Hybrid Optimization of Vehicle Routing Problems

Edward Lam

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

October 2018

School of Computing and Information Systems
University of Melbourne

Abstract

Vehicle routing problems are combinatorial optimization problems that aspire to design vehicle routes that minimize some measure of cost, such as the total distance traveled or the time at which the last vehicle returns to a depot, while adhering to various restrictions. Vehicle routing problems are of profound interest in both academia and industry because they are opportunities to study graph structures and algorithms, and because they underpin practical applications in a multitude of industries, but notably, the transportation and logistics industries. This dissertation presents two applications relevant to industry and develops a fully hybrid method for solving a classical vehicle routing problem.

The first application combines vehicle routing with crew scheduling. In industry, vehicle routing and crew scheduling are usually performed in stages due to the large search space of integrated models. The first stage finds minimal-cost vehicle routes with little consideration to crew constraints and objectives. The second stage schedules crews on the routes from the first stage. Disregarding crew constraints in the first stage can lead to suboptimality or even infeasibility of the overall problem in the second stage. To quantify the suboptimality of staged optimization models, two formulations of the integrated problem are developed. The first is an ordinary mixed integer programming model, and the second is a constraint programming model containing a linear relaxation global constraint that performs cost-based filtering. The two integrated models are solved using a branch-and-bound search and a highly specialized large neighborhood search. The large neighborhood search exploits the substructures linking the vehicle routing and crew scheduling elements of the problem, and when executed on the constraint programming model, is shown to perform significantly better than the other approaches.

The second application introduces a number of scheduling constraints to the Vehicle Routing Problem with Pickup and Delivery and Time Windows. The scheduling constraints arise from a lack of loading bays or equipment that unloads and loads incoming vehicles. These constraints limit the number of vehicles present or in service at any particular site by requiring the arrival of vehicles to be scheduled around the availabilities of a scarce resource. A mixed integer programming model, a constraint programming model and a sequential model are implemented for the problem but are shown to be inferior to a branch-and-price-and-check model, which hybridizes column generation and constraint programming with nogood learning.

This thesis concludes with a hybrid method, named Branch-and-Check with Explanations, that unifies linear programming, constraint programming and Boolean satisfiability. The method

begins with a linear programming model that omits several critical constraints. The solver uses the linear programming model to find objective bounds and candidate solutions, which are checked by a constraint programming model for feasibility of the omitted constraints. A Boolean satisfiability model performs conflict analysis on infeasible candidate solutions to derive nogood cuts, which are placed into the linear programming model and the constraint programming model. The method is implemented in a proof-of-concept solver for the Vehicle Routing Problem with Time Windows and is shown to be competitive against a branch-and-cut model while avoiding the intricacies involved in developing the cutting planes and separation algorithms required in branch-and-cut.

Declaration

I, Edward Lam, certify that:

- this thesis comprises only my original work,
- due acknowledgement has been made in the text to all other material used, and
- the thesis is fewer than 100,000 words in length, exclusive of tables, bibliographies, footnotes and appendices.

Signed: _____

Date: _____

Preface

The work contained in this thesis is conducted under the supervision of Pascal Van Hentenryck between December 2013 and October 2018. The contributions of this thesis can be found in the third, fourth and fifth chapters. The main findings of these chapters are respectively published in the following papers:

Lam, E., P. Van Hentenryck and P. Kilby (2015). 'Joint Vehicle and Crew Routing and Scheduling'. In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31 – September 4, 2015, Proceedings*. Ed. by G. Pesant. Springer, Cham, pp. 654–670.

Lam, E. and P. Van Hentenryck (2016). 'A branch-and-price-and-check model for the vehicle routing problem with location congestion'. In: *Constraints* 21.3, pp. 394–412.

Lam, E. and P. Van Hentenryck (2017). 'Branch-and-Check with Explanations for the Vehicle Routing Problem with Time Windows'. In: *Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*. Ed. by J. C. Beck. Springer, Cham, pp. 579–595.

Acknowledgements

First and foremost, I would like to thank my supervisor Pascal Van Hentenryck. I am grateful for his expertise and his encouragement, especially during the times when I didn't believe my work will be successful. I appreciate him donating many evenings to argue with me when I didn't accept his opinion, which turned out correct more times than I want to admit. I could not have completed this thesis nor tasted unforgettable Belgian fries without Pascal.

I wish to thank Phil Kilby and Andreas Schutt for the advice and discussions in Pascal's absence. I am also appreciative of Peter Stuckey, Sandra Stasi, Simon Dunstall and Tim Miller for administrating my candidature. Almost every time I talk to them, it was to beg for travel funding or to sign some paperwork, which is always urgent in my view.

I am indebted to my family for supporting me throughout this endeavor as well as all aspects of life. They all have contributed to my success however minor. I owe my uncle for teaching me algorithms and coding during my childhood, to whom I repaid by erasing his entire hard drive repeatedly over the years. I wish to thank my partner, a food critic, who has prevented me from starving during my candidature, even though sometimes I would rather starve than visit yet another restaurant. Finally, I have to thank my mother in particular for encouraging me to strive for excellence in all of my studies, from my childhood until today, despite not having the opportunity herself as a war refugee.

The work in this thesis is fully funded by CSIRO's Data61, formerly NICTA.

Contents

List of Figures

List of Tables

List of Algorithms

To Lillian, Ethan and Winston

Chapter 1

Introduction

A vehicle routing problem aims to design minimal-cost routes for a fleet of vehicles that visit a number of locations to perform tasks while adhering to various restrictions. The original problem was formulated by Dantzig and Ramser (1959) but has since expanded into a large family of related problems (e.g., Lahyani, Khemakhem and Semet 2015). Some variants of the problem family are academic curiosities but others are highly relevant to many industries, notably, the transportation and logistics industries (e.g., Golden, Assad and Wasil 2001).

Finding efficient routes has historically been performed by human planners, but recently, algorithms have mostly supplanted human planning (e.g., Irnich, Toth and Vigo 2014). The past five decades of research into vehicle routing problems have resulted in intricate algorithms that can find routes with significantly lower cost than routes found by human planners. These algorithms are grounded in the study of combinatorial optimization. Combinatorial optimization is an interdisciplinary subfield of mathematics and computer science. The field is backed by mathematical arguments but appeals to practitioners in a wide range of industries since many business problems can be modeled as combinatorial optimization problems and then solved rigorously using scientific methods. Solutions to these optimization problems can be used to advise decision makers, which may result in significant monetary or time savings or other efficiencies.

Combinatorial optimization problems are usually formulated as mathematical models. These models generally consist of

- an objective function that measures the quality of a solution,
- discrete- or integer-valued variables that represent decisions or indivisible quantities,
- continuous real-valued variables that represent divisible quantities, and
- constraints that express relationships between the variables.

Solving a model involves finding values for the variables that minimize or maximize the objective function without violating any constraint.

Sections 1.1 and 1.2 briefly summarize mixed integer programming and constraint programming, which are two technologies that can be used to solve vehicle routing problems and general combinatorial optimization problems. In particular, these two sections describe the strengths and weaknesses of the two technologies. Section 1.3 describes hybridizations as a

path to mitigating weaknesses in the individual techniques. Section 1.4 outlines the structure of this thesis.

This thesis argues that hybrid models of vehicle routing problems that exploit the individual strengths of mixed integer programming and constraint programming can solve problems that are far too difficult for either technology alone. This thesis experimentally compares pure mixed integer programming and pure constraint programming models of vehicle routing problems to hybrid models.

1.1 Mixed Integer Programming Models

Vehicle routing problems are commonly formulated as mixed integer programming models (e.g., Vigo and Toth 2014). These models admit integer- and real-valued variables but restrict the objective function and the constraints to be linear functions of the variables.

Mixed integer programming models are frequently solved using a branch-and-bound tree-search algorithm (e.g., Kianfar 2010). At every node of the tree, branch-and-bound solves a subproblem, known as the linear relaxation, that disregards the requirement that integer-valued variables take on integer values. If the constraints are violated in the linear relaxation solution, the search algorithm explores another node. If any integer-valued variable has a fractional value in the linear relaxation solution, the search algorithm creates two or more nodes that forbid the current linear relaxation solution. Otherwise, the linear relaxation solution is a valid solution to the original mixed integer programming problem. If the value of the objective function, called the objective value, is better than that of the incumbent solution, then the new solution is stored as the incumbent solution.

At every node of the search tree, the objective value of the linear relaxation solution is a bound on the objective value of the entire subtree. If the objective bound is worse than the objective value of the incumbent solution, the subtree can be pruned. Hence, strong bounds enable large subtrees to be pruned, resulting in less work required to exhaustively explore the search tree.

General combinatorial substructure, such as cycles and bijections, must be implemented using multiple constraints since mixed integer programming only admits linear constraints. Linearizing certain constraints, such as cycle constraints and logical constraints, is not only cumbersome but also severely degrades the performance of search since it weakens the objective bounds (e.g., Hooker 2010).

The linear relaxations can be solved using the simplex algorithm (e.g., Cochran 2010), which performs elementary row operations on a matrix representing the constraints. Since row operations can combine multiple constraints, the constraints are said to communicate. Furthermore, since the objective function is essentially another row in the matrix, the constraints can also communicate with the objective function and vice versa. For some models, the communication between the objective function and the constraints gives rise to asymptotically tight objective bounds (Bramel and Simchi-Levi 1997), which allow these models to completely dominate many other approaches (e.g., Pecin et al. 2014, Røpke 2012).

1.2 Constraint Programming Models

Vehicle routing problems can also be formulated as constraint programming models. Constraint programming models are more general than mixed integer programming models and allow general constraints and variables.

Constraint programming models are commonly solved using a tree-search algorithm similar to branch-and-bound (e.g., Michel and Van Hentenryck 2010). Constraint programming lacks the linear relaxation of mixed integer programming and does not contain an all-encompassing data structure, like the linear relaxation matrix, that links the variables and constraints. Instead, every variable is initialized at the root of the search tree with a set of possible values, known as its domain, which is continually reduced by the constraints as the search progresses deeper into the tree. The search continues until either the domain of every variable contains exactly one value or until the domain of at least one variable is empty. In the first case, the values represent a solution, and in the second case, the empty domain represents a violation of a constraint.

Constraints are implemented using an algorithm known as a propagator (e.g., Bessiere 2010). Propagators are a key component of constraint programming solvers, which repeatedly call propagators to reduce the domains of variables. Many propagators can exist for a particular constraint, and the choice of propagator is selected based on a trade-off that balances speed and strength, i.e., the number of values removed from the domains.

The constraints in constraint programming models are independent and only communicate by removing values from the domains. Hence, the objective bounds are highly dependent on strong propagators for constraints that involve variables in the objective function. Propagators for linear functions are known to be weak, and unfortunately, linear functions appear frequently as objective functions in vehicle routing problems (e.g., Vigo and Toth 2014). Because of this, vehicle routing problems are rarely formulated as constraint programming models in the literature.

Unlike constraints in mixed integer programming, constraints in constraint programming can be general. For example, constraints can be logical relations or incorporate entire combinatorial substructures. Propagators for combinatorial substructures typically implement dedicated logic, making them much more effective than linearizations in mixed integer programming models. Furthermore, bespoke propagators can be built to combine multiple constraints into one. These propagators should be considered when separate propagators are able to deduce information that can significantly reduce the domains when combined but are independently useless.

1.3 Hybrid Models

For linear objective functions, the linear relaxation of mixed integer programming provides objective bounds that are tighter than those of constraint programming models. Conversely, constraint programming models directly implement logical relations and many combinatorial substructures, and with strong propagators, constraint programming models are competitive against many mixed integer programming models (e.g., Benchimol et al. 2012). However, these

stereotypes are not guaranteed: there is no definitive feature that makes a problem better suited to mixed integer programming or constraint programming.

One approach to mitigating inherent weaknesses of mixed integer programming and constraint programming is to hybridize the two technologies. Hybridization has become a highly active area of study, especially during the past several years, because the weaknesses of both mixed integer programming and constraint programming are not yet alleviated after decades of ongoing development (Hooker and van Hoeve 2018).

Vehicle routing problems are historically solved using mixed integer programming because their objective functions are usually linear, making constraint programming ineffective. However, many rich vehicle routing problems contain difficult real-world constraints that are better modeled using logical relations from constraint programming.

1.4 Preview of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2 formalizes mixed integer programming and constraint programming, presents existing hybridization techniques and introduces several basic vehicle routing problems. Readers familiar with this material are encouraged to proceed to the main contributions of this thesis, which are found in Chapters 3 to 5.

Chapter 3 considers an application that integrates vehicle routing and crew scheduling, which are usually solved in separate stages. Staged models are compared to integrated models based on mixed integer programming and constraint programming. The constraint programming model wraps a linear relaxation inside a global constraint to calculate objective bounds. Results demonstrate that the constraint programming model with the bounding constraint coupled with a bespoke search algorithm outperforms the other models.

Chapter 4 presents a vehicle routing problem that requires vehicles to be scheduled around the availability of scarce parking bays and loading equipment. Constraint programming excels at scheduling but has considerable difficulty in optimizing linear objective functions commonly seen in vehicle routing problems. The opposite is often true for mixed integer programming. An elaborate hybrid model is developed and compared to naive mixed integer programming and constraint programming models. Results indicate that the hybrid model outperforms both the mixed integer programming and constraint programming models.

Chapter 5 develops a method that unifies mixed integer programming and constraint programming. The method is implemented for a classical vehicle routing problem known as the Vehicle Routing Problem with Time Windows. The hybrid model is compared to mixed integer programming and constraint programming models as well as a mixed integer programming model that implements an advanced technique known as branch-and-cut. Experimental results show that the hybrid model outperforms the other approaches while avoiding the intricacies of developing specialized algorithms necessary for branch-and-cut.

Chapter 6 concludes this thesis with a summary of the main findings and recommendations for future research.

Chapter 2

Background

This chapter reviews background material that underpins the main contributions of this thesis. Section 2.1 introduces constrained optimization problems, which are specialized into a few important classes in Sections 2.2 to 2.5. Section 2.2 presents linear programming, which is generalized to mixed integer programming in Section 2.3. Section 2.4 considers Boolean satisfiability. Section 2.5 generalizes Boolean satisfiability to constraint programming. Section 2.6 presents several techniques for hybridizing mixed integer programming and constraint programming. Section 2.7 introduces a few basic vehicle routing problems, and Section 2.8 summarizes several influential models of vehicle routing problems.

Where relevant, the vector operators $=$, $\leq$ and $\geq$ perform component-wise comparison. The symbols $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$, $\mathbb{Z}_+$ and $\mathbb{B}$ respectively denote the real numbers, the non-negative real numbers, the integers, the non-negative integers and the Boolean domain, i.e., the set {false, true} or equivalently $\{0, 1\}$ by an abuse of notation.

2.1 Constrained Optimization Problems

Constrained optimization problems are mathematical problems tasked with finding values to variables such that the values obey various restrictions, called *constraints*, and that a function, called the *objective function*, attains either a minimum or maximum at the values. The following definitions formalize constrained optimization problems.

Definition 2.1 (Constrained Minimization Problem)**.** A *constrained minimization problem* $\mathcal{P}$ asks to find a vector

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in S} f(\mathbf{x}),$$

where $S = c_1 \cap \ldots \cap c_m \subseteq \mathbb{R}^n$, called the *feasible space*, is the intersection of *constraints* $c_1, \ldots, c_m \subseteq \mathbb{R}^n$, $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $x_1, \ldots, x_n$ are *variables*, and $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*. Every $\mathbf{x} \in S$ is a *feasible solution* with *objective value* $f(\mathbf{x})$. The vector $\mathbf{x}^*$ is called an *optimal solution*, and the value $f(\mathbf{x}^*)$ is called the *optimal value*. If $f(\mathbf{x}_k) \to -\infty$ for a sequence $k = 1, 2, \ldots$, the problem is said to be *unbounded*. If $S = \varnothing$, the problem is said to be *infeasible*.

Definition 2.2 (Constrained Maximization Problem)**.** A *constrained maximization problem* with objective function $f(\mathbf{x})$ is a constrained minimization problem with objective function $-f(\mathbf{x})$.

Maximization problems can be expressed as minimization problems and vice versa since maximizing $f(\mathbf{x})$ is equivalent to minimizing $-f(\mathbf{x})$ and recovering the value $f(\mathbf{x}^*)$ by negating $-f(\mathbf{x}^*)$. The remainder of this thesis only considers minimization problems unless specified otherwise.

Definition 2.3 (Constrained Optimization Problem)**.** A *constrained optimization problem* is either a constrained minimization problem or a constrained maximization problem.

Constrained optimization problems seldom have closed-form solutions. Instead, iterative algorithms are used to find numerical solutions. These algorithms are implemented in a software package called a *solver*. Solvers are only applicable to certain classes of problems. These classes are categorized according to the form of their objective functions, variables and constraints. Each of the next four sections surveys a class of constrained optimization problems and its solution algorithms. Rather than solving a problem directly, some of these algorithms exploit a relaxation of the problem, defined as follows.

Definition 2.4 (Relaxation)**.** Let $\mathcal{P}$ be a constrained minimization problem with objective function $f_{\mathcal{P}}$, feasible space $S_{\mathcal{P}}$ and variables $\mathbf{x} \in S_{\mathcal{P}}$. A *relaxation* $\mathcal{R}$ of $\mathcal{P}$ is a constrained minimization problem with objective function $f_{\mathcal{R}}$, feasible space $S_{\mathcal{R}} \supseteq S_{\mathcal{P}}$ and variables $\mathbf{y} \in S_{\mathcal{R}}$ such that $f_{\mathcal{R}}(\mathbf{y}) \leq f_{\mathcal{P}}(\text{proj}_{S_{\mathcal{P}}}(\mathbf{y}))$ for all $\mathbf{y} \in S_{\mathcal{R}}$.

2.2 Linear Programming

Linear programming (LP) seeks to solve a class of constrained optimization problems known as linear programs (also abbreviated as LP).

Variables in linear programs can only take non-negative real-values, but linear programs can model general real-valued variables because any real-valued variable $x \in \mathbb{R}$ can be expressed as $x = x_+ - x_-$ with $x_+, x_- \in \mathbb{R}_+$.

The objective function is linear and the constraints are linear inequalities. However, constraints can include linear equations since they can be expressed as inequalities. A linear equation $\alpha_1 + \ldots + \alpha_n = \beta$ can be written as the two inequalities $\alpha_1 + \ldots + \alpha_n \geq \beta$ and $\alpha_1 + \ldots + \alpha_n \leq \beta$. A linear inequality $\alpha_1 + \ldots + \alpha_n \geq \beta$ can also be written as $\alpha_1 + \ldots + \alpha_n - s = \beta$, where $s \geq 0$ is a variable, called a *surplus variable*, that absorbs the difference. Similarly, $\alpha_1 + \ldots + \alpha_n \leq \beta$ can be stated as $\alpha_1 + \ldots + \alpha_n + t = \beta$, where $t \geq 0$ is a *slack variable*. Furthermore, a linear inequality $\alpha_1 + \ldots + \alpha_n \geq \beta$ can be written as $-\alpha_1 - \ldots - \alpha_n \leq -\beta$ and vice versa. Therefore, linear programs can model any linear equation or inequality constraint. Even though linear programs cannot directly model strict inequality (i.e., $<$ or $>$) nor disequality (i.e., $\neq$), they can be extended to model these constraints in certain cases (Van Hentenryck and Graf 1992).

The following definitions summarize well-known results of linear programming. Readers seeking further details are recommended to consult the excellent textbook by Rader (2010).

Other sources for introductory material include the textbooks by Nemhauser and Wolsey (1999), Vanderbei (2014) and Winston (2004).

Definition 2.5 (Linear Program)**.** A (minimization) *linear program* $\mathcal{P}$ is a tuple $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ such that $A\mathbf{x} \geq \mathbf{b}$, where $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n_+$ is a vector of independent variables, $z = \mathbf{c} \cdot \mathbf{x} \in \mathbb{R}$ is a dependent variable, $A \in \mathbb{R}^{m \times n}$ is a constant matrix, and $\mathbf{b} = (b_1, \dots, b_m) \in \mathbb{R}^m$ and $\mathbf{c} = (c_1, \dots, c_n) \in \mathbb{R}^n$ are constant vectors.

Definition 2.6 (Variable)**.** Given a linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$, each component of $\mathbf{x}$, i.e., $x_1, \dots, x_n$, is a *variable* of $\mathcal{P}$.

Definition 2.7 (Constraint)**.** A *constraint* of a linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ represents the set

$$\left\{\mathbf{x} \in \mathbb{R}^n_+ | \sum_{j=1}^{n} A_{i,j} x_j \geq b_i\right\}$$

for any $i = 1, \dots, m$. For brevity, this constraint is usually stated as

$$\sum_{j=1}^{n} A_{i,j} x_j \geq b_i.$$

Definition 2.8 (Objective Function)**.** The dependent variable $z = \mathbf{c} \cdot \mathbf{x}$ is called the *objective function* of the linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$.

Example 2.1. Given independent variables x_1, x_2, $\mathbf{x} = (x_1, x_2)$, dependent variable z, $A = \left[\begin{smallmatrix} 6 & 4 \\ 2 & 3 \end{smallmatrix}\right]$, $\mathbf{b} = (8, 5)$ and $\mathbf{c} = (1, 2)$, then $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ is a linear program. It has constraints

$$6x_1 + 4x_2 \geq 8$$

and

$$2x_1 + 3x_2 \geq 5,$$

and has objective function

$$z = x_1 + 2x_2.$$

Definition 2.9 (Feasible Space)**.** The *feasible space* or *solution space* S of a linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ is the intersection of all its constraints, i.e., $S = \{\mathbf{x} \in \mathbb{R}^n_+ | A\mathbf{x} \geq \mathbf{b}\}$.

The feasible space in many vehicle routing problems, including those relevant to this thesis, are bounded. The discussion below makes this assumption, which simplifies many of the following definitions and results.

Definition 2.10 (Feasible Solution)**.** Given a linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ and its feasible space S, any $\hat{\mathbf{x}} \in S$ is a *feasible solution* or simply *solution* of $\mathcal{P}$. The value $z = \mathbf{c} \cdot \hat{\mathbf{x}}$ is the *objective value* of $\hat{\mathbf{x}}$.

For brevity, this thesis defines a *solution* as a feasible solution. This definition is widely used in the constraint programming literature but is uncommon in the linear programming literature, which usually defines a solution as an optimal solution.

Example 2.2. Consider the linear program in Example 2.1. The vector $\hat{\mathbf{x}} = (1, 1)$ is a feasible solution with objective value $z = \mathbf{c} \cdot \hat{\mathbf{x}} = (1, 2) \cdot (1, 1) = 3$.

Definition 2.11 (Solving a Linear Program)**.** Let $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ be a linear program with bounded feasible space S. *Solving* $\mathcal{P}$ refers to (1) finding a feasible solution

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in S} z,$$

in which case $\mathbf{x}^*$ and $z = \mathbf{c} \cdot \mathbf{x}^*$ are respectively called an *optimal solution* and the *optimal value*, or (2) showing that $S = \varnothing$, in which case $\mathcal{P}$ is described as *infeasible*.

Definition 2.12 (Standard Form)**.** A linear program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ can be written in *standard form* as

$$\begin{array}{lrcccccccll}
\min & z & = & c_1 x_1 & + & \dots & + & c_n x_n & & & \\
\text{subject to} & & & A_{1,1} x_1 & + & \dots & + & A_{1,n} x_n & \geq & b_1, & \\
& & & \vdots & + & \ddots & + & \vdots & \geq & \vdots & \\
& & & A_{m,1} x_1 & + & \dots & + & A_{m,n} x_n & \geq & b_m, & \\
& & & & & & & x_j & \geq & 0 & \forall j = 1, \dots, n.
\end{array}$$

Example 2.3. The linear program in Example 2.1 can be written in standard form as

$$\begin{array}{lrcccccll}
\min & z & = & x_1 & + & 2x_2 & & & \\
\text{subject to} & & & 6x_1 & + & 4x_2 & \geq & 8, & \\
& & & 2x_1 & + & 3x_2 & \geq & 5, & \\
& & & x_1 & & & \geq & 0, & \\
& & & & & x_2 & \geq & 0. &
\end{array}$$

Constraints in standard form are sometimes stated with = or ≤ signs to ease understanding. These constraints are no different to ≥ constraints because of the rules mentioned at the start of this section.

Recall from Section 2.1 that maximization problems can be written as minimization problems and vice versa. Maximization linear programs can be written in standard form using max instead of min. For example, consider the following maximization linear program $\mathcal{P}$ written in standard form:

$$\begin{array}{lrcccccccll}
\max & z & = & c_1 x_1 & + & \dots & + & c_n x_n & & & \\
\text{subject to} & & & A_{1,1} x_1 & + & \dots & + & A_{1,n} x_n & \geq & b_1, & \\
& & & \vdots & + & \ddots & + & \vdots & \geq & \vdots & \\
& & & A_{m,1} x_1 & + & \dots & + & A_{m,n} x_n & \geq & b_m, & \\
& & & & & & & x_j & \geq & 0 & \forall j = 1, \dots, n.
\end{array}$$

It can be presented in the tuple form of Definition 2.5 by replacing z with $-z$ and $\mathbf{c}$ with $-\mathbf{c}$. That is, $\mathcal{P}$ is equivalent to the minimization linear program $\mathcal{P}' = (\mathbf{x}, -z, A, \mathbf{b}, -\mathbf{c})$. This minimization

problem is written in standard form as

$$\begin{array}{lrcrcrcrcll}
\min & -z & = & -c_1x_1 & - & \dots & - & c_nx_n & & & \\
\text{subject to} & & & A_{1,1}x_1 & + & \dots & + & A_{1,n}x_n & \geq & b_1, & \\
& & & \vdots & + & \ddots & + & \vdots & \geq & \vdots & \\
& & & A_{m,1}x_1 & + & \dots & + & A_{m,n}x_n & \geq & b_m, & \\
& & & & & & & x_j & \geq & 0 & \forall j = 1, \dots, n.
\end{array}$$

The value $-z$ is the objective value of solutions in the equivalent minimization problem $\mathcal{P}'$. Objective values in the original maximization problem $\mathcal{P}$ are recovered in z.

The following discussion relies on the maximization rewriting to define duality. A review of duality is needed to discuss advanced techniques later in this chapter.

Definition 2.13 (Dual)**.** Let $\mathcal{P} = (\mathbf{x}, z_\mathcal{P}, A, \mathbf{b}, \mathbf{c})$ be a linear program. The *dual* $\mathcal{D}$ of $\mathcal{P}$ is the linear program $\mathcal{D} = (\mathbf{y}, -z_\mathcal{D}, -A^\top, -\mathbf{c}, -\mathbf{b})$, where $\mathbf{y} = (y_1, \dots, y_m) \in \mathbb{R}^m_+$ and $y_1, \dots, y_m$ are the variables.

The dual is usually considered a maximization problem. It is presented in Definition 2.13 as a minimization problem with objective value $-z_\mathcal{D} = -\mathbf{b} \cdot \mathbf{y}$. As a maximization problem, it has objective value $z_\mathcal{D} = \mathbf{b} \cdot \mathbf{y}$.

Example 2.4. The dual of $\mathcal{P}$ in Examples 2.1 and 2.3 is the linear program

$$\begin{array}{lrcrcrcrl}
\min & -z_\mathcal{D} & = & -8y_1 & - & 5y_2 & & \\
\text{subject to} & & & -6y_1 & - & 2y_2 & \geq & -1, \\
& & & -4y_1 & - & 3y_2 & \geq & -2, \\
& & & y_1 & & & \geq & 0, \\
& & & & & y_2 & \geq & 0,
\end{array}$$

or equivalently,

$$\begin{array}{lrcrcrcrl}
\max & z_\mathcal{D} & = & 8y_1 & + & 5y_2 & & \\
\text{subject to} & & & 6y_1 & + & 2y_2 & \leq & 1, \\
& & & 4y_1 & + & 3y_2 & \leq & 2, \\
& & & y_1 & & & \geq & 0, \\
& & & & & y_2 & \geq & 0.
\end{array}$$

Definition 2.14 (Primal)**.** Let $\mathcal{P}$ be a linear program and $\mathcal{D}$ be its dual. In the context of a dual, $\mathcal{P}$ is called the *primal*.

Proposition 2.1. The dual of the dual of a primal $\mathcal{P}$ is $\mathcal{P}$ itself.

Theorem 2.2 (Weak Duality)**.** Let $\mathcal{P} = (\mathbf{x}, z_\mathcal{P}, A, \mathbf{b}, \mathbf{c})$ be a linear program with dual $\mathcal{D}$, then for every feasible solution $\hat{\mathbf{x}}$ of $\mathcal{P}$ and every feasible solution $\hat{\mathbf{y}}$ of $\mathcal{D}$,

$$\mathbf{c} \cdot \hat{\mathbf{x}} \leq \mathbf{b} \cdot \hat{\mathbf{y}}.$$

Proof. See Theorem 9.1 of Rader (2010). □

Theorem 2.3 (Strong Duality)**.** Let $\mathcal{P} = (\mathbf{x}, z_{\mathcal{P}}, A, \mathbf{b}, \mathbf{c})$ be a linear program and let $\mathcal{D}$ be its dual. The problem $\mathcal{P}$ has an optimal solution $\mathbf{x}^*$ if and only if $\mathcal{D}$ has an optimal solution $\mathbf{y}^*$ such that

$$\mathbf{c} \cdot \mathbf{x}^* = \mathbf{b} \cdot \mathbf{y}^*.$$

Proof. See Theorem 9.2 of Rader (2010). □

Definition 2.15 (Objective Bound)**.** Let $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$ be a linear program with a primal feasible solution $\hat{\mathbf{x}}$. Let $\mathcal{D}$ be the dual of $\mathcal{P}$ and $\hat{\mathbf{y}}$ be a dual feasible solution. Then, the value $\mathbf{c} \cdot \hat{\mathbf{x}}$ is a *lower bound* and $\mathbf{b} \cdot \hat{\mathbf{y}}$ is an *upper bound* to the optimal value $\mathbf{c} \cdot \mathbf{x}^*$ by the Weak Duality Theorem and the Strong Duality Theorem.

2.2.1 Solutions to Linear Programs

The following discussion pertains to an algebraic derivation of the form of the solutions to linear programs. As explained previously, linear inequalities can be written as linear equations and vice versa. Therefore, any linear program can be written as

$$\begin{aligned} \min \quad & z = \mathbf{c}^\top \mathbf{x} && \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}, && (2.1)\\ & \mathbf{x} \geq \mathbf{0}, && (2.2) \end{aligned}$$

where $\mathbf{x}$ includes a surplus variable for every row of A. Without loss of generality, reorder the columns of A and then partition the columns into an invertible submatrix B and a submatrix N, i.e., $A = [B\ N]$. Using the partition, the above linear program can be written as

$$\begin{aligned} \min \quad & z = \mathbf{c}_B^\top \mathbf{x}_B + \mathbf{c}_N^\top \mathbf{x}_N && (2.3)\\ \text{subject to} \quad & B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b}, && (2.4)\\ & \mathbf{x}_B \geq \mathbf{0}, && (2.5)\\ & \mathbf{x}_N \geq \mathbf{0}, && (2.6) \end{aligned}$$

where $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ and $\mathbf{c} = (\mathbf{c}_B, \mathbf{c}_N)$ are also partitioned accordingly. The variables in the components of $\mathbf{x}_B$ and $\mathbf{x}_N$ are respectively called the *basic variables* and the *non-basic variables* because the matrix B, being invertible, is a basis for $\mathbb{R}^m$.

Since B is invertible, Constraint (2.4) can be rearranged to

$$\mathbf{x}_B = B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N.$$

Using this equation, any point $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$, with $\mathbf{x}_N \geq \mathbf{0}$, can be expressed as

$$\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N) = (B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N, \mathbf{x}_N). \tag{2.7}$$

This point can be substituted into Objective Function (2.3), resulting in

$$z = \mathbf{c}_B^\top(B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N) + \mathbf{c}_N^\top \mathbf{x}_N = \mathbf{c}_B^\top B^{-1}\mathbf{b} + (\mathbf{c}_N^\top - \mathbf{c}_B^\top B^{-1}N)\mathbf{x}_N. \tag{2.8}$$

Each component in the row vector $\mathbf{c}_N^\top - \mathbf{c}_B^\top B^{-1}N$ is called the *reduced cost* of the corresponding variable in $\mathbf{x}_N$. By observing the form of this objective function, the reduced cost of a variable can be understood as a bound on the change in the objective value for every unit increase in the variable's value.

Next, set $\mathbf{x}_N = \mathbf{0}$ in Equation (2.7), yielding the *basic solution* $\hat{\mathbf{x}} = (B^{-1}\mathbf{b}, \mathbf{0})$. Assuming that $B^{-1}\mathbf{b} \geq \mathbf{0}$, then $\hat{\mathbf{x}}$ satisfies Constraints (2.4) to (2.6), and hence, is a *basic feasible solution.* Substituting $\hat{\mathbf{x}}$ into Equation (2.8) gives its objective value as

$$\mathbf{c}_B^\top B^{-1}\mathbf{b}.$$

Any point that has a better (lower) objective value than $\mathbf{c}_B^\top B^{-1}\mathbf{b}$ must have $(\mathbf{c}_N^\top - \mathbf{c}_B^\top B^{-1}N)\mathbf{x}_N < 0$ in Equation (2.8). This term is negative if at least one of the non-basic variables have negative reduced cost because $\mathbf{x}_N \geq \mathbf{0}$ by definition. Conversely, it can be shown that the basic feasible solution $\hat{\mathbf{x}}$ is an optimal solution if all reduced costs are non-negative. Any basic feasible solution can be easily checked to determine whether it is an optimal solution using this criterion.

Once a basic feasible solution is proven to be optimal, a corresponding dual solution can be constructed. The Strong Duality Theorem (Theorem 2.3) states that the optimal values of the primal and the dual are equal. Given an optimal basic feasible solution $\mathbf{x}^* = (B^{-1}\mathbf{b}, \mathbf{0})$ with objective value $\mathbf{c}_B^\top B^{-1}\mathbf{b}$, an optimal dual solution $\mathbf{y}^*$ can be constructed by equating its objective value to $\mathbf{c}_B^\top B^{-1}\mathbf{b}$, i.e.,

$$\mathbf{y}^{*\top}\mathbf{b} = \mathbf{c}_B^\top B^{-1}\mathbf{b}.$$

Hence,

$$\mathbf{y}^{*\top} = \mathbf{c}_B^\top B^{-1}$$

is a dual solution called the *complementary dual solution* of the primal solution $\mathbf{x}^*$. Furthermore, the vector of reduced costs can be expressed in terms of $\mathbf{y}^*$ as

$$\mathbf{c}_N^\top - \mathbf{y}^{*\top}N. \tag{2.9}$$

2.2.2 Solution Methods

The set given by Constraints (2.4) to (2.6), i.e., the feasible space, defines a polyhedron. An optimal solution $\mathbf{x}^* = (B^{-1}\mathbf{b}, \mathbf{0})$ corresponds to a vertex of the polyhedron. Almost all modern linear programming algorithms find optimal solutions by either moving on the surface or in the interior of the polyhedron towards an optimal vertex (Bixby 2002).

The simplex algorithm, developed by Dantzig in 1947, is widely used to solve linear programs. The simplex algorithm is a surface algorithm; it crawls along the edges of the feasible space from vertex to vertex until it converges to an optimal solution if one exists (e.g., Prabhu 2010). Even though its worst-case time complexity is exponential (e.g., Prabhu 2010), it performs very well in practice and is implemented in practically every commercial linear programming solver (Bixby 2002).

Khachiyan (1979), Karmarkar (1984) and many others developed interior point methods.

These algorithms move inside the feasible space towards an optimal vertex (e.g., Peng and Salahi 2011). Although these algorithms have polynomial-time complexity (e.g., Peng and Salahi 2011, Terlaky 2010), they do not necessarily dominate the simplex algorithm. Many commercial linear programming solvers also implement an interior point algorithm (e.g., Benson 2011). Whether a problem is best solved using the simplex algorithm or an interior point algorithm is difficult to determine and usually requires experimental comparisons. For a thorough discussion on interior point methods, interested readers can consult the works by Gondzio (2012), Peng and Salahi (2011), Terlaky (2010) and Wright (1997).

Other historical algorithms for solving linear programs are surveyed by Murty (2010). Solution methods for linear programs are not explored any further as they are not relevant to the main contributions of this thesis.

2.3 Mixed Integer Programming

Two prominent techniques in the field of mathematical programming are linear programming and mixed integer programming (MIP). Mixed integer programming generalizes linear programming to allow integer-valued variables. This class of problems is known as mixed integer programs (also abbreviated as MIP). An enormous number of real-world problems can be modeled using mixed integer programs. Integer-valued variables can be used to model indivisible quantities, for example. The variables can also be restricted to binary values (i.e., 0 or 1). Binary-valued variables can be used to indicate whether a choice is selected.

The following definitions formalize mixed integer programs. Many of the definitions are similar or identical to those of linear programming. To make this section self-contained, the definitions are repeated in this section. A thorough introduction to mixed integer programming can be found in the textbooks by Nemhauser and Wolsey (1999), Rader (2010), Vanderbei (2014) and Winston (2004). A recount of the history of mixed integer programming can be found in the book by Jünger et al. (2009).

Definition 2.16 (Mixed Integer Program)**.** A (minimization) *mixed integer program* $\mathcal{P}$ is a tuple $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ such that $A\mathbf{x} \geq \mathbf{b}$, where $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n_+$ is a vector of independent variables, $z = \mathbf{c} \cdot \mathbf{x} \in \mathbb{R}$ is a dependent variable, $A \in \mathbb{R}^{m \times n}$ is a constant matrix, $\mathbf{b} = (b_1, \dots, b_m) \in \mathbb{R}^m$ and $\mathbf{c} = (c_1, \dots, c_n) \in \mathbb{R}^n$ are constant vectors, and $I \subseteq \{1, \dots, n\}$ is a subset of the index set of the components of $\mathbf{x}$.

Definition 2.17 (Variable)**.** Given a mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$, each component of $\mathbf{x}$, i.e., $x_1, \dots, x_n$, is a *variable* of $\mathcal{P}$. Every x_j where $j \in I$ is an *integer variable*, and every x_j such that $j = 1, \dots, n, j \notin I$ is a *continuous variable*.

