additive bounding procedure for combinatorial optimization problems," *Operations Research*, vol. 37, no. 2, pp. pp. 319–328, 1989.

R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. L. Reis, E. Uchoa, and R. F. F. Werneck, "Robust branch-and-cut-and-price for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 106, no. 3, pp. 491–511, 2006.

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

A. Geoffrion, "Generalized Benders decomposition," *Journal of Optimization Theory and Applications*, vol. 10, pp. 237 – 260, 1972.

——, "Lagrangean relaxation for integer programming," in *Approaches to Integer Programming*, ser. Mathematical Programming Studies, M. Balinski, Ed. Springer Berlin Heidelberg, 1974, vol. 2, pp. 82–114.

L. Gouveia and S. Voss, "A classification of formulations for the (time-dependent) traveling salesman problem," *European Journal of Operational Research*, vol. 83, no. 1, pp. 69 – 82, 1995.

L. D. Graham, D. R. Forbes, and S. D. Smith, "Modeling the ready mixed concrete delivery system with neural networks," *Automation in Construction*, vol. 15, no. 5, pp. 656 – 663, 2006.

M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

A. Hertz, M. Uldry, and M. Widmer, "Integer linear programming models for a cement delivery problem," *European Journal of Operational Research*, vol. 222, no. 3, pp. 623 – 631, 2012.

J. Hooker, "Planning and scheduling by logic-based benders decomposition," *Operations Research*, vol. 55, pp. 588 – 602, 2007.

——, "Lecture notes in linear programming," Lecture Notes, 2013.

E. A. Hoshino and C. C. de Souza, "A branch-and-cut-and-price approach for the capacitated m-ring-star problem," *Discrete Appl. Math.*, vol. 160, no. 18, pp. 2728–2741, Dec. 2012.

——, "A branch-and-cut-and-price approach for the capacitated m-ring-star problem," *Electronic Notes in Discrete Mathematics*, vol. 35, pp. 103 – 108, 2009.

N. Katoh and K. Iwano, "Efficient algorithms for minimum range cut problems," *Networks*, vol. 24, no. 7, pp. 395–407, 1994.

J. Kinable, F. Spieksma, and G. Vanden Berghe, "School bus routing—a column generation approach," *International Transactions in Operational Research*, vol. 21, no. 3, pp. 453–478, 2014.

J. Kinable, T. Wauters, and G. Vanden Berghe, "The concrete delivery problem," *Computers and Operations Research*, vol. 48, pp. 53–68, 2014.

J. Kinable and M. Trick, *A Logic Based Benders' Approach to the Concrete Delivery Problem*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8451, pp. 176–192.

J. Kinable and T. Wauters, "CDPLib." https://sites.google.com/site/cdplib/, 2013.

P. Laborie and J. Rogerie, "Reasoning with conditional time-intervals," in *FLAIRS Conference*, 2008, pp. 555–560.

P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "Reasoning with conditional time-intervals. part ii: An algebraical model for resources," in *FLAIRS Conference*, 2009.

J. Larusic and A. P. Punnen, "The balanced traveling salesmanproblem," *Computers and Operations Research*, vol. 38, no. 5, pp. 868–875, May 2011.

L. S. Lasdon, *Optimization theory for large systems*, ser. Macmillan series in operations research. Macmillan, 1970.

P.-C. Lin, J. Wang, S.-H. Huang, and Y.-T. Wang, "Dispatching ready mixed concrete trucks under demand postponement and weight limit regulation," *Automation in Construction*, vol. 19, no. 6, pp. 798 – 807, 2010.

M. E. Lübbecke, "Column generation," *Wiley Encyclopedia of Operations Research and Management Science (EORMS)*, 2010.

M. E. Lübbecke and J. Desrosiers, "Selected topics in column generation," *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2002.

S. Martello, W. Pulleyblank, P. Toth, and D. de Werra, "Balanced optimization problems," *Operations Research Letters*, vol. 3, no. 5, pp. 275 – 278, 1984.

A. Martin, "Integer programs with block structure," Konrad-Zuse-Zentrum für Informationstechnik Berlin, Preprint SC 99-03, 1999, habilitation Thesis.

O. D. Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, "Stabilized column generation," *Discrete Math*, vol. 194, pp. 229–237, 1997.

J. J. Miranda-Bront, I. Méndez-Díaz, and P. Zabala, "An integer programming approach for the time-dependent tsp," *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 351–358, 2010.