Definition 2.18 (Constraint)**.** A *constraint* of a mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ represents the set

$$\left\{ \mathbf{x} \in \mathbb{R}^n_+ \middle| \sum_{j=1}^{n} A_{i,j} x_j \geq b_i \right\}$$

for any $i = 1, \dots, m$. For brevity, this constraint is usually stated as

$$\sum_{j=1}^{n} A_{i,j} x_j \geq b_i.$$

Definition 2.19 (Objective Function)**.** The dependent variable $z = \mathbf{c} \cdot \mathbf{x}$ is called the *objective function* of the mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$

Example 2.5. Given independent variables x_1, x_2, $\mathbf{x} = (x_1, x_2)$, dependent variable z, $A = \left[\begin{smallmatrix} 6 & 4 \\ 2 & 3 \end{smallmatrix}\right]$, $\mathbf{b} = (8, 5)$, $\mathbf{c} = (1, 2)$ and $I = \{1, 2\}$, then $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ is a mixed integer program. Both variables x_1 and x_2 are integer variables.

Definition 2.20 (Linear Relaxation)**.** The *linear relaxation* or *linear programming relaxation* of a mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ is the linear program $\mathcal{P}_{\text{LP}} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c})$.

Example 2.6. The linear relaxation $\mathcal{P}_{\text{LP}}$ of the problem $\mathcal{P}$ in Example 2.5 is the linear program in Example 2.1.

Definition 2.21 (Feasible Space)**.** Let $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ be a mixed integer program and let S_{LP} be the feasible space of its linear relaxation. The *feasible space* or *solution space* S of $\mathcal{P}$ is the set $S = \{\mathbf{x} \in S_{\text{LP}} | x_j \in \mathbb{Z} \text{ for all } j \in I\} = \{\mathbf{x} \in \mathbb{R}^n_+ | A\mathbf{x} \geq \mathbf{b}, x_j \in \mathbb{Z} \text{ for all } j \in I\}$.

Like in the previous section, the feasible space is assumed to be bounded. This assumption is valid for many models of applications, and in particular, all concerned vehicle routing problems.

Definition 2.22 (Feasible Solution)**.** Given a mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ and its feasible space S, any $\hat{\mathbf{x}} \in S$ is a *feasible solution* or simply *solution* of $\mathcal{P}$. The value $z = \mathbf{c} \cdot \hat{\mathbf{x}}$ is the *objective value* of $\hat{\mathbf{x}}$.

Example 2.7. Recall that the linear relaxation $\mathcal{P}_{\text{LP}}$ of the mixed integer program $\mathcal{P}$ in Example 2.5 is the linear program in Example 2.1. Example 2.2 shows that the vector $\hat{\mathbf{x}} = (1, 1)$ is a feasible solution to the linear relaxation. Since x_1 and x_2 are integer variables and $\hat{x}_1 = 1$ and $\hat{x}_2 = 1$ are integral, $\hat{\mathbf{x}}$ is also a feasible solution to $\mathcal{P}$.

Definition 2.23 (Solving a Mixed Integer Program)**.** Let $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ be a mixed integer program with bounded feasible space S. *Solving* $\mathcal{P}$ refers to (1) finding a feasible solution

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in S} z,$$

in which case $\mathbf{x}^*$ and $z = \mathbf{c} \cdot \mathbf{x}^*$ are respectively called an *optimal solution* and the *optimal value*, or (2) showing that $S = \varnothing$, in which case $\mathcal{P}$ is described as *infeasible*.

Definition 2.24 (Standard Form)**.** Every mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$ can be

written in *standard form* as

$$
\begin{array}{lrcccccccll}
\min & z & = & c_1x_1 & + & \dots & + & c_nx_n & & & \\
\text{subject to} & & & A_{1,1}x_1 & + & \dots & + & A_{1,n}x_n & \geq & b_1, & \\
& & & \vdots & + & \ddots & + & \vdots & \geq & \vdots & \\
& & & A_{m,1}x_1 & + & \dots & + & A_{m,n}x_n & \geq & b_m, & \\
& & & & & & & x_j & \geq & 0 & \forall j = 1, \dots, n, \\
& & & & & & & x_j & & \text{integer} & \forall j \in I.
\end{array}
$$

The last line is known as the *integrality constraints*.

Example 2.8. The mixed integer program from Example 2.5 can be written in standard form as

$$
\begin{array}{lrcccccll}
\min & z & = & x_1 & + & 2x_2 & & & \\
\text{subject to} & & & 6x_1 & + & 4x_2 & \geq & 8, & \\
& & & 2x_1 & + & 3x_2 & \geq & 5, & \\
& & & x_1 & & & \geq & 0, & \\
& & & & & x_2 & \geq & 0, & \\
& & & x_1, & & x_2 & & \text{integer.} &
\end{array}
$$

2.3.1 Branch-and-Bound

Mixed integer programs can be solved using a variety of methods. Even though they are NP-hard in general, some classes of mixed integer programs can be solved in polynomial time (e.g., Nemhauser and Wolsey 1999, Rader 2010). These classes of problems have a totally unimodular matrix; hence, their linear-relaxation optimal solutions are also optimal for the mixed integer programs. The Assignment Problem is one such example. Even though it can be solved using polynomial-time algorithms that are specifically tailored to the problem (e.g., Kuhn 1955), it can also be modeled using a mixed integer program and solved in polynomial time using a general-purpose linear programming solver.

General mixed integer programs lack dedicated algorithms, and instead, are commonly solved using an algorithm based on branch-and-bound (e.g., Kianfar 2010, Lodi 2010). Branch-and-bound, developed by Land and Doig (1960), is a general-purpose tree-search framework for solving optimization problems with discrete-valued variables. The framework is based on the divide-and-conquer concept.

Branch-and-bound initializes the root node of the search tree with the linear relaxation of the mixed integer program and then solves the linear relaxation. Recall that integer variables can take on fractional values in the linear relaxation. If any integer variable has a fractional value, then branch-and-bound removes the current solution by branching. For example, if the integer variable x has a value of 6.5 in the linear relaxation solution, branch-and-bound can create two children nodes with the addition of the constraint $x \leq 6$ in one node and $x \geq 7$ in the other. Solving and branching recur until the algorithm reaches either an integer node (a node whose linear relaxation solution satisfies the integrality constraints) or a suboptimal node (a node whose objective bound is worse than the objective value of the incumbent solution). It

discards the node and then backtracks to solve other nodes.

The basic algorithm for solving mixed integer programs is presented in Algorithm 2.1. Lines 1 and 2 initialize the optimal value z^* as infinity, representing no incumbent solution, and the set O of open nodes with the input problem $\mathcal{P}$. Line 3 is the main loop.

Lines 4 and 5 extract a node, which is represented by its internal mixed integer program $\mathcal{S}$. The `SelectNode` function uses various criteria to choose a node. Two common heuristics are *depth-first search* and *best bound-first search* (e.g., Kianfar 2010, Lodi 2010). Depth-first search selects the deepest node and intends to find feasible solutions quickly by enforcing decisions via branching. Best bound-first search selects the node with the smallest objective bound and strives to find high-quality solutions quickly but may suffer if the problem is difficult in terms of feasibility.

Line 6 solves the linear relaxation $\mathcal{S}_{\text{LP}}$ of $\mathcal{S}$ and returns an optimal solution $\mathbf{x}^*_{\text{LP}}$ and the optimal value z^*_{LP}. If $\mathcal{S}_{\text{LP}}$ is infeasible, then $z^*_{\text{LP}} = \infty$. Line 7 discards the current node unless $z^*_{\text{LP}} < z^*$, i.e., the subtree can contain a solution with an objective value better than the current z^*.

If all integer variables take on integer values in $\mathbf{x}^*_{\text{LP}}$ (Line 8), then the linear relaxation solution $\mathbf{x}^*_{\text{LP}}$ is valid for the original mixed integer program $\mathcal{P}$. Line 9 stores $\mathbf{x}^*_{\text{LP}}$ as the new incumbent solution $\mathbf{x}^*$. Line 10 stores the objective value z^*_{LP} of $\mathbf{x}^*_{\text{LP}}$.

If some integer variables have fractional values, then the linear relaxation solution is not a valid solution to the original mixed integer program. The algorithm splits $\mathcal{S}$ into subproblems that do not contain the current solution (Line 12).

A basic implementation of the `CreateChildren` function applies the most fractional branching rule, which selects a variable x_j where $j \in \arg\min_{j \in I} |(x^*_{\text{LP}})_j - \lfloor (x^*_{\text{LP}})_j \rfloor|$ and creates one child node with the constraint $x_j \leq \lfloor (x^*_{\text{LP}})_j \rfloor$ and another child node with $x_j \geq \lceil (x^*_{\text{LP}})_j \rceil$. A comprehensive evaluation of various branching rules is conducted by Achterberg, Koch and Martin (2005).

Solving and branching continues until the tree is fully explored. If a feasible solution is found (Line 13), the algorithm returns the optimal solution $\mathbf{x}^*$ (Line 14). Otherwise, the problem is infeasible and the algorithm terminates (Line 16).

Example 2.9. Figure 2.1 shows a branch-and-bound search tree for the mixed integer program $\mathcal{P}$ in Example 2.8. At the root node (node 1), it solves the linear relaxation $\mathcal{P}_{\text{LP}}$, which results in the linearly-feasible solution $(2.5, 0)$. Despite being feasible for the linear relaxation, this solution is not feasible for $\mathcal{P}$ because x_1 is an integer variable. Branch-and-bound creates two children nodes and imposes the constraint $x_1 \leq 2$ in one node (node 2) and $x_1 \geq 3$ in the other (node 3). Next, it chooses to solve node 2. The linear relaxation $\mathcal{P}_{\text{LP}}$ is augmented with the constraint $x_1 \leq 2$ and then solved, which results in the solution $(2, 0.33)$. This vector is not a feasible solution to $\mathcal{P}$ since x_2 is an integer variable. The algorithm branches again and creates a child node with the constraint $x_2 \leq 0$ (node 4) and another with $x_2 \geq 1$ (node 5). Node 4 is found to be infeasible. Branch-and-bound then proceeds to node 5. It solves the linear relaxation, which results in $\mathbf{x}^*_{\text{LP}} = (1, 1)$. Since $(x^*_{\text{LP}})_1 = 1$ and $(x^*_{\text{LP}})_2 = 1$ are integral, $\mathbf{x}^*_{\text{LP}}$ is a feasible solution to $\mathcal{P}$. The solution $\mathbf{x}^*_{\text{LP}}$, with objective value $z^*_{\text{LP}} = 3$, is stored as the incumbent solution

Algorithm 2.1: The basic branch-and-bound algorithm for mixed integer programs.

Input: A mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$
Output: An optimal solution $\mathbf{x}^*$ to $\mathcal{P}$ if one exists

```
1  z* ← ∞                                          /* initialize optimal value */
2  O ← {P}                                         /* initialize set of open nodes */
3  while O ≠ ∅                                     /* loop until all nodes are solved */
4      S ← SelectNode(O)                           /* get subproblem of a node */
5      O ← O \ {S}                                 /* delete selected node */
6      (x*_LP, z*_LP) ← Solve(S_LP)                /* solve linear relaxation */
7      if z*_LP < z* then                          /* if not suboptimal */
8          if (x*_LP)_j ∈ Z for all j ∈ I then
9              x* ← x*_LP                          /* if integer, store new solution */
10             z* ← z*_LP
11         else
12             O ← O ∪ CreateChildren(S)           /* otherwise, branch */
13 if z* < ∞ then
14     return x*                                   /* return optimal solution */
15 else
16     report P is infeasible and terminate
```

$\mathbf{x}^*$ to $\mathcal{P}$. Branch-and-bound then backtracks and continues exploring the search tree. It proceeds to node 3. At this node, it solves $\mathcal{P}_{\text{LP}}$ with $x_1 \geq 3$. The resulting solution has an objective value of 3, which is not better than the incumbent solution, whose objective value is also 3. The objective value of the linear relaxation solution serves as the objective bound for the entire subtree, and hence, node 3 can be discarded. The search tree is now entirely explored and $\mathbf{x}^*$ is an optimal solution to $\mathcal{P}$.

Example 2.10. Figure 2.2 shows a graphical representation of the feasible space of the subproblem in each node of the search tree in Figure 2.1. The feasible space of the mixed integer program in each node is given by the dots while the feasible space of its linear relaxation is

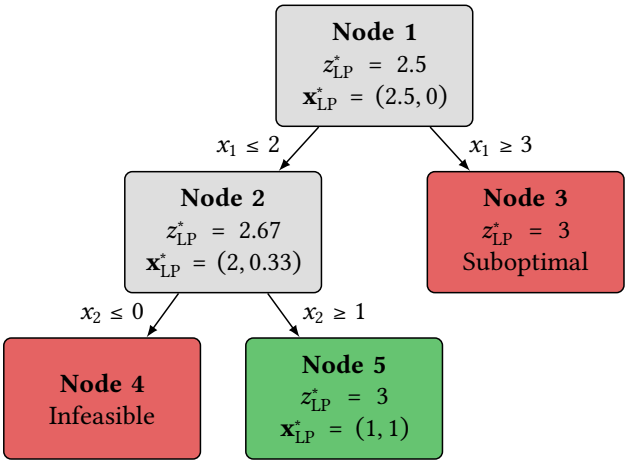


Figure 2.1: Example of a branch-and-bound search tree.

given by the grey area. As the branch-and-bound algorithm progresses deeper into the search tree, the feasible space of the linear relaxation becomes smaller.

Example 2.11. Figure 2.2a shows the feasible space of the mixed integer program and its linear relaxation in the root node (node 1). The linear relaxation solution is located at the point $(2.5, 0)$. If the feasible space of the linear relaxation happens to be the convex hull of the feasible space of the mixed integer program, i.e., the convex hull of the dots, then the linear relaxation solution will be either $(1, 1)$ or $(3, 0)$, giving an objective value of 3. In this case, no branching is required since both of these solutions are integral.

In general, if the feasible space of the linear relaxation is the convex hull of the feasible space of the mixed integer program, solving the linear relaxation will also solve the mixed integer program (e.g., Rader 2010).

2.3.2 Branch-and-Cut

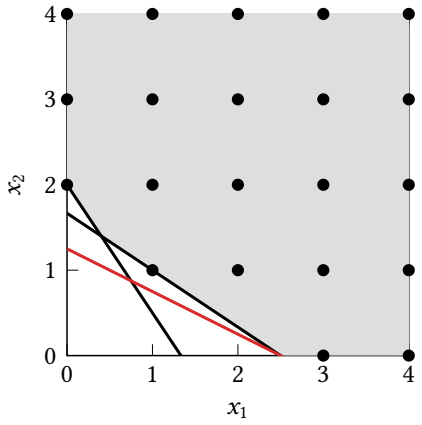
An alternative to branch-and-bound is the cutting plane algorithm by Gomory (1958, 1960, 1963). Instead of branching, the cutting plane algorithm adds constraints to the linear relaxation to enforce integrality. The intention is to remove large portions of the linear relaxation feasible space and bring it towards the convex hull of the feasible space of the mixed integer program. In the context of the cutting plane algorithm, the added constraints are known as *cuts* or *cutting planes*.

As the cutting plane algorithm progresses, the incoming cuts gradually prune smaller sections of the feasible space and become increasingly ineffective. Because of this, the cutting plane algorithm is rarely used in practice, even if it is elegant in theory (Mitchell 2010). Branch-and-cut, developed by Padberg and Rinaldi (1991), mitigates this issue by coupling branch-and-bound with the cutting plane method. Whenever the quality of the cuts tails off, branch-and-cut branches to two or more subproblems instead of continuing with generating cuts.

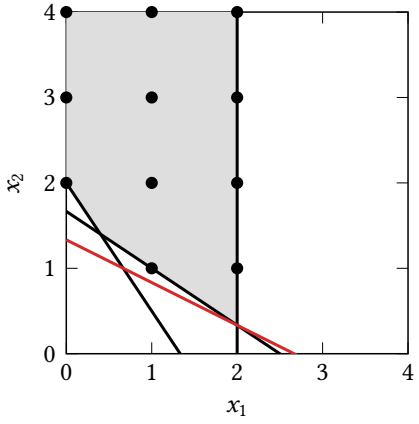
The following discussion summarizes branch-and-cut; a detailed account is given by Mitchell (2010). Consider the mixed integer program $\mathcal{P} = (\mathbf{x}, z, \left[\begin{smallmatrix} A \\ C \end{smallmatrix}\right], (\mathbf{b}, \mathbf{d}), \mathbf{c}, I)$, where $C \in \mathbb{R}^{m \times n}$ has a large number of rows: perhaps exponential in the number of columns, i.e., $m \approx 2^n$. The problem $\mathcal{P}$ can be written in standard form as

$$
\begin{aligned}
\min \quad & \mathbf{c} \cdot \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \geq \mathbf{b}, \\
& C\mathbf{x} \geq \mathbf{d}, && (2.10) \\
& \mathbf{x} \geq \mathbf{0}, \\
& x_j \text{ integer} \qquad \forall j \in I. && (2.11)
\end{aligned}
$$

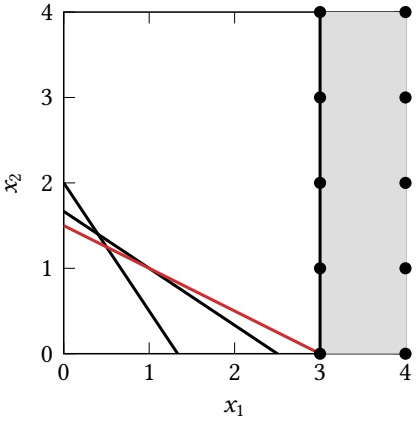
The constraints given by $C\mathbf{x} \geq \mathbf{d}$ may be implicit, i.e., they are not explicitly specified by the user but instead are implicitly included in the model by the solver after deducing that adding certain constraints would improve the model. Examples of these cuts include Gomory cuts, clique cuts and cover cuts.



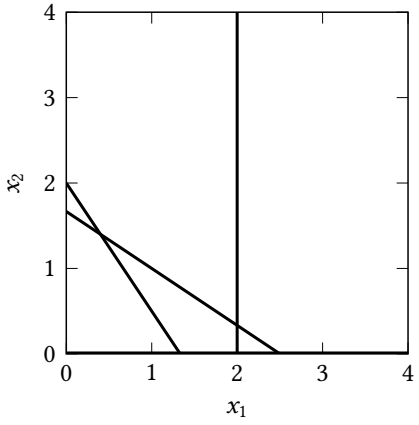
(a) Node 1.



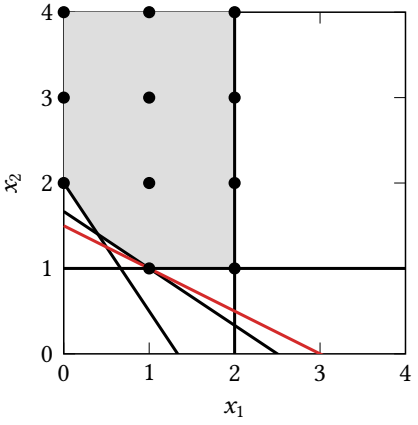
(b) Node 2.



(c) Node 3.



(d) Node 4.



(e) Node 5.

Figure 2.2: A graphical representation of the feasible space for each node in the branch-and-bound tree in Figure 2.1. The black lines are the boundary of the constraints. The grey area is the feasible space of the linear relaxation and the dots are the feasible space of the mixed integer program. The red line represents the objective function. The optimal solution of the linear relaxation is where the red line intersects the feasible space.

Algorithm 2.2: The basic branch-and-cut algorithm.

Input: A mixed integer program $\mathcal{P} = (\mathbf{x}, z, A, \mathbf{b}, \mathbf{c}, I)$
Output: An optimal solution $\mathbf{x}^*$ to $\mathcal{P}$ if one exists

```
1  z* ← ∞
2  O ← {P}
3  while O ≠ ∅
4      S ← SelectNode(O)
5      O ← O \ {S}
6      repeat                                          /* cut separation loop */
7          (x*_LP, z*_LP) ←Solve(S_LP)                 /* solve linear relaxation */
8          γ ← SeparateCuts(S, x*_LP)                  /* find cuts */
9          S ← S ∪ γ                                   /* add cuts */
10     until γ = ∅
11     if z*_LP < z* then
12         if (x*_LP)_j ∈ Z for all j ∈ I then
13             x* ← x*_LP
14             z* ← z*_LP
15         else
16             O ← O ∪ CreateChildren(S)
17 if z* < ∞ then
18     return x*
19 else
20     report P is infeasible and terminate
```

Without loss of generality, consider a relaxed problem $\mathcal{P}'$ defined as

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \geq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \\ & x_j \text{ integer} \qquad \forall j \in I' \subseteq I. \end{aligned}$$

The key idea behind branch-and-cut is to repeatedly solve the linear relaxation $\mathcal{P}'_{\text{LP}}$ of $\mathcal{P}'$ and add subsets of Constraints (2.10) and (2.11) that are implicitly violated. Adding these cuts discards the current solution and forces the solver to find another candidate solution. This continues until the cuts remove exceedingly small sections of the feasible space, at which point the solver branches.

The branch-and-cut algorithm is shown in Algorithm 2.2. It differs from branch-and-bound by the addition of the loop in Line 6. Line 7 solves the linear relaxation. Line 8 solves *separation subproblems* to find or *separate* cuts. Line 9 adds the new cuts.

There are many classes or *families* of cuts in the literature. Each family of cuts is independently derived using mathematical proofs and intends to remove a specific type of section of the feasible space. Ideal families of cuts are those proven to be facets of the convex hull of the feasible space of the mixed integer program because these cuts remove the largest possible portions of the feasible space. These cuts are termed *polyhedral cuts*. Many families of cuts, especially polyhedral cuts, rely on assumptions of the problem and/or model, limiting their

reuse in other problems or even other models of the same problem. A list of families of cuts relevant to vehicle routing problems is given by Semet, Toth and Vigo (2014).

The families of cuts fall into one of two categories:

- *Implied cuts* are redundant constraints that shrink the feasible space of the linear relaxation. Their main purpose is to lift solutions of $\mathcal{P}'_{\text{LP}}$ to solutions of $\mathcal{P}'$ or $\mathcal{P}$. Implied cuts are usually implicitly omitted in the problem specification, and instead, are found internally by a mixed integer programming solver. An example of implied cuts is the Gomory cuts, which round fractional solutions, thereby removing fractional solutions that are $\mathcal{P}'_{\text{LP}}$-feasible but are $\mathcal{P}'$-infeasible or $\mathcal{P}$-infeasible.
- *Problem cuts* are necessary constraints for correctly solving $\mathcal{P}$ but are deliberately omitted because, e.g., there are too many. Their main purpose is to lift solutions of $\mathcal{P}'$ to solutions of $\mathcal{P}$. Problem cuts are typically programmed explicitly into a mixed integer program solver by the user. Examples of problem cuts include subtour elimination cuts, which are discussed in Example 2.12 and in the main chapters.

Since the purpose of the cuts is to lift solutions of $\mathcal{P}'_{LP}$ and $\mathcal{P}'$ to solutions of $\mathcal{P}$, branch-and-cut can be viewed as working towards feasibility from an infeasible starting point. This statement will be contrasted to an orthogonal but complementary statement in the next section.

Cuts are found within a separation subproblem. Separation subproblems implement a *separation algorithm*, of which many can exist for a family of cuts. A separation algorithm checks solutions of the linear relaxation against the idea and logic of its family of cuts, and if the check fails, derives cuts to disallow the current solution as well as other solutions that violate these cuts.

Let $\mathcal{S}$ be the feasible space of $\mathcal{P}'_{\text{LP}}$. Each call to a separation algorithm derives an implicit polyhedron $\mathcal{S}' \supseteq \mathcal{S}$. Given a feasible solution vector $\mathbf{x} \in \mathcal{S}$, a separation algorithm attempts to determine whether $\mathbf{x} \in \mathcal{S}'$, and if not, finds at least one of Constraints (2.10) and (2.11) that removes $\mathbf{x}$ from $\mathcal{S}$.

Example 2.12 (Traveling Salesman Problem)**.** Consider a set $\mathcal{N} = \{1, \dots, N\}$ of nodes labeled from 1 to N. Let $\mathcal{A} = \{(i,j) : i,j \in \mathcal{N}\}$ be the complete set of arcs and associate a cost $c_{i,j} \in \mathbb{R}_+$ with every arc $(i,j) \in \mathcal{A}$. The Traveling Salesman Problem (TSP) attempts to find a cycle of minimal cost that visits every node exactly once. Let $x_{i,j} \in \{0,1\}$ be a binary decision variable indicating if arc $(i,j) \in \mathcal{A}$ is included in the cycle, then the TSP can be formulated as

$$
\begin{aligned}
\text{min} \quad & \sum_{(i,j)\in\mathcal{A}} c_{i,j}\, x_{i,j} & & & \\
\text{subject to} \quad & \sum_{j:(i,j)\in\mathcal{A}} x_{i,j} &= 1 & \forall i \in \mathcal{N}, & (2.12)\\
& \sum_{h:(h,i)\in\mathcal{A}} x_{h,i} &= 1 & \forall i \in \mathcal{N}, & (2.13)\\
& \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}} x_{i,j} &\le |\mathcal{S}| - 1 & \forall \mathcal{S} \subseteq \mathcal{N}\setminus\{1\}, |\mathcal{S}| \ge 2. & (2.14)
\end{aligned}
$$

Constraints (2.12) and (2.13) require an arc entering and leaving every node, which can include

self-cycles. Constraint (2.14) is the subtour elimination constraints, which enforce exactly one cycle through the arcs. The number of subtour elimination constraints is exponential in the number of nodes. Hence, stating all these constraints initially is impractical. Instead, the branch-and-cut model of the TSP starts as the Assignment Problem, i.e., only with Constraints (2.12) and (2.13), and progressively adds a subset of Constraint (2.14) whenever a linear relaxation solution violates some of these constraints. Constraint (2.14) is named the DFJ subtour elimination constraints after its authors Dantzig, Fulkerson and Johnson (1954). There are various kinds of subtour elimination constraints, including some that are polynomial in the number of nodes (e.g., Miller, Tucker and Zemlin 1960). Applegate et al. (2003) and Desrochers and Laporte (1991) discuss these constraints in detail.

Implementing branch-and-cut models requires significant expertise. Proving the validity of a family of cuts can require deep understanding of the problem and the shape of its feasible space. Furthermore, developing efficient separation algorithms may require extensive experimentation with a number of implementations, including heuristics, since some families of cuts can only be separated in exponential time.

Despite these difficulties, branch-and-cut is highly effective for many classes of problems. Today, all state-of-the-art commercial and open-source mixed integer programming solvers use a branch-and-cut algorithm and internally implement many families of cuts in order to remain competitive (e.g., Lodi 2010).

2.3.3 Branch-and-Price

Branch-and-price, invented by Dantzig and Wolfe (1960) and independently by Gilmore and Gomory (1961, 1963), is an advanced variant of branch-and-bound. In contrast to branch-and-cut, branch-and-price extends branch-and-bound with a loop that adds variables instead of constraints. Since variables are implemented as columns in the linear relaxation, this loop is known as *column generation*.

Unlike branch-and-cut, which can be executed on any mixed integer program, branch-and-price can only be invoked on problems that have a form amenable to adding variables. Models of this form are called *column generation models*. Unfortunately, many real-world problems are difficult or even impossible to formulate as a column generation model. Nonetheless, vehicle routing problems usually have formulations that can be transformed into column generation form. Furthermore, many vehicle routing problems can be directly written in column generation form without a reformulation (Desrosiers and Lübbecke 2010). Column generation models of vehicle routing problems can be found in the guide by Feillet (2010).

The discussion below formalizes branch-and-price by reformulating a linear program into column generation form and then proposing the column generation model as the linear relaxation of a mixed integer program. Further details can be found in the works by Barnhart, Johnson et al. (1998), Chung (2010), Desaulniers, Desrosiers and Solomon (2005), Desrosiers and Lübbecke (2010), Lübbecke (2010) and Lübbecke and Desrosiers (2005).

Consider a mixed integer program $\mathcal{P} = (\mathbf{x}, z_{\mathcal{P}}, \left[\begin{smallmatrix} A \\ C \end{smallmatrix}\right], (\mathbf{b}, \mathbf{d}), \mathbf{c}, I)$ with n variables and with

bounded feasible space, written in standard form as

$$\begin{aligned}
\min \quad & z_{\mathcal{P}} = \mathbf{c} \cdot \mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \geq \mathbf{b}, && (2.15)\\
& C\mathbf{x} \geq \mathbf{d}, && (2.16)\\
& \mathbf{x} \geq \mathbf{0}, && (2.17)\\
& x_j \text{ integer} \quad \forall j \in I. && (2.18)
\end{aligned}$$

Constraint (2.15) is called the *linking* or *complicating constraints*. Constraint (2.16) is called the *structural constraints*. The structural constraints typically defines a combinatorial substructure that has a dedicated algorithm in the absence of the complicating constraints. The goal of reformulating a model into column generation form is to separate the complicating constraints and the structural constraints into two problems such that the problem containing only the structural constraints can be solved with dedicated algorithms. This reformulation process is called the *Dantzig-Wolfe decomposition*.

By the definition of $\mathcal{P}$, the set $S = \{\mathbf{x} \in \mathbb{R}^n_+ | C\mathbf{x} \geq \mathbf{d}, x_j \in \mathbb{Z} \text{ for all } j \in I\} \neq \varnothing$, the intersection of Constraints (2.16) to (2.18), is bounded. Its convex hull $\text{conv}(S)$ is a bounded polyhedron that has a finite number k of extreme points $\{\mathbf{s}_1, \dots \mathbf{s}_k\}$. By well-known results from Minkowski and Weyl, $\text{conv}(S)$ can be written as a convex combination of its extreme points. That is, any point $\mathbf{x} \in \text{conv}(S)$ can be expressed as

$$\mathbf{x} = \sum_{j=1}^{k} \mathbf{s}_j \lambda_j, \tag{2.19}$$

where $\lambda_j \geq 0$ and $\sum_{j=1}^{k} \lambda_j = 1$.

Incorporating these ideas into the problem $\mathcal{P}$ and relaxing the integrality constraints on $\mathbf{x}$ results in the column generation model $\mathcal{C}$, defined as

$$\begin{aligned}
\min \quad & z_{\mathcal{C}} = \sum_{j=1}^{k} c_j \lambda_j \\
\text{subject to} \quad & \sum_{j=1}^{k} \mathbf{a}_j \lambda_j \geq \mathbf{b}, \\
& \sum_{j=1}^{k} \lambda_j = 1, && (2.20)\\
& \boldsymbol{\lambda} \geq \mathbf{0},
\end{aligned}$$

where $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_k)$, $c_j = \mathbf{c} \cdot \mathbf{s}_j$ and $\mathbf{a}_j = A\mathbf{s}_j$ for all $j = 1, \dots, k$. Constraint (2.20) is called the *convexity constraint*, and the linear program $\mathcal{C}$ is called the *master problem*.

Enumerating the extreme points of $\text{conv}(S)$ is usually impossible in practice because k is very large: perhaps exponential in the number of constraints. Instead, columns are iteratively added to a related problem $\mathcal{C}'$, called the *restricted master problem*, that initially contains a small number of columns of $\mathcal{C}$. Since not all variables are necessary to prove optimality, column generation can be terminated prior to finding all k of the $\boldsymbol{\lambda}$ variables.

New columns are found by solving a subproblem known as a *pricing subproblem*. Recall from Section 2.2.1 that a linear program solution is optimal if all reduced costs are non-negative. If the reduced cost of a variable is negative, incrementing the value of this variable from zero to some positive value will change the objective value by the reduced cost per unit increment. The pricing subproblem is itself an optimization problem that asks to find a variable with the most negative reduced cost. Formally, the pricing subproblem attempts to find a vector

$$\mathbf{x}^* \in \underset{\mathbf{x}\in\text{conv}(S)}{\arg\min} \{\mathbf{c}\cdot\mathbf{x} - \boldsymbol{\pi}^* A\mathbf{x} | \mathbf{c}\cdot\mathbf{x} - \boldsymbol{\pi}^* A\mathbf{x} < 0\}. \tag{2.21}$$

Observe that the objective function of the pricing problem is the reduced cost of a yet undetermined column by comparing it to Equation (2.9) on page 11. The term $\mathbf{c}\cdot\mathbf{x}$ is the cost coefficient of the upcoming column and is equivalent to a component of $\mathbf{c}_N^\top$ in Equation (2.9). The vector $\boldsymbol{\pi}^*$ is the dual solution of $\mathcal{C}'$ and is equivalent to $\mathbf{y}^{*\top}$ in Equation (2.9). The vector $A\mathbf{x}$ is the column coefficients of the upcoming variable and is equivalent to a column of N in Equation (2.9).

The $\boldsymbol{\lambda}$ variables of $\mathcal{C}$ hide the structural constraints of $\mathcal{P}$, a highly non-trivial set. Hence, the set S must be carefully chosen to give the required reformulation. Usually, the $\boldsymbol{\lambda}$ variables are crafted to represent alternative options. The restricted master problem selects an optimal subset of the options and the pricing subproblem progressively finds better options. *In contrast to branch-and-cut, column generation can be viewed as working towards optimality from a suboptimal starting point.*

Branch-and-price models are mixed integer programs that use $\mathcal{C}$ as its linear relaxation. In branch-and-price, column generation is interleaved with branching. In practice, branching usually occurs on the $\mathbf{x}$ variables of $\mathcal{P}$ because it is difficult to prevent $\boldsymbol{\lambda}$ variables fixed to 0 by branching from being regenerated in a future round of column generation if the dual values dictate that having these columns is beneficial. Recall that the $\mathbf{x}$ variables are readily available within the pricing subproblem; hence, branching on the $\mathbf{x}$ variables simply translates to removing incompatible $\boldsymbol{\lambda}$ variables from $\mathcal{C}'$ and enforcing the branching constraint within the pricing subproblem.

A major benefit of branch-and-price is that the column generation model has stronger dual bounds than the linear relaxation of the original problem. The problem $\mathcal{C}$ has feasible space $\{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} \geq \mathbf{b}\} \cap \text{conv}(S)$, which is the same as or smaller than the feasible space $\{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} \geq \mathbf{b}, C\mathbf{x} \geq \mathbf{d}\}$ of the linear relaxation of $\mathcal{P}$. For some problems, such as set cover problems and vehicle routing problems, column generation models have significantly tighter bounds.

Algorithm 2.3 summarizes the basic branch-and-price algorithm. Line 6 is the column generation loop. Line 7 solves the linear relaxation $\mathcal{S}_{\text{LP}}$ and also returns a solution $\boldsymbol{\pi}^*_{\text{LP}}$ to the dual of $\mathcal{S}_{\text{LP}}$. The `PriceVariables` function in Line 8 solves a pricing subproblem to find new columns. The vector $\mathbf{x}^*$ in Equation (2.21) represents a new column, which is added to the restricted master problem via a λ_j variable with cost coefficient $c_j = \mathbf{c}\cdot\mathbf{x}^*$ and matrix coefficients $\mathbf{a}_j = A\mathbf{x}^*$. In practice, a large number of columns is found by the pricing subproblem in order to hasten convergence by reducing the number of iterations of the column generation loop. Hence, the `PriceVariables` function returns a set γ of columns, which includes an optimal

Algorithm 2.3: The basic branch-and-price algorithm.

Input: A restricted master problem $\mathcal{C}' = (\boldsymbol{\lambda}, z, A, \mathbf{b}, \mathbf{c}, I)$
Output: An optimal solution $\boldsymbol{\lambda}^*$ to $\mathcal{C}'$ if one exists

1 $z^* \longleftarrow \infty$
2 $O \longleftarrow \{\mathcal{C}'\}$
3 **while** $O \neq \varnothing$
4 $\quad \mathcal{S} \longleftarrow$ `SelectNode(`O`)`
5 $\quad O \longleftarrow O \setminus \{\mathcal{S}\}$
6 $\quad$ **repeat** `/* column generation loop */`
7 $\quad\quad (\boldsymbol{\lambda}^*_{\text{LP}}, \boldsymbol{\pi}^*_{\text{LP}}, z^*_{\text{LP}}) \longleftarrow$`Solve(`$\mathcal{S}_{LP}$`)` `/* solve linear relaxation */`
8 $\quad\quad \gamma \longleftarrow$ `PriceVariables(`$\mathcal{S}, \boldsymbol{\pi}^*_{LP}$`)` `/* find columns */`
9 $\quad\quad \mathcal{S} \longleftarrow \mathcal{S} \cup \gamma$ `/* add columns */`
10 $\quad$ **until** $\gamma = \varnothing$
11 $\quad$ **if** $z^*_{\text{LP}} < z^*$ **then**
12 $\quad\quad$ **if** $(\lambda^*_{\text{LP}})_j \in \mathbb{B}$ for all j **then**
13 $\quad\quad\quad \boldsymbol{\lambda}^* \longleftarrow \boldsymbol{\lambda}^*_{\text{LP}}$
14 $\quad\quad\quad z^* \longleftarrow z^*_{\text{LP}}$
15 $\quad\quad$ **else**
16 $\quad\quad\quad O \longleftarrow O \cup$ `CreateChildren(`$\mathcal{S}$`)`
17 **if** $z^* < \infty$ **then**
18 $\quad$ **return** $\boldsymbol{\lambda}^*$
19 **else**
20 $\quad$ report $\mathcal{C}'$ is infeasible and terminate

column, rather than solely one optimal column. Line 9 adds the new columns γ to $\mathcal{S}$. Column generation proceeds until no new columns are found (Line 10).

2.3.4 Branch-and-Cut-and-Price

Branch-and-cut-and-price incorporates both column and cut generation. Recall that column generation gradually reduces suboptimality, while cut generation reduces infeasibility. The integration of both aspects enables branch-and-cut-and-price models to benefit from orthogonal views of a problem.

Since branch-and-price operates on a transformed formulation rather than the original formulation, families of cuts developed for branch-and-cut models cannot be directly implemented in branch-and-cut-and-price. However, for some problems, such as vehicle routing problems, cuts on the original formulation are easily adapted into cuts for the column generation reformulation because the variables in the original formulation are available in the pricing subproblems (Fukasawa et al. 2006). Hence, branch-and-cut-and-price for vehicle routing problems can utilize all conventional cuts.

Furthermore, branch-and-cut-and-price models admit cuts over the variables of the column generation model. For vehicle routing problems, some of these cuts, such as some variations of clique cuts, are extremely strong, but are very difficult to separate because they interfere with the pricing problem (e.g., Jepsen et al. 2008). The inclusion of cuts over the column generation variables enables branch-and-cut-and-price models to break previous world records on many

vehicle routing benchmarks (e.g., Pecin et al. 2014).

Poggi and Uchoa (2014) list several families of cuts applicable to branch-and-cut-and-price models of vehicle routing problems.

2.4 Boolean Satisfiability

This section reviews Boolean satisfiability (SAT) problems, which are decision problems, not optimization problems. Decision problems are optimization problems without an objective function, or equivalently, a constant objective function.

Definition 2.25 (Decision Problem)**.** A *decision problem* $\mathcal{P}$ attempts to find a vector

$$\mathbf{x} \in S,$$

where $S = c_1 \cap \ldots \cap c_m \subseteq \mathbb{R}^n$, called the *feasible space*, is the intersection of *constraints* $c_1, \ldots, c_m \subseteq \mathbb{R}^n$, $\mathbf{x} = (x_1, \ldots, x_n) \in S$, and $x_1, \ldots, x_n$ are *variables*. Every $\mathbf{x} \in S$ is a *feasible solution*. If $S = \varnothing$, the problem is described as *infeasible* or *unsatisfiable*.

Boolean satisfiability problems permit only binary variables and clauses as constraints. A clause is a disjunction of variables and their negations. Conjunctions can be written as clauses by using De Morgan's laws. Even though this class of problems is highly restricted, modern Boolean satisfiability solvers can solve problems with up to millions of variables; therefore, enabling many real-world problems to be formulated and solved as Boolean satisfiability problems by rewriting them with binary indicator variables. The following discussion formalizes Boolean satisfiability. Franco and Martin (2009) produced a comprehensive introduction to Boolean satisfiability.

Definition 2.26 (Variable)**.** A *variable* x is a name or symbol that represents a *value*.

Definition 2.27 (Domain)**.** The *domain* D_x of a variable x is a subset of the Boolean domain that represents the set of possible values that x can take, i.e., $x \in D_x \subseteq \mathbb{B}$.

Definition 2.28 (Literal)**.** A *literal* of a variable x is either x itself or its negation $\neg x$. The literal $\neg x$ is a *negated literal*.

Definition 2.29 (Clause)**.** A *clause* c is a disjunction of literals, i.e.,

$$l_1 \vee \ldots \vee l_k,$$

where $l_1, \ldots, l_k$ are literals.

Definition 2.30 (Boolean Satisfiability Problem)**.** A *Boolean satisfiability problem* $\mathcal{P}$ is a triple $\mathcal{P} = (\mathbf{x}, D, C)$, where $\mathbf{x} = (x_1, \ldots, x_n) \in D$, $x_1, \ldots, x_n$ are variables, $D = D_1 \times \ldots \times D_n$, $D_j \subseteq \mathbb{B}$ is the domain of x_j for $j = 1, \ldots, n$, and $C = \{c_1, \ldots, c_m\}$ is the set of clauses.

Example 2.13. Let $\mathbf{x} = (x_1, x_2, x_3, x_4)$, $D = \mathbb{B}^4$, $C = \{c_1, c_2, c_3\}$, $c_1 \equiv \neg x_1 \vee \neg x_2$, $c_2 \equiv x_2 \vee \neg x_3 \vee x_4$ and $c_3 \equiv \neg x_1 \vee \neg x_3 \vee \neg x_4$. The triple $\mathcal{P} = (\mathbf{x}, D, C)$ is a Boolean satisfiability problem.

Definition 2.31 (Variable Assignment)**.** For a Boolean satisfiability problem $\mathcal{P} = (\mathbf{x}, D, C)$ and any $j = 1, \dots, n$, the variable x_j is *unassigned* if $|D_j| = 2$ and is *assigned* if $|D_j| = 1$. If x_j is assigned, it is *assigned true* if $D_j = \{1\}$ and *assigned false* if $D_j = \{0\}$.

Definition 2.32 (Literal Assignment)**.** A literal x is assigned *true* (respectively *false*) if its variable is assigned true (respectively false). A negated literal $\neg x$ is assigned *true* (respectively *false*) if its variable is assigned false (respectively true).

Definition 2.33 (Clause Satisfaction)**.** A clause is *satisfied* if at least one of its literals is assigned true. A clause is *unsatisfied* if all its literals are assigned false. A clause is *unsatisfiable* if no set of assignments exists that make the clause satisfied.

Example 2.14. Consider the Boolean satisfiability problem in Example 2.13 but with $D = \{0\}^2 \times \{1\} \times \{0\}$. The clause c_1 is satisfied because the literal $\neg x_1$ (or equivalently $\neg x_2$) is assigned true. The clause c_2 is unsatisfiable because all its literals are assigned false.

Definition 2.34 (Solution Space)**.** Let $\mathcal{P} = (\mathbf{x}, D, C)$ be a Boolean satisfiability problem. For a clause $c \in C$, let $x_1^-, \dots, x_{k^-}^-$ be the variables corresponding to its negated literals, and let $x_1^+, \dots, x_{k^+}^+$ be the variables of the remaining literals, then c represents the set

$$\{\mathbf{x} \in \mathbb{B}^n | x_1^- = 0\} \cup \dots \cup \{\mathbf{x} \in \mathbb{B}^n | x_{k^-}^- = 0\} \cup \{\mathbf{x} \in \mathbb{B}^n | x_1^+ = 1\} \cup \dots \cup \{\mathbf{x} \in \mathbb{B}^n | x_{k^+}^+ = 1\}.$$

The *solution space* of $\mathcal{P}$ is the intersection of the variable domains and the clauses, i.e.,

$$S = D \cap \bigcap_{c \in C} c.$$

Definition 2.35 (Solution)**.** Given a Boolean satisfiability problem $\mathcal{P}$ and its solution space S, any vector $\hat{\mathbf{x}} \in S$ is a *solution* to $\mathcal{P}$.

Example 2.15. The vector $(0, 0, 0, 0)$ is a solution to $\mathcal{P}$ in Example 2.13.

Definition 2.36 (Solving a Boolean Satisfiability Problem)**.** Let $\mathcal{P} = (\mathbf{x}, D, C)$ be a Boolean satisfiability problem with solution space S. *Solving* $\mathcal{P}$ aims to either (1) find a solution $\hat{\mathbf{x}} \in S$, or (2) show that $S = \varnothing$, in which case $\mathcal{P}$ is described as *unsatisfiable* or *infeasible*.

2.4.1 The Davis-Putnam-Logemann-Loveland Algorithm

Davis and Putnam (1960) conceived an algorithm for solving Boolean satisfiability problems. It was later extended by Davis, Logemann and Loveland (1962). The resultant algorithm, termed the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, is the basis of all modern Boolean satisfiability solvers.

The DPLL algorithm is a tree-search algorithm that resembles branch-and-bound but differs in two main ways. First, it has no suboptimality checking step because Boolean satisfiability problems are decision problems. Second, instead of solving each node using a linear relaxation like in mixed integer programming, *unit propagation* is executed on each node. Unit propagation is an algorithm that makes assignments by reasoning about clauses that are *unit*.

Algorithm 2.4: The unit propagation procedure in the DPLL algorithm.

Input: A Boolean satisfiability problem $\mathcal{P} = (\mathbf{x}, D, C)$, where $D = D_1 \times \ldots \times D_n$
Output: Reduced domains $D' \subseteq D$

```
1  D' ← D                                              /* copy current domains */
2  repeat
3      changed ← false                                  /* mark as not changed */
4      for all c ∈ C such that c is unit
5          l ← unassigned literal of c
6          if l = ¬x_j for some j then
7              D'_j ← {x ∈ D'_j | x_j = 0}             /* assign variable to false */
8              changed ← true                           /* mark as changed */
9          else
10             D'_j ← {x ∈ D'_j | x_j = 1}              /* assign variable to true */
11             changed ← true                           /* mark as changed */
12 until ¬changed
13 return D'
```

Definition 2.37 (Unit Clause)**.** A clause is *unit* if all but one of its literals are assigned false and the remaining literal is unassigned.

According to Definition 2.33, the unassigned literal of a unit clause must be assigned true in order to satisfy the clause. Unit propagation loops through unit clauses and makes this assignment. It is formally presented in Algorithm 2.4. Line 1 creates a copy of the initial domains, which is gradually reduced during the loop in Line 2. Line 3 marks the domains as unchanged since the previous iteration of the loop. Line 4 loops through all unit clauses. Line 5 retrieves the unassigned literal of the clause. If the literal is a negated literal (Line 6), then the variable is assigned false to make the literal true (Line 7). Otherwise, the literal is simply a variable (Line 9), so it is assigned true (Line 10). Lines 8 and 11 mark the domains as changed. The loop is repeated until the domains do not change (Line 12). After the loop terminates, all possible propagations have been completed and the problem reaches a state known as a *fixed point*. Line 13 returns the reduced domains.

The DPLL algorithm is laid out in Algorithm 2.5. Line 5 calls the unit propagation procedure from Algorithm 2.4. Line 6 replaces the domains of the subproblem with the new domains. If all domains are singletons (Line 7), then the value of each variable can be read from the domains. Line 8 returns these values. If the new domains are not empty, the algorithm branches (Lines 9 and 10). Otherwise, the node is discarded because the subproblem is infeasible.

The `CreateChildren` function selects an unassigned variable and branches to two children nodes with the variable assigned to 0 in one node and to 1 in the other. Formally, given a Boolean satisfiability problem $\mathcal{P} = (\mathbf{x}, D, C)$, it selects a variable x_j such that $|D_j| = 2$ and returns $\{\mathcal{P}_0, \mathcal{P}_1\}$ where $\mathcal{P}_0 = (\mathbf{x}, \{\mathbf{x} \in D | x_j = 0\}, C)$ and $\mathcal{P}_1 = (\mathbf{x}, \{\mathbf{x} \in D | x_j = 1\}, C)$.

The DPLL algorithm terminates in Line 8 if a feasible solution is found. Otherwise, it terminates in Line 11 because the problem is infeasible.

Practically every Boolean satisfiability solver uses the depth-first node selection rule in the `SelectNode` function of Line 3. This rule always selects the deepest open node. Using this

Algorithm 2.5: The DPLL algorithm.

Input: A Boolean satisfiability problem $\mathcal{P} = (\mathbf{x}, D, C)$, where $D = D_1 \times \ldots \times D_n$
Output: A solution $\mathbf{x}^*$ to $\mathcal{P}$ if one exists

```
1  O ← {P}
2  while O ≠ ∅
3      S ← SelectNode(O)
4      O ← O \ {S}
5      D^S ← UnitPropagation(S)                    /* call unit propagation */
6      S ← (x, D^S, C)                             /* replace domains */
7      if |D_j^S| = 1 for all j = 1, ..., n then   /* if all domains are singletons */
8          return x* where x* ∈ D^S                /* return solution */
9      else if D^S ≠ ∅ then                        /* if all domains not empty */
10         O ← O ∪ CreateChildren(S)                /* branch */
11 report P is infeasible and terminate
```

rule enables the entire search tree to be stored in a stack data structure, which allows for easy implementations. Other node selection rules severely complicate the implementation of the search tree and remain a difficult research question to this day.

Example 2.16. Figure 2.3 shows a search tree in the DPLL algorithm for the problem from Example 2.13. No propagation occurs at the root level (node 1) and all variables remain unassigned. The DPLL algorithm then selects the literal x_1 and branches on it (node 2). Unit propagation assigns $\neg x_2$ for c_1 to be satisfied. Next, the algorithm selects the literal x_3 and branches on it (node 3). Unit propagation assigns x_4 in order to satisfy c_2. The DPLL algorithm then detects that c_3 cannot be satisfied since all its literals are false. It backtracks, undoing the last decision x_3 and deciding $\neg x_3$ (node 4). At this stage, all clauses are satisfied. The x_4 variable is not yet assigned but can simply be assigned to any value, say $x_4 = 0$. The solution $(1, 0, 0, 1)$ can then be read from the singleton domains.

2.4.2 Conflict Analysis

Exploring a search tree naively can result in a subtree being explored repeatedly. This is particularly inefficient if the subtree is infeasible. Marques Silva and Sakallah (1996) invented what is now called *conflict analysis* or *conflict-driven clause learning* (CDCL) to prevent infeasible subtrees from being explored more than once. Conflict analysis is widely adopted in competitive Boolean satisfiability solvers of today (e.g., Eén and Sörensson 2004, Moskewicz et al. 2001).

Whenever the DPLL algorithm reaches an infeasible node, it performs conflict analysis, which inspects the chain of propagations that lead to the infeasibility and then creates a clause, called a *nogood*, that prevents the infeasibility from reoccurring. Any subtree that contains the same propagations can be discarded because the nogood cannot be satisfied; hence, the same infeasibility cannot reoccur anywhere in the search tree.

Every assignment made by unit propagation is recorded in a graph known as the *implication graph*. To derive nogoods, conflict analysis begins with a tentative nogood and repeatedly replaces literals in the nogood with literals previously added to the implication graph until it

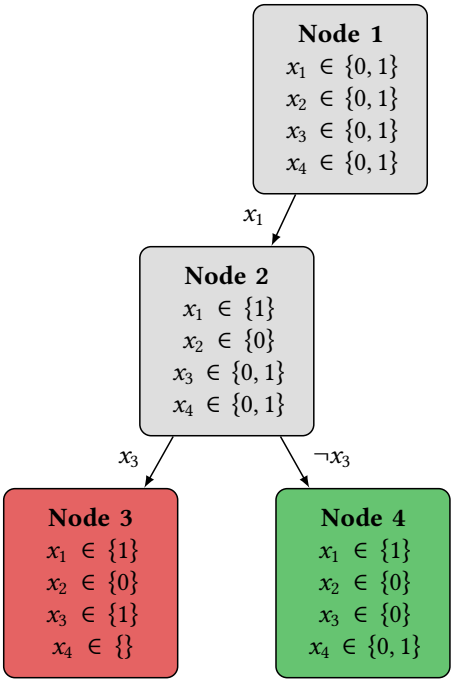


Figure 2.3: Example of a search tree in the DPLL algorithm.

fulfils a stopping criterion specified by the type of nogood implemented. The final nogood is then added to the problem clauses and is henceforth treated as a regular clause (although it may be removed if excessively many nogoods are found). Conflict analysis is formalized as follows.

Definition 2.38 (Nogood)**.** A *nogood* is a clause that is not explicitly stated in a model but is implied by the problem clauses.

Definition 2.39 (Reason for Propagation)**.** A *reason* for a propagation is a logical relation of the form $l_1 \wedge \ldots \wedge l_k \longrightarrow l_{k+1}$, where $l_1, \ldots, l_{k+1}$ are literals.

Definition 2.40 (Reason for Infeasibility)**.** A *reason* for an infeasibility is a logical relation of the form $l_1 \wedge \ldots \wedge l_k \longrightarrow$ false, where $l_1, \ldots, l_k$ are literals.

A reason explains a propagation or an infeasibility in terms of the assignments that lead to the propagation or infeasibility. Reasons can be stated as clauses and vice versa since

$$(l_1 \wedge \ldots \wedge l_k \longrightarrow l_{k+1}) \equiv (\neg l_1 \vee \ldots \vee \neg l_k \vee l_{k+1})$$

and

$$(l_1 \wedge \ldots \wedge l_k \longrightarrow \text{false}) \equiv (\neg l_1 \vee \ldots \vee \neg l_k \vee \text{false}) \equiv (\neg l_1 \vee \ldots \vee \neg l_k).$$

Definition 2.41 (Implication Graph)**.** Let $\mathcal{P} = (\mathbf{x}, D, C)$ be a Boolean satisfiability problem where $\mathbf{x} = (x_1, \ldots, x_n)$. An *implication graph* $G = (V, E)$ is an acyclic directed graph with one vertex for every literal, i.e., $V = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ and no initial edges (i.e., $E = \varnothing$). The edges E is populated by the DPLL algorithm. Associated with every edge $e \in E$ is a label $d_e \in \{0, 1, 2, \ldots\}$.

Without loss of generality, assume that unit propagation is considering the clause

$$l_1 \vee \ldots \vee l_k \vee l_{k+1},$$

where $l_1, \ldots, l_k$ are previously assigned false and l_{k+1} is unassigned. Unit propagation will assign l_{k+1} with the reason

$$\neg l_1 \wedge \ldots \wedge \neg l_k \longrightarrow l_{k+1}.$$

This reason is stored in the implication graph by adding edges from every literal on the left-hand side of the clause to the literal on the right-hand side, i.e., the edges

$$(\neg l_1, l_{k+1}), \ldots, (\neg l_k, l_{k+1}).$$

For each $e \in \{(\neg l_1, l_{k+1}), \ldots, (\neg l_k, l_{k+1})\}$, the current depth of the search tree is stored in d_e.

A conflict occurs whenever all literals in a clause are assigned false. Consider the clause

$$l_1 \vee \ldots \vee l_k$$

where all $l_1, \ldots, l_k$ are assigned false. Unit propagation fails with the reason

$$\neg l_1 \wedge \ldots \wedge \neg l_k \longrightarrow \text{false}.$$

This reason serves as the initial nogood. Conflict analysis walks the implication graph and repeatedly replaces each literal l on the left-hand side with the conjunction of all literals that lead to l in the implication graph, i.e., the conjunction of all l' such that $(l', l) \in E$. This occurs until a stopping criterion is met. The stopping criterion is specified by the type of nogood implemented. Once conflict analysis is terminated, the final reason

$$\bar{l}_1 \wedge \ldots \wedge \bar{l}_k \longrightarrow \text{false}$$

is rewritten as the clause

$$\neg\bar{l}_1 \vee \ldots \vee \neg\bar{l}_k,$$

which is then added to the problem clauses C. After adding this clause, the DPLL algorithm backtracks, selects the next node at some depth d and then removes all $e \in E$ with $d_e > d$.

Because conflict analysis replaces each literal l in the reason with a conjunction of literals that lead to l being assigned, the incoming literals are all assigned either at the current node of the search tree or at an ancestor node. *Since conflict analysis uses information in ancestors to derive nogoods, it is naturally most effective when coupled with a depth-first node selection rule.*

The *first unique implication point* (1UIP) type of nogood is commonly used as it is known to be efficient (Zhang et al. 2001). To derive 1UIP nogoods, conflict analysis continues replacing the literals on the left-hand side until exactly one literal is propagated at the current depth of the search tree.

Example 2.17. Figure 2.4 shows the implication graph corresponding to the search tree in Example 2.16 and Figure 2.3. No propagation occurs at the root level. After branching on x_1,

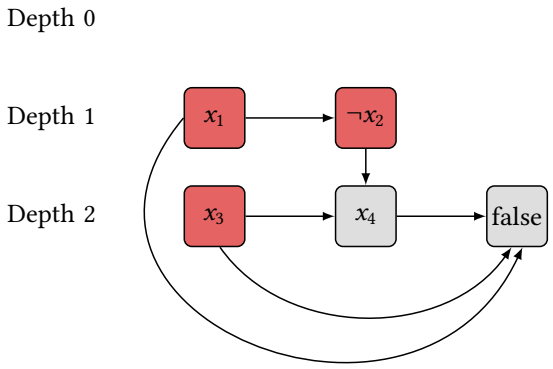


Figure 2.4: Example of an implication graph. All irrelevant nodes are omitted.

unit propagation assigns $\neg x_2$ with the reason $x_1 \longrightarrow \neg x_2$. Next, the algorithm branches on x_3. At the second level, x_4 is propagated with the reason $\neg x_2 \wedge x_3 \longrightarrow x_4$. The algorithm then identifies that the clause $\neg x_1 \vee \neg x_3 \vee \neg x_4$ cannot be satisfied since all its literals are false. Conflict analysis starts with the tentative nogood

$$x_1 \wedge x_3 \wedge x_4 \longrightarrow \text{false},$$

and, according to the 1UIP stopping criterion, replaces each literal ordered from the deepest to shallowest depth until exactly one literal is propagated at the current depth. It replaces x_4 with the left-hand side of its reason, resulting in the nogood

$$x_1 \wedge x_3 \wedge (\neg x_2 \wedge x_3) \longrightarrow \text{false}.$$

The literal x_3 is duplicated, and hence, is removed, resulting in

$$x_1 \wedge x_3 \wedge \neg x_2 \longrightarrow \text{false}.$$

Of all the literals in the nogood, only x_3 is propagated at the current level. Hence, the nogood is a valid 1UIP nogood and conflict analysis stops. The reason is rewritten as the clause

$$\neg x_1 \vee \neg x_3 \vee x_2,$$

which is then added to C.

2.5 Constraint Programming

Constraint programming (CP) was developed by the artificial intelligence community in the late 1980s by Jaffar and Lassez (1987), Jaffar, Michaylov et al. (1992) and Van Hentenryck (1989). Today, it successfully solves many decision problems and optimization problems. Optimization problems can be solved using constraint programming by formulating them as constraint optimization problems. This section directly reviews constraint optimization problems without considering their underlying constraint satisfaction problems.

Constraint programming admits general variables unlike linear programming, mixed integer programming and Boolean satisfiability. Examples of variables include set-value variables and graph-valued variables. Many constraint programming solvers limit the variables to take on finite integer values. As such, this thesis makes the same assumption. Constraint programming also allows general constraints, which can be stated using a declarative language like in mixed integer programming and Boolean satisfiability but are individually implemented and enforced using an algorithm called a propagator. The following definitions formalize constraint optimization problems with a focus on minimization. Detailed introductions to constraint programming can be found in the works by Michel and Van Hentenryck (2010) and Rossi, Van Beek and Walsh (2006).

Definition 2.42 (Variable)**.** A *variable* x is a name or symbol that represents a *value*.

Definition 2.43 (Domain)**.** The *domain* D_x of a variable x is a finite subset of the integers that represents the set of possible values that x can take, i.e., $x \in D_x \subset \mathbb{Z}$.

Definition 2.44 (Constraint)**.** A *constraint* c over a vector $\mathbf{x} = (x_1, \dots, x_n)$ of variables is a subset of $\mathbb{Z}^n$, i.e., $c \subseteq \mathbb{Z}^n$.

Definition 2.45 (Constraint Optimization Problem)**.** A (minimization) *constraint optimization problem* $\mathcal{P}$ is a tuple $\mathcal{P} = (\mathbf{x}, D, C, f)$, where $\mathbf{x} = (x_1, \dots, x_n) \in D$, $x_1, \dots, x_n$ are variables, $D = D_1 \times \dots \times D_n$, $D_j \subset \mathbb{Z}$ is the domain of x_j for $j = 1, \dots, n$, $C = \{c_1, \dots, c_m\}$ is the set of constraints over $\mathbf{x}$, and $f \in \{1, \dots, n\}$ is the index of the variable x_f that represents the objective value.

Example 2.18. Given $\mathbf{x} = (x_1, x_2)$, $D = [0, 10] \times [0, 10]$, $C = \{c_1, c_2\}$, $c_1 = \{(x_1, x_2) \in D | x_1 \geq x_2\}$, $c_2 = \{(x_1, x_2) \in D | x_2 = 8\}$ and $f = 1$, then the tuple $\mathcal{P} = (\mathbf{x}, D, C, f)$ is a constraint optimization problem. For brevity, c_1 can be written as $x_1 \geq x_2$ and c_2 as $x_2 = 8$.

Definition 2.46 (Variable Assignment)**.** Let $\mathcal{P} = (\mathbf{x}, D, C, f)$ be a constraint optimization problem. For any $j = 1, \dots, n$, the variable x_j is *unassigned* if $|D_j| \geq 2$ and is *assigned* if $|D_j| = 1$.

Definition 2.47 (Solution Space)**.** The set intersection $S = D \cap \bigcap_{c \in C} c$ is the *solution space* of the constraint optimization problem $\mathcal{P} = (\mathbf{x}, D, C, f)$.

Definition 2.48 (Solution)**.** Given a constraint optimization problem $\mathcal{P} = (\mathbf{x}, D, C, f)$ and its solution space S, any vector $\hat{\mathbf{x}} \in S$ is a *solution* to $\mathcal{P}$ and has *objective value* $\hat{x}_f$.

Example 2.19. The set $\{(8, 8), (9, 8), (10, 8)\}$ is the solution space of $\mathcal{P}$ from Example 2.18, and $(8, 8)$ is a solution with objective value 8.

Definition 2.49 (Solving a Constraint Optimization Problem)**.** Let $\mathcal{P} = (\mathbf{x}, D, C, f)$ be a constraint optimization problem with solution space S. *Solving* $\mathcal{P}$ aims to either (1) find a solution

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in S} x_f,$$

in which case $\mathbf{x}^*$ and x_f^* are respectively called an *optimal solution* and the *optimal value*, or (2) show that $S = \varnothing$, in which case $\mathcal{P}$ is described as *unsatisfiable* or *infeasible*.

Algorithm 2.6: The constraint programming propagation engine.

Input: A constraint optimization problem $\mathcal{P} = (\mathbf{x}, D, C, f)$ and a propagator p_c for each constraint $c \in C$
Output: Reduced domains $D' \subseteq D$

```
1  D' ← D                                          /* copy current domains */
2  repeat
3      changed ← false                              /* mark as not changed */
4      for c ∈ C
5          D'' ← p_c(D')                           /* propagate constraint */
6          if D'' ≠ D' then
7              D' ← D''                            /* store new domains */
8              changed ← true                      /* mark as changed */
9  until ¬changed
10 return D'
```

2.5.1 Branch-and-Prune

Constraint optimization problems are solved using a branch-and-prune algorithm similar to the branch-and-bound algorithms of mixed integer programming (e.g., Michel and Van Hentenryck 2010, Milano 2010). The main difference is that each node of the search tree is solved using propagation instead of via a linear relaxation.

Definition 2.50 (Propagator)**.** Let c be a constraint over some variables $\mathbf{x} \in D$. A *propagator* $p : D \longrightarrow D$ of c is a monotonically non-increasing function from domains to domains, i.e., $p(D'') \subseteq p(D')$ whenever $D'' \subseteq D' \subseteq D$.

Example 2.20. Consider the constraint c_1 from Example 2.18. One possible propagator for c_1 is the function $p(D_1, D_2) = ([\min(D_2), \max(D_1)] \times [\min(D_2), \max(D_1)]) \cap ([\min(D_1), \max(D_1)] \times [\min(D_2), \max(D_2)])$.

Algorithm 2.6 outlines the main propagation loop, which resembles the unit propagation subroutine in the DPLL algorithm for solving Boolean satisfiability problems. Line 4 loops through every constraint. Line 5 calls the propagator of a constraint. If the domains have changed (Line 6), then the domains are stored (Line 7). Line 8 marks the domains as changed so that another iteration of propagation occurs. When the procedure exits, the solver is in a state known as a *fixed point*.

Algorithm 2.7 presents the constraint programming branch-and-prune algorithm. It differs from branch-and-bound for mixed integer programs between Lines 6 and 14. The `Propagate` function in Line 6 runs Algorithm 2.6 to propagate the constraints and returns the domains after propagation. Line 7 replaces the domains of $\mathcal{S}$ with the reduced domains $D^{\mathcal{S}}$. Line 8 discards suboptimal nodes, i.e., if the minimum of the domain of the objective variable is greater or equal to the objective value of the incumbent solution. If all domains are singletons (Line 9), the solution (Line 10) and objective value (Line 11) are extracted from the domains.

Line 12 decreases the upper bound of the variable denoting the objective value in all open nodes. The `SetNewUpperBound` function takes a set of open nodes O and a new upper bound z^* then sets the domain D_f of x_f to $D_f \cap [\min(D_f), z^* - 1]$ for every $\mathcal{P} = (\mathbf{x}, D, C, f) \in O$.

Algorithm 2.7: The basic branch-and-prune algorithm for constraint optimization problems.

Input: A constraint optimization problem $\mathcal{P} = (\mathbf{x}, D, C, f)$, where $D = D_1 \times \ldots \times D_n$
Output: An optimal solution $\mathbf{x}^*$ to $\mathcal{P}$ if one exists

```
1  z* ← ∞
2  O ← {P}
3  while O ≠ ∅
4      S ← SelectNode(O)
5      O ← O \ {S}
6      D^S ← Propagate(S)                          /* propagate constraints */
7      S ← (x, D^S, C, f)                          /* replace domains */
8      if |D_f^S| > 0 and min(D_f^S) < z* then     /* if not suboptimal */
9          if |D_j^S| = 1 for all j = 1, ..., n then   /* if all domains are singletons */
10             x* ← x̂ where x̂ ∈ D^S                 /* store solution */
11             z* ← x*_f                            /* store objective value */
12             O ← SetNewUpperBound(O, z*)          /* set new upper bound in all open
                   nodes */
13         else if D^S ≠ ∅ then                     /* if all domains not empty */
14             O ← O ∪ CreateChildren(S)            /* branch */
15 if z* < ∞ then
16     return x*
17 else
18     report P is infeasible and terminate
```

Line 14 creates new children nodes by branching. Branching in constraint programming differs from branching in mixed integer programming because variables in constraint programming do not have a value until their domains are singletons. Branching in constraint programming partitions the domain of a variable and enforces one of the partitions in each of the children nodes. Formally, given a constraint optimization problem $\mathcal{P} = (\mathbf{x}, D, C, f)$, the `CreateChildren` function

1. finds a variable x_j such that $|D_j| \geq 2$,
2. finds k partitions of D_j, i.e., finds $D_j^1, \ldots, D_j^k$ such that $D_j = D_j^1 \cup \ldots \cup D_j^k$ and $D_j^{k_1} \cap D_j^{k_2} = \varnothing$ for all $k_1, k_2 = 1, \ldots, k$, where $k_1 \neq k_2$, and
3. creates k children nodes with constraint optimization problems $\mathcal{P}^1 = (\mathbf{x}, \{\mathbf{x} \in D | x_j \in D_j^1\}, C, f)$, ..., $\mathcal{P}^k = (\mathbf{x}, \{\mathbf{x} \in D | x_j \in D_j^k\}, C, f)$.

2.5.2 Conflict Analysis

Conflict analysis can also be implemented within a constraint programming solver (e.g., Feydy and Stuckey 2009, Jussien and Lhomme 2002, Ohrimenko, Stuckey and Codish 2009). It operates in a similar fashion to Boolean satisfiability. In fact, conflict analysis in constraint programming frequently relies on an underlying Boolean satisfiability solver (e.g., Feydy and Stuckey 2009). Propagators return reasons for domain changes rather than making the changes directly. The reasons are sent to the Boolean satisfiability solver, which tracks the domain changes using binary indicator variables.

Let $\mathcal{P} = (\mathbf{x}, D, C, f)$ be a constraint optimization problem where $\mathbf{x} = (x_1, \ldots, x_n)$ and $D =$

$D_1 \times \dots D_n$. For every possible value $v \in D_j$ of every variable x_j, the underlying Boolean satisfiability solver is augmented with two indicator variables named $[\![x_j = v]\!]$ and $[\![x_j \leq v]\!]$ and two clauses linking the indicator variables. The variable $[\![x_j = v]\!]$ indicates whether $D_j = \{v\}$, and $[\![x_j \leq v]\!]$ indicates whether $v' \leq v$ for all $v' \in D_j$. For brevity, writing $[\![x_j \neq v]\!]$, $[\![x_j < v]\!]$, $[\![x_j > v]\!]$ and $[\![x_j \geq v]\!]$ respectively refers to the literals $\neg[\![x_j = v]\!]$, $[\![x_j \leq v - 1]\!]$, $\neg[\![x_j \leq v]\!]$ and $\neg[\![x_j \leq v - 1]\!]$. The two indicator variables $[\![x_j = v]\!]$ and $[\![x_j \leq v]\!]$ are linked with the clauses

$$[\![x_j = i]\!] \longrightarrow [\![x_j \leq i]\!]$$

and

$$[\![x_j \leq i - 1]\!] \longrightarrow [\![x_j \leq i]\!].$$

Using this framework, nogoods can be naturally translated into domain changes in the constraint programming model and vice versa. Hence, constraint programming solvers implementing conflict analysis are themselves hybrid solvers.

Constraint programming solvers that implement conflict analysis are highly effective and are consistently ranked amongst the fastest solvers today (Stuckey et al. 2014). These solvers have successfully closed many problems, and in particular, many challenging scheduling problems (Schutt et al. 2010).

2.6 Hybridization Techniques

This section discusses several methods for hybridizing mixed integer programming and constraint programming. It begins with a review of global optimization constraints, followed by constraint-based Lagrangian relaxation, constraint-based column generation, and finally, logic-based Benders decomposition and branch-and-check. Readers interested in a comprehensive appraisal of hybridization techniques can refer to the survey by Hooker and van Hoeve (2018) and book by Van Hentenryck and Milano (2011).

2.6.1 Global Optimization Constraints

Without loss of generality, consider a minimization constraint program $\mathcal{P} = (\mathbf{x}, C, D, f)$ with feasible space $S_{\mathcal{P}}$. Assume that $\mathcal{P}$ has an objective function $f_{\mathcal{P}}$ that calculates the objective variable x_f, i.e., $x_f = f_{\mathcal{P}}(\mathbf{x})$. The smallest value of the domain D_f of variable x_f is the current lower bound. This lower bound is used to prune suboptimal subtrees in Line 8 of Algorithm 2.7. Hence, having tight bounds greatly reduces the size of the search tree. Unfortunately, the link between x_f and the other variables is generally weak; perhaps only after assigning many variables will the minimum of the objective variable increase.

Focacci, Lodi and Milano (1999, 2000, 2002, 2004) remedied this deficiency by including a global constraint, called a *global optimization constraint*, in $\mathcal{P}$ that contains a relaxation $\mathcal{R}$ of $\mathcal{P}$. The relaxation $\mathcal{R}$ aims to propagate a tighter lower bound than the existing constraints of $\mathcal{P}$. Furthermore, some relaxations, such as linear relaxations, provide reduced costs, which are used to remove values from domains.

Let $f_{\mathcal{R}}$ be the objective function and $S_{\mathcal{R}}$ be the feasible space of $\mathcal{R}$. Recall that $S_{\mathcal{P}} \subseteq S_{\mathcal{R}}$ and $f_{\mathcal{R}}(\mathbf{x}) \leq f_{\mathcal{P}}(\mathbf{x})$ for every $\mathbf{x} \in S_{\mathcal{P}}$. Hence, the optimal value $z^*_{\mathcal{R}}$ of $\mathcal{R}$ is a lower bound to x_f. The relaxation $\mathcal{R}$ is wrapped within a global constraint r and included in $\mathcal{P}$. Whenever the constraint programming propagation engine calls the propagator of r, it solves $\mathcal{R}$ to compute the optimal value $z^*_{\mathcal{R}}$ and then tightens the domain with, e.g., $D_f \longleftarrow D_f \cap [z^*_{\mathcal{R}}, \max(D_f)]$.

In some relaxations, reduced costs can also be used to remove certain values from the domains. Assume that $\mathcal{P}$ has a variable x with domain $D_x = \{a, \dots, b\}$. Also assume that $\mathcal{R}$ is a linear relaxation with variables $y_a, \dots, y_b \in \{0, 1\}$ and the constraint $\sum_{j=a}^{b} y_j = 1$, which together are used as a one-to-one mapping to the values in D_x, i.e.,

$$x = k \leftrightarrow y_k = 1.$$

Solving $\mathcal{R}$ produces an objective value $z^*_{\mathcal{R}}$ and reduced costs $\bar{y}_a, \dots, \bar{y}_b$. For $j = a, \dots, b$, whenever $z^*_{\mathcal{R}} + \bar{y}^j \geq z^*$, then j can be removed from D_x since setting $x = j$ would result in the objective value being worse than that of the incumbent solution z^*.

2.6.2 Constraint-based Lagrangian Relaxation

The mathematical programming community developed Lagrangian relaxation to compute stronger objective bounds. Its central idea is to relax a problem by moving difficult constraints into the objective function and weighting them with a penalty whenever they are violated. The relaxed problem, called a *Lagrangian relaxation*, minimizes the original objective function in conjunction with the penalties. A byproduct of moving constraints into the objective function is that coupling constraints can sometimes be removed, resulting in multiple independent subproblems. These can be solved using any appropriate technology, and hence, Lagrangian relaxation can be used for hybridization even though it is originally built for a different purpose.

Fontaine, Michel and Van Hentenryck (2014) generalized Lagrangian relaxation to constraint programming in constraint-based Lagrangian relaxation. The following discussion summarizes constraint-based Lagrangian relaxation with a focus on hybridization. The derivation for the general case is available in the original work. Bergman, Cire and van Hoeve (2015) and Sellmann (2004) developed alternative ideas on constraint-based Lagrangian relaxation.

Consider the general problem $\mathcal{P}$

$$\min_{\mathbf{x} \in D_{\mathbf{x}}, \mathbf{y} \in D_{\mathbf{y}}} \{f(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in c_1, \mathbf{y} \in c_2, (\mathbf{x}, \mathbf{y}) \in c_3\},$$

where $\mathbf{x} \in D_{\mathbf{x}}$ and $\mathbf{y} \in D_{\mathbf{y}}$ are the variables, $D_{\mathbf{x}}$ and $D_{\mathbf{y}}$ respectively are the domains of $\mathbf{x}$ and $\mathbf{y}$, and $c_1, c_2, c_3 \subseteq D_{\mathbf{x}} \times D_{\mathbf{y}}$ are constraints. The *satisfiability degree* $\sigma : D_{\mathbf{x}} \times D_{\mathbf{y}} \longrightarrow \mathbb{R}$ is a function such that

$$\sigma(\mathbf{x}, \mathbf{y}) \leq 0 \leftrightarrow (\mathbf{x}, \mathbf{y}) \in c_3.$$

The satisfiability degree $\sigma(\mathbf{x}, \mathbf{y})$ measures the violation of constraint c_3 in a vector $(\mathbf{x}, \mathbf{y})$ and is non-positive whenever c_3 is satisfied. For any constant $\lambda \in \mathbb{R}_+$, called the *Lagrangian multiplier*,

the generalized Lagrangian relaxation $\mathcal{L}(\lambda)$ is defined as

$$\min_{\mathbf{x}\in D_{\mathbf{x}},\mathbf{y}\in D_{\mathbf{y}}} \{f(\mathbf{x},\mathbf{y}) + \lambda\sigma(\mathbf{x},\mathbf{y}) | \mathbf{x} \in c_1, \mathbf{y} \in c_2\}.$$

The additional term $\lambda\sigma(\mathbf{x},\mathbf{y})$ in the objective function penalizes the infeasibility of c_3. Whenever c_3 is satisfied, $\lambda\sigma(\mathbf{x},\mathbf{y}) \leq 0$ because $\lambda \geq 0$ and $\sigma(\mathbf{x},\mathbf{y}) \leq 0$ for all $(\mathbf{x},\mathbf{y}) \in c_3$.