M. Misir, W. Vancroonenburg, K. Verbeeck, and G. Vanden Berghe, "A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete," in *Proceedings of the 9th Metaheuristics International Conference,*, L. Di Gaspero, A. Schaerf, and T. Stützle, Eds., Jul. 2011, pp. 289–298.

D. Naso, M. Surico, B. Turchiano, and U. Kaymak, "Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2069 – 2099, 2007.

A. Oukil, H. B. Amor, J. Desrosiers, and H. E. Gueddari, "Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems," *Computers and Operations Research*, vol. 34, no. 3, pp. 817 – 834, 2007.

J. Park and B.-I. Kim, "The school bus routing problem: A review," *European Journal of Operational Research*, vol. 202, no. 2, pp. 311 – 319, 2010.

J. Park, H. Tae, and B.-I. Kim, "A post-improvement procedure for the mixed load school bus routing problem," *European Journal of Operational Research*, vol. 217, no. 1, pp. 204 – 213, 2012.

G. Pesant, C.-G. Quimper, and A. Zanarini, "Counting-based search: Branching heuristics for constraint satisfaction problems," *Journal of Artificial Intelligence Research*, vol. 43, pp. 173–210, 2012.

J. C. Picard and M. Queyranne, "The Time-dependent Traveling Salesman Problem and its Application to the Tardiness Problem in One-machine Scheduling," *Operations Research*, vol. 26, no. 1, pp. 86–110, 1978.

R. Rasmussen and M. Trick, "A Benders approach for the constrained minimum break problem," *European Journal of Operational Research*, vol. 177, no. 1, pp. 198–213, 2007.

C. R. Reeves, Ed., *Modern Heuristic Techniques for Combinatorial Problems*. New York, NY, USA: John Wiley & Sons, Inc., 1993.

J. Riera-Ledesma and J. J. Salazar-González, "Solving school bus routing using the multiple vehicle traveling purchaser problem: A branch-and-cut approach," *Computers and Operations Research*, vol. 39, no. 2, pp. 391 – 404, 2012.

——, "A column generation approach for a school bus routing problem with resource constraints," *Computers and Operations Research*, vol. 40, no. 2, pp. 566 – 583, 2013.

S. Ropke and J.-F. Cordeau, "Branch and cut and price for the pickup and delivery problem with time windows," *Transportation Science*, vol. 43, pp. 267–286, August 2009.

L. Rousseau, M. Gendreau, and D. Feillet, "Interior point stabilization for column generation," *Operations Research Letters*, vol. 35, no. 5, pp. 660–668, Sep. 2007.

P. Schittekat, J. Kinable, K. Sörensen, M. Sevaux, F. Spieksma, and J. Springael, "A metaheuristic for the school bus routing problem with bus stop selection," *European Journal of Operational Research*, vol. 229, pp. 518–528, 2013.

V. Schmid, K. F. Doerner, R. F. Hartl, M. W. P. Savelsbergh, and W. Stoecher, "A hybrid solution approach for ready-mixed concrete delivery," *Transportation Science*, vol. 43, no. 1, pp. 70–85, Feb. 2009.

V. Schmid, K. F. Doerner, R. F. Hartl, and J.-J. Salazar-González, "Hybridization of very large neighborhood search for ready-mixed concrete delivery problems," *Computers and Operations Research*, vol. 37, no. 3, pp. 559 – 574, 2010.

C. Silva, J. M. Faria, P. Abrantes, J. M. C. Sousa, M. Surico, and D. Naso, "Concrete Delivery using a combination of GA and ACO," in *Decision and*

Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on, 2005, pp. 7633–7638.

T. T. Tran and J. C. Beck, "Logic-based Benders decomposition for alternative resource scheduling with sequence dependent setups." in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 242. IOS Press, 2012, pp. 774–779.

F. Vanderbeck, "Decomposition and column generation for integer programs," Ph.D. dissertation, Louvain-La-Neuve, September 1994.

J. Verstichel, J. Kinable, G. Vanden Berghe, and P. De Causmaecker, "A combinatorial Benders decomposition for the lock scheduling problem," KU Leuven, Tech. Rep., September 2013, http://allserv.kahosl.be/~jannes/lockscheduling/combinatorialBenders_19112013.pdf.

L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988, pp. 323 – 337.