Let $S_{\mathcal{P}} = c_1 \cap c_2 \cap c_3$ and $S_{\mathcal{L}(\lambda)} = c_1 \cap c_2$ be the feasible space of $\mathcal{P}$ and $\mathcal{L}(\lambda)$ respectively, and let $z^*_{\mathcal{P}}$ and $z^*_{\mathcal{L}(\lambda)}$ be the optimal value of $\mathcal{P}$ and $\mathcal{L}(\lambda)$. Obviously, $S_{\mathcal{P}} \subseteq S_{\mathcal{L}(\lambda)}$ and $f(\mathbf{x},\mathbf{y}) + \lambda\sigma(\mathbf{x},\mathbf{y}) \leq f(\mathbf{x},\mathbf{y})$ for all $(\mathbf{x},\mathbf{y}) \in S_{\mathcal{P}}$. Therefore, $\mathcal{L}(\lambda)$ is a relaxation of $\mathcal{P}$ by Definition 2.4. As such, $z^*_{\mathcal{L}(\lambda)} \leq z^*_{\mathcal{P}}$ for any $\lambda \geq 0$. The tightest possible bound is found by maximizing $\mathcal{L}(\lambda)$ over λ, i.e.,

$$\max_{\lambda\in\mathbb{R}_+} \min_{\mathbf{x}\in D_{\mathbf{x}},\mathbf{y}\in D_{\mathbf{y}}} \{f(\mathbf{x},\mathbf{y}) + \lambda\sigma(\mathbf{x},\mathbf{y}) | \mathbf{x} \in c_1, \mathbf{y} \in c_2\}.$$

This problem is called the *generalized Lagrangian dual.* Now, assume that $f(\mathbf{x},\mathbf{y}) + \lambda\sigma(\mathbf{x},\mathbf{y})$ can be expressed as a sum of functions of $\mathbf{x}$ and functions of $\mathbf{y}$, i.e.,

$$f(\mathbf{x},\mathbf{y}) + \lambda\sigma(\mathbf{x},\mathbf{y}) = g_1(\mathbf{x},\lambda) + g_2(\mathbf{y},\lambda).$$

Then, the generalized Lagrangian dual can be stated as

$$\max_{\lambda\in\mathbb{R}_+} \left\{ \min_{\mathbf{x}\in D_{\mathbf{x}}} \{g_1(\mathbf{x},\lambda) | \mathbf{x} \in c_1\} + \min_{\mathbf{y}\in D_{\mathbf{y}}} \{g_2(\mathbf{y},\lambda) | \mathbf{y} \in c_2\} \right\}.$$

The two inner optimization problems can be solved using different technologies, e.g., one with mixed integer programming and the other with constraint programming.

Lagrangian duals are usually solved using a subgradient method, which moves towards an optimal solution in the $(\mathbf{x},\mathbf{y},\lambda)$-space. A subgradient algorithm begins with an initial value for λ, solves the two inner optimization problems with this fixed value of λ and then uses the inner solutions to calculate the next value of λ. This repeats until it reaches a local extremum. For convex optimization problems, local extrema are global extrema.

2.6.3 Constraint-based Column Generation

Junker et al. (1999) introduced constraint-based column generation. This framework hybridizes mixed integer programming and constraint programming in column generation models. It solves the restricted master problem using mixed integer programming and the pricing subproblem using constraint programming.

Recall that the pricing subproblem attempts to find the vector $\mathbf{x}^*$ in the minimization problem of Equation (2.21). This vector can be found using any method provided that it satisfies several criteria; namely, it has negative reduced cost and is a valid point in the original formulation. Pricing algorithms are commonly based on dynamic programming as these algorithms can be specialized to specific problems, making them highly efficient. Of course, pricing subproblems can also be solved using constraint programming.

However, constraint programming is not ideal for solving pricing subproblems because the solvers generally perform depth-first search. In depth-first search, successive columns are quite

similar; the matrix coefficients of one column to the next may only swap one value. Due to the nature of the master problem, which selects a subset from a diverse set of columns, having too many similar columns forces more rounds of column generation prior to convergence. Feillet, Gendreau and Rousseau (2007) mitigated this issue with advanced search heuristics.

Interested readers can consult the survey by Gualandi and Malucelli (2013) or the book chapter by Castro, Grossmann and Rousseau (2011) for further details.

2.6.4 Logic-based Benders Decomposition and Branch-and-Check

Benders (1962) developed a decomposition technique now called Benders decomposition. Its key idea is to divide a problem into a master problem and a subproblem if fixing the value of some difficult variables simplifies the problem. The master problem retains only the difficult variables and the subproblem contains the remaining variables. Benders decomposition begins by solving the master problem to optimality, which results in a candidate solution for the difficult variables. Next, it fixes the value of these variables in the subproblem according to the candidate solution and then solves the subproblem. The subproblem is easier to solve because the difficult variables are fixed. If the candidate solution is suboptimal or infeasible in the subproblem, a cut removing the current candidate solution is added to the master problem. The master problem is then reoptimized to compute a new candidate solution. Benders decomposition iterates between the master problem and the subproblem until the candidate solution is optimal.

Since Benders decomposition adds cuts to the master problem to discard candidate solutions, it is, in a way, related to branch-and-cut. The key difference between Benders decomposition and branch-and-cut is that the Benders subproblem encompasses an entire optimization problem with its own constraints and variables, whereas the separation subproblem in branch-and-cut only checks specific aspects of the problem (i.e., one family of cuts).

The master problem in Benders decomposition is a mixed integer program and the subproblems are linear programs. Hooker (1994) defined the inference dual of a problem, and using this dual, generalized Benders decomposition to logic-based Benders decomposition. Logic-based Benders decomposition can be used for hybridization by implementing the master problem using mixed integer programming and the subproblem using constraint programming, for example.

Thorsteinsson (2001) proposed the branch-and-check framework. Branch-and-check is today recognized to be essentially the same as logic-based Benders decomposition even though it is originally developed from a different perspective. The subproblem in Benders decomposition is traditionally solved only after the master problem is optimized. Branch-and-check is developed specifically with the intention that the subproblem is solved throughout the search tree. However, the classical and logic-based Benders decomposition frameworks do not necessarily stipulate that the subproblem must be solved only after the master problem is optimized. Due to this, the terminology is somewhat blurred in recent literature; this dissertation takes the traditional view and focuses on branch-and-check.

A crucial disadvantage of adding cuts after the master problem is optimized is that the search may return a superoptimal solution that is infeasible for the original problem because the master problem lacks knowledge about the constraints in the subproblem. Another disadvantage is that

the entire search tree is restarted after adding a cut, and hence, suboptimal and infeasible subtrees are explored again. Branch-and-check avoids these two issues by solving the subproblem throughout the search tree, and hence, infeasible and suboptimal subtrees are only explored once. Even though the subproblem can be solved throughout the search tree, it is not necessarily solved at every node. The frequency of solving the subproblem is controlled by the implementation and should balance the difficulty of the master problem and the subproblem against the strength of the cuts. Since branch-and-check adds cuts into the master problem during search, it is essentially a branch-and-cut method that uses Benders subproblems for cut separation.

A minor disadvantage of branch-and-check is that cuts excluding suboptimal solutions unnecessarily accumulate during the search. The impact of an excessive number of cuts is not specifically studied in the literature but is not believed to be significant due to the efficiency of modern mixed integer programming solvers. Of course these cuts can be periodically removed during the search.

Branch-and-check is summarized as follows. A thorough treatment can be found in the work by Beck (2010). Branch-and-check decomposes a problem $\mathcal{P}$ of the form

$$\min_{\mathbf{x}\in D_{\mathbf{x}},\mathbf{y}\in D_{\mathbf{y}}} \{f_1(\mathbf{x}) + f_2(\mathbf{x},\mathbf{y}) | c_1(\mathbf{x}), c_2(\mathbf{x},\mathbf{y})\},$$

where $\mathbf{x}$ and $\mathbf{y}$ are the variables, $(D_{\mathbf{x}}, D_{\mathbf{y}})$ is the initial domains and c_1 and c_2 are constraints. The constraints c_2 link $\mathbf{x}$ and $\mathbf{y}$. The master problem $\mathcal{M}$ is a relaxation of $\mathcal{P}$, defined as

$$\min_{\mathbf{x}\in D_{\mathbf{x}},z\in\mathbb{R}} \{f_1(\mathbf{x}) + z | c_1(\mathbf{x}), c_2'(\mathbf{x}), f_2'(\mathbf{x}) \leq z\},$$

where $c_2'(\mathbf{x})$ and $f_2'(\mathbf{x})$ respectively are relaxations of $c_2(\mathbf{x},\mathbf{y})$ and $f_2(\mathbf{y})$ based only on $\mathbf{x}$. Now, assume that f_1, c_1 and c_2' are linear. In this case, branch-and-check solves the linear relaxation of $\mathcal{M}$ in every node of the search tree. This results in a candidate solution $\hat{\mathbf{x}}$, which is fed to the subproblem $\mathcal{S}$. Given $\hat{\mathbf{x}}$, the subproblem $\mathcal{S}$ solves

$$\min_{\mathbf{y}\in D_{\mathbf{y}}} \{f_2(\hat{\mathbf{x}},\mathbf{y}) | c_2(\hat{\mathbf{x}},\mathbf{y})\}.$$

If the subproblem finds that $\hat{\mathbf{x}}$ is suboptimal with regards to $\mathbf{y}$, then the constraint

$$\alpha_{\hat{\mathbf{x}}}(\mathbf{x}) \leq z$$

is added to the master problem, where $\alpha_{\hat{\mathbf{x}}}(\mathbf{x})$ is a function of $\mathbf{x}$. This constraint prohibits the candidate solution $\hat{\mathbf{x}}$ by enforcing a lower bound that is higher than its objective value. If the subproblem finds that $\hat{\mathbf{x}}$ is infeasible in consideration of $\mathbf{y}$, a constraint

$$\beta_{\hat{\mathbf{x}}}(\mathbf{x})$$

is added to the master problem that removes $\hat{\mathbf{x}}$. In both cases, the new constraint forces the master problem to find another candidate solution.

2.7 Vehicle Routing Problems

Vehicle routing problems model a fleet of vehicles that visit various locations to perform their duties. The task is to find least-cost routes that adhere to various restrictions on the routes and on the vehicles. The family of vehicle routing problems is extensive and this section only summarizes four basic problems. Interested readers seeking an extensive introduction to vehicle routing problems can consult the book by Vigo and Toth (2014).

The Capacitated Vehicle Routing Problem The Capacitated Vehicle Routing Problem (CVRP) is a basic problem that requires vehicles to depart a central depot to pick up goods and then deliver them to the same depot at the end of their routes. Each pickup, called a *request*, is associated with a weight, called its *load*. All vehicles are identical, and each vehicle can carry requests up to a maximum total weight, called the *vehicle capacity*. Since the vehicle capacity and the load of each request are fixed, the number of vehicles necessary to pick up all requests can be calculated exactly by solving a bin-packing problem. The goal of the CVRP is to find routes for these vehicles that pick up all requests while minimizing the total travel distance.

The Vehicle Routing Problem with Time Windows The Vehicle Routing Problem with Time Windows (VRPTW) extends the CVRP with time constraints. Every request is associated with a time frame, called its *time window*, within which the request must be picked up. Vehicles can arrive at the location of a request prior to the opening of its time window but must wait until its time window opens before commencing service. Due to the time windows, the number of vehicles necessary to pick up all requests cannot be calculated beforehand in the VRPTW, unlike the CVRP. Hence, the VRPTW contains an additional source of variability in that the number of vehicles must also be determined. In particular, this can complicate models that have vehicle symmetries. The number of vehicles may or may not be limited in the VRPTW, but each vehicle can perform at most one route, i.e., it must depart and return to the central depot at most once. Some variants of the VRPTW first minimize the number of vehicles required to pick up all requests and then minimize the total travel distance.

The Pickup and Delivery Problem The Pickup and Delivery Problem (PDP), also called the Vehicle Routing Problem with Pickup and Delivery (VRPPD), is an extension of the CVRP. The problem consists of a number of *pickup-delivery pairs* instead of single pickup requests. Each pair is associated with both a pickup request and a delivery request. Every pickup must be brought to its destination during a route instead of the central depot at the end of a route. The load of a vehicle cannot exceed its capacity at any time along its route; although making a delivery will reduce its load and free up the capacity to pick up other requests. Since all pickups must be delivered, all vehicles return empty to the central depot.

The Pickup and Delivery Problem with Time Windows The Pickup and Delivery Problem with Time Windows (PDPTW), also known as the Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW), extends the PDP in the same manner that the

VRPTW extends the CVRP. Every pickup request and every delivery request retains its load but additionally has a time window within which the request must be serviced.

Solomon (1987) published a set of problem instances for the VRPTW. Over the years, these instances have become the standard benchmarks for the VRPTW: practically all VRPTW models are evaluated against each other using the Solomon problems. The instances consist of 25, 50 and 100 requests with varying difficulty. The instances are grouped into three classes. The R class instances have requests that are randomly located, whereas the C class instances have requests clustered together. The RC instances have a mixture of both randomly located requests and clustered requests.

Benchmark problems also exist for the CVRP, PDP and PDPTW but these are not relevant to this thesis.

2.8 Models of Vehicle Routing Problems

This section summarizes several influential mixed integer programming and constraint programming models of the basic vehicle routing problems.

2.8.1 Mixed Integer Programming Models

Vehicle routing problems are historically modeled using mixed integer programming. Basic formulations are based on a *three-index model* (e.g., Desaulniers, Madsen and Røpke 2014). The main decision variables in this model are binary variables that indicate if a particular vehicle traverses an arc. The model is named after the number of components in the indices of the main decision variables, which are a vehicle and the head and tail of an arc. In the three-index model, the time windows, vehicle load and other logical constraints are captured using *big-M* constraints, which are linearizations of logical constraints. The big-M rewritings weaken the linear relaxation and inhibits three-index models from scaling to larger instances. Furthermore, since the decision variables are indexed by vehicle, the model exhibits vehicle symmetry, which significantly hinders its performance. A benefit of the model, however, is that all the constraints and variables are stated in the model, and hence, the model can be input directly into a mixed integer programming solver.

The vehicle symmetries and the big-M linearizations are eliminated in branch-and-cut approaches. Branch-and-cut models of vehicle routing problem use a *two-index model*, which generalizes the standard formulation of the Traveling Salesman Problem (TSP) seen in Example 2.12. The primary decision variables in this model are indexed by only the arcs; hence the name two-index model. The vehicles are unnumbered, and therefore, vehicle symmetries do not exist. Furthermore, the load and time constraints are hidden behind separation algorithms. Instead of stating all the load and time constraints, as in the three-index model, the two-index branch-and-cut model calls separation subproblems to check the feasibility of the load and time constraints. Load-infeasible or time-infeasible partial paths are prevented using cuts that forbid

combinations of arcs (e.g., Kallehauge, Boland and Madsen 2007). The linear relaxation itself has no knowledge of the time and load constraints.

Bard, Kontoravdis and Yu (2002) developed a branch-and-cut model of the VRPTW that inherits the capacity cuts from branch-and-cut models of the CVRP. Capacity cuts generalize the subtour elimination cuts of the TSP to consider vehicle capacity. Hence, they serve the purpose of excluding both subtours and partial paths that exceed the vehicle capacity. This model also implements infeasible path cuts to exclude partial paths that violate the time windows. Infeasible path cuts require at least one arc in an infeasible partial path to be unused.

Kallehauge, Boland and Madsen (2007) solved the VRPTW with a branch-and-cut model that uses the subtour elimination constraints from the TSP instead of the capacity cuts. Vehicle capacity constraints are enforced by the same infeasible path cuts that enforce the time windows. They also proved that both the subtour elimination cuts and the infeasible path cuts can be strengthened using ideas conceived by Mak (2001) for a variant of the TSP.

Desrochers, Desrosiers and Solomon (1992) built a branch-and-price model of the VRPTW based on an earlier model by Desrosiers, Soumis and Desrochers (1984) that excludes vehicle capacity constraints. The model uses a set partitioning problem as the master problem and a resource-constrained shortest path problem as the pricing subproblem. The pricing algorithm finds paths that can have cycles. The cycles make each node in the branch-and-bound tree easier to solve, but degrade the lower bounds, and hence, increase the number of nodes in the search tree. This model, developed early in the study of vehicle routing problems, is now fundamental to all modern branch-and-price models of vehicle routing problems.

Baldacci, Mingozzi and Roberti (2011) presented a sophisticated branch-and-cut-and-price model of the VRPTW based on the earlier work by Baldacci, Christofides and Mingozzi (2008). The model uses a series of heuristics that move the dual solution from each iteration towards their optimal values; thus bypassing the early iterations of column generation. It then invokes a new pricing algorithm using these dual solutions. This model solved all but one of the Solomon benchmarks.

Røpke (2012) produced a branch-and-cut-and-price model of the VRPTW that incorporates many ideas in the literature. Its main contribution is validating that strong branching has a significant impact. Strong branching solves many candidate children nodes near the root to optimality, evaluates the candidate children nodes against some criteria and then commits to branch on only one set of children nodes (e.g., Achterberg, Koch and Martin 2005). This model successfully solves the last open Solomon instance. This is only achieved after the tremendous amount of study of the VRPTW over the previous 25 years, which attests to the difficulty of the VRPTW in general.

Røpke, Cordeau and Laporte (2007) implemented two branch-and-cut models of the PDPTW. The first model uses variables that represent time and vehicle load, while the second indirectly models time and vehicle capacity using infeasible path cuts. Subtour elimination and infeasible path constraints are lifted to also consider the precedence relationships that arise from the pickup-delivery constraints. Both models also feature many other families of cuts. Their results indicate that the more compact second model outperforms the first.

Dumas, Desrosiers and Soumis (1991) formulated the first branch-and-price model of the PDPTW. The model is a straightforward extension of existing branch-and-price models of vehicle routing problems. They derived new dominance criteria for the PDPTW that enable the pricing subproblem to eliminate paths that cannot contribute to an optimal solution.

Røpke and Cordeau (2009) constructed a branch-and-cut-and-price model of the PDPTW that unites many of the existing ideas in the literature. The model is paired to one of two pricing algorithms: the first solves an elementary shortest path problem and second solves the non-elementary version. Solving the linear relaxation of column generation model with non-elementary paths is known to be easier but results in weaker lower bounds, and hence, more nodes in the search tree. Their experiments show that the performance of the two pricing algorithms are roughly similar. The authors also show that certain families of cuts are implied by the elementary shortest path problem, and hence, are unnecessary when using this pricing problem.

2.8.2 Constraint Programming Models

There are only a small number of constraint programming models of vehicle routing problems. The constraint programming community generally avoids vehicle routing problems because the objective function is linear and linear functions are known to have weak propagators.

Backer et al. (2000) developed a constraint programming model for vehicle routing problems using successor variables. Successor variables describe a path starting at a vehicle's starting node to its end node via the requests it visits. The model is tested on the Solomon instances and is solved using local search coupled with three metaheuristics. This model improved the upper bounds for four of the instances.

Bent and Van Hentenryck (2004) presented a constraint programming model of the VRPTW. This model uses two stages to minimize the number of vehicles and then minimize the total travel distance. The first stage is solved using simulated annealing and the second using large neighborhood search. This model was instrumental in finding many best solutions to the Solomon benchmarks at the time. Bent and Van Hentenryck (2006) later extended this model to the PDPTW.

Rousseau, Gendreau, Pesant and Focacci (2004) applied constraint programming-based column generation to the VRPTW. In this model, the column generation master problem is a set covering problem and is solved regularly using mixed integer programming. The pricing problem is an elementary shortest path problem that also contains global optimization constraints to assist with the objective bounds.

Feillet, Gendreau and Rousseau (2007) applied limited discrepancy search in the pricing subproblem of a branch-and-price model of the VRPTW. Limited discrepancy search is a search strategy related to depth-first search that uses a heuristic to guide the tree-search algorithm towards promising nodes by backtracking prior to a complete depth-first exploration (Harvey and Ginsberg 1995). As the search progresses, deeper subtrees are permitted, allowing the search to focus on potentially good subtrees in the early stages. The model uses a standard dynamic programming algorithm in the pricing subproblem but complements its underlying

graph with additional nodes for the limited discrepancy search.

Recently, Benchimol et al. (2012) developed an effective propagator for the WeightedCircuit global constraint, which can be used to infer objective bounds in vehicle routing problems. However, it is not yet applied to vehicle routing problems.

Chapter 3

The Joint Vehicle and Crew Routing and Scheduling Problem

Vehicle routing problems are studied in academic circles for their combinatorial properties. As an academic curiosity, algorithms are developed to solve larger and larger instances of highly simplified problems such as the Capacitated Vehicle Routing Problem (e.g., Pecin et al. 2014). However, these simplified problems bear little relevance to the challenges faced in the logistics industry (Bräysy and Hasle 2014).

A transportation problem commonly encountered is the routing and scheduling of vehicles and crews. This problem is computationally difficult because the routing and scheduling subproblems are high interdependent: a change to a vehicle route allows an exponential number of new crew schedules. To reduce the computational complexity, vehicle routing and crew scheduling are usually solved sequentially in practice (e.g., Barnhart, Lu and Shenoi 1998). This involves devising vehicle routes, on which crews are then scheduled. By designing vehicle routes first, sequential approaches may lead to suboptimal or even infeasible crew schedules since decisions in the vehicle routing phase may ignore crew constraints and objectives. Therefore, it is desirable to simultaneously consider vehicle and crew constraints and objectives, particularly for cases in which crew constraints are tight or crew costs exceed vehicle costs.

This chapter proposes the Joint Vehicle and Crew Routing and Scheduling Problem (JVCRSP), which adds a second layer of routing for crews to the Pickup and Delivery Problem with Time Windows (PDPTW) introduced in Section 2.7. In many applications of vehicle routing problems, goods are moved from one location to another, usually across the course of a day. The JVCRSP is motivated by applications in humanitarian and military logistics; in these contexts, vehicles (e.g., airplanes) travel long routes and transport food and medical supplies across time horizons that can span several days, and hence, determining crew schedules becomes an important part of the problem. For example, vehicles must be operated by crews, which have limitations on their duty times. Crews are able to interchange vehicles at different locations and to travel as passengers before and after their duty times. The JVCRSP is extremely challenging computationally because

As described in the Preface, early results of this chapter are published in the paper titled "Joint Vehicle and Crew Routing and Scheduling".

vehicle routes and crew routes are interdependent. Allowing crews to interchange vehicles adds an additional time element to the problem since two vehicles must be synchronized in order for an exchange to proceed. It is thus necessary to decide whether vehicles wait and for how long at a location because both vehicles must be present at the same location for an interchange to occur.

This chapter develops a mixed integer programming and a constraint programming formulation of the JVCRSP that jointly optimize vehicle and crew routing and scheduling in the hope of remedying some limitations of sequential approaches. The formulations overlay crew routing constraints over the PDPTW and add a number of synchronization constraints to link the vehicles and crews. In addition, the constraint programming formulation includes a novel global optimization constraint that uses a linear relaxation to check whether the current crew partial routes are feasible and to bound crew costs, which is crucial in the early stages of the search when the focus is on vehicle routing.

Both the mixed integer programming model and the constraint programming model are each developed into two additional models that sequentialize the vehicle routing and scheduling components in order to evaluate the impacts of simultaneously optimizing these decisions. These six models are then solved using a regular branch-and-bound complete tree search and a large neighborhood search, giving a total of twelve methods.

Experimental results on instances with up to 100 requests and three cost functions indicate that (1) the joint optimization of vehicle and crew routing can produce considerable benefits over sequential methods, (2) the combination of constraint programming and large neighborhood search scales significantly better than pure constraint programming and mixed integer programming approaches, and (3) vehicle interchanges are critical for obtaining high-quality solutions. These findings indicate that it is now in the realm of optimization technology to simultaneously optimize vehicle and crew routing and scheduling in a single model, and that concurrently modeling vehicle and crew routing may bring significant benefits in cost reduction compared to methods that sequentialize these decisions.

The remainder of this chapter is organized as follows. Section 3.1 reviews existing work on related problems. Section 3.2 describes the JVCRSP. Section 3.3 discusses a high-level model of the problem. Sections 3.4 and 3.5 concretizes the high-level model as a mixed integer programming model and a constraint programming model. Section 3.6 describes the large neighborhood search common to both the mixed integer programming and constraint programming models. Section 3.7 reports experimental results, and Section 3.8 concludes this chapter.

3.1 Literature Review

Simultaneous vehicle routing and crew scheduling problems have not attracted much interest in the literature at this point, probably due to their inherent complexity (Drexl 2012). This section reviews three relevant problems from the literature.

Kim, Koo and Park (2010) considered a problem in which vehicles transport teams to service customers. Vehicles are able to move without any team on board. The problem features three

types of tasks that must be serviced in order, and all customers have one task of each type. Each team can only service one compatible type of task. The mixed integer programming formulation has variables indexed by five dimensions and is intractable. The paper develops a simple local search algorithm that is embedded within a particle swarm metaheuristic. This approach was developed specifically for the problem and cannot easily accommodate side constraints.

Hollis, Forbes and Douglas (2006) solved a mail distribution problem that features multiple depots at which vehicle routes begin and end. The model was solved using a two-stage heuristic column generation approach, which cannot be guaranteed to solve the problem to optimality. In the first stage, trips that begin and end at the depots are computed. The second stage takes a scheduling approach and assigns vehicles and crews to the trips. Vehicle interchange can only occur at the depots at the start or end of a trip. In addition, the model features a 24-hour cyclic time period and variables indexed by discretized blocks of time.

Drexl et al. (2013) considered a problem that includes European legislation and relay stations where drivers rest and interchange vehicles. Vehicles must wait a fixed amount of time upon reaching a relay station. Vehicle interchange can only occur if other drivers arrive at this relay station during this time interval. The problem also provides a shuttle service, separate from the fleet of vehicles, that can be used to move drivers between relay stations. The problem is solved using a two-stage large neighborhood search method. In the first stage, a vehicle routing problem is solved and the resulting routes form the customers of another vehicle routing problem in the second stage, in which the crews perform the role of vehicles. Observe that this approach also cannot jointly optimize vehicle and crew routing. The model features several fixed parameters, such as the duration during which a vehicle waits at a relay station, and a search procedure that only explores a limited number of nearby relay stations. Both these restrictions greatly reduce the search space but negatively impact vehicle interchange, leading the authors to conclude that allowing for vehicle interchange does not significantly improve the objective value.

Drexl (2012) classified simultaneous vehicle routing and crew scheduling problems as a Vehicle Routing Problem with Multiple Synchronization constraints (VRPMS). Synchronization is a feature present in some vehicle routing problems, in which decisions about one object (e.g., vehicle, route, request) imply actions that may or must be taken on other objects.

Drexl (2007, 2014) developed a VRPMS called the Vehicle Routing Problem with Trailers and Transshipments (VRPTT). It features two vehicle classes: lorries which can move independently, and trailers which must be towed by an accompanying lorry. All lorries begin at a single depot with or without a trailer. A lorry can detach its trailer at transshipment locations to visit customers who are unable to accommodate a trailer (e.g., due to size). Lorries can transfer load into and/or attach with any trailer at any transshipment location. A lorry that has detached its trailer can also return to the depot without reattaching a trailer, leaving its trailer behind at a transshipment location to be collected by another lorry at a future time. Several sophisticated mixed integer programming formulations were presented, which were solved using branch-and-cut on instances with up to eight customers, eight transshipment locations and eight vehicles. Drexl (2013) argued that simultaneous vehicle routing and crew scheduling problems can be

reformulated into the VRPTT by casting crews as lorries with zero capacity, and vehicles as trailers.

The VRPTT is most closely related to the JVCRSP since lorry routes and trailer routes are jointly computed, and the search space is not artificially limited. A key difference is that the VRPTT includes load synchronization, which is not considered in the JVCRSP. Load synchronization refers to the ability to either transfer load from one vehicle to another while both are present, or partially or fully transporting load to a transshipment location and then having a different vehicle retrieve the load to deliver it to its final destination.

Finally, observe that vehicle and crew *scheduling* problems are thoroughly studied (e.g., Cordeau et al. 2001, Freling, Huisman and Wagelmans 2001, 2003, Freling, Wagelmans and Paixão 1999, Haase, Desaulniers and Desrosiers 2001, Mercier, Cordeau and Soumis 2005, Mercier and Soumis 2007, Mesquita and Paias 2008). These problems aim to assign vehicles and crews to a predetermined set of trips, with each trip consisting of a fixed route and usually with fixed arrival and departure times. Trips in these problems correspond to parts of a route in the JVCRSP, which are not available a priori, but instead, must be computed during search, thereby increasing the computational challenges.

3.2 Problem Description

This section describes the application and introduces the JVCRSP as an abstraction of interesting elements of the original problem. Given the complexity of the overall problem, the research methodology is to find effective solution approaches to interesting aspects of the application before addressing the problem as a whole.

The JVCRSP is motivated by an air transportation problem faced by the Royal Australian Air Force. The challenge is to design effective routes for moving goods, known as parcels, around the world. The majority of parcels originate in Australia and are destined for the Asia-Pacific region. Most parcels only have a due date for arrival; parcels rarely have a due date for departure, possibly because of ample storage space at military bases. The parcels are associated with a weight and a 3-dimensional size of length, width and height. The sizes of the parcels vary: they can be as small as a spare part for a plane or as large as a tank. They can also include doctors moving to other bases in order to perform urgent medical procedures on patients, moving soldiers on and off rotations, moving food and medical supplies to bases, keeping inventories of spare parts, delivering mail, etc. To simplify the problem, this study only considers the weights and due dates. Obviously, the full problem contains 3D-packing constraints for the airplanes as well.

The planes are operated by crews, who have limited flying time. They will often fly to a base and hand over the plane to another crew. The crew will then spend some recovery time at that base and then continue in the next plane that arrives. Crews can also be moved on planes to other bases if needed. The abstracted problem considered in this chapter only accounts for one duty period. The full problem expands the number of duty periods, which is modeled exactly as in the chapter with the addition of a minimum rest time between duty periods. The minimum

rest time depends on the previous flight times, e.g., 12 hours rest for 24 hours of flight, 60 hours rest for 96 hours of flight, and 4 days rest for 14 days of flight.

Additionally, the full problem restricts the number of planes that can occupy an airfield at any given time because of the limited availability of taxiing and parking space. This aspect of the motivating application is studied in the next chapter.

The JVCRSP aims at capturing interesting elements of the motivating application surrounding the two layers of interdependent vehicle and crew routing. In particular, the JVCRSP generalizes the PDPTW introduced in Section 2.7 with two major additions.

First, it groups requests by location, i.e., every request has an attribute for its location. This contrasts with traditional vehicle routing problems, in which locations are synonymous with requests. Grouping requests by location makes it possible to model crews interchanging vehicles because the model can recognize if two vehicles are present at the same location.

Second, the JVCRSP adds crews to the PDPTW. Crews must travel on a vehicle when traveling from one location to another, and are free to switch vehicles at any location. Hence, crews can exit a location on a vehicle different to the one on which they entered. Every vehicle can carry an unlimited number of crews onboard as passengers, and one of the crews onboard, known as the driver, must operate the vehicle whenever it travels from one location to another.

Each crew is restricted to at most one driving segment, which is defined as the time period from the beginning of a crew's first drive to the end of the crew's last drive. The driving segment is limited to a maximum duration. During the driving segment, the crew may interchange vehicles to drive on other vehicles and may travel as a passenger to reach a vehicle before recommencing driving on this vehicle. The time taken for traveling as a passenger is included within the driving segment. Crews can travel on any vehicle to reach the location of their first drive, and crews can travel on any vehicle back to the depot after driving. No distance or time limitations are placed on crews before and after the driving segment.

The JVCRSP minimizes a weighted sum of the number of vehicles and crews used, as well as the total vehicle and crew travel distances.

3.3 High-Level Modeling Concepts

This section presents several modeling decisions underlying the constraint programming and mixed integer programming models and concepts relevant to solving the JVCRSP.

Figure 3.1 illustrates an example of several vehicle routes and crew routes for a problem with three vehicles, five crews and five locations. Every crew begins at the common crew start node (CS) and either moves directly to the crew end node (CE), signifying that the crew is unused, or proceeds to a vehicle start node (S1 to S3) to board a vehicle. The vehicles and crews visit the locations (L1 to L5) to service requests and then proceed to the vehicle end nodes (E1 to E3), where the vehicles complete their routes. The crews then disembark the vehicles and proceed to the crew end node (CE). Observe that the yellow crew changes vehicles at L3 and then drives the blue vehicle. Also, observe that the blue crew changes vehicles at L3, travels as a passenger on the red vehicle and then changes vehicles at L5 to drive on the green vehicle. Because the

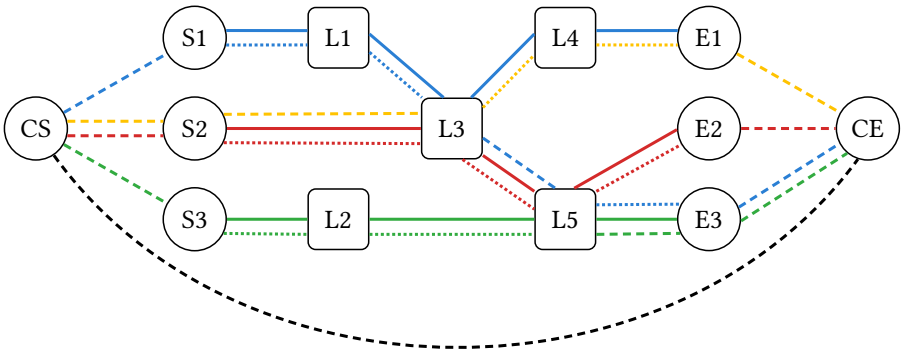


Figure 3.1: Example of vehicle routes and crew routes. Vehicle routes are marked by solid lines and crew routes are marked by dashed lines and dotted lines. Crews travel as passengers on dashed lines and as drivers on dotted lines.

driving segment of a crew is defined from the moment that it starts driving to the moment that it finishes driving, the driving segment of the blue crew is its entire route from S1 to E3.

Like in conventional vehicle routing problems, a route in the JVCRSP is a sequence of requests. However, each request is associated with an additional input parameter that maps it to a location. Along a route, a subsequence of requests at the same location can be thought of as a visit to the location, with an entry at the first request in the subsequence and an exit at the last request in the subsequence.

Figure 3.2 illustrates a route that visits two locations. A vehicle departs the starting node S to service requests 1 to 3 at a location and then moves to the next location to service requests 4 to 6 before finishing at the end node E. Even though a vehicle route is a sequence of requests, semantically, the vehicle enters the first location at request 1, departs it at request 3, enters the second location at request 4 and departs it at request 6.

The model employs a number of crew routing constraints, which mirror the vehicle routing constraints. Vehicles and crews are synchronized using constraints that require

- crews to move with a vehicle when moving from one location to another,
- vehicles to have exactly one driver onboard when moving from one location to another, and
- the driver of a vehicle to be one of the crews onboard.

These requirements allow vehicles and crews to move independently within a location.

Figure 3.3 illustrates two crews switching vehicles at a location. The location contains requests 1 to 6. A vehicle and crew enter the location at request 1, and another vehicle and crew enter the location at request 4. While the two vehicles are servicing requests, the first crew moves to the departure request of the second vehicle (request 6), and the second crew moves to the departure request of the first vehicle (request 3). The two crews wait for their new vehicles to complete servicing the requests then leave the location. In order for this interchange to occur, both vehicles must be at the same location at the same time. Suppose requests 1 to 3 require 10 units of time for service, and requests 4 to 6 require 5 units of time for service. The first crew can move to the second vehicle because the crew arrived (at time 10) before the departure of

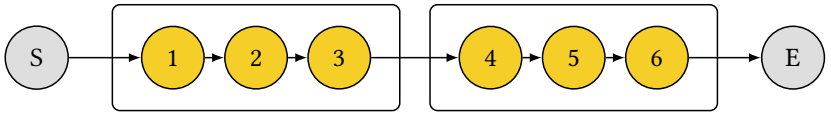


Figure 3.2: Example of locations along a vehicle route.

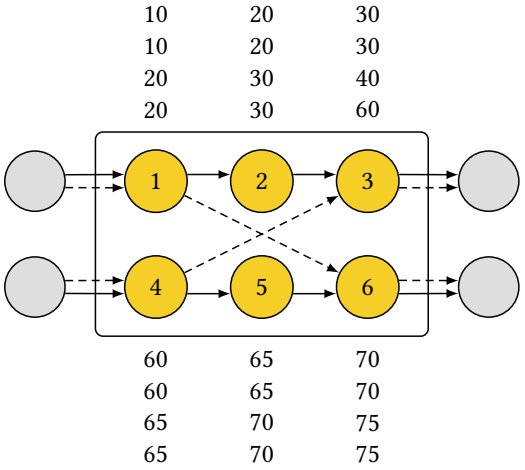
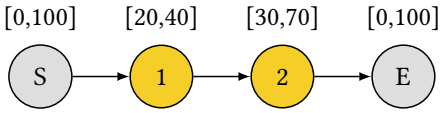


Figure 3.3: Example of two crews interchanging vehicles at a location. Vehicles are marked by solid lines and crews are marked by dashed lines. Each node has its arrival time, start of service, end of service and departure time labeled from top to bottom.

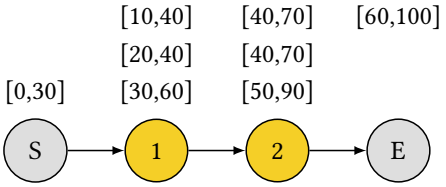
the second vehicle (at time 75). The second crew can move to the first vehicle which waits until time 60 to depart, even though the service for request 3 is completed at time 40.

The crew constraints described above permit many symmetrical solutions due to the numerous subpaths that a crew can travel on within a location. The specific requests that a crew visits at a particular location are not important; what is significant is the two vehicles that the crew uses to arrive at and depart from the location. In other words, there are many symmetrical solutions that differ only by the path that a crew takes within a location. These symmetries are best explained using Figure 3.3. It is possible for the first crew to enter the location at request 1, visit requests 2 to 5 in order then exit on the second vehicle at request 6. However, this path is equivalent, for all practical purposes, to one in which the crew moves directly from request 1 to request 6. Hence, the search space can be reduced by requiring crews to shortcut intermediate nodes within a location by visiting at most a single entry node and a single exit node per location. This is accomplished by ensuring that crews cannot visit a subsequence of three or more requests at the same location. This restriction allows for vehicle interchanges without considering all the symmetric solutions that have no impact on the objective.

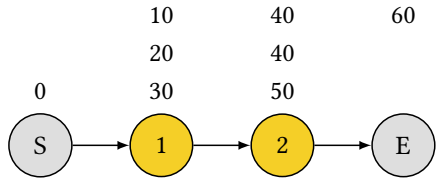
The JVCRSP features a temporal element that emerges from the interdependency between vehicle routes and crew routes. This time complexity is not present in traditional vehicle routing problems. Consider a route in the PDPTW, as shown in Figures 3.4a and 3.4b. Figures 3.4c and 3.4d show two different schedules for the same vehicle route. Delaying the departure times along one route does not impact other routes nor invalidate the solution (provided that



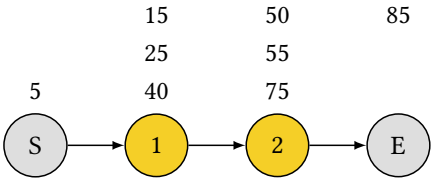
(a) A route consisting of two requests and the start and end depot nodes, and their time windows for the start of service. Every request requires 10 time units for service, and every arc requires 10 time units of travel time.



(b) Lower and upper bounds for the arrival time (top), service start time (middle) and departure time (bottom) at each request along the route can be easily computed by considering travel time and time window constraints.



(c) Since early arrival is permitted, arrival times, service times and departure times can be fixed to their earliest possible once a route is determined.



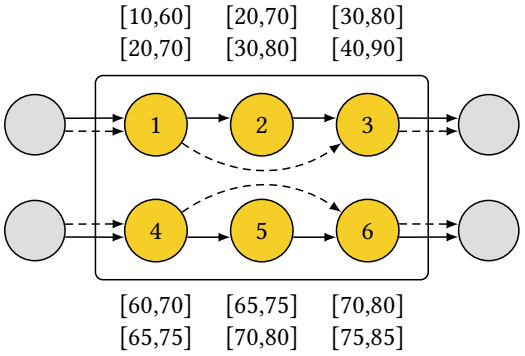
(d) Delaying service or departure along a route, although possible, is not beneficial since an amount of time is wasted, which could be better used to service other requests.

Figure 3.4: Example of two different schedules for the same route in classical vehicle routing problems without time synchronization.

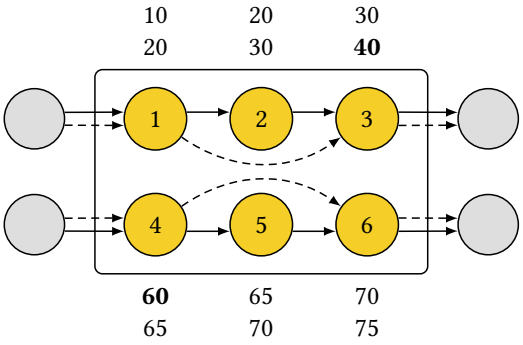
the delayed route satisfies the time windows). The time variables in classical vehicle routing problems are used solely to enforce the feasibility of the time windows. Solvers frequently use this knowledge to avoid branching on time variables by fixing all time variables to their earliest possible once all routes are determined. Because solvers only need to search on the arcs (i.e., space) variables and not the time variables, classical vehicle routing problems are said to possess one spatial degree of freedom and zero temporal degrees of freedom.

Figure 3.5 shows two crews that switch vehicles at a location. Delaying the top vehicle to depart at time 60, instead of departing immediately after service at time 40, allows the bottom crew to move onboard. This delay alters the departure times and can cause a cascade of events farther along this route and on other routes. For example, the delay may allow a crew to interchange vehicles at one location but prevent a crew from interchanging vehicles at another location. Hence, searching on the arrival and departure time variables is essential in the JVCRSP.

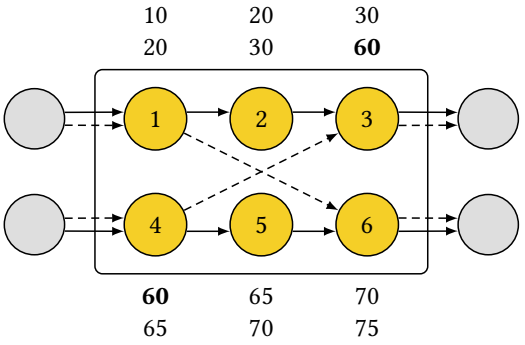
The JVCRSP has two spatial degrees of freedom because vehicles and crews can move independently in space. However, the JVCRSP has only one temporal degree of freedom because the crew time variables are tightly coupled to the vehicle time variables since the crews move with vehicles between locations and crews have no notion of time within locations. Hence, the crew time variables serve a similar purpose to the vehicle time variables in classical vehicle routing problems, such as the PDPTW, in that they exist only to ensure the feasibility of the maximum driving duration constraints.



(a) Consider two vehicles (solid lines) and two crews (dashed lines) that arrive at a location. Next to each request at the location is the lower and upper bounds for its arrival time (top) and departure time (bottom). Requests 1 to 3 require 10 units of time for service, and requests 4 to 6 require 5 units of time for service.



(b) Setting all time variables to the earliest possible will prohibit the bottom crew from changing to the top vehicle because the top vehicle departs the location (at time 40) before the arrival of the bottom vehicle and crew (at time 60).



(c) Delaying the departure of the top vehicle to time 60 allows both the top and bottom crews to exchange vehicles.

Figure 3.5: Example of the significance of branching on the time variables when vehicle routes are interdependent.

It is the temporal degree of freedom that makes the JVCRSP difficult, especially when the time horizon is large, because the solver must test many combinations of assignments to the time variables to determine if crews can interchange vehicles or if they have reached the maximum driving duration.

3.4 The Mixed Integer Programming Model

This section concretizes the high-level JVCRSP model into a mixed integer program.

The inputs and decision variables for the model are listed in Table 3.1. The problem is defined on a time interval $\mathcal{T} = [0, T]$, where $T > 0$ is the time horizon when all $V \in \{1, \dots, \infty\}$ vehicles and $C \in \{V, \dots, \infty\}$ crews must have completed their routes. The vehicles and crews are represented by the sets $\mathcal{V} = \{1, \dots, V\}$ and $\mathcal{C} = \{1, \dots, C\}$ respectively. Each vehicle has a load capacity of $Q \geq 0$ and each crew has a maximum driving duration of $\bar{T} \in \mathcal{T}$.

Assume that there are $P \in \{1, \dots, \infty\}$ pickup-delivery pairs, and hence, $2P$ requests in total. Define $\mathcal{P} = \{1, \dots, P\}$ and $\mathcal{D} = \{P + 1, \dots, 2P\}$ as the set of pickup nodes and delivery nodes respectively, and let $\mathcal{R} = \mathcal{P} \cup \mathcal{D}$ be the set of all request nodes. Every vehicle $v \in \mathcal{V}$ has a unique start node s_v and end node e_v. These are grouped in the sets $\mathcal{S} = \{s_1, \dots, s_V\}$ and $\mathcal{E} = \{e_1, \dots, e_V\}$. Also, define $\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$ to be all nodes that vehicles can visit. Finally, let s_{crew} and e_{crew} be the common crew start node and end node respectively.

Each vehicle can traverse arcs from its start node to any pickup, from any request to any other request, from deliveries to its end node, and if the vehicle is unused, from its start node to end node. For every vehicle $v \in \mathcal{V}$, define this set of arcs as

$$\mathcal{A}_v = \{(s_v, i) | i \in \mathcal{P}\} \cup \{(i, j) | i \in \mathcal{R}, j \in \mathcal{R}, i \neq j\} \cup \{(i, e_v) | i \in \mathcal{D}\} \cup \{(s_v, e_v)\}. \tag{3.1}$$

Crews can traverse all vehicle arcs except the arcs indicating that a vehicle is unused. Additionally, crews can traverse arcs from the crew start node to any vehicle start node, from any vehicle end node to the crew end node, and directly from the crew start node to the crew end node. Define the common crew arcs as

$$\mathcal{A} = \{(s_{\text{crew}}, i) | i \in \mathcal{S}\} \cup \bigcup_{v \in \mathcal{V}} \mathcal{A}_v \cup \{(i, e_{\text{crew}}) | i \in \mathcal{E}\} \cup \{(s_{\text{crew}}, e_{\text{crew}})\} \setminus \{(s_v, e_v) | v \in \mathcal{V}\}. \tag{3.2}$$

Define $\mathcal{L}$ as the set of locations, including one depot location. For every node $i \in \mathcal{N}$, let $l_i \in \mathcal{L}$ be its location, $a_i \in \mathcal{T}$ and $b_i \in \mathcal{T}$ be its earliest time and latest time to start service, $t_i \in \mathcal{T}$ be its service duration and $q_i \in [-Q, Q]$ be its load demand. For every arc $(i, j) \in \bigcup_{v \in \mathcal{V}} \mathcal{A}_v \cup \mathcal{A}$, define $d_{i,j} \in \mathcal{T}$ as the distance and travel time along the arc. Finally, let $w_1 > 0$ and $w_2 > 0$ be the cost of using one vehicle and one crew, and let $w_3 > 0$ and $w_4 > 0$ be the cost of one unit of distance traveled by a vehicle and by a crew.

The primary decision variables are the usual vehicle flow variables $\text{veh}_{v,i,j} \in \{0, 1\}$, which indicate whether vehicle $v \in \mathcal{V}$ traverses $(i, j) \in \mathcal{A}_v$. The variables $\text{arr}_{v,i} \in \mathcal{T}$, $\text{serv}_{v,i} \in [a_i, b_i]$ and $\text{dep}_{v,i} \in \mathcal{T}$ represent the arrival time, service start time and departure time of vehicle $v \in \mathcal{V}$ at node $i \in \mathcal{N}$. The $\text{load}_{v,i} \in [0, Q]$ variable contains the load of vehicle $v \in \mathcal{V}$ after it services

Name	Description
$T > 0$	Time horizon.
$\mathcal{T} = [0, T]$	Time interval.
$V \in \{1, \dots, \infty\}$	Number of vehicles.
$\mathcal{V} = \{1, \dots, V\}$	Set of vehicles.
$Q \geq 0$	Vehicle capacity.
$C \in \{V, \dots, \infty\}$	Number of crews.
$\mathcal{C} = \{1, \dots, C\}$	Set of crews.
$\bar{T} \in \mathcal{T}$	Crew maximum driving duration.
$P \in \{1, \dots, \infty\}$	Number of pickup-delivery pairs.
$\mathcal{P} = \{1, \dots, P\}$	Set of pickup nodes.
$\mathcal{D} = \{P + 1, \dots, 2P\}$	Set of delivery nodes.
$\mathcal{R} = \mathcal{P} \cup \mathcal{D}$	Set of all requests.
s_v	Start node of vehicle $v \in \mathcal{V}$.
e_v	End node of vehicle $v \in \mathcal{V}$.
$\mathcal{S} = \{s_1, \dots, s_V\}$	Set of vehicle start nodes.
$\mathcal{E} = \{e_1, \dots, e_V\}$	Set of vehicle end nodes.
$\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$	Set of all requests and vehicle start and end nodes.
s_{crew}	Start node of all crews.
e_{crew}	End node of all crews.
$\mathcal{A}_v$	Arcs that can be traversed by vehicle $v \in \mathcal{V}$. Defined in Equation (3.1).
$\mathcal{A}$	Arcs that can be traversed by crews. Defined in Equation (3.2).
$\mathcal{L}$	Set of locations, including one depot location.
$l_i \in \mathcal{L}$	Location of $i \in \mathcal{N}$.
$a_i \in \mathcal{T}$	Earliest start of service at $i \in \mathcal{N}$.
$b_i \in \mathcal{T}$	Latest start of service at $i \in \mathcal{N}$.
$t_i \in \mathcal{T}$	Service duration of $i \in \mathcal{N}$.
$q_i \in [-Q, Q]$	Load demand at $i \in \mathcal{N}$.
$d_{i,j} \in \mathcal{T}$	Distance and travel time along the arc $(i, j) \in \bigcup_{v \in \mathcal{V}} \mathcal{A}_v \cup \mathcal{A}$.
$w_1 > 0$	Cost of using one vehicle.
$w_2 > 0$	Cost of using one crew.
$w_3 > 0$	Cost of one unit of vehicle distance.
$w_4 > 0$	Cost of one unit of crew distance.
$\text{veh}_{v,i,j} \in \{0, 1\}$	Indicates if vehicle $v \in \mathcal{V}$ traverses $(i, j) \in \mathcal{A}_v$.
$\text{arr}_{v,i} \in \mathcal{T}$	Arrival time of vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{serv}_{v,i} \in [a_i, b_i]$	Start of service by vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{dep}_{v,i} \in \mathcal{T}$	Departure time of vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{load}_{v,i} \in [0, Q]$	Load of vehicle $v \in \mathcal{V}$ after servicing $i \in \mathcal{N}$.
$\text{crew}_{c,i,j} \in \{0, 1\}$	Indicates if crew $c \in \mathcal{C}$ traverses $(i, j) \in \mathcal{A}$.
$\text{crewTime}_{c,i} \in \mathcal{T}$	Time when crew $c \in \mathcal{C}$ is at $i \in \mathcal{N}$.
$\text{driver}_{c,i,j} \in \{0, 1\}$	Indicates if crew $c \in \mathcal{C}$ drives on $(i, j) \in \mathcal{A}$, $l_i \neq l_j$.
$\text{driveStart}_c \in \mathcal{T}$	Start time of driving for crew $c \in \mathcal{C}$.
$\text{driveEnd}_c \in \mathcal{T}$	End time of driving for crew $c \in \mathcal{C}$.
$\text{driveDur}_c \in [0, \bar{T}]$	Driving duration of crew $c \in \mathcal{C}$.

Table 3.1: The data and decision variables of the mixed integer programming model.

node $i \in \mathcal{N}$.

The secondary decision variables are the crew flow variables $\text{crew}_{c,i,j} \in \{0, 1\}$, which indicate whether crew $c \in \mathcal{C}$ traverses $(i, j) \in \mathcal{A}$. Variable $\text{crewTime}_{c,i} \in \mathcal{T}$ stores a moment when crew $c \in \mathcal{C}$ is present at node $i \in \mathcal{N}$. In our solutions, it will be either the arrival or the departure time at the node (or both if they are equal). The $\text{driver}_{c,i,j} \in \{0, 1\}$ variable indicates whether crew $c \in \mathcal{C}$ drives the vehicle that traverses $(i, j) \in \mathcal{A}$ if $l_i \neq l_j$. The start and end time of the driving segment of crew $c \in \mathcal{C}$ is given by $\text{driveStart}_c \in \mathcal{T}$ and $\text{driveEnd}_c \in \mathcal{T}$, and the total driving duration by $\text{driveDur}_c \in [0, \bar{T}]$.

The constraints of the mixed integer programming model are separated into a vehicle component and a crew component. The vehicle component, depicted in Figure 3.6, is the standard three-index flow model of the PDPTW with the addition of arrival and departure time variables and the duplication of the start and end node for each vehicle. Constraints (3.3) to (3.5) are the usual flow constraints, which ensure that each vehicle follows a path from its start node to its end node. Constraint (3.6) is the request cover constraint, which requires every (pickup) request to be visited. Constraints (3.7) and (3.8) are the pickup-delivery constraints, which ensure that delivery requests are serviced by the same vehicle that serviced their associated pickup request and are serviced after the associated pickup request. Constraints (3.9) and (3.10) order the arrival, service and departure times at each request. Constraint (3.11) constrains each start node and end node to one common arrival/service/departure time. Constraints (3.12) and (3.13) are the travel time constraints, which linearize the constraint

$$\text{veh}_{v,i,j} = 1 \longrightarrow \text{dep}_{v,i} + d_{i,j} = \text{arr}_{v,j} \qquad \forall v \in \mathcal{V}, (i, j) \in \mathcal{A}_v.$$

Constraints (3.14) to (3.16) bound the vehicle load after service of a request. Vehicle loads along a route are accumulated by Constraint (3.17). Constraint (3.18) is a redundant constraint that prunes the search space by allowing each vehicle to visit a request at most once. Constraint (3.19) is a redundant constraint that breaks symmetry between vehicles by forcing a vehicle to be unused if a lower-numbered vehicle is unused. M_1 to M_3 are big-M constants.

The crew component, depicted in Figure 3.7, overlays the vehicle component with crew constraints to obtain the JVCRSP. It contains routing constraints similar to those in the vehicle component but also includes synchronization constraints to couple the vehicles and crews. Constraints (3.20) to (3.22) are the crew flow constraints, which ensure that all crews follow a path beginning at the crew start node through to the crew end node. Constraints (3.23) and (3.24) are space synchronization constraints that require crews to move with a vehicle and vehicles to move with a driver onboard when moving from one location to another. Constraint (3.25) is another space synchronization constraint that restricts the driver along an arc to be one of the crews that traverses the arc. Constraint (3.26) is the crew travel time constraint. Constraint (3.27) is a time synchronization constraint that allows crews to be at a node only while a vehicle is present. Having bounds on the crew time variables, rather than strict equality, is essential to modeling vehicle interchange because it enables crews to both move off a vehicle when it arrives at a location and move on a vehicle when it departs a location. In an equality constraint, crews must either always move off a vehicle at arrival or at departure, which disallow some

$$\sum_{j:(s_v,j)\in\mathcal{A}_v} \text{veh}_{v,s_v,j} = 1 \qquad \forall v \in \mathcal{V}, \quad (3.3)$$

$$\sum_{h:(h,i)\in\mathcal{A}_v} \text{veh}_{v,h,i} = \sum_{j:(i,j)\in\mathcal{A}_v} \text{veh}_{v,i,j} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \quad (3.4)$$

$$\sum_{h:(h,e_v)\in\mathcal{A}_v} \text{veh}_{v,h,e_v} = 1 \qquad \forall v \in \mathcal{V}, \quad (3.5)$$

$$\sum_{v\in\mathcal{V}} \sum_{h:(h,i)\in\mathcal{A}_v} \text{veh}_{v,h,i} = 1 \qquad \forall i \in \mathcal{P}, \quad (3.6)$$

$$\sum_{h:(h,i)\in\mathcal{A}_v} \text{veh}_{v,h,i} = \sum_{h:(h,P+i)\in\mathcal{A}_v} \text{veh}_{v,h,P+i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \quad (3.7)$$

$$\text{dep}_{v,i} + d_{i,P+i} \le \text{arr}_{v,P+i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \quad (3.8)$$

$$\text{arr}_{v,i} \le \text{serv}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \quad (3.9)$$

$$\text{serv}_{v,i} + t_i \le \text{dep}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \quad (3.10)$$

$$\text{arr}_{v,i} = \text{serv}_{v,i} = \text{dep}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{S} \cup \mathcal{E}, \quad (3.11)$$

$$\text{dep}_{v,i} + d_{i,j} - \text{arr}_{v,j} \le M_1 \left(1 - \text{veh}_{v,i,j}\right) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \quad (3.12)$$

$$\text{arr}_{v,j} - \text{dep}_{v,i} - d_{i,j} \le M_2 \left(1 - \text{veh}_{v,i,j}\right) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \quad (3.13)$$

$$\text{load}_{v,i} = 0 \qquad \forall v \in \mathcal{V}, i \in \mathcal{S} \cup \mathcal{E}, \quad (3.14)$$

$$q_i \le \text{load}_{v,i} \le Q \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \quad (3.15)$$

$$0 \le \text{load}_{v,i} \le Q + q_i \qquad \forall v \in \mathcal{V}, i \in \mathcal{D}, \quad (3.16)$$

$$\text{load}_{v,i} + q_j - \text{load}_{v,j} \le M_3 \left(1 - \text{veh}_{v,i,j}\right) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \quad (3.17)$$

$$\sum_{h:(h,i)\in\mathcal{A}_v} \text{veh}_{v,h,i} \le 1 \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \quad (3.18)$$

$$\text{veh}_{v,s_v,e_v} \le \text{veh}_{v+1,s_{v+1},e_{v+1}} \qquad \forall v \in \{1,\dots,V-1\}. \quad (3.19)$$

Figure 3.6: The vehicle component of the mixed integer programming model.

$$\sum_{j:(s_{\text{crew}},j)\in\mathcal{A}} \text{crew}_{c,s_{\text{crew}},j} = 1 \qquad \forall c \in \mathcal{C}, \quad (3.20)$$

$$\sum_{h:(h,i)\in\mathcal{A}} \text{crew}_{c,h,i} = \sum_{j:(i,j)\in\mathcal{A}} \text{crew}_{c,i,j} \qquad \forall c \in \mathcal{C}, i \in \mathcal{N}, \quad (3.21)$$

$$\sum_{h:(h,e_{\text{crew}})\in\mathcal{A}} \text{crew}_{c,h,e_{\text{crew}}} = 1 \qquad \forall c \in \mathcal{C}, \quad (3.22)$$

$$\text{crew}_{c,i,j} \leq \sum_{v\in\mathcal{V}:(i,j)\in\mathcal{A}_v} \text{veh}_{v,i,j} \qquad \forall c \in \mathcal{C}, (i,j) \in \mathcal{A}, l_i \neq l_j, \quad (3.23)$$

$$\sum_{v\in\mathcal{V}:(i,j)\in\mathcal{A}_v} \text{veh}_{v,i,j} = \sum_{c\in\mathcal{C}} \text{driver}_{c,i,j} \qquad \forall (i,j) \in \mathcal{A}, l_i \neq l_j, \quad (3.24)$$

$$\text{driver}_{c,i,j} \leq \text{crew}_{c,i,j} \qquad \forall c \in \mathcal{C}, (i,j) \in \mathcal{A}, l_i \neq l_j, \quad (3.25)$$

$$\text{crewTime}_{c,i} + d_{i,j} - \text{crewTime}_{c,j} \leq M_4\left(1 - \text{crew}_{c,i,j}\right) \qquad \forall c \in \mathcal{C}, (i,j) \in \mathcal{A}, \quad (3.26)$$

$$\text{arr}_{v,i} \leq \text{crewTime}_{c,i} \leq \text{dep}_{v,i} \qquad \forall v \in \mathcal{V}, c \in \mathcal{C}, i \in \mathcal{N}, \quad (3.27)$$

$$\text{driveStart}_c - \text{crewTime}_{c,i} \leq M_5\left(1 - \sum_{j:(i,j)\in\mathcal{A}} \text{driver}_{c,i,j}\right) \qquad \forall c \in \mathcal{C}, i \in \mathcal{R} \cup \mathcal{S}, \quad (3.28)$$

$$\text{crewTime}_{c,i} - \text{driveEnd}_c \leq M_6\left(1 - \sum_{h:(h,i)\in\mathcal{A}} \text{driver}_{c,h,i}\right) \qquad \forall c \in \mathcal{C}, i \in \mathcal{R} \cup \mathcal{E}, \quad (3.29)$$

$$\text{driveDur}_c = \text{driveEnd}_c - \text{driveStart}_c \qquad \forall c \in \mathcal{C}, \quad (3.30)$$

$$\sum_{h:(h,i)\in\mathcal{A}} \text{crew}_{c,h,i} \leq 1 \qquad \forall c \in \mathcal{C}, i \in \mathcal{N}, \quad (3.31)$$

$$\text{crew}_{c,s_{\text{crew}},e_{\text{crew}}} \leq \text{crew}_{c+1,s_{\text{crew}},e_{\text{crew}}} \qquad \forall c \in \{1,\ldots,C-1\}, \quad (3.32)$$

$$\text{crew}_{c,i,j} + \text{crew}_{c,j,k} \leq 1 \qquad \forall c \in \mathcal{C}, (i,j,k) : (i,j) \in \mathcal{A}, (j,k) \in \mathcal{A}, l_i = l_j = l_k, \quad (3.33)$$

$$\sum_{(i,j)\in\mathcal{A}:i,j\in S} \text{crew}_{c,i,j} \leq |S| - 1 \qquad \forall c \in \mathcal{C}, S \subseteq \mathcal{N}. \quad (3.34)$$

Figure 3.7: The crew component of the mixed integer programming model.

interchanges. Constraints (3.28) and (3.29) determine the start and end of driving of each crew. Constraint (3.28) is a linearization of the constraint

$$\sum_{j:(i,j)\in\mathcal{A}} \text{driver}_{c,i,j} = 1 \longrightarrow \text{driveStart}_c \leq \text{crewTime}_{c,i} \qquad \forall c \in \mathcal{C}, i \in \mathcal{R} \cup \mathcal{S},$$

which imposes that, if a driver departs the node i, the driver must have already started driving before or at the departure time at i. Similarly, Constraint (3.29) states that if a crew drives to node i, the end of driving of the crew must be later than or at the arrival time of i. Constraint (3.30) calculates the driving duration of each crew. Constraints (3.31) and (3.32) are redundant constraints and are equivalent to Constraints (3.18) and (3.19). Constraint (3.33) prevents crews from visiting subsequences of three or more requests at the same location, as explained in Section 3.3. M_4 to M_6 are big-M constants.

When a crew travels between two requests i and j within a location, the distance and travel time is zero ($d_{i,j} = 0$), and hence, Constraint (3.26) fails to perform subtour elimination. There are two possible remedies. The first option replaces $d_{i,j}$ in Constraint (3.26) with a new crew travel time cost that has a positive value when traveling between two requests within a location. This value can be interpreted as the time required for a crew to switch vehicles. The alternative option is to use the subtour elimination constraints specified by Constraint (3.34). Our algorithm uses the second approach but adds these constraints lazily in a branch-and-cut scheme.

Objective Function (3.35) minimizes a weighted sum of the number of vehicles and crews used and the total vehicle and crew travel distances.

$$\begin{aligned}\min\ & w_1 \sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{P}} \text{veh}_{v,s_v,j} + w_2 \sum_{c\in\mathcal{C}} \sum_{j\in\mathcal{S}} \text{crew}_{c,s_{\text{crew}},j} + \\ & w_3 \sum_{v\in\mathcal{V}} \sum_{(i,j)\in\mathcal{A}_v} d_{i,j}\text{veh}_{v,i,j} + w_4 \sum_{c\in\mathcal{C}} \sum_{(i,j)\in\mathcal{A}} d_{i,j}\text{crew}_{c,i,j}.\end{aligned} \tag{3.35}$$

3.5 The Constraint Programming Model

This section discusses the constraint programming formulation of the high-level model of the JVCRSP.

The inputs and decision variables of the constraint programming model are listed in Table 3.2. The definitions of many of the variables are identical to those in the mixed integer programming model but differ in that some sets are discrete instead of continuous. The problem is defined on a discrete time interval $\mathcal{T} = \{0, \dots, T\}$, where $T \in \{1, \dots, \infty\}$ is the time horizon. Let $V \in \{1, \dots, \infty\}$ be the number of vehicles, and $Q \in \{0, \dots, \infty\}$ be the load capacity of each vehicle. Let $C \in \{V, \dots, \infty\}$ be the number of crews, and $\bar{T} \in \mathcal{T}$ be the maximum driving duration. The vehicles and crews are represented by the sets $\mathcal{V} = \{1, \dots, V\}$ and $\mathcal{C} = \{1, \dots, C\}$ respectively. The set $\mathcal{C}_0 = \mathcal{C} \cup \{0\}$ extends the set of crews with a dummy value 0 that indicates no crew.

The problem has $P \in \{1, \dots, \infty\}$ pickup-delivery pairs, giving a total of $R = 2P$ requests. Define $\mathcal{P} = \{1, \dots, P\}$ and $\mathcal{D} = \{P + 1, \dots, R\}$ as the set of pickups and deliveries respectively, and group them in the set $\mathcal{R} = \mathcal{P} \cup \mathcal{D}$. For every vehicle $v \in \mathcal{V}$, define its unique start and end node as $s(v) = R + v$ and $e(v) = R + V + v$. The start and end nodes are grouped in $\mathcal{S} = \{R + 1, \dots, R + V\}$

Name	Description
$T \in \{1, \dots, \infty\}$	Time horizon.
$\mathcal{T} = \{0, \dots, T\}$	Time interval.
$V \in \{1, \dots, \infty\}$	Number of vehicles.
$\mathcal{V} = \{1, \dots, V\}$	Set of vehicles.
$Q \in \{0, \dots, \infty\}$	Vehicle capacity.
$C \in \{V, \dots, \infty\}$	Number of crews.
$\mathcal{C} = \{1, \dots, C\}$	Set of crews.
$\mathcal{C}_0 = \mathcal{C} \cup \{0\}$	Set of crews, including a 0 value indicating no crew.
$\bar{T} \in \mathcal{T}$	Maximum driving duration of a crew.
$P \in \{1, \dots, \infty\}$	Number of pickup-delivery pairs.
$R = 2P$	Number of requests.
$\mathcal{P} = \{1, \dots, P\}$	Set of pickup nodes.
$\mathcal{D} = \{P+1, \dots, R\}$	Set of delivery nodes.
$\mathcal{R} = \mathcal{P} \cup \mathcal{D}$	Set of all requests.
$s(v) = R + v$	Start node of vehicle $v \in \mathcal{V}$.
$e(v) = R + V + v$	End node of vehicle $v \in \mathcal{V}$.
$\mathcal{S} = \{R+1, \dots, R+V\}$	Set of vehicle start nodes.
$\mathcal{E} = \{R+V+1, \dots, R+2V\}$	Set of vehicle end nodes.
$\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$	Set of all requests and vehicle start and end nodes.
$\mathcal{N}_0 = \mathcal{N} \cup \{0\}$	Set of all nodes, including the crew depot node 0.
$\mathcal{L}$	Set of locations, including one depot location.
$l(i) \in \mathcal{L}$	Location of $i \in \mathcal{N}$.
$a(i) \in \mathcal{T}$	Earliest start of service at $i \in \mathcal{N}$.
$b(i) \in \mathcal{T}$	Latest start of service at $i \in \mathcal{N}$.
$t(i) \in \{1, \dots, \infty\}$	Service duration of $i \in \mathcal{N}$.
$q(i) \in \{-Q, \dots, Q\}$	Load demand at $i \in \mathcal{N}$.
$d(i,j) \in \mathcal{T}$	Distance and travel time from $i \in \mathcal{N}$ to $j \in \mathcal{N}$.
$w_1 \in \{1, \dots, \infty\}$	Cost of using one vehicle.
$w_2 \in \{1, \dots, \infty\}$	Cost of using one crew.
$w_3 \in \{1, \dots, \infty\}$	Cost of one unit of vehicle distance.
$w_4 \in \{1, \dots, \infty\}$	Cost of one unit of crew distance.
succ$(i) \in \mathcal{N}$	Successor of $i \in \mathcal{N}$.
veh$(i) \in \mathcal{V}$	Vehicle that visits $i \in \mathcal{N}$.
arr$(i) \in \mathcal{T}$	Arrival time at $i \in \mathcal{N}$.
serv$(i) \in \{a(i), \dots, b(i)\}$	Start of service at $i \in \mathcal{N}$.
dep$(i) \in \mathcal{T}$	Departure time at $i \in \mathcal{N}$.
load$(i) \in \{0, \dots, Q\}$	Load of vehicle veh(i) after servicing $i \in \mathcal{N}$.
vehUsed$(v) \in \{0, 1\}$	Indicates if vehicle $v \in \mathcal{V}$ is used.
crewSucc$(c, i) \in \mathcal{N}_0$	Successor of $i \in \mathcal{N}_0$ for crew $c \in \mathcal{C}_0$.
crewTime$(i) \in \mathcal{T}$	Time when all crews that visit $i \in \mathcal{N}$ is present at i.
crewUsed$(c) \in \{0, 1\}$	Indicates if crew $c \in \mathcal{C}$ is used.
crewDist$(c) \in \{0, \dots, \infty\}$	Distance traveled by crew $c \in \mathcal{C}$.
driver$(i) \in \mathcal{C}_0$	Driver of vehicle veh(i) from $i \in \mathcal{R} \cup \mathcal{S}$ to succ(i), with the value 0 indicating no driver.
driveStart$(c) \in \mathcal{T}$	Start time of driving for crew $c \in \mathcal{C}_0$.
driveEnd$(c) \in \mathcal{T}$	End time of driving for crew $c \in \mathcal{C}_0$.
driveDur$(c) \in \{0, \dots, \bar{T}\}$	Driving duration of crew $c \in \mathcal{C}$.

Table 3.2: The data and decision variables of the constraint programming model.

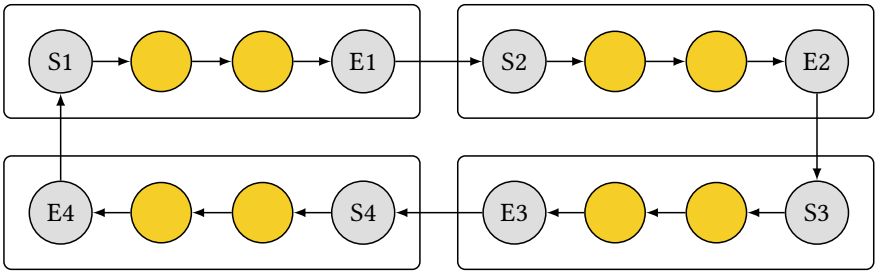


Figure 3.8: Example of four vehicle routes as modeled by successor variables. The S nodes and E nodes respectively are the start nodes and end nodes of the four vehicles.

and $\mathcal{E} = \{R + V + 1, \ldots, R + 2V\}$. Let $\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$ be the set of all nodes, and $\mathcal{N}_0 = \mathcal{N} \cup \{0\}$ be the set of all nodes plus the combined crew start and end depot node 0.

Define $\mathcal{L}$ as the set of locations, including one depot location. For every node $i \in \mathcal{N}$, define $l(i) \in \mathcal{L}$ as its location, $a(i) \in \mathcal{T}$ and $b(i) \in \mathcal{T}$ as the opening and closing of its time window, $t(i) \in \{1, \ldots, \infty\}$ as its service duration, and $q(i) \in \{-Q, \ldots, Q\}$ as its load demand. Let $d(i, j) \in \mathcal{T}$ be the distance and travel time from $i \in \mathcal{N}$ to $j \in \mathcal{N}$.

Let $w_1 \in \{1, \ldots, \infty\}$ and $w_2 \in \{1, \ldots, \infty\}$ be the cost of using one vehicle and one crew, and let $w_3 \in \{1, \ldots, \infty\}$ and $w_4 \in \{1, \ldots, \infty\}$ be the cost of one unit of distance traveled by a vehicle and by a crew.

The primary decision variables are the vehicle successor variables. Successor variables are frequently seen in constraint programming models of vehicle routing problems (e.g., Kilby, Prosser and Shaw 2000, Rousseau, Gendreau and Pesant 2002) and serve the same purpose as the flow variables in the mixed integer programming model. For every node $i \in \mathcal{N}$, $\text{succ}(i) \in \mathcal{N}$ denotes the direct successor of i on its route. For example, if $\text{succ}(i) = j$, then the arc (i, j) is used. The successor of a vehicle's end node is the start node of the following vehicle, and the successor of the last vehicle's end node is the start node of the first vehicle. The time and load resources are accumulated along a route and then reset at an end node prior to the start of the next route. Under this modeling, the successor variables describe a Hamiltonian cycle. Figure 3.8 shows an example of the Hamiltonian cycle formed by four vehicle routes. This modeling was developed by Christofides and Eilon (1969) and subsequently called the *giant tour* representation (e.g., Irnich 2008).

Since the successor variables are not indexed by vehicle, the model uses the $\text{veh}(i) \in \mathcal{V}$ variable to store the vehicle that visits node $i \in \mathcal{N}$. This modeling, which uses only two vectors, is more succinct than the three-dimensional flow variables found in the mixed integer programming model.

Variables $\text{arr}(i) \in \mathcal{T}$, $\text{serv}(i) \in \{a(i), \ldots, b(i)\}$ and $\text{dep}(i) \in \mathcal{T}$ represent the arrival time, service start time and departure time at node $i \in \mathcal{N}$. The load after servicing node $i \in \mathcal{N}$ is stored in $\text{load}(i) \in \{0, \ldots, Q\}$. Variable $\text{vehUsed}(v) \in \{0, 1\}$ indicates whether vehicle $v \in \mathcal{V}$ is used.

There is a significant difference between vehicles and crews in terms of routing. For vehicles, every request is visited exactly once, which allows the use of a single set of successor variables. In contrast, multiple crews can be on a vehicle when it visits a node, and hence, it is not

possible to associate a single crew successor variable with every node. Instead, the model needs a crew successor variable for every crew and every node. However, for any given crew, its successor variables do not need to cover all requests. The crew successor variables are a matrix $\text{crewSucc}(c, i) \in \mathcal{N}_0$ that stores the immediate successor of node $i \in \mathcal{N}_0$ for crew $c \in \mathcal{C}_0$. If crew c does not visit a node i, then $\text{crewSucc}(c, i) = i$. The $\text{crewTime}(i) \in \mathcal{T}$ variable stores a moment when every crew that visits $i \in \mathcal{N}$ is present at i. For every crew $c \in \mathcal{C}$, variable $\text{crewUsed}(c) \in \{0, 1\}$ indicates whether the crew is used, and $\text{crewDist}(c) \in \{0, \dots, \infty\}$ stores the distance traveled by the crew.

Every node $i \in \mathcal{R} \cup \mathcal{S}$ has an associated $\text{driver}(i) \in \mathcal{C}_0$ variable that either denotes the driver of vehicle $\text{veh}(i)$ if the vehicle travels from i to its successor $\text{succ}(i)$ at a different location, or takes the value 0, indicating that no driver is necessary, if the successor is at the same location. The variables $\text{driveStart}(c) \in \mathcal{T}$ and $\text{driveEnd}(c) \in \mathcal{T}$ store the start and end time of driving of crew $c \in \mathcal{C}_0$, and $\text{driveDur}(c) \in \{0, \dots, \bar{T}\}$ stores the total driving duration of crew $c \in \mathcal{C}$.

Like the mixed integer programming model, the constraints of the constraint programming model are also divided into a vehicle component and a crew component. The vehicle component, depicted in Figure 3.9, models a PDPTW. Constraints (3.36) to (3.38) restrict the possible values of the successor variables. Constraint (3.36) states that a vehicle can only move from its start node to any pickup node or its end node. Constraint (3.37) states that a vehicle can move from a pickup node to any pickup or delivery node, and Constraint (3.38) states that a vehicle can move from a delivery node to any pickup, delivery or end node. Constraints (3.39) and (3.40) join the end nodes to the start nodes. Using the giant tour modeling, the Circuit global constraint from Constraint (3.41) performs subtour elimination by imposing a Hamiltonian cycle through the $\text{succ}(\cdot)$ variables. Constraints (3.42) and (3.43) allow only the associated vehicle to visit the start nodes and end nodes. Constraint (3.43) is needed to prevent vehicles from visiting the end node of other vehicles, which is permitted by Constraint (3.38). Constraint (3.44) tracks vehicles along their routes. Constraints (3.45) and (3.46) are the pickup and delivery constraints. Constraints (3.47) and (3.48) order the arrival, service and departure times at each request. Constraint (3.49) restricts each start and end node to one common arrival/service/departure time. Constraint (3.50) enforces travel times. Constraints (3.51) to (3.53) bound the vehicle loads. Constraint (3.54) is the load constraint. Constraints (3.55) and (3.56) state that a vehicle is used if and only if it does not travel from its start node to its end node or if it visits any request. Since they are equivalences, only one of these constraints is necessary but, in practice, stating the two constraints achieves stronger propagation. Constraint (3.57) breaks vehicle symmetry in a manner similar to Constraint (3.19).