L. A. Wolsey, *Integer programming*. Wiley-Interscience, 1998.

S. Yan and W. Lai, "An optimal scheduling model for ready mixed concrete supply with overtime considerations," *Automation in Construction*, vol. 16, no. 6, pp. 734 – 744, 2007.

Z. Zeitlin, "Minimization of maximum absolute deviation in integers," *Discrete Applied Mathematics*, vol. 3, no. 3, pp. 203 – 220, 1981.

List of Publications

IT (Articles in internationally reviewed academic journals)

Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden Berghe, G., Verstichel, J. (2015). The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem: The MISTA 2013 challenge. *Journal of Scheduling*.

Verstichel, J., Kinable, J., De Causmaecker, P., Vanden Berghe, G. (2015). A Combinatorial Benders' decomposition for the lock scheduling problem. *Computers & Operations Research*, 54, 117-128. (most recent IF: 1.72).

Kinable, J., Wauters, T., Vanden Berghe, G. (2014). The Concrete Delivery Problem. *Computers & Operations Research*, 48, 53-68. (most recent IF: 1.72).

Kinable, J., Spieksma, F., Vanden Berghe, G. (2014). School bus Routing - A Column Generation Approach. *International Transactions in Operational Research*, 21 (3), 453-478. (most recent IF: 0.48).

Schittekat, P., Kinable, J., Sörensen, K., Sevaux, M., Spieksma, F., Springael, J. (2013). A metaheuristic for the school bus routing problem with bus stop selection. *European Journal of Operational Research*, 229 (2), 518-528. (citations: 3) (IF publication year: 1.84) (most recent IF: 1.84).

Kinable, J., Orestis, K. (2011). Malware classification based on call graph clustering. *Journal in Computer Virology*, 7 (4), 233-245.

IC (Papers at international scientific conferences and symposia, published in full in proceedings)

Kinable, J., Trick, M. (2014). A Logic Based Benders' Approach to the Concrete Delivery Problem. In Lecture Notes in Computer Science: Vol. 8451. *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming.* Cork, 19-23 May 2014 (pp. 176-192) Springer.

IMa (Meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals)

Kinable, J., Cire, A., Van Hoeve, W-J. (2014). Integrating CP, MIP and Decision Diagrams for the Time-Dependent TSP. *INFORMS 2014.* San Francisco, 9-12 November 2014.

Kinable, J., Wauters, T. (2013). Crux 25 - Circle Packing. *INFORMS 2013.* Minneapolis, 6-9 October 2013.

Kinable, J., Wauters, T. (2013). Crux 25 - Circle Packing. *International Conference on Mathematics in Sport.* Leuven, Belgium, 5-7 June 2013.

Kinable, J., Spieksma, F., Vanden Berghe, G. (2012). School Bus Routing - A Column Generation Approach. *INFORMS 2012.* Beijing, 23-27 Juni 2012.

Schittekat, P., Kinable, J., Sörensen, k., Sevaux, M., Spieksma, F., Springael, J. (2012). An efficient metaheuristic for the school bus routing problem. *Recherche Opérationnelle et d'Aide à la Décision (ROADEF).* France, 11-13 April 2012.

Orestis, K., Kinable, J., Mahmoudi, H., Mustonen, K. (2011). Improved call graph comparison using simulated annealing. SAC '11 Proceedings of the 2011 ACM Symposium on Applied Computing. *ACM Symposium on Applied Computing.* Taiwan, 21-24 May 2011, 1516-1523.

IMa-p (Meeting abstracts, presented at international professionally oriented conferences and symposia, published or not published in proceedings or journals)

Wauters, T., Kinable, J., Vanden Berghe, G. (2013). Using Advanced Algorithms to Optimize Concrete Delivery. *iMinds The Conference*. Brussels, Belgium, 5 December 2013.

AMa (Meeting abstracts, presented at other scientific conferences and symposia, published or not published in proceedings or journals)

Kinable, J., Spieksma, F., Vanden Berghe, G. (2012). School Bus Routing - A column generation approach. *ORBEL 2012*. Belgium, Ghent, 2-3 February 2012.

Kinable, J. (2009). Trust Management Framework for Multi-level Clustered Wireless Sensor Networks. Current Internet Trends - Seminar on Internetworking. Current Internet Trends - *Seminar on Internetworking*. Finland, Espoo, 20 April 2009.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
COMBINATORIAL OPTIMISATION AND DECISION SUPPORT (CODES)
Celestijnenlaan 200A box 2402
B-3001 Heverlee
joris.kinable@kuleuven.be