The crew component, depicted in Figure 3.10, overlays the vehicle component with crew decisions. Constraints (3.58) to (3.62) are the domain restrictions. Constraint (3.58) requires crews to either leave the crew depot node for a vehicle start node or remain at the crew depot node. Constraints (3.59) to (3.61) are similar to Constraints (3.36) to (3.38) with the exception that the successor of a node is itself if it is not visited. Constraint (3.62) states that a crew either moves from a vehicle end node to the crew depot node or the crew does not visit the end node. The Subcircuit global constraint of Constraint (3.63) enforces connectivity and eliminates

$$\text{succ}(s(v)) \in \mathcal{P} \cup \{e(v)\} \qquad \forall v \in \mathcal{V}, \quad (3.36)$$

$$\text{succ}(i) \in \mathcal{P} \cup \mathcal{D} \qquad \forall i \in \mathcal{P}, \quad (3.37)$$

$$\text{succ}(i) \in \mathcal{P} \cup \mathcal{D} \cup \mathcal{E} \qquad \forall i \in \mathcal{D}, \quad (3.38)$$

$$\text{succ}(e(v)) = s(v+1) \qquad \forall v \in \{1, \dots, V-1\}, \quad (3.39)$$

$$\text{succ}(e(V)) = s(1), \quad (3.40)$$

$$\textsc{Circuit}(\text{succ}(\cdot)), \quad (3.41)$$

$$\text{veh}(s(v)) = v \qquad \forall v \in \mathcal{V}, \quad (3.42)$$

$$\text{veh}(e(v)) = v \qquad \forall v \in \mathcal{V}, \quad (3.43)$$

$$\text{veh}(\text{succ}(i)) = \text{veh}(i) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.44)$$

$$\text{veh}(i) = \text{veh}(P+i) \qquad \forall i \in \mathcal{P}, \quad (3.45)$$

$$\text{dep}(i) + d(i, P+i) \leq \text{arr}(P+i) \qquad \forall i \in \mathcal{P}, \quad (3.46)$$

$$\text{arr}(i) \leq \text{serv}(i) \qquad \forall i \in \mathcal{R}, \quad (3.47)$$

$$\text{serv}(i) + t(i) \leq \text{dep}(i) \qquad \forall i \in \mathcal{R}, \quad (3.48)$$

$$\text{arr}(i) = \text{serv}(i) = \text{dep}(i) \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \quad (3.49)$$

$$\text{dep}(i) + d(i, \text{succ}(i)) = \text{arr}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.50)$$

$$\text{load}(i) = 0 \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \quad (3.51)$$

$$q(i) \leq \text{load}(i) \leq Q \qquad \forall i \in \mathcal{P}, \quad (3.52)$$

$$0 \leq \text{load}(i) \leq Q + q(i) \qquad \forall i \in \mathcal{D}, \quad (3.53)$$

$$\text{load}(i) + q(\text{succ}(i)) = \text{load}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.54)$$

$$\text{vehUsed}(v) \leftrightarrow \text{succ}(s(v)) \neq e(v) \qquad \forall v \in \mathcal{V}, \quad (3.55)$$

$$\text{vehUsed}(v) \leftrightarrow \bigvee_{i \in \mathcal{R}} \text{veh}(i) = v \qquad \forall v \in \mathcal{V}, \quad (3.56)$$

$$\text{vehUsed}(v) \geq \text{vehUsed}(v+1) \qquad \forall v \in \{1, \dots, V-1\}. \quad (3.57)$$

Figure 3.9: The vehicle component of the constraint programming model.

$$\text{crewSucc}(c, 0) \in \mathcal{S} \cup \{0\} \qquad \forall c \in \mathcal{C}, \quad (3.58)$$

$$\text{crewSucc}(c, i) \in \mathcal{P} \cup \{i\} \qquad \forall c \in \mathcal{C}, i \in \mathcal{S}, \quad (3.59)$$

$$\text{crewSucc}(c, i) \in \mathcal{P} \cup \mathcal{D} \qquad \forall c \in \mathcal{C}, i \in \mathcal{P}, \quad (3.60)$$

$$\text{crewSucc}(c, i) \in \mathcal{P} \cup \mathcal{D} \cup \mathcal{E} \qquad \forall c \in \mathcal{C}, i \in \mathcal{D}, \quad (3.61)$$

$$\text{crewSucc}(c, i) \in \{0, i\} \qquad \forall c \in \mathcal{C}, i \in \mathcal{E}, \quad (3.62)$$

$$\textsc{Subcircuit}(\text{crewSucc}(c, \cdot)) \qquad \forall c \in \mathcal{C}, \quad (3.63)$$

$$l(\text{succ}(i)) = l(i) \leftrightarrow \text{driver}(i) = 0 \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.64)$$

$$\text{crewSucc}(\text{driver}(i), i) = \text{succ}(i) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.65)$$

$$l(\text{crewSucc}(c, i)) = l(i) \vee \text{crewSucc}(c, i) = \text{succ}(i) \qquad \forall c \in \mathcal{C}, i \in \mathcal{R} \cup \mathcal{S}, \quad (3.66)$$

$$\textsc{CrewShortcut}(\text{succ}(\cdot), \text{crewSucc}(\cdot, \cdot), \text{arr}(\cdot), \text{dep}(\cdot), \mathcal{C}, \mathcal{R}, \mathcal{S}, \mathcal{E}, l(\cdot)), \quad (3.67)$$

$$\text{arr}(i) \leq \text{crewTime}(i) \leq \text{dep}(i) \qquad \forall i \in \mathcal{N}, \quad (3.68)$$

$$\text{crewTime}(i) \leq \text{crewTime}(\text{crewSucc}(c, i)) \qquad \forall c \in \mathcal{C}, i \in \mathcal{R} \cup \mathcal{S}, \quad (3.69)$$

$$\text{driveStart}(\text{driver}(i)) \leq \text{dep}(i) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.70)$$

$$\text{driveEnd}(\text{driver}(i)) \geq \text{arr}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (3.71)$$

$$\text{driveDur}(c) = \text{driveEnd}(c) - \text{driveStart}(c) \qquad \forall c \in \mathcal{C}, \quad (3.72)$$

$$\text{crewUsed}(c) \leftrightarrow \bigvee_{i \in \mathcal{N}_0} \text{crewSucc}(c, i) \neq i \qquad \forall c \in \mathcal{C}, \quad (3.73)$$

$$\text{crewUsed}(c) \leftrightarrow \bigvee_{i \in \mathcal{R} \cup \mathcal{S}} \text{driver}(i) = c \qquad \forall c \in \mathcal{C}, \quad (3.74)$$

$$\text{crewUsed}(c) \geq \text{crewUsed}(c + 1) \qquad \forall c \in \{1, \dots, C - 1\}, \quad (3.75)$$

$$\textsc{CrewBound}(\text{crewDist}(c), \text{crewSucc}(c, \cdot), \text{crewTime}(\cdot), \mathcal{R}, \mathcal{S}, \mathcal{E}, d(\cdot, \cdot)) \qquad \forall c \in \mathcal{C}. \quad (3.76)$$

Figure 3.10: The crew component of the constraint programming model.

subtours (Francis and Stuckey 2014). It differs from the CIRCUIT constraint seen in the vehicle component by allowing some nodes to be excluded from the Hamiltonian cycle.

Constraint (3.64) states that vehicles have no driver when moving within a location. Constraint (3.65) requires drivers to move with their vehicles. Constraint (3.66) requires crews to either move to another node at their current location or move with a vehicle (to a different location). The CREWSHORTCUT global constraint in Constraint (3.67) removes crew subsequence symmetries within locations, and is detailed in Section 3.5.1. Constraint (3.68) allows crews to be at a node only while a vehicle is present. Constraint (3.69) forces crews to move forward in time only. Constraint (3.70) states that the driver on the arc $(i, \text{succ}(i))$ starts driving at or before departing i. Similarly, Constraint (3.71) states that the driver on the arc $(i, \text{succ}(i))$ ends driving at or after arriving at $\text{succ}(i)$. Constraint (3.72) calculates the total driving duration of each crew. Constraints (3.73) and (3.74) state that a crew is used if and only if it visits at least one node or if it drives along any arc. Constraint (3.75) is a symmetry-breaking constraint. Constraint (3.76) is a global optimization constraint, described in Section 3.5.2, that bounds crew distances and checks whether crews can return to the depot.

Objective Function (3.77) minimizes a weighted sum of the vehicle and crew counts and the total vehicle and crew travel distances.

$$\min w_1 \sum_{v \in \mathcal{V}} \text{vehUsed}(v) + w_2 \sum_{c \in \mathcal{C}} \text{crewUsed}(c) + w_3 \sum_{i \in \mathcal{R} \cup \mathcal{S}} d(i, \text{succ}(i)) + w_4 \sum_{c \in \mathcal{C}} \text{crewDist}(c). \quad (3.77)$$

3.5.1 Breaking Crew Subpath Symmetries within Locations

The CREWSHORTCUT global constraint of Constraint (3.67) removes symmetric subsequences of requests within locations, which are previously explained in Section 3.3. Define $\text{pred}(i) \in \mathcal{N}$ as the vehicle predecessor of the node $i \in \mathcal{N}$, and $\text{crewPred}(c, i) \in \mathcal{N}_0$ as the predecessor of $i \in \mathcal{N}_0$ for crew $c \in \mathcal{C}_0$. CREWSHORTCUT implements the following two propagation rules:

$$l(\text{pred}(i)) = l(i) = l(\text{succ}(i)) \leftrightarrow \bigwedge_{c \in \mathcal{C}} \text{crewSucc}(c, i) = i \qquad \forall i \in \mathcal{R}, \quad (3.78)$$

$$l(i) = l(\text{crewSucc}(c, i)) \wedge \text{crewSucc}(c, i) \neq i \longrightarrow$$

$$l(i) \neq l(\text{crewSucc}(c, \text{crewSucc}(c, i))) \wedge l(i) \neq l(\text{crewPred}(c, i)) \qquad \forall c \in \mathcal{C}, i \in \mathcal{R}. \quad (3.79)$$

Rule (3.78) states that if a vehicle visits a sequence of three requests at the same location, then crews cannot visit the second request of the sequence. Rule (3.79) states that if a crew visits a node i and then another node at the same location, then the crew cannot visit a third node at the same location and the crew must reach i from a different location.

3.5.2 Feasibility and Bounding of Crew Routes

The constraint programming model presented so far has fewer variables than the mixed integer programming model and prunes infeasible solutions effectively. However, removing infeasible arcs by pruning values from the domains of successor variables may have little or no impact on the lower bound of the objective value. This limitation of constraint programming can be

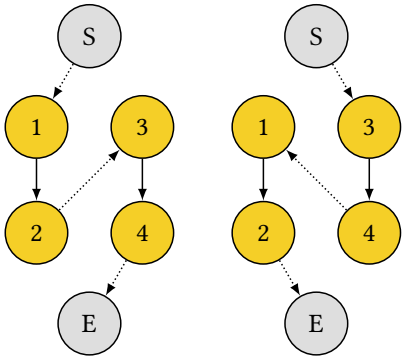


Figure 3.11: Example of a partial crew route obtained at an early stage of the search. The S and E nodes are the start node and end node respectively. Solid arrows represent crew assignments and dotted arrows represent possible crew assignments.

addressed using global optimization constraints, which are previously introduced in Section 2.6.1. This section presents a global optimization constraint that checks the feasibility of the crew partial routes and computes lower bounds to the crew objectives.

The search procedure of the constraint programming model branches on the vehicle successor variables before the crew successor variables. This means that, while searching for vehicle routes, the domains of the crew successor variables are large, leading to weak lower bounds to the crew distance terms in the objective function. For example, Figure 3.11 shows a partial crew route found in an early stage of the search when the focus is on vehicle routes. The crew is allocated to traverse the arcs $(1, 2)$ and $(3, 4)$, but it is not yet known if a path from the start node S to the end node E via these two arcs exists, and how long this path is if it exists.

A global optimization constraint is used to determine whether each crew has a feasible route and to compute lower bounds to the crew distance. Such a constraint needs to find, for every crew, a shortest path from a start node to an end node that includes all arcs known to be traversed by the crew. Since this problem is NP-hard (Dreyfus 1969, Ibaraki 1973, Laporte, Mercure and Norbert 1984, Volgenant and Jonker 1987), the CrewBound constraint, presented below, uses its linear relaxation.

Whenever a crewSucc$(c, \cdot)$ variable is fixed, the CrewBound constraint solves the linear program defined in Figure 3.12 for the crew $c \in \mathcal{C}$. The inputs to this linear program include four sets extracted from the current domains of the variables in the main constraint programming model. Let the $D(\cdot)$ function denote the current domain of a variable, then the four input sets are

- cSucc$(i) = D(\text{crewSucc}(c, i))$ for $i \in \mathcal{R} \cup \mathcal{S}$,
- cTime$(i) = D(\text{crewTime}(i))$ for $i \in \mathcal{N}$,
- cDist $= D(\text{crewDist}(c))$, and
- $\mathcal{B} = \{(i, j) | i \in \mathcal{R} \cup \mathcal{S}, j \in \text{cSucc}(i), i \neq j\}$, which represents the current set of arcs that can be traversed by the crew.

The $x_{i,j}$ variable indicates whether the crew traverses arc $(i, j) \in \mathcal{B}$, and the t_i variable stores the arrival time to node i.

$$\min \sum_{(i,j)\in\mathcal{B}} d(i,j)x_{i,j} \tag{3.80}$$

subject to

$$\min(\text{cDist}) \le \sum_{(i,j)\in\mathcal{B}} d(i,j)x_{i,j} \le \max(\text{cDist}), \tag{3.81}$$

$$\sum_{(i,j)\in\mathcal{B}\,:\,i\in\mathcal{S}} x_{i,j} = 1, \tag{3.82}$$

$$\sum_{h\,:\,(h,i)\in\mathcal{B}} x_{h,i} = \sum_{j\,:\,(i,j)\in\mathcal{B}} x_{i,j} \qquad \forall i \in \mathcal{R}, \tag{3.83}$$

$$\sum_{(h,i)\in\mathcal{B}\,:\,i\in\mathcal{E}} x_{h,i} = 1, \tag{3.84}$$

$$\sum_{j\,:\,(i,j)\in\mathcal{B}} x_{i,j} \le 1 \qquad \forall i \in \mathcal{R}\cup\mathcal{S}, \tag{3.85}$$

$$t_i + d(i,j) - t_j \le M(1 - x_{i,j}) \qquad \forall (i,j) \in \mathcal{B}, \tag{3.86}$$

$$x_{i,j} \in [0,1] \qquad \forall (i,j) \in \mathcal{B}, \tag{3.87}$$

$$t_i \in [\min(\text{cTime}(i)), \max(\text{cTime}(i))] \qquad \forall i \in \mathcal{N}. \tag{3.88}$$

Figure 3.12: The linear relaxation of the shortest path problem in the CrewBound optimization constraint.

Objective Function (3.80) minimizes the total distance. Constraint (3.81) bounds the objective value using information from the current domains of the variables in the main constraint programming model. Constraints (3.82) to (3.84) are the flow constraints, which ensure the existence of a path from a start node to an end node. Constraint (3.85) is a redundant constraint that strengthens the linear program. Constraint (3.86) is the time-based subtour elimination constraint, where M is a big-M constant. When $d(i,j) = 0$, this constraint fails to perform subtour elimination; however, the linear program remains a valid relaxation of the shortest path problem. Constraints (3.87) and (3.88) restrict the domains of the $x_{i,j}$ and t_i variables respectively.

The CrewBound constraint performs three tasks. First, it checks feasibility, i.e., the existence of a route for the crew that satisfies all constraints. Second, it constrains the lower bound of crewDist(c) to be greater than the objective value of the linear program. Third, it prunes the search space using the reduced costs at optimality. For every arc $(i,j) \in \mathcal{B}$, it prunes j from the domain of crewSucc(c, i) if

$$\min(\text{cDist}) + \bar{x}_{i,j} > \max(\text{cDist}),$$

where $\bar{x}_{i,j}$ is the reduced cost of $x_{i,j}$.

Algorithm 3.1: Sketch of the procedure that assigns vehicle routes.

```
1 for all v ∈ V
2     i ← s(v)
3     while i ≠ e(v)
4         try all j ∈ D(succ(i)) ordered by min(l(i) ≠ l(j), min(D(ser(j))))
5             succ(i) ← j
6         i ← succ(i)
```

3.5.3 The Search Procedures

The search procedure first assigns vehicle routes and then crew routes. Once all routes are assigned, it assigns values to the time variables.

Vehicle routes are assigned according to Algorithm 3.1. The procedure considers each vehicle in turn (Line 1) and labels the successor variables from its start node (Line 2) to its end node (Line 3). To choose successors, the algorithm assigns nodes to successor variables using a heuristic that lexicographically prefers nodes at the same location and then nodes with earlier service start times (Line 4). Line 5 makes the assignment, and Line 6 advances to the chosen successor node. The **try all** instruction defines a branching decision.

Crew routes assigned using Algorithm 3.2. The intuition behind the algorithm is to start with a node i that has no assigned driver, assign a driver c to this node, and then complete the route of this crew. In assigning the route of crew c, preference is given to assigning the crew to drive as much as possible. More precisely, the search procedure begins by ordering all nodes $i \in \mathcal{R} \cup \mathcal{S}$ that have an unassigned driver(i) variable by earliest departure time (Line 2). The search procedure then selects the first of these nodes and assigns it a driver (Line 4). Only a single driver (here, the one with smallest index) must be considered at this point because all crews are identical (Line 3). For this driving crew, the search procedure constructs a path covering the vehicle routes from the node to the depot (Line 6). It simultaneously labels the crew successor variables from the node to any vehicle end node (Line 8) and the driver variables until the crew exceeds its maximum driving duration (Line 9). Note that the **try** instruction in Line 9 tries to assign crew c as the driver at node i. If this is not possible, the instruction has no effect. The value selection heuristic for crew successor variables favors the node that is visited next by the current vehicle of the crew, provided that the crew can drive from this node; otherwise, the successor is chosen randomly. Once the crew reaches an end node, it is disallowed from driving at any other node (Lines 11 and 12). This process is repeated until all driver($\cdot$) variables are fixed. The search procedure then completes the crew routes by labeling the crew successor variables from the first drive node of each crew back to any vehicle start node (Lines 13 to 17).

Algorithm 3.3 shows the procedure that assigns vehicle schedules and crew schedules. This procedure is best explained using Figure 3.2. The search procedure selects a vehicle route and divides it into segments consisting of requests at the same location. These segments correspond to requests 1 to 3 and requests 4 to 6 in the example. The search procedure then labels the departure time variable at the exit request in each segment (i.e., requests 3 and 6). It only needs

Algorithm 3.2: Sketch of the procedure that assigns drivers and crew routes.

```
1  initialize list startDrive of tuples
2  for all i ∈ ℛ ∪ 𝒮 such that |D(driver(i))| > 1 ordered by min(D(dep(i)))
3      c ← min(D(driver(i)))
4      driver(i) ← c
5      startDrive.append(c, i)
6      while i ∉ ℰ
7          try all j ∈ D(crewSucc(c, i)) ordered by max(c ∈ D(driver(i)) ∧ j ∈ D(succ(i)), rand())
8              crewSucc(c, i) ← j
9              try driver(i) ← c
10         i ← crewSucc(c, i)
11     for all i ∈ ℛ ∪ 𝒮 such that c ∈ D(driver(i)) ∧ |D(driver(i))| > 1
12         driver(i) ≠ c
13 for all (c, i) ∈ startDrive
14     while i ∉ 𝒮
15         try all h ∈ 𝒩 such that i ∈ D(crewSucc(c, h))
16             crewSucc(c, h) ← i
17         i ← h
```

to branch on the departure time variables at these requests because the arrival time at the entry requests (i.e., requests 1 and 4) are specified by the travel time constraints, and all three time variables at the intermediate nodes (i.e., requests 2 and 5) are bounded by the arrival time at entry requests and the departure time at the exit requests. Once the arrival and departure time of each segment is known, the search procedure simply fixes the time variables at intermediate nodes to their earliest possible. This is always feasible since the travel time and service time constraints already bound the time variables at intermediate requests during the construction of the routes. Crew schedules are simply fixed to their earliest possible since they are tied to the vehicle schedules.

Note that branching on the departure times is essential because of the constraint that limits the maximum driving duration. This constraint is a simple subtraction and propagates extremely weakly unless its variables are fixed.

3.6 The Large Neighborhood Search

A large neighborhood search is also applied to the mixed integer programming and constraint programming models. Large neighborhood search iteratively finds a sequence of improving solutions by destroying parts of a solution and reconstructing it using an underlying solution method. In particular, large neighborhood search uses a neighborhood to fix a number of variables to their values in the incumbent solution and then calls the underlying solver to determine values for the remaining relaxed variables. The solver is given a limited run time since large neighborhood search is an incomplete method and no proof of optimality is available. The large neighborhood search is applied to both the mixed integer programming and constraint programming models with the following four neighborhoods:

Algorithm 3.3: Sketch of the procedure that assigns vehicle and crew schedules.

```
1  for all v ∈ 𝒱 ordered by min(D(dep(s(v))))
2      i ← s(v)
3      while i ∉ ℰ
4          while l(i) = l(succ(i))
5              dep(i) ← min(D(dep(i)))
6              ser(i) ← min(D(ser(i)))
7              i ← succ(i)
8          try all t ∈ D(dep(i))
9              dep(i) ← t
10         ser(i) ← min(D(ser(i)))
11         i ← succ(i)
12 for all c ∈ 𝒞
13     i ← crewSucc(c, 0)
14     while i ≠ 0
15         crewTime(i) ← min(D(crewTime(i)))
16         i ← crewSucc(c, i)
```

- **Vehicle Route Neighborhood:** This neighborhood fixes a number of vehicle routes and destroys all other vehicle routes and all existing crew routes. This neighborhood destroys large portions of an existing solution, and hence, has the potential to explore diverse parts of the search space. However, the neighborhood is difficult to explore exhaustively because of its size. Because of this, this neighborhood performs better early in optimization and has difficulty improving the solutions near optimality.

- **Request Neighborhood:** This neighborhood relaxes several pickup-delivery pairs and all crew routes. The relaxed requests are then inserted into existing routes. This neighborhood complements the vehicle route neighborhood since it attempts to obtain incremental improvement to an existing solution by destroying small portions of the solution. This neighborhood performs better than the vehicle routing neighborhood later in the solution process because it is significantly smaller and can be explored more exhaustively.

- **Crew Route Neighborhood:** This neighborhood fixes all vehicle routes and relaxes a number of crew routes. It aims to improve the crew objectives using the same reasoning as for the Vehicle Route Neighborhood.

- **Crew Passenger Neighborhood:** This neighborhood fixes all vehicle routes and the driving segments of all crews, and relaxes the passenger segments before and after the driving segment of each crew. This neighborhood attempts to obtain minor improvement to an existing solution by optimizing the pre-driving and post-driving passenger segments of crews. These two segments are only loosely coupled to the vehicles, whereas the driving segment of each crew is tightly coupled to the vehicles. This means that there is more opportunity to optimize these two segments.

3.7 Experimental Results

This section describes the experiments and analyzes the results.

3.7.1 The Instances

The instances are generated to capture the essence of applications in humanitarian and military logistics. These problems typically feature fewer locations than traditional vehicle routing applications but comprise multiple requests at each location. First, five sets of seven locations and five sets of eleven locations are generated on a 50 × 50 Euclidean grid. Next, 5, 10, 20, 30, 40 and 50 pickup-delivery pairs are generated and assigned to the locations. The instances are then duplicated using three different cost functions:

- $w_1 = 1000$, $w_2 = 1000$, $w_3 = 1$, $w_4 = 1$,
- $w_1 = 1000$, $w_2 = 5000$, $w_3 = 1$, $w_4 = 1$, and
- $w_1 = 5000$, $w_2 = 1000$, $w_3 = 1$, $w_4 = 1$.

In total, there are $10 \times 6 \times 3 = 180$ instances. Service durations vary between 1 and 20, and load demands vary between 1 and 15. The time windows of requests are randomly chosen.

3.7.2 The Methods

Both the mixed integer programming and constraint programming models are solved in two stages. First, the vehicle component is solved to produce an initial set of feasible vehicle routes. Then the full model, consisting of the vehicle and crew components, is started using these vehicle routes. If the vehicle routing stage is unable to find a feasible solution, the main model is started with a solution consisting of one pickup-delivery pair per vehicle.

Additionally, the full model is solved with (1) the vehicle route variables fixed according to the initial solution, and (2) the vehicle route variables fixed according to the initial solution and the vehicle time variables fixed to their earliest possible. These two models are respectively named Fixed and Semi-flexible. The original model with both variable vehicle routes and schedules is named Flexible.

All three models are started using the same vehicle routes, enabling any improvement in the objective value to be attributed to the joint vehicle and crew optimization rather than to other causes, such as the branching decisions used to obtain an initial solution. Because the vehicle routes in Fixed and Semi-flexible are fixed to the initial vehicle routing solution but Semi-flexible also searches over vehicle schedules, any improvement in crew routing objectives seen in Semi-flexible can be attributed to better vehicle scheduling. Similarly, the impact of rerouting vehicles according to the crew objectives can be examined by comparing Flexible against Semi-flexible since Semi-flexible has fixed vehicle routes but Flexible does not.

The mixed integer programming models and constraint programming models are respectively implemented in Gurobi and Objective-CP (Van Hentenryck and Michel 2013). All six models are solved using branch-and-bound and the large neighborhood search procedures detailed in Section 3.6. The two search techniques are respectively named BB and LNS.

	MIP-BB	CP-BB	MIP-LNS	CP-LNS
Fixed	118	180	151	180
Semi-flexible	121	180	144	180
Flexible	94	180	143	180

Table 3.3: Number of instances with feasible solutions for each method.

The following parameters are used for the large neighborhood search:

- The four neighborhoods are selected with equal probability. For the first stage of the sequential methods, there is an equal probability of selecting any of the two vehicle neighborhoods.
- For the vehicle route neighborhood, between two and five vehicle routes are destroyed. If the current solution has fewer than the chosen number of vehicle routes to destroy, one route will be retained and the rest destroyed.
- For the request neighborhood, between two and seven pickup-delivery pairs are destroyed but at least one is retained.
- For the crew route neighborhood, between two and seven crew routes are destroyed but at least one is retained.
- For the crew passenger neighborhood, the passenger segments of all crews are destroyed but the driving segments are retained.
- For MIP-LNS, Gurobi is called to determine values for the relaxed variables until it reaches a time limit of one minute.
- For CP-LNS, Objective-CP is used to fix values for the relaxed variables until it reaches a failure limit of 800.

The vehicle routing initialization is solved for one hour and the main model is solved for three hours on an Intel Xeon E5-2660 V3 CPU at 2.6 GHz.

3.7.3 Feasible Solutions

Table 3.3 shows the number of instances for which the methods find at least one feasible solution. The CP-based methods find feasible solutions to all 180 instances but the MIP-based methods face significant difficulty in even finding feasible solutions.

3.7.4 The Impacts of Rescheduling Vehicles

The advantages seen in Semi-flexible in comparison to Fixed can be attributed to the ability to co-optimize vehicle schedules and crews. Table 3.4 compares the four Semi-flexible models against the Fixed models. The results show that the four Semi-flexible approaches perform substantially better than their Fixed counterparts, achieving overall cost reductons of between 4.40% and 12.82%. The difference is most apparant in MIP-LNS, which performs up to 50.32% better than Fixed MIP-LNS. However, it also performs up to 129.03% worse. Semi-flexible MIP-BB

		MIP-BB	CP-BB	MIP-LNS	CP-LNS
Fixed	Average	−12.82%	−9.64%	−4.40%	−9.52%
	Minimum	−45.36%	−45.68%	−50.32%	−31.12%
	Maximum	1.22%	0.00%	129.03%	0.00%
	Standard deviation	10.96%	8.11%	22.38%	7.83%
Best Fixed	Average	98.59%	12.19%	28.69%	−9.28%
	Minimum	−31.02%	−31.02%	−31.02%	−31.05%
	Maximum	918.69%	754.83%	560.02%	0.06%
	Standard deviation	173.77%	71.24%	95.37%	7.60%

Table 3.4: Comparison of the four Semi-flexible methods against their Fixed counterparts and the best Fixed method of each instance.

performs up to 1.22% worse than Fixed MIP-BB, while Semi-flexible CP-BB and CP-LNS perform no worse than their Fixed variants.

A comparison of the four Semi-flexible methods against the a posteriori best Fixed method of each instance is also presented in the table. The results indicate that Semi-flexible MIP-BB (98.59%), CP-BB (12.19%) and MIP-LNS (28.69%) all perform worse than the best Fixed method on average. Only Semi-flexible CP-LNS improves on the best Fixed approach on average (9.28%). In fact, CP-LNS only performs up to 0.06% worse than the best Fixed approach on difficult instances, whereas the other three methods perform from five to over nine times worse. However, all four Semi-flexible approaches can perform up to 31% better. The standard deviations of comparing MIP-BB, CP-BB and MIP-LNS against the best Fixed approach is particularly large, indicating that their performance is highly dependent on the instances. The standard deviation of CP-LNS is much smaller, suggesting that this method is more consistent in its performance.

The excellent behavior of Semi-flexible CP-LNS indicates that there is significant value in jointly optimizing vehicle schedules and crew routes in a two-stage approach consisting of an initial vehicle routing phase and a vehicle scheduling and crew routing and scheduling step.

3.7.5 The Impacts of Rerouting Vehicles

The effects of rerouting vehicles according to crew objectives can be observed by comparing Flexible to Semi-flexible. Table 3.5 provides statistics comparing the four Flexible models against the Semi-flexible models. On average, Flexible MIP-BB (0.49%), CP-BB (0.30%) and CP-LNS (1.61%) perform better than their Semi-flexible counterparts. However, allowing variable vehicle routes is detrimental to Flexible MIP-LNS. It displays significant difficulty in finding good feasible solutions, performing 3.67% worse than Semi-flexible MIP-LNS overall. Flexible MIP-BB, CP-BB and MIP-LNS all can perform up to 22.52% better than their Semi-flexible variants, while CP-LNS can reach 30.76% better. Due to the short time limit and the larger search space, all four Flexible methods can perform worse than their Semi-flexible counterparts. In particular, Flexible MIP-LNS can be up to 58.89% worse off than Semi-flexible MIP-LNS.

The trends seen in comparing Flexible against the best Semi-flexible model is similar to comparing Semi-flexible against the best Fixed method in the previous subsection. On average,

		MIP-BB	CP-BB	MIP-LNS	CP-LNS
Semi-flexible	Average	−0.49%	−0.30%	3.67%	−1.61%
	Minimum	−22.52%	−22.52%	−22.52%	−30.76%
	Maximum	5.73%	1.55%	58.89%	9.02%
	Standard deviation	3.34%	2.14%	10.73%	4.80%
Best Semi-flexible	Average	103.08%	24.45%	48.48%	−1.45%
	Minimum	−22.52%	−22.52%	−22.52%	−30.66%
	Maximum	1054.08%	867.52%	739.96%	9.02%
	Standard deviation	215.79%	83.86%	117.43%	4.72%

Table 3.5: Comparison of the four Flexible methods against their Semi-flexible counterparts and the best Semi-flexible method of each instance.

only Flexible CP-LNS performs better than the best Semi-flexible model (1.45%), and the other three approaches perform significantly worse (24.45% to 103.08%). Even on difficult instances, Flexible CP-LNS only performs up to 9.02% worse than the best Semi-flexible model; the other three models perform from seven to over ten times worse. The standard deviation is again small for CP-LNS, indicating that its performance is much more consistent than the other three methods.

It appears that the concepts of rerouting vehicles for crew optimization are only realized by Flexible CP-LNS, which improves upon the best Semi-flexible method with cost reductions of up to 30.66% and 1.45% on average. These results indicate that improvements can be found by rerouting vehicles but are computationally demanding, warranting further investigation on the consequences of vehicle rerouting for crew routing and scheduling.

3.7.6 The Impacts of Rerouting and Rescheduling Vehicles

The combined effects of both rerouting and rescheduling vehicles can be analyzed in a comparison between the Flexible and Fixed methods. Table 3.6 shows summary statistics of this comparison. Note that there is some inconsistency with the MIP results in Tables 3.4 and 3.5 due to the different instances for which feasible solutions are available. The results demonstrate that Flexible MIP-BB and MIP-LNS are inferior to Semi-flexible MIP-BB and MIP-LNS on average. Flexible MIP-BB and MIP-LNS find fewer feasible solutions and improve less on Fixed MIP-BB and MIP-LNS. Contrastingly, Flexible CP-BB and CP-LNS perform 9.90% and 11.01% better than Fixed CP-BB and CP-LNS, which are more than the 9.64% and 9.52% improvements found by Semi-flexible CP-BB and CP-LNS. MIP-BB, CP-BB and CP-LNS achieve benefits of up to 45.36%, 45.68% and 47.04% at their best, and never perform worse than their Fixed counterparts. Flexible MIP-LNS improves on its own Fixed variant the most (50.32%), probably because Fixed MIP-LNS performed poorly. Flexible MIP-LNS is also the only method not dominating its Fixed counterpart, finding solutions up to 131.06% worse.

A comparison of Flexible against the a posteriori best Fixed results is also almost identical to the previous discussions. Flexible MIP-BB, CP-BB and MIP-LNS average 78.56%, 11.94% and 33.25% worse than the best Fixed result, and can exceed nine times worse. Only Flexible CP-LNS

		MIP-BB	CP-BB	MIP-LNS	CP-LNS
Fixed	Average	−11.13%	−9.90%	−0.86%	−11.01%
	Minimum	−45.36%	−45.68%	−50.32%	−47.04%
	Maximum	0.00%	0.00%	131.06%	0.00%
	Standard deviation	10.89%	8.49%	26.09%	8.34%
Best Fixed	Average	78.56%	11.94%	33.25%	−10.77%
	Minimum	−36.82%	−36.82%	−36.82%	−46.99%
	Maximum	918.69%	754.83%	560.02%	0.00%
	Standard deviation	182.18%	71.37%	97.38%	8.16%

Table 3.6: Comparison of the four Flexible methods against their Fixed counterparts and the best Fixed method of each instance.

improves on its Fixed variant on average (10.77%) and never performs any worse.

These results indicate that given an appropriate formulation and search technique, it is possible to find improved solutions by rerouting and rescheduling vehicles. Considering that Flexible CP-LNS dominates the best Fixed and improves on Semi-flexible as discussed previously, it is easy to conclude that Flexible CP-LNS is the best method of those evalulated.

3.7.7 Detailed Analysis

This section presents the main discoveries for each method. All solutions from MIP-BB, CP-BB, MIP-LNS and CP-LNS are respectively reported in Tables 3.7 to 3.10 at the end of this chapter on pages 81 to 104.

Analysis of MIP-BB A number of findings related to MIP-BB are listed below:

- Fixed is only able to find feasible solutions to 118 of the 180 instances despite being initialized with feasible vehicle routes. Semi-flexible and Flexible respectively find feasible solutions to 121 and 94 instances. MIP-BB performs particularly poorly on the instances with 40 and 50 pickup-delivery pairs. Fixed, Semi-flexible and Flexible are only able to find feasible solutions to 7, 10 and 9 of the 60 largest instances respectively.
- Semi-flexible does not dominate Fixed; Semi-flexible performs worse on two instances (1.22% and 0.42%). On average, Semi-flexible performs 12.82% better than Fixed on the 106 instances for which they both find feasible solutions. For the cost functions $w_1 = 1000$ and $w_2 = 1000$, $w_1 = 1000$ and $w_2 = 5000$, and $w_1 = 5000$ and $w_2 = 1000$, Semi-flexible respectively averages 14.48%, 17.48% and 6.76% better than Fixed on the 36, 34 and 36 instances with feasible solutions from both models. As discussed previously, these improvements are likely a consequence of allowing variable vehicle schedules.
- There are 82 instances for which Semi-flexible reschedules the vehicles for an improved overall cost and a reduction in the number of crews. For these instances, 7.52 fewer crews are needed on average, resulting in cost savings of 16.55%.
- Flexible dominates Fixed. It finds savings of 11.13% better overall, at least 20% savings on 16 instances, and 45.36% savings on one instance. For the three cost functions, Flexible

performs 12.46%, 15.41% and 5.94% better than Fixed on average on the 29, 25 and 28 instances for which both models find feasible solutions.

- There are 25 instances for which Flexible trades crew costs for vehicle costs in order to find overall better solutions. On these instances, overall costs reduced by 12.97%, vehicle costs increased by 4.20% and crew costs reduced by 19.18%. Two of these 25 instances require one more vehicle but one fewer crew, resulting in 0.50% and 10.49% cost savings overall, which comprises of an increase in vehicle costs of 44.93% and 48.01%, and a decrease in crew costs of 13.98% and 14.13%.
- Flexible performs 0.49% better on average than Semi-flexible. There are 12 instances for which Flexible performs worse than Semi-flexible, and 28 instances for which it performs better. For the three cost functions, Flexible averages 0.49%, 0.70% and 0.26% better than Semi-flexible on the 31, 31 and 30 instances for which both approaches find feasible solutions.
- There are 80 instances for which all three variants find feasible solutions. On these instances, Semi-flexible and Flexible perform 10.77% and 11.41% better than Fixed respectively, and Flexible improves upon Semi-flexible by 0.72%. These results suggest that Flexible MIP-BB is superior to Semi-flexible; although it is difficult to definitively argue this case considering that Flexible only finds feasible solutions to 94 instances whereas Semi-flexible finds feasible solutions to 121 instances.

Analysis of CP-BB Several findings focusing on CP-BB are discussed below:

- Fixed, Semi-flexible and Flexible find feasible solutions to all 180 instances.
- CP-BB Fixed, Semi-flexible and Flexible averages 10.81%, 8.72% and 8.62% better than MIP-BB Fixed, Semi-flexible and Flexible on the 80 instances where the three variants of MIP-BB find solutions. CP-BB Fixed performs better than MIP-BB Fixed on 20 instances and worse on 15 instances. CP-BB Semi-flexible performs better than MIP-BB Semi-flexible on 20 instances and worse on 15. CP-BB Flexible performs better than MIP-BB Flexible on 21 instances and worse on 16 instances.
- Semi-flexible performs 9.64% better overall than Fixed. For the three cost functions, it averages 10.39%, 12.96% and 5.59% better. These numbers are not as pronounced as MIP-BB because Fixed CP-BB already performs better than Fixed MIP-BB.
- Semi-flexible dominates Fixed, which shows that CP-BB can improve crew optimization by searching over vehicle schedules. On 144 instances, Semi-flexible finds solutions with 4.47 fewer crews on average and reductions of 12.03% in cost compared to Fixed.
- Flexible dominates Fixed and averages 9.90% better overall, and 10.70%, 13.28% and 5.71% better for the three cost functions.
- Flexible performs almost identically to Semi-flexible, achieving 0.30% better solutions in general. It performs better than Semi-flexible on 28 instances, achieving reductions of 14.20% in cost. It also performs worse on three instances, with increased costs of 0.54% on average.

- There are 24 instances on which Flexible finds improved solutions compared to Fixed by increasing vehicle costs. Increasing vehicle costs by 0.25% allows crew costs to be decreased by 21.20%, achieving an overall reduction of 14.65%. On the same instances, relaxing the vehicle routes allows Flexible to improve upon Semi-flexible by decreasing crew costs by 3.81%, resulting in an overall decrease of 2.26% in costs.

Analysis of MIP-LNS An analysis of MIP-LNS is presented below:

- Fixed, Semi-flexible and Flexible find feasible solutions to 151, 144 and 143 instances respectively; that is, 33, 23 and 49 more than MIP-BB.
- Semi-flexible sees benefits of 4.40% compared to Fixed. It performs worse (23.18%) on five instances and better (11.62%) on 115 instances. For the three cost functions, Semi-flexible finds cost savings of 5.27%, 5.05% and 2.88%.
- Semi-flexible is able to reschedule the vehicles to reduce the total cost and the total number of crews compared to Fixed on 99 instances. On these instances, 2.97 fewer crews are required and 13.11% cost savings are available.
- Flexible improves upon Fixed by 0.86% overall. The cost reductions on 13 instances exceed 20%, and reaches 50.32% on one instance. For the three cost functions, it performs 2.59% better, 1.24% worse and 1.31% better.
- Flexible performs 3.67% worse than Semi-flexible overall. The solutions are 2.77%, 6.29% and 1.83% worse when separated into the three cost functions.
- Flexible reroutes vehicles to increase vehicle costs but decrease crew costs on 36 instances. Vehicle costs are increased by 0.28% on average when compared to Fixed. In return, crew costs are decreased by up to 39.97% but averages 18.20%. On these 36 instances, overall costs are reduced by 13.05%.
- MIP-LNS Fixed, Semi-flexible and Flexible averages 11.06%, 5.94% and 4.84% better than MIP-BB on the 80 instances that all six variants find feasible solutions, and 8.58%, 16.40% and 21.23% worse than CP-BB on the 140 instances with feasible solutions.

Analysis of CP-LNS A number of key results are highlighted below:

- Fixed, Semi-flexible and Flexible find feasible solutions to all 180 instances.
- Semi-flexible dominates Fixed, performing 9.52% better on average, and 10.29%, 12.79% and 5.49% better for the three cost functions.
- Semi-flexible reschedules the vehicles on 141 instances to find lower total costs, which are reduced by 12.13%. On average, 2.98 fewer crews are necessary.
- Flexible dominates Fixed, improving upon Fixed by 11.01% overall, and 11.70%, 14.72% and 6.62% for the three cost functions.
- There are 30 instances for which Flexible performs worse (1.84%) and 98 instances for which it performs better (3.51%) than Semi-flexible. Generally, Flexible improves on Semi-flexible by 1.61%, unlike the other three search methods, for which Flexible performs nearly identically or worse than Semi-flexible.

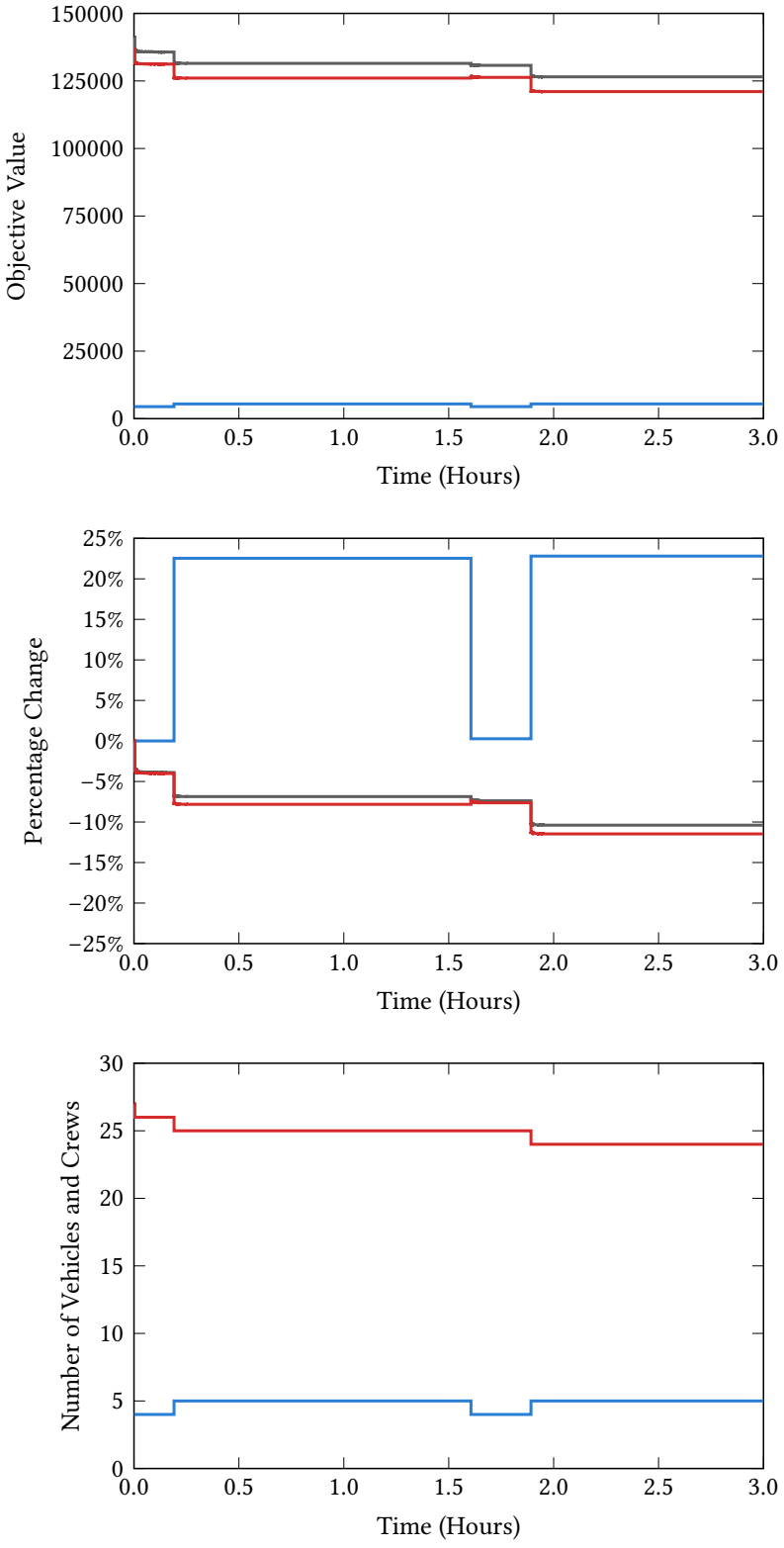


Figure 3.13: Plots of the objective value, the percentage change and the number of vehicles and crews over time for Flexible CP-LNS on an instance with $|\mathcal{L}| = 11$, $|\mathcal{P}| = 30$, $w_1 = 1000$ and $w_2 = 5000$. Lines colored blue represent vehicles, red represent crews and grey represent the total objective.

- Flexible often finds solutions in which vehicle costs are higher than those from Fixed. On 75 instances, it increases vehicle costs by 1.48% and decreases crew costs by 17.83%, resulting in savings of 12.82% on average.
- Figure 3.13 shows three plots related to Flexible CP-LNS on an instance with $|\mathcal{L}| = 11$, $|\mathcal{P}| = 30$, $w_1 = 1000$ and $w_2 = 5000$. Notice that the vehicle costs and crew costs do not monotonically decrease. Furthermore, the plots show that the solver twice finds a solution that uses one additional vehicle to decrease overall costs. The final solutions from Fixed and Semi-flexible require 4 vehicles and 29 and 26 crews respectively. The final solution of Flexible requires 5 vehicles but only 24 crews. This solution is 16.09% better than Fixed and 6.76% better than Semi-flexible. Vehicle costs are increased by 22.81% but crew costs are decreased by 17.27% compared to Fixed and 7.76% compared to Semi-flexible.
- On average, CP-LNS Fixed, Semi-flexible and Flexible respectively perform 12.35%, 11.37% and 13.07% better than MIP-BB, 12.17%, 12.32% and 13.62% better than CP-BB and 11.93%, 13.41% and 17.41% better than MIP-LNS on the instances for which the six variants in each comparison find feasible solutions.

3.8 Conclusion

This chapter presents the Joint Vehicle and Crew Routing and Scheduling Problem, which is motivated by applications arising in humanitarian and military logistics. The problem routes and schedules vehicles and crews to pick up and deliver requests. Because crews are able to interchange vehicles, both vehicle routes and crew routes become highly interdependent in space and time.

This chapter develops a high-level model of the problem, which is formulated as a mixed integer programming model and a constraint programming model. The two models overlay crew routing constraints over the Pickup and Delivery Problem with Time Windows. The constraint programming model features a symmetry-breaking global constraint and a global optimization constraint to detect infeasibility and to bound crew objectives. Both the mixed integer programming and constraint programming models are solved using a regular branch-and-bound search and a large neighborhood search. In order to compare the effects of rerouting and rescheduling vehicles on crew optimization, both models are extended with constraints that (1) fix the vehicle routes and (2) fix the vehicle routes and vehicle schedules. Comparing the full model, called Flexible, against one with fixed vehicle routes, called Semi-flexible, allows the impacts of rerouting vehicles for crew optimization to be quantified. Similarly, comparing Semi-flexible to the model with fixed vehicle routes and schedules, named Fixed, allows the benefits of rescheduling vehicles to be observed.

Experimental results indicate that jointly optimizing vehicle and crew routing and scheduling achieves significant benefits, and that the constraint programming model coupled with large neighborhood search finds the greatest reductions in costs compared to the other approaches. In particular, it produces average improvements of 10.77% compared to the a posteriori best Fixed sequential method and 1.45% compared to the a posteriori best partially sequential Semi-flexible

method.

All models and search methods found solutions that trade crew costs for vehicle costs in order to produce an overall improved solution. The ability to use fewer crews at the expense of more vehicles is a key feature of integrated models and is highly difficult, if not impossible, to replicate in sequential methods.

These results highlight the benefits of jointly optimizing vehicles and crew, and in particular, suggest that many of the benefits of jointly optimizing vehicle and crew routing originate in the ability to reschedule vehicles according to crew objectives. There are also benefits in rerouting vehicles for crew optimization but, at this stage, they seem much more computationally demanding.

There are several interesting research avenues going forward. Given the strong performance of the Semi-flexible methods, it would be interesting to study whether vehicle scheduling and crew routing can be solved to optimality using combinations of constraint programming and mathematical programming. Another direction for future research is to evaluate a three-stage optimization process that runs Fixed, Semi-flexible and Flexible in sequence with each stage initialized using the solution from the previous stage. Furthermore, another promising direction for future research is to develop neighborhoods that span vehicle and crew routes and/or schedules because the existing neighborhoods only consider vehicle routes or crew routes independently. Finally, it will also be interesting to compare the existing routing models against state-of-the-art scheduling models based on constraint programming, as suggested by a reviewer.

				Fixed			Semi-flexible						Flexible					
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value			Objective Value			Change (Fixed)			Objective Value			Change (Fixed)			Change (Semi-flexible)		
6	5	1,000	1,000	8,312 (	2,078 +	6,234)	7,268 (	2,078 +	5,190)	−12.6% (	0.0%,	−16.7%)	7,268 (	2,078 +	5,190)	−12.6% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				4,166 (	1,047 +	3,119)	4,144 (	1,047 +	3,097)	−0.5% (	0.0%,	−0.7%)	4,135 (	1,048 +	3,087)	−0.7% (	0.1%,	−1.0%)	−0.2% (	0.1%,	−0.3%)
				4,098 (	1,035 +	3,063)	3,105 (	1,035 +	2,070)	−24.2% (	0.0%,	−32.4%)	3,089 (	1,037 +	2,052)	−24.6% (	0.2%,	−33.0%)	−0.5% (	0.2%,	−0.9%)
				4,410 (	1,151 +	3,259)	3,453 (	1,151 +	2,302)	−21.7% (	0.0%,	−29.4%)	3,352 (	1,159 +	2,193)	−24.0% (	0.7%,	−32.7%)	−2.9% (	0.7%,	−4.7%)
				7,432 (	3,202 +	4,230)	6,404 (	3,202 +	3,202)	−13.8% (	0.0%,	−24.3%)	6,404 (	3,202 +	3,202)	−13.8% (	0.0%,	−24.3%)	0.0% (	0.0%,	0.0%)
		1,000	5,000	32,312 (	2,078 +	30,234)	27,268 (	2,078 +	25,190)	−15.6% (	0.0%,	−16.7%)	27,268 (	2,078 +	25,190)	−15.6% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				16,166 (	1,047 +	15,119)	16,144 (	1,047 +	15,097)	−0.1% (	0.0%,	−0.1%)	16,135 (	1,048 +	15,087)	−0.2% (	0.1%,	−0.2%)	−0.1% (	0.1%,	−0.1%)
				16,098 (	1,035 +	15,063)	11,105 (	1,035 +	10,070)	−31.0% (	0.0%,	−33.1%)	11,089 (	1,037 +	10,052)	−31.1% (	0.2%,	−33.3%)	−0.1% (	0.2%,	−0.2%)
				16,410 (	1,151 +	15,259)	11,453 (	1,151 +	10,302)	−30.2% (	0.0%,	−32.5%)	11,352 (	1,159 +	10,193)	−30.8% (	0.7%,	−33.2%)	−0.9% (	0.7%,	−1.1%)
				23,432 (	3,202 +	20,230)	18,404 (	3,202 +	15,202)	−21.5% (	0.0%,	−24.9%)	18,404 (	3,202 +	15,202)	−21.5% (	0.0%,	−24.9%)	0.0% (	0.0%,	0.0%)
		5,000	1,000	16,312 (	10,078 +	6,234)	15,268 (	10,078 +	5,190)	−6.4% (	0.0%,	−16.7%)	15,268 (	10,078 +	5,190)	−6.4% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				8,166 (	5,047 +	3,119)	8,144 (	5,047 +	3,097)	−0.3% (	0.0%,	−0.7%)	8,135 (	5,048 +	3,087)	−0.4% (	0.0%,	−1.0%)	−0.1% (	0.0%,	−0.3%)
				8,098 (	5,035 +	3,063)	7,105 (	5,035 +	2,070)	−12.3% (	0.0%,	−32.4%)	7,089 (	5,037 +	2,052)	−12.5% (	0.0%,	−33.0%)	−0.2% (	0.0%,	−0.9%)
				8,410 (	5,151 +	3,259)	7,453 (	5,151 +	2,302)	−11.4% (	0.0%,	−29.4%)	7,352 (	5,159 +	2,193)	−12.6% (	0.2%,	−32.7%)	−1.4% (	0.2%,	−4.7%)
				19,432 (	15,202 +	4,230)	18,404 (	15,202 +	3,202)	−5.3% (	0.0%,	−24.3%)	18,404 (	15,202 +	3,202)	−5.3% (	0.0%,	−24.3%)	0.0% (	0.0%,	0.0%)
6	10	1,000	1,000	13,462 (	4,160 +	9,302)	13,462 (	4,160 +	9,302)	0.0% (	0.0%,	0.0%)	13,462 (	4,160 +	9,302)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				8,194 (	2,064 +	6,130)	8,189 (	2,064 +	6,125)	−0.1% (	0.0%,	−0.1%)	8,189 (	2,064 +	6,125)	−0.1% (	0.0%,	−0.1%)	0.0% (	0.0%,	0.0%)
				7,191 (	2,071 +	5,120)	6,176 (	2,071 +	4,105)	−14.1% (	0.0%,	−19.8%)	6,176 (	2,071 +	4,105)	−14.1% (	0.0%,	−19.8%)	0.0% (	0.0%,	0.0%)
				5,671 (	1,204 +	4,467)	4,566 (	1,204 +	3,362)	−19.5% (	0.0%,	−24.7%)	4,542 (	1,237 +	3,305)	−19.9% (	2.7%,	−26.0%)	−0.5% (	2.7%,	−1.7%)
				13,776 (	6,380 +	7,396)	12,760 (	6,380 +	6,380)	−7.4% (	0.0%,	−13.7%)	12,760 (	6,380 +	6,380)	−7.4% (	0.0%,	−13.7%)	0.0% (	0.0%,	0.0%)
		1,000	5,000	49,462 (	4,160 +	45,302)	49,462 (	4,160 +	45,302)	0.0% (	0.0%,	0.0%)	49,455 (	4,162 +	45,293)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				32,194 (	2,064 +	30,130)	32,189 (	2,064 +	30,125)	0.0% (	0.0%,	0.0%)	32,189 (	2,064 +	30,125)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				27,191 (	2,071 +	25,120)	22,176 (	2,071 +	20,105)	−18.4% (	0.0%,	−20.0%)	22,176 (	2,071 +	20,105)	−18.4% (	0.0%,	−20.0%)	0.0% (	0.0%,	0.0%)
				21,671 (	1,204 +	20,467)	16,566 (	1,204 +	15,362)	−23.6% (	0.0%,	−24.9%)	16,542 (	1,237 +	15,305)	−23.7% (	2.7%,	−25.2%)	−0.1% (	2.7%,	−0.4%)
				41,776 (	6,380 +	35,396)	36,760 (	6,380 +	30,380)	−12.0% (	0.0%,	−14.2%)	36,760 (	6,380 +	30,380)	−12.0% (	0.0%,	−14.2%)	0.0% (	0.0%,	0.0%)
		5,000	1,000	29,462 (	20,160 +	9,302)	29,462 (	20,160 +	9,302)	0.0% (	0.0%,	0.0%)	29,462 (	20,160 +	9,302)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				16,194 (	10,064 +	6,130)	16,189 (	10,064 +	6,125)	0.0% (	0.0%,	−0.1%)	16,189 (	10,064 +	6,125)	0.0% (	0.0%,	−0.1%)	0.0% (	0.0%,	0.0%)
				15,191 (	10,071 +	5,120)	14,176 (	10,071 +	4,105)	−6.7% (	0.0%,	−19.8%)	14,176 (	10,071 +	4,105)	−6.7% (	0.0%,	−19.8%)	0.0% (	0.0%,	0.0%)
				9,689 (	5,196 +	4,493)	8,566 (	5,204 +	3,362)	−11.6% (	0.2%,	−25.2%)	8,566 (	5,204 +	3,362)	−11.6% (	0.2%,	−25.2%)	0.0% (	0.0%,	0.0%)
				37,776 (	30,380 +	7,396)	36,760 (	30,380 +	6,380)	−2.7% (	0.0%,	−13.7%)	36,760 (	30,380 +	6,380)	−2.7% (	0.0%,	−13.7%)	0.0% (	0.0%,	0.0%)

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	20	1,000	1,000	40,228 (11,449 + 28,779)	34,127 (11,449 + 22,678)	−15.2% (0.0%, −21.2%)	–	–	–
				60,852 (20,333 + 40,519)	40,666 (20,333 + 20,333)	−33.2% (0.0%, −49.8%)	40,666 (20,333 + 20,333)	−33.2% (0.0%, −49.8%)	0.0% (0.0%, 0.0%)
				57,884 (20,353 + 37,531)	41,716 (20,353 + 21,363)	−27.9% (0.0%, −43.1%)	–	–	–
				–	42,694 (21,347 + 21,347)	–	42,694 (21,347 + 21,347)	–	0.0% (0.0%, 0.0%)
				25,604 (11,780 + 13,824)	23,560 (11,780 + 11,780)	−8.0% (0.0%, −14.8%)	23,560 (11,780 + 11,780)	−8.0% (0.0%, −14.8%)	0.0% (0.0%, 0.0%)
		1,000	5,000	152,228 (11,449 + 140,779)	122,120 (11,449 + 110,671)	−19.8% (0.0%, −21.4%)	–	–	–
				220,852 (20,333 + 200,519)	120,666 (20,333 + 100,333)	−45.4% (0.0%, −50.0%)	120,666 (20,333 + 100,333)	−45.4% (0.0%, −50.0%)	0.0% (0.0%, 0.0%)
				205,884 (20,353 + 185,531)	125,721 (20,353 + 105,368)	−38.9% (0.0%, −43.2%)	–	–	–
				–	122,694 (21,347 + 101,347)	–	122,694 (21,347 + 101,347)	–	0.0% (0.0%, 0.0%)
				77,604 (11,780 + 65,824)	67,560 (11,780 + 55,780)	−12.9% (0.0%, −15.3%)	67,560 (11,780 + 55,780)	−12.9% (0.0%, −15.3%)	0.0% (0.0%, 0.0%)
		5,000	1,000	84,228 (55,449 + 28,779)	78,127 (55,449 + 22,678)	−7.2% (0.0%, −21.2%)	–	–	–
				140,852 (100,333 + 40,519)	120,666 (100,333 + 20,333)	−14.3% (0.0%, −49.8%)	120,666 (100,333 + 20,333)	−14.3% (0.0%, −49.8%)	0.0% (0.0%, 0.0%)
				137,884 (100,353 + 37,531)	121,716 (100,353 + 21,363)	−11.7% (0.0%, −43.1%)	–	–	–
				–	122,694 (101,347 + 21,347)	–	122,694 (101,347 + 21,347)	–	0.0% (0.0%, 0.0%)
				69,604 (55,780 + 13,824)	67,560 (55,780 + 11,780)	−2.9% (0.0%, −14.8%)	67,560 (55,780 + 11,780)	−2.9% (0.0%, −14.8%)	0.0% (0.0%, 0.0%)
6	30	1,000	1,000	86,296 (26,835 + 59,461)	–	–	–	–	–
				89,323 (29,481 + 59,842)	58,962 (29,481 + 29,481)	−34.0% (0.0%, −50.7%)	60,970 (29,481 + 31,489)	−31.7% (0.0%, −47.4%)	3.4% (0.0%, 6.8%)
				80,312 (27,515 + 52,797)	59,076 (27,515 + 31,561)	−26.4% (0.0%, −40.2%)	60,124 (27,515 + 32,609)	−25.1% (0.0%, −38.2%)	1.8% (0.0%, 3.3%)
				–	63,914 (31,957 + 31,957)	–	63,914 (31,957 + 31,957)	–	0.0% (0.0%, 0.0%)
				44,508 (21,232 + 23,276)	42,464 (21,232 + 21,232)	−4.6% (0.0%, −8.8%)	42,464 (21,232 + 21,232)	−4.6% (0.0%, −8.8%)	0.0% (0.0%, 0.0%)
		1,000	5,000	318,296 (26,835 + 291,461)	263,214 (26,835 + 236,379)	−17.3% (0.0%, −18.9%)	–	–	–
				–	174,962 (29,481 + 145,481)	–	184,988 (29,481 + 155,507)	–	5.7% (0.0%, 6.9%)
				–	183,096 (27,515 + 155,581)	–	193,118 (27,515 + 165,603)	–	5.5% (0.0%, 6.4%)
				–	183,914 (31,957 + 151,957)	–	183,914 (31,957 + 151,957)	–	0.0% (0.0%, 0.0%)
				132,508 (21,232 + 111,276)	122,464 (21,232 + 101,232)	−7.6% (0.0%, −9.0%)	122,464 (21,232 + 101,232)	−7.6% (0.0%, −9.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	190,308 (130,835 + 59,473)	–	–	–	–	–
				201,239 (145,481 + 55,758)	174,962 (145,481 + 29,481)	−13.1% (0.0%, −47.1%)	176,976 (145,481 + 31,495)	−12.1% (0.0%, −43.5%)	1.2% (0.0%, 6.8%)
				188,308 (135,515 + 52,793)	168,078 (135,515 + 32,563)	−10.7% (0.0%, −38.3%)	169,118 (135,515 + 33,603)	−10.2% (0.0%, −36.3%)	0.6% (0.0%, 3.2%)
				–	183,914 (151,957 + 31,957)	–	183,914 (151,957 + 31,957)	–	0.0% (0.0%, 0.0%)
				124,508 (101,232 + 23,276)	122,464 (101,232 + 21,232)	−1.6% (0.0%, −8.8%)	122,464 (101,232 + 21,232)	−1.6% (0.0%, −8.8%)	0.0% (0.0%, 0.0%)

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	40	1,000	1,000	–	–	–	–	–	–
				–	82,304 (40,650 + 41,654)	–	–	–	–
				–	–	–	–	–	–
				–	85,282 (42,641 + 42,641)	–	85,282 (42,641 + 42,641)	–	0.0% (0.0%, 0.0%)
				84,064 (42,032 + 42,032)	85,092 (42,032 + 43,060)	1.2% (0.0%, 2.4%)	84,064 (42,032 + 42,032)	0.0% (0.0%, 0.0%)	−1.2% (0.0%, −2.4%)
		1,000	5,000	–	–	–	–	–	–
				–	246,304 (40,650 + 205,654)	–	–	–	–
				–	–	–	–	–	–
				–	245,282 (42,641 + 202,641)	–	245,282 (42,641 + 202,641)	–	0.0% (0.0%, 0.0%)
				244,064 (42,032 + 202,032)	244,064 (42,032 + 202,032)	0.0% (0.0%, 0.0%)	244,064 (42,032 + 202,032)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	–	–	–	–	–	–
				–	243,314 (200,650 + 42,664)	–	–	–	–
				–	–	–	–	–	–
				–	245,282 (202,641 + 42,641)	–	245,282 (202,641 + 42,641)	–	0.0% (0.0%, 0.0%)
				244,064 (202,032 + 42,032)	245,092 (202,032 + 43,060)	0.4% (0.0%, 2.4%)	244,064 (202,032 + 42,032)	0.0% (0.0%, 0.0%)	−0.4% (0.0%, −2.4%)
6	50	1,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				105,036 (52,518 + 52,518)	–	–	105,036 (52,518 + 52,518)	0.0% (0.0%, 0.0%)	–
		1,000	5,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	306,550 (53,275 + 253,275)	–	311,594 (53,275 + 258,319)	–	1.6% (0.0%, 2.0%)
				305,036 (52,518 + 252,518)	–	–	–	–	–
		5,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				305,036 (252,518 + 52,518)	–	–	305,036 (252,518 + 52,518)	0.0% (0.0%, 0.0%)	–

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible			Flexible			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)		Objective Value	Change (Fixed)		Change (Semi-flexible)	
10	5	1,000	1,000	4,360 (1,090 + 3,270)	4,308 (1,090 + 3,218)	−1.2% (0.0%, −1.6%)		4,291 (1,093 + 3,198)	−1.6% (0.3%, −2.2%)		−0.4% (0.3%, −0.6%)	
				7,440 (1,093 + 6,347)	6,405 (1,093 + 5,312)	−13.9% (0.0%, −16.3%)		6,380 (1,096 + 5,284)	−14.2% (0.3%, −16.7%)		−0.4% (0.3%, −0.5%)	
				7,509 (1,111 + 6,398)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)		6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				7,395 (2,112 + 5,283)	6,336 (2,112 + 4,224)	−14.3% (0.0%, −20.0%)		5,307 (2,130 + 3,177)	−28.2% (0.9%, −39.9%)		−16.2% (0.9%, −24.8%)	
				6,327 (2,109 + 4,218)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)		6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	16,360 (1,090 + 15,270)	16,308 (1,090 + 15,218)	−0.3% (0.0%, −0.3%)		16,291 (1,093 + 15,198)	−0.4% (0.3%, −0.5%)		−0.1% (0.3%, −0.1%)	
				31,440 (1,093 + 30,347)	26,405 (1,093 + 25,312)	−16.0% (0.0%, −16.6%)		26,380 (1,096 + 25,284)	−16.1% (0.3%, −16.7%)		−0.1% (0.3%, −0.1%)	
				31,509 (1,111 + 30,398)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)		26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				27,395 (2,112 + 25,283)	22,336 (2,112 + 20,224)	−18.5% (0.0%, −20.0%)		17,307 (2,130 + 15,177)	−36.8% (0.9%, −40.0%)		−22.5% (0.9%, −25.0%)	
				22,327 (2,109 + 20,218)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)		22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	8,360 (5,090 + 3,270)	8,308 (5,090 + 3,218)	−0.6% (0.0%, −1.6%)		8,291 (5,093 + 3,198)	−0.8% (0.1%, −2.2%)		−0.2% (0.1%, −0.6%)	
				11,440 (5,093 + 6,347)	10,405 (5,093 + 5,312)	−9.0% (0.0%, −16.3%)		10,380 (5,096 + 5,284)	−9.3% (0.1%, −16.7%)		−0.2% (0.1%, −0.5%)	
				11,509 (5,111 + 6,398)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)		10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				15,395 (10,112 + 5,283)	14,336 (10,112 + 4,224)	−6.9% (0.0%, −20.0%)		13,307 (10,130 + 3,177)	−13.6% (0.2%, −39.9%)		−7.2% (0.2%, −24.8%)	
				14,327 (10,109 + 4,218)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)		14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
10	10	1,000	1,000	9,529 (2,173 + 7,356)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)		7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)		0.0% (0.0%, 0.0%)	
				10,907 (1,166 + 9,741)	10,907 (1,166 + 9,741)	0.0% (0.0%, 0.0%)		10,907 (1,166 + 9,741)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				12,744 (2,178 + 10,566)	11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)		11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)		0.0% (0.0%, 0.0%)	
				9,656 (2,210 + 7,446)	9,656 (2,210 + 7,446)	0.0% (0.0%, 0.0%)		9,608 (3,203 + 6,405)	−0.5% (44.9%, −14.0%)		−0.5% (44.9%, −14.0%)	
				9,644 (2,227 + 7,417)	8,638 (2,227 + 6,411)	−10.4% (0.0%, −13.6%)		8,638 (2,227 + 6,411)	−10.4% (0.0%, −13.6%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	37,529 (2,173 + 35,356)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)		27,483 (2,173 + 25,310)	−26.8% (0.0%, −28.4%)		0.0% (0.0%, 0.0%)	
				46,907 (1,166 + 45,741)	46,907 (1,166 + 45,741)	0.0% (0.0%, 0.0%)		46,875 (1,180 + 45,695)	−0.1% (1.2%, −0.1%)		−0.1% (1.2%, −0.1%)	
				52,744 (2,178 + 50,566)	47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)		47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)		0.0% (0.0%, 0.0%)	
				37,656 (2,210 + 35,446)	37,656 (2,210 + 35,446)	0.0% (0.0%, 0.0%)		33,707 (3,271 + 30,436)	−10.5% (48.0%, −14.1%)		−10.5% (48.0%, −14.1%)	
				37,644 (2,227 + 35,417)	32,638 (2,227 + 30,411)	−13.3% (0.0%, −14.1%)		32,638 (2,227 + 30,411)	−13.3% (0.0%, −14.1%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	17,529 (10,173 + 7,356)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)		15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)		0.0% (0.0%, 0.0%)	
				14,907 (5,166 + 9,741)	14,907 (5,166 + 9,741)	0.0% (0.0%, 0.0%)		14,907 (5,166 + 9,741)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				20,744 (10,178 + 10,566)	19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)		19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)		0.0% (0.0%, 0.0%)	
				17,656 (10,210 + 7,446)	17,656 (10,210 + 7,446)	0.0% (0.0%, 0.0%)		17,656 (10,210 + 7,446)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				17,644 (10,227 + 7,417)	16,638 (10,227 + 6,411)	−5.7% (0.0%, −13.6%)		16,638 (10,227 + 6,411)	−5.7% (0.0%, −13.6%)		0.0% (0.0%, 0.0%)	

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	20	1,000	1,000	53,545 (17,577 + 35,968)	39,244 (17,577 + 21,667)	−26.7% (0.0%, −39.8%)	39,299 (17,577 + 21,722)	−26.6% (0.0%, −39.6%)	0.1% (0.0%, 0.3%)
				56,573 (17,546 + 39,027)	43,330 (17,546 + 25,784)	−23.4% (0.0%, −33.9%)	–	–	–
				24,363 (3,386 + 20,977)	22,300 (3,386 + 18,914)	−8.5% (0.0%, −9.8%)	–	–	–
				59,451 (20,956 + 38,495)	45,014 (20,956 + 24,058)	−24.3% (0.0%, −37.5%)	–	–	–
				52,139 (19,877 + 32,262)	42,859 (19,877 + 22,982)	−17.8% (0.0%, −28.8%)	43,906 (19,877 + 24,029)	−15.8% (0.0%, −25.5%)	2.4% (0.0%, 4.6%)
		1,000	5,000	193,545 (17,577 + 175,968)	123,261 (17,577 + 105,684)	−36.3% (0.0%, −39.9%)	123,310 (17,577 + 105,733)	−36.3% (0.0%, −39.9%)	0.0% (0.0%, 0.0%)
				208,573 (17,546 + 191,027)	148,346 (17,546 + 130,800)	−28.9% (0.0%, −31.5%)	–	–	–
				104,446 (3,377 + 101,069)	94,310 (3,386 + 90,924)	−9.7% (0.3%, −10.0%)	–	–	–
				207,451 (20,956 + 186,495)	137,014 (20,956 + 116,058)	−34.0% (0.0%, −37.8%)	–	–	–
				176,139 (19,877 + 156,262)	130,859 (19,877 + 110,982)	−25.7% (0.0%, −29.0%)	–	–	–
		5,000	1,000	121,545 (85,577 + 35,968)	107,244 (85,577 + 21,667)	−11.8% (0.0%, −39.8%)	107,317 (85,577 + 21,740)	−11.7% (0.0%, −39.6%)	0.1% (0.0%, 0.3%)
				124,573 (85,546 + 39,027)	111,336 (85,546 + 25,790)	−10.6% (0.0%, −33.9%)	–	–	–
				36,363 (15,386 + 20,977)	34,310 (15,386 + 18,924)	−5.6% (0.0%, −9.8%)	–	–	–
				139,451 (100,956 + 38,495)	125,014 (100,956 + 24,058)	−10.4% (0.0%, −37.5%)	–	–	–
				128,139 (95,877 + 32,262)	118,859 (95,877 + 22,982)	−7.2% (0.0%, −28.8%)	–	–	–
10	30	1,000	1,000	91,507 (30,947 + 60,560)	64,964 (30,947 + 34,017)	−29.0% (0.0%, −43.8%)	–	–	–
				92,505 (28,897 + 63,608)	–	–	–	–	–
				87,372 (26,850 + 60,522)	70,071 (26,850 + 43,221)	−19.8% (0.0%, −28.6%)	–	–	–
				87,455 (29,309 + 58,146)	64,834 (29,309 + 35,525)	−25.9% (0.0%, −38.9%)	–	–	–
				83,306 (31,344 + 51,962)	–	–	–	–	–
		1,000	5,000	–	–	–	–	–	–
				340,505 (28,897 + 311,608)	–	–	–	–	–
				323,363 (26,850 + 296,513)	228,041 (26,850 + 201,191)	−29.5% (0.0%, −32.1%)	–	–	–
				311,454 (29,309 + 282,145)	200,829 (29,309 + 171,520)	−35.5% (0.0%, −39.2%)	–	–	–
				283,306 (31,344 + 251,962)	–	–	–	–	–
		5,000	1,000	211,507 (150,947 + 60,560)	187,002 (150,947 + 36,055)	−11.6% (0.0%, −40.5%)	–	–	–
				204,505 (140,897 + 63,608)	–	–	–	–	–
				191,371 (130,850 + 60,521)	172,037 (130,850 + 41,187)	−10.1% (0.0%, −31.9%)	–	–	–
				199,454 (141,309 + 58,145)	179,883 (141,309 + 38,574)	−9.8% (0.0%, −33.7%)	–	–	–
				203,306 (151,344 + 51,962)	–	–	–	–	–

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	40	1,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
		1,000	5,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				414,574 (41,766 + 372,808)	–	–	–	–	–
		5,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
10	50	1,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
		1,000	5,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
		5,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–

Table 3.7: Comparison of the solutions from MIP-BB. Vehicle and crew objective values are shown inside parenthesis.

				Fixed			Semi-flexible						Flexible					
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value			Objective Value			Change (Fixed)			Objective Value			Change (Fixed)			Change (Semi-flexible)		
6	5	1,000	1,000	8,312 (	2,078 +	6,234)	7,268 (	2,078 +	5,190)	−12.6% (	0.0%,	−16.7%)	7,268 (	2,078 +	5,190)	−12.6% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				4,166 (	1,047 +	3,119)	4,144 (	1,047 +	3,097)	−0.5% (	0.0%,	−0.7%)	4,135 (	1,048 +	3,087)	−0.7% (	0.1%,	−1.0%)	−0.2% (	0.1%,	−0.3%)
				4,098 (	1,035 +	3,063)	3,105 (	1,035 +	2,070)	−24.2% (	0.0%,	−32.4%)	3,089 (	1,037 +	2,052)	−24.6% (	0.2%,	−33.0%)	−0.5% (	0.2%,	−0.9%)
				4,410 (	1,151 +	3,259)	3,453 (	1,151 +	2,302)	−21.7% (	0.0%,	−29.4%)	3,352 (	1,159 +	2,193)	−24.0% (	0.7%,	−32.7%)	−2.9% (	0.7%,	−4.7%)
				6,404 (	3,202 +	3,202)	6,404 (	3,202 +	3,202)	0.0% (	0.0%,	0.0%)	6,404 (	3,202 +	3,202)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
		1,000	5,000	32,312 (	2,078 +	30,234)	27,268 (	2,078 +	25,190)	−15.6% (	0.0%,	−16.7%)	27,268 (	2,078 +	25,190)	−15.6% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				16,166 (	1,047 +	15,119)	16,144 (	1,047 +	15,097)	−0.1% (	0.0%,	−0.1%)	16,135 (	1,048 +	15,087)	−0.2% (	0.1%,	−0.2%)	−0.1% (	0.1%,	−0.1%)
				16,098 (	1,035 +	15,063)	11,105 (	1,035 +	10,070)	−31.0% (	0.0%,	−33.1%)	11,089 (	1,037 +	10,052)	−31.1% (	0.2%,	−33.3%)	−0.1% (	0.2%,	−0.2%)
				16,410 (	1,151 +	15,259)	11,453 (	1,151 +	10,302)	−30.2% (	0.0%,	−32.5%)	11,352 (	1,159 +	10,193)	−30.8% (	0.7%,	−33.2%)	−0.9% (	0.7%,	−1.1%)
				18,404 (	3,202 +	15,202)	18,404 (	3,202 +	15,202)	0.0% (	0.0%,	0.0%)	18,404 (	3,202 +	15,202)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
		5,000	1,000	16,312 (	10,078 +	6,234)	15,268 (	10,078 +	5,190)	−6.4% (	0.0%,	−16.7%)	15,268 (	10,078 +	5,190)	−6.4% (	0.0%,	−16.7%)	0.0% (	0.0%,	0.0%)
				8,166 (	5,047 +	3,119)	8,144 (	5,047 +	3,097)	−0.3% (	0.0%,	−0.7%)	8,135 (	5,048 +	3,087)	−0.4% (	0.0%,	−1.0%)	−0.1% (	0.0%,	−0.3%)
				8,098 (	5,035 +	3,063)	7,105 (	5,035 +	2,070)	−12.3% (	0.0%,	−32.4%)	7,089 (	5,037 +	2,052)	−12.5% (	0.0%,	−33.0%)	−0.2% (	0.0%,	−0.9%)
				8,410 (	5,151 +	3,259)	7,453 (	5,151 +	2,302)	−11.4% (	0.0%,	−29.4%)	7,352 (	5,159 +	2,193)	−12.6% (	0.2%,	−32.7%)	−1.4% (	0.2%,	−4.7%)
				18,404 (	15,202 +	3,202)	18,404 (	15,202 +	3,202)	0.0% (	0.0%,	0.0%)	18,404 (	15,202 +	3,202)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
6	10	1,000	1,000	13,462 (	4,160 +	9,302)	13,462 (	4,160 +	9,302)	0.0% (	0.0%,	0.0%)	13,462 (	4,160 +	9,302)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				8,194 (	2,064 +	6,130)	8,189 (	2,064 +	6,125)	−0.1% (	0.0%,	−0.1%)	8,189 (	2,064 +	6,125)	−0.1% (	0.0%,	−0.1%)	0.0% (	0.0%,	0.0%)
				7,227 (	2,077 +	5,150)	6,216 (	2,077 +	4,139)	−14.0% (	0.0%,	−19.6%)	6,207 (	2,078 +	4,129)	−14.1% (	0.0%,	−19.8%)	−0.1% (	0.0%,	−0.2%)
				5,701 (	1,205 +	4,496)	4,604 (	1,205 +	3,399)	−19.2% (	0.0%,	−24.4%)	4,596 (	1,208 +	3,388)	−19.4% (	0.2%,	−24.6%)	−0.2% (	0.2%,	−0.3%)
				13,776 (	6,380 +	7,396)	12,760 (	6,380 +	6,380)	−7.4% (	0.0%,	−13.7%)	12,760 (	6,380 +	6,380)	−7.4% (	0.0%,	−13.7%)	0.0% (	0.0%,	0.0%)
		1,000	5,000	49,462 (	4,160 +	45,302)	49,462 (	4,160 +	45,302)	0.0% (	0.0%,	0.0%)	49,462 (	4,160 +	45,302)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				32,194 (	2,064 +	30,130)	32,189 (	2,064 +	30,125)	0.0% (	0.0%,	0.0%)	32,189 (	2,064 +	30,125)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				27,227 (	2,077 +	25,150)	22,216 (	2,077 +	20,139)	−18.4% (	0.0%,	−19.9%)	22,207 (	2,078 +	20,129)	−18.4% (	0.0%,	−20.0%)	0.0% (	0.0%,	0.0%)
				21,701 (	1,205 +	20,496)	16,604 (	1,205 +	15,399)	−23.5% (	0.0%,	−24.9%)	16,596 (	1,208 +	15,388)	−23.5% (	0.2%,	−24.9%)	0.0% (	0.2%,	−0.1%)
				41,776 (	6,380 +	35,396)	36,760 (	6,380 +	30,380)	−12.0% (	0.0%,	−14.2%)	36,760 (	6,380 +	30,380)	−12.0% (	0.0%,	−14.2%)	0.0% (	0.0%,	0.0%)
		5,000	1,000	29,462 (	20,160 +	9,302)	29,462 (	20,160 +	9,302)	0.0% (	0.0%,	0.0%)	29,462 (	20,160 +	9,302)	0.0% (	0.0%,	0.0%)	0.0% (	0.0%,	0.0%)
				16,194 (	10,064 +	6,130)	16,189 (	10,064 +	6,125)	0.0% (	0.0%,	−0.1%)	16,189 (	10,064 +	6,125)	0.0% (	0.0%,	−0.1%)	0.0% (	0.0%,	0.0%)
				15,227 (	10,077 +	5,150)	14,216 (	10,077 +	4,139)	−6.6% (	0.0%,	−19.6%)	14,207 (	10,078 +	4,129)	−6.7% (	0.0%,	−19.8%)	−0.1% (	0.0%,	−0.2%)
				9,701 (	5,205 +	4,496)	8,604 (	5,205 +	3,399)	−11.3% (	0.0%,	−24.4%)	8,596 (	5,208 +	3,388)	−11.4% (	0.1%,	−24.6%)	−0.1% (	0.1%,	−0.3%)
				37,776 (	30,380 +	7,396)	36,760 (	30,380 +	6,380)	−2.7% (	0.0%,	−13.7%)	36,760 (	30,380 +	6,380)	−2.7% (	0.0%,	−13.7%)	0.0% (	0.0%,	0.0%)

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	20	1,000	1,000	35,146 (10,374 + 24,772)	31,037 (10,374 + 20,663)	−11.7% (0.0%, −16.6%)	31,037 (10,374 + 20,663)	−11.7% (0.0%, −16.6%)	0.0% (0.0%, 0.0%)
				19,566 (5,175 + 14,391)	17,494 (5,175 + 12,319)	−10.6% (0.0%, −14.4%)	17,494 (5,175 + 12,319)	−10.6% (0.0%, −14.4%)	0.0% (0.0%, 0.0%)
				17,627 (4,198 + 13,429)	14,502 (4,198 + 10,304)	−17.7% (0.0%, −23.3%)	14,502 (4,198 + 10,304)	−17.7% (0.0%, −23.3%)	0.0% (0.0%, 0.0%)
				8,710 (2,256 + 6,454)	6,670 (2,256 + 4,414)	−23.4% (0.0%, −31.6%)	6,618 (2,256 + 4,362)	−24.0% (0.0%, −32.4%)	−0.8% (0.0%, −1.2%)
				28,642 (13,801 + 14,841)	28,642 (13,801 + 14,841)	0.0% (0.0%, 0.0%)	28,642 (13,801 + 14,841)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	131,146 (10,374 + 120,772)	111,037 (10,374 + 100,663)	−15.3% (0.0%, −16.7%)	111,037 (10,374 + 100,663)	−15.3% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				75,566 (5,175 + 70,391)	65,494 (5,175 + 60,319)	−13.3% (0.0%, −14.3%)	65,494 (5,175 + 60,319)	−13.3% (0.0%, −14.3%)	0.0% (0.0%, 0.0%)
				69,627 (4,198 + 65,429)	54,502 (4,198 + 50,304)	−21.7% (0.0%, −23.1%)	54,502 (4,198 + 50,304)	−21.7% (0.0%, −23.1%)	0.0% (0.0%, 0.0%)
				32,710 (2,256 + 30,454)	22,670 (2,256 + 20,414)	−30.7% (0.0%, −33.0%)	22,618 (2,256 + 20,362)	−30.9% (0.0%, −33.1%)	−0.2% (0.0%, −0.3%)
				84,642 (13,801 + 70,841)	84,642 (13,801 + 70,841)	0.0% (0.0%, 0.0%)	84,642 (13,801 + 70,841)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	75,146 (50,374 + 24,772)	71,037 (50,374 + 20,663)	−5.5% (0.0%, −16.6%)	71,037 (50,374 + 20,663)	−5.5% (0.0%, −16.6%)	0.0% (0.0%, 0.0%)
				39,566 (25,175 + 14,391)	37,494 (25,175 + 12,319)	−5.2% (0.0%, −14.4%)	37,494 (25,175 + 12,319)	−5.2% (0.0%, −14.4%)	0.0% (0.0%, 0.0%)
				33,627 (20,198 + 13,429)	30,502 (20,198 + 10,304)	−9.3% (0.0%, −23.3%)	30,502 (20,198 + 10,304)	−9.3% (0.0%, −23.3%)	0.0% (0.0%, 0.0%)
				16,710 (10,256 + 6,454)	14,670 (10,256 + 4,414)	−12.2% (0.0%, −31.6%)	14,618 (10,256 + 4,362)	−12.5% (0.0%, −32.4%)	−0.4% (0.0%, −1.2%)
				80,642 (65,801 + 14,841)	80,642 (65,801 + 14,841)	0.0% (0.0%, 0.0%)	80,642 (65,801 + 14,841)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
6	30	1,000	1,000	59,003 (16,627 + 42,376)	53,826 (16,627 + 37,199)	−8.8% (0.0%, −12.2%)	53,826 (16,627 + 37,199)	−8.8% (0.0%, −12.2%)	0.0% (0.0%, 0.0%)
				32,802 (9,252 + 23,550)	28,673 (9,252 + 19,421)	−12.6% (0.0%, −17.5%)	28,673 (9,252 + 19,421)	−12.6% (0.0%, −17.5%)	0.0% (0.0%, 0.0%)
				30,025 (7,280 + 22,745)	23,791 (7,280 + 16,511)	−20.8% (0.0%, −27.4%)	23,802 (7,280 + 16,522)	−20.7% (0.0%, −27.4%)	0.0% (0.0%, 0.1%)
				10,216 (2,388 + 7,828)	8,008 (2,388 + 5,620)	−21.6% (0.0%, −28.2%)	8,008 (2,388 + 5,620)	−21.6% (0.0%, −28.2%)	0.0% (0.0%, 0.0%)
				45,460 (22,210 + 23,250)	45,460 (22,210 + 23,250)	0.0% (0.0%, 0.0%)	45,460 (22,210 + 23,250)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	223,003 (16,627 + 206,376)	197,826 (16,627 + 181,199)	−11.3% (0.0%, −12.2%)	197,826 (16,627 + 181,199)	−11.3% (0.0%, −12.2%)	0.0% (0.0%, 0.0%)
				124,802 (9,252 + 115,550)	104,673 (9,252 + 95,421)	−16.1% (0.0%, −17.4%)	104,673 (9,252 + 95,421)	−16.1% (0.0%, −17.4%)	0.0% (0.0%, 0.0%)
				118,025 (7,280 + 110,745)	87,802 (7,280 + 80,522)	−25.6% (0.0%, −27.3%)	87,791 (7,280 + 80,511)	−25.6% (0.0%, −27.3%)	0.0% (0.0%, 0.0%)
				38,216 (2,388 + 35,828)	28,008 (2,388 + 25,620)	−26.7% (0.0%, −28.5%)	28,008 (2,388 + 25,620)	−26.7% (0.0%, −28.5%)	0.0% (0.0%, 0.0%)
				133,460 (22,210 + 111,250)	133,460 (22,210 + 111,250)	0.0% (0.0%, 0.0%)	133,460 (22,210 + 111,250)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	123,003 (80,627 + 42,376)	117,826 (80,627 + 37,199)	−4.2% (0.0%, −12.2%)	117,826 (80,627 + 37,199)	−4.2% (0.0%, −12.2%)	0.0% (0.0%, 0.0%)
				68,802 (45,252 + 23,550)	64,673 (45,252 + 19,421)	−6.0% (0.0%, −17.5%)	65,677 (45,252 + 20,425)	−4.5% (0.0%, −13.3%)	1.6% (0.0%, 5.2%)
				58,025 (35,280 + 22,745)	51,791 (35,280 + 16,511)	−10.7% (0.0%, −27.4%)	51,802 (35,280 + 16,522)	−10.7% (0.0%, −27.4%)	0.0% (0.0%, 0.1%)
				18,216 (10,388 + 7,828)	16,008 (10,388 + 5,620)	−12.1% (0.0%, −28.2%)	16,008 (10,388 + 5,620)	−12.1% (0.0%, −28.2%)	0.0% (0.0%, 0.0%)
				129,460 (106,210 + 23,250)	129,460 (106,210 + 23,250)	0.0% (0.0%, 0.0%)	129,460 (106,210 + 23,250)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	40	1,000	1,000	74,525 (21,828 + 52,697)	69,345 (21,828 + 47,517)	−7.0% (0.0%, −9.8%)	69,345 (21,828 + 47,517)	−7.0% (0.0%, −9.8%)	0.0% (0.0%, 0.0%)
				38,938 (10,300 + 28,638)	32,810 (10,300 + 22,510)	−15.7% (0.0%, −21.4%)	32,810 (10,300 + 22,510)	−15.7% (0.0%, −21.4%)	0.0% (0.0%, 0.0%)
				37,197 (9,362 + 27,835)	31,030 (9,362 + 21,668)	−16.6% (0.0%, −22.2%)	31,030 (9,362 + 21,668)	−16.6% (0.0%, −22.2%)	0.0% (0.0%, 0.0%)
				18,454 (3,715 + 14,739)	15,009 (3,715 + 11,294)	−18.7% (0.0%, −23.4%)	15,009 (3,715 + 11,294)	−18.7% (0.0%, −23.4%)	0.0% (0.0%, 0.0%)
				57,166 (27,549 + 29,617)	57,166 (27,549 + 29,617)	0.0% (0.0%, 0.0%)	57,166 (27,549 + 29,617)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	278,525 (21,828 + 256,697)	253,345 (21,828 + 231,517)	−9.0% (0.0%, −9.8%)	253,345 (21,828 + 231,517)	−9.0% (0.0%, −9.8%)	0.0% (0.0%, 0.0%)
				150,938 (10,300 + 140,638)	120,810 (10,300 + 110,510)	−20.0% (0.0%, −21.4%)	120,810 (10,300 + 110,510)	−20.0% (0.0%, −21.4%)	0.0% (0.0%, 0.0%)
				145,197 (9,362 + 135,835)	115,030 (9,362 + 105,668)	−20.8% (0.0%, −22.2%)	115,030 (9,362 + 105,668)	−20.8% (0.0%, −22.2%)	0.0% (0.0%, 0.0%)
				70,454 (3,715 + 66,739)	55,009 (3,715 + 51,294)	−21.9% (0.0%, −23.1%)	55,009 (3,715 + 51,294)	−21.9% (0.0%, −23.1%)	0.0% (0.0%, 0.0%)
				169,166 (27,549 + 141,617)	169,166 (27,549 + 141,617)	0.0% (0.0%, 0.0%)	169,166 (27,549 + 141,617)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	158,525 (105,828 + 52,697)	153,345 (105,828 + 47,517)	−3.3% (0.0%, −9.8%)	153,345 (105,828 + 47,517)	−3.3% (0.0%, −9.8%)	0.0% (0.0%, 0.0%)
				78,938 (50,300 + 28,638)	72,810 (50,300 + 22,510)	−7.8% (0.0%, −21.4%)	72,810 (50,300 + 22,510)	−7.8% (0.0%, −21.4%)	0.0% (0.0%, 0.0%)
				73,197 (45,362 + 27,835)	67,030 (45,362 + 21,668)	−8.4% (0.0%, −22.2%)	67,030 (45,362 + 21,668)	−8.4% (0.0%, −22.2%)	0.0% (0.0%, 0.0%)
				30,454 (15,715 + 14,739)	27,009 (15,715 + 11,294)	−11.3% (0.0%, −23.4%)	27,009 (15,715 + 11,294)	−11.3% (0.0%, −23.4%)	0.0% (0.0%, 0.0%)
				161,166 (131,549 + 29,617)	161,166 (131,549 + 29,617)	0.0% (0.0%, 0.0%)	161,166 (131,549 + 29,617)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
6	50	1,000	1,000	98,500 (27,078 + 71,422)	89,262 (27,078 + 62,184)	−9.4% (0.0%, −12.9%)	89,262 (27,078 + 62,184)	−9.4% (0.0%, −12.9%)	0.0% (0.0%, 0.0%)
				50,422 (13,416 + 37,006)	44,164 (13,416 + 30,748)	−12.4% (0.0%, −16.9%)	44,164 (13,416 + 30,748)	−12.4% (0.0%, −16.9%)	0.0% (0.0%, 0.0%)
				48,383 (12,455 + 35,928)	41,230 (12,455 + 28,775)	−14.8% (0.0%, −19.9%)	41,230 (12,455 + 28,775)	−14.8% (0.0%, −19.9%)	0.0% (0.0%, 0.0%)
				160,385 (53,275 + 107,110)	106,550 (53,275 + 53,275)	−33.6% (0.0%, −50.3%)	106,550 (53,275 + 53,275)	−33.6% (0.0%, −50.3%)	0.0% (0.0%, 0.0%)
				77,078 (36,990 + 40,088)	75,008 (36,990 + 38,018)	−2.7% (0.0%, −5.2%)	75,008 (36,990 + 38,018)	−2.7% (0.0%, −5.2%)	0.0% (0.0%, 0.0%)
		1,000	5,000	374,500 (27,078 + 347,422)	329,262 (27,078 + 302,184)	−12.1% (0.0%, −13.0%)	329,262 (27,078 + 302,184)	−12.1% (0.0%, −13.0%)	0.0% (0.0%, 0.0%)
				194,422 (13,416 + 181,006)	164,164 (13,416 + 150,748)	−15.6% (0.0%, −16.7%)	164,164 (13,416 + 150,748)	−15.6% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				188,383 (12,455 + 175,928)	153,230 (12,455 + 140,775)	−18.7% (0.0%, −20.0%)	153,230 (12,455 + 140,775)	−18.7% (0.0%, −20.0%)	0.0% (0.0%, 0.0%)
				564,385 (53,275 + 511,110)	306,550 (53,275 + 253,275)	−45.7% (0.0%, −50.4%)	306,550 (53,275 + 253,275)	−45.7% (0.0%, −50.4%)	0.0% (0.0%, 0.0%)
				229,078 (36,990 + 192,088)	219,008 (36,990 + 182,018)	−4.4% (0.0%, −5.2%)	219,008 (36,990 + 182,018)	−4.4% (0.0%, −5.2%)	0.0% (0.0%, 0.0%)
		5,000	1,000	202,500 (131,078 + 71,422)	193,262 (131,078 + 62,184)	−4.6% (0.0%, −12.9%)	193,262 (131,078 + 62,184)	−4.6% (0.0%, −12.9%)	0.0% (0.0%, 0.0%)
				102,422 (65,416 + 37,006)	96,164 (65,416 + 30,748)	−6.1% (0.0%, −16.9%)	96,164 (65,416 + 30,748)	−6.1% (0.0%, −16.9%)	0.0% (0.0%, 0.0%)
				96,383 (60,455 + 35,928)	89,230 (60,455 + 28,775)	−7.4% (0.0%, −19.9%)	89,230 (60,455 + 28,775)	−7.4% (0.0%, −19.9%)	0.0% (0.0%, 0.0%)
				360,385 (253,275 + 107,110)	306,550 (253,275 + 53,275)	−14.9% (0.0%, −50.3%)	306,550 (253,275 + 53,275)	−14.9% (0.0%, −50.3%)	0.0% (0.0%, 0.0%)
				217,078 (176,990 + 40,088)	215,008 (176,990 + 38,018)	−1.0% (0.0%, −5.2%)	215,008 (176,990 + 38,018)	−1.0% (0.0%, −5.2%)	0.0% (0.0%, 0.0%)

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible			Flexible				
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)		Objective Value	Change (Fixed)		Change (Semi-flexible)	
10	5	1,000	1,000	4,360 (1,090 + 3,270)	4,308 (1,090 + 3,218)	−1.2% (0.0%, −1.6%)		4,291 (1,093 + 3,198)	−1.6% (0.3%, −2.2%)		−0.4% (0.3%, −0.6%)	
				7,440 (1,093 + 6,347)	6,405 (1,093 + 5,312)	−13.9% (0.0%, −16.3%)		6,380 (1,096 + 5,284)	−14.2% (0.3%, −16.7%)		−0.4% (0.3%, −0.5%)	
				7,509 (1,111 + 6,398)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)		6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				7,395 (2,112 + 5,283)	6,336 (2,112 + 4,224)	−14.3% (0.0%, −20.0%)		5,307 (2,130 + 3,177)	−28.2% (0.9%, −39.9%)		−16.2% (0.9%, −24.8%)	
				6,327 (2,109 + 4,218)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)		6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	16,360 (1,090 + 15,270)	16,308 (1,090 + 15,218)	−0.3% (0.0%, −0.3%)		16,291 (1,093 + 15,198)	−0.4% (0.3%, −0.5%)		−0.1% (0.3%, −0.1%)	
				31,440 (1,093 + 30,347)	26,405 (1,093 + 25,312)	−16.0% (0.0%, −16.6%)		26,380 (1,096 + 25,284)	−16.1% (0.3%, −16.7%)		−0.1% (0.3%, −0.1%)	
				31,509 (1,111 + 30,398)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)		26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				27,395 (2,112 + 25,283)	22,336 (2,112 + 20,224)	−18.5% (0.0%, −20.0%)		17,307 (2,130 + 15,177)	−36.8% (0.9%, −40.0%)		−22.5% (0.9%, −25.0%)	
				22,327 (2,109 + 20,218)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)		22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	8,360 (5,090 + 3,270)	8,308 (5,090 + 3,218)	−0.6% (0.0%, −1.6%)		8,291 (5,093 + 3,198)	−0.8% (0.1%, −2.2%)		−0.2% (0.1%, −0.6%)	
				11,440 (5,093 + 6,347)	10,405 (5,093 + 5,312)	−9.0% (0.0%, −16.3%)		10,380 (5,096 + 5,284)	−9.3% (0.1%, −16.7%)		−0.2% (0.1%, −0.5%)	
				11,509 (5,111 + 6,398)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)		10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				15,395 (10,112 + 5,283)	14,336 (10,112 + 4,224)	−6.9% (0.0%, −20.0%)		13,307 (10,130 + 3,177)	−13.6% (0.2%, −39.9%)		−7.2% (0.2%, −24.8%)	
				14,327 (10,109 + 4,218)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)		14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
10	10	1,000	1,000	9,529 (2,173 + 7,356)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)		7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)		0.0% (0.0%, 0.0%)	
				10,923 (1,166 + 9,757)	10,923 (1,166 + 9,757)	0.0% (0.0%, 0.0%)		10,923 (1,166 + 9,757)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				12,744 (2,178 + 10,566)	11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)		11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)		0.0% (0.0%, 0.0%)	
				8,655 (2,210 + 6,445)	8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)		8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)		0.0% (0.0%, 0.0%)	
				9,644 (2,227 + 7,417)	8,609 (2,227 + 6,382)	−10.7% (0.0%, −14.0%)		8,609 (2,227 + 6,382)	−10.7% (0.0%, −14.0%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	37,529 (2,173 + 35,356)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)		27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)		0.0% (0.0%, 0.0%)	
				46,923 (1,166 + 45,757)	46,923 (1,166 + 45,757)	0.0% (0.0%, 0.0%)		46,923 (1,166 + 45,757)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				52,744 (2,178 + 50,566)	47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)		47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)		0.0% (0.0%, 0.0%)	
				32,655 (2,210 + 30,445)	32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)		32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)		0.0% (0.0%, 0.0%)	
				37,644 (2,227 + 35,417)	32,609 (2,227 + 30,382)	−13.4% (0.0%, −14.2%)		32,609 (2,227 + 30,382)	−13.4% (0.0%, −14.2%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	17,529 (10,173 + 7,356)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)		15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)		0.0% (0.0%, 0.0%)	
				14,923 (5,166 + 9,757)	14,923 (5,166 + 9,757)	0.0% (0.0%, 0.0%)		14,923 (5,166 + 9,757)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				20,744 (10,178 + 10,566)	19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)		19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)		0.0% (0.0%, 0.0%)	
				16,655 (10,210 + 6,445)	16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)		16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)		0.0% (0.0%, 0.0%)	
				17,644 (10,227 + 7,417)	16,609 (10,227 + 6,382)	−5.9% (0.0%, −14.0%)		16,609 (10,227 + 6,382)	−5.9% (0.0%, −14.0%)		0.0% (0.0%, 0.0%)	

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	20	1,000	1,000	15,161 (3,310 + 11,851)	14,088 (3,310 + 10,778)	−7.1% (0.0%, −9.1%)	14,088 (3,310 + 10,778)	−7.1% (0.0%, −9.1%)	0.0% (0.0%, 0.0%)
				18,302 (2,255 + 16,047)	16,167 (2,255 + 13,912)	−11.7% (0.0%, −13.3%)	16,167 (2,255 + 13,912)	−11.7% (0.0%, −13.3%)	0.0% (0.0%, 0.0%)
				22,384 (3,345 + 19,039)	21,355 (3,345 + 18,010)	−4.6% (0.0%, −5.4%)	21,355 (3,345 + 18,010)	−4.6% (0.0%, −5.4%)	0.0% (0.0%, 0.0%)
				20,584 (4,444 + 16,140)	18,465 (4,444 + 14,021)	−10.3% (0.0%, −13.1%)	18,465 (4,444 + 14,021)	−10.3% (0.0%, −13.1%)	0.0% (0.0%, 0.0%)
				21,662 (5,490 + 16,172)	21,614 (5,490 + 16,124)	−0.2% (0.0%, −0.3%)	21,614 (5,490 + 16,124)	−0.2% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
		1,000	5,000	59,161 (3,310 + 55,851)	54,088 (3,310 + 50,778)	−8.6% (0.0%, −9.1%)	54,088 (3,310 + 50,778)	−8.6% (0.0%, −9.1%)	0.0% (0.0%, 0.0%)
				78,302 (2,255 + 76,047)	68,167 (2,255 + 65,912)	−12.9% (0.0%, −13.3%)	68,167 (2,255 + 65,912)	−12.9% (0.0%, −13.3%)	0.0% (0.0%, 0.0%)
				94,384 (3,345 + 91,039)	89,355 (3,345 + 86,010)	−5.3% (0.0%, −5.5%)	89,355 (3,345 + 86,010)	−5.3% (0.0%, −5.5%)	0.0% (0.0%, 0.0%)
				80,584 (4,444 + 76,140)	70,465 (4,444 + 66,021)	−12.6% (0.0%, −13.3%)	70,465 (4,444 + 66,021)	−12.6% (0.0%, −13.3%)	0.0% (0.0%, 0.0%)
				81,662 (5,490 + 76,172)	81,614 (5,490 + 76,124)	−0.1% (0.0%, −0.1%)	81,614 (5,490 + 76,124)	−0.1% (0.0%, −0.1%)	0.0% (0.0%, 0.0%)
		5,000	1,000	27,161 (15,310 + 11,851)	26,088 (15,310 + 10,778)	−4.0% (0.0%, −9.1%)	26,088 (15,310 + 10,778)	−4.0% (0.0%, −9.1%)	0.0% (0.0%, 0.0%)
				26,302 (10,255 + 16,047)	24,167 (10,255 + 13,912)	−8.1% (0.0%, −13.3%)	24,167 (10,255 + 13,912)	−8.1% (0.0%, −13.3%)	0.0% (0.0%, 0.0%)
				34,378 (15,345 + 19,033)	33,355 (15,345 + 18,010)	−3.0% (0.0%, −5.4%)	33,355 (15,345 + 18,010)	−3.0% (0.0%, −5.4%)	0.0% (0.0%, 0.0%)
				36,584 (20,444 + 16,140)	34,465 (20,444 + 14,021)	−5.8% (0.0%, −13.1%)	34,465 (20,444 + 14,021)	−5.8% (0.0%, −13.1%)	0.0% (0.0%, 0.0%)
				41,662 (25,490 + 16,172)	41,614 (25,490 + 16,124)	−0.1% (0.0%, −0.3%)	41,614 (25,490 + 16,124)	−0.1% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
10	30	1,000	1,000	33,840 (7,550 + 26,290)	27,598 (7,550 + 20,048)	−18.4% (0.0%, −23.7%)	27,598 (7,550 + 20,048)	−18.4% (0.0%, −23.7%)	0.0% (0.0%, 0.0%)
				36,147 (5,488 + 30,659)	32,997 (5,488 + 27,509)	−8.7% (0.0%, −10.3%)	32,997 (5,488 + 27,509)	−8.7% (0.0%, −10.3%)	0.0% (0.0%, 0.0%)
				38,482 (5,549 + 32,933)	36,512 (5,549 + 30,963)	−5.1% (0.0%, −6.0%)	36,512 (5,549 + 30,963)	−5.1% (0.0%, −6.0%)	0.0% (0.0%, 0.0%)
				30,286 (6,658 + 23,628)	27,057 (6,658 + 20,399)	−10.7% (0.0%, −13.7%)	27,057 (6,658 + 20,399)	−10.7% (0.0%, −13.7%)	0.0% (0.0%, 0.0%)
				22,486 (5,504 + 16,982)	21,465 (5,504 + 15,961)	−4.5% (0.0%, −6.0%)	21,465 (5,504 + 15,961)	−4.5% (0.0%, −6.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	133,840 (7,550 + 126,290)	103,598 (7,550 + 96,048)	−22.6% (0.0%, −23.9%)	103,598 (7,550 + 96,048)	−22.6% (0.0%, −23.9%)	0.0% (0.0%, 0.0%)
				152,147 (5,488 + 146,659)	136,997 (5,488 + 131,509)	−10.0% (0.0%, −10.3%)	136,997 (5,488 + 131,509)	−10.0% (0.0%, −10.3%)	0.0% (0.0%, 0.0%)
				162,482 (5,549 + 156,933)	152,512 (5,549 + 146,963)	−6.1% (0.0%, −6.4%)	152,512 (5,549 + 146,963)	−6.1% (0.0%, −6.4%)	0.0% (0.0%, 0.0%)
				118,286 (6,658 + 111,628)	103,057 (6,658 + 96,399)	−12.9% (0.0%, −13.6%)	103,057 (6,658 + 96,399)	−12.9% (0.0%, −13.6%)	0.0% (0.0%, 0.0%)
				86,486 (5,504 + 80,982)	81,465 (5,504 + 75,961)	−5.8% (0.0%, −6.2%)	81,465 (5,504 + 75,961)	−5.8% (0.0%, −6.2%)	0.0% (0.0%, 0.0%)
		5,000	1,000	61,840 (35,550 + 26,290)	55,598 (35,550 + 20,048)	−10.1% (0.0%, −23.7%)	55,598 (35,550 + 20,048)	−10.1% (0.0%, −23.7%)	0.0% (0.0%, 0.0%)
				56,147 (25,488 + 30,659)	52,997 (25,488 + 27,509)	−5.6% (0.0%, −10.3%)	52,997 (25,488 + 27,509)	−5.6% (0.0%, −10.3%)	0.0% (0.0%, 0.0%)
				58,482 (25,549 + 32,933)	56,512 (25,549 + 30,963)	−3.4% (0.0%, −6.0%)	56,512 (25,549 + 30,963)	−3.4% (0.0%, −6.0%)	0.0% (0.0%, 0.0%)
				54,286 (30,658 + 23,628)	51,057 (30,658 + 20,399)	−5.9% (0.0%, −13.7%)	51,057 (30,658 + 20,399)	−5.9% (0.0%, −13.7%)	0.0% (0.0%, 0.0%)
				42,486 (25,504 + 16,982)	41,465 (25,504 + 15,961)	−2.4% (0.0%, −6.0%)	41,465 (25,504 + 15,961)	−2.4% (0.0%, −6.0%)	0.0% (0.0%, 0.0%)

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	40	1,000	1,000	52,853 (11,765 + 41,088)	43,349 (11,765 + 31,584)	−18.0% (0.0%, −23.1%)	43,349 (11,765 + 31,584)	−18.0% (0.0%, −23.1%)	0.0% (0.0%, 0.0%)
				46,823 (7,660 + 39,163)	45,764 (7,660 + 38,104)	−2.3% (0.0%, −2.7%)	45,764 (7,660 + 38,104)	−2.3% (0.0%, −2.7%)	0.0% (0.0%, 0.0%)
				48,067 (6,697 + 41,370)	42,906 (6,697 + 36,209)	−10.7% (0.0%, −12.5%)	42,906 (6,697 + 36,209)	−10.7% (0.0%, −12.5%)	0.0% (0.0%, 0.0%)
				46,011 (10,905 + 35,106)	40,660 (10,905 + 29,755)	−11.6% (0.0%, −15.2%)	40,660 (10,905 + 29,755)	−11.6% (0.0%, −15.2%)	0.0% (0.0%, 0.0%)
				33,405 (7,755 + 25,650)	30,269 (7,755 + 22,514)	−9.4% (0.0%, −12.2%)	30,269 (7,755 + 22,514)	−9.4% (0.0%, −12.2%)	0.0% (0.0%, 0.0%)
		1,000	5,000	208,853 (11,765 + 197,088)	163,349 (11,765 + 151,584)	−21.8% (0.0%, −23.1%)	163,349 (11,765 + 151,584)	−21.8% (0.0%, −23.1%)	0.0% (0.0%, 0.0%)
				194,823 (7,660 + 187,163)	189,764 (7,660 + 182,104)	−2.6% (0.0%, −2.7%)	189,764 (7,660 + 182,104)	−2.6% (0.0%, −2.7%)	0.0% (0.0%, 0.0%)
				204,067 (6,697 + 197,370)	178,906 (6,697 + 172,209)	−12.3% (0.0%, −12.7%)	178,906 (6,697 + 172,209)	−12.3% (0.0%, −12.7%)	0.0% (0.0%, 0.0%)
				178,011 (10,905 + 167,106)	152,660 (10,905 + 141,755)	−14.2% (0.0%, −15.2%)	152,660 (10,905 + 141,755)	−14.2% (0.0%, −15.2%)	0.0% (0.0%, 0.0%)
				129,405 (7,755 + 121,650)	114,269 (7,755 + 106,514)	−11.7% (0.0%, −12.4%)	114,269 (7,755 + 106,514)	−11.7% (0.0%, −12.4%)	0.0% (0.0%, 0.0%)
		5,000	1,000	96,853 (55,765 + 41,088)	87,349 (55,765 + 31,584)	−9.8% (0.0%, −23.1%)	87,349 (55,765 + 31,584)	−9.8% (0.0%, −23.1%)	0.0% (0.0%, 0.0%)
				74,823 (35,660 + 39,163)	73,764 (35,660 + 38,104)	−1.4% (0.0%, −2.7%)	73,764 (35,660 + 38,104)	−1.4% (0.0%, −2.7%)	0.0% (0.0%, 0.0%)
				72,067 (30,697 + 41,370)	66,906 (30,697 + 36,209)	−7.2% (0.0%, −12.5%)	66,906 (30,697 + 36,209)	−7.2% (0.0%, −12.5%)	0.0% (0.0%, 0.0%)
				86,011 (50,905 + 35,106)	80,660 (50,905 + 29,755)	−6.2% (0.0%, −15.2%)	80,660 (50,905 + 29,755)	−6.2% (0.0%, −15.2%)	0.0% (0.0%, 0.0%)
				61,418 (35,755 + 25,663)	58,269 (35,755 + 22,514)	−5.1% (0.0%, −12.3%)	58,269 (35,755 + 22,514)	−5.1% (0.0%, −12.3%)	0.0% (0.0%, 0.0%)
10	50	1,000	1,000	72,725 (18,078 + 54,647)	63,221 (18,078 + 45,143)	−13.1% (0.0%, −17.4%)	63,221 (18,078 + 45,143)	−13.1% (0.0%, −17.4%)	0.0% (0.0%, 0.0%)
				67,107 (10,831 + 56,276)	59,744 (10,831 + 48,913)	−11.0% (0.0%, −13.1%)	59,744 (10,831 + 48,913)	−11.0% (0.0%, −13.1%)	0.0% (0.0%, 0.0%)
				61,639 (8,844 + 52,795)	57,523 (8,844 + 48,679)	−6.7% (0.0%, −7.8%)	57,523 (8,844 + 48,679)	−6.7% (0.0%, −7.8%)	0.0% (0.0%, 0.0%)
				52,691 (12,128 + 40,563)	46,366 (12,128 + 34,238)	−12.0% (0.0%, −15.6%)	46,366 (12,128 + 34,238)	−12.0% (0.0%, −15.6%)	0.0% (0.0%, 0.0%)
				43,265 (9,987 + 33,278)	40,961 (9,987 + 30,974)	−5.3% (0.0%, −6.9%)	40,961 (9,987 + 30,974)	−5.3% (0.0%, −6.9%)	0.0% (0.0%, 0.0%)
		1,000	5,000	280,725 (18,078 + 262,647)	235,221 (18,078 + 217,143)	−16.2% (0.0%, −17.3%)	235,221 (18,078 + 217,143)	−16.2% (0.0%, −17.3%)	0.0% (0.0%, 0.0%)
				279,107 (10,831 + 268,276)	243,744 (10,831 + 232,913)	−12.7% (0.0%, −13.2%)	243,744 (10,831 + 232,913)	−12.7% (0.0%, −13.2%)	0.0% (0.0%, 0.0%)
				261,639 (8,844 + 252,795)	241,523 (8,844 + 232,679)	−7.7% (0.0%, −8.0%)	241,523 (8,844 + 232,679)	−7.7% (0.0%, −8.0%)	0.0% (0.0%, 0.0%)
				204,691 (12,128 + 192,563)	174,366 (12,128 + 162,238)	−14.8% (0.0%, −15.7%)	174,366 (12,128 + 162,238)	−14.8% (0.0%, −15.7%)	0.0% (0.0%, 0.0%)
				167,265 (9,987 + 157,278)	156,961 (9,987 + 146,974)	−6.2% (0.0%, −6.6%)	156,961 (9,987 + 146,974)	−6.2% (0.0%, −6.6%)	0.0% (0.0%, 0.0%)
		5,000	1,000	140,725 (86,078 + 54,647)	131,221 (86,078 + 45,143)	−6.8% (0.0%, −17.4%)	131,221 (86,078 + 45,143)	−6.8% (0.0%, −17.4%)	0.0% (0.0%, 0.0%)
				107,107 (50,831 + 56,276)	99,744 (50,831 + 48,913)	−6.9% (0.0%, −13.1%)	99,744 (50,831 + 48,913)	−6.9% (0.0%, −13.1%)	0.0% (0.0%, 0.0%)
				93,639 (40,844 + 52,795)	89,523 (40,844 + 48,679)	−4.4% (0.0%, −7.8%)	89,523 (40,844 + 48,679)	−4.4% (0.0%, −7.8%)	0.0% (0.0%, 0.0%)
				96,691 (56,128 + 40,563)	90,366 (56,128 + 34,238)	−6.5% (0.0%, −15.6%)	90,366 (56,128 + 34,238)	−6.5% (0.0%, −15.6%)	0.0% (0.0%, 0.0%)
				79,265 (45,987 + 33,278)	76,961 (45,987 + 30,974)	−2.9% (0.0%, −6.9%)	76,961 (45,987 + 30,974)	−2.9% (0.0%, −6.9%)	0.0% (0.0%, 0.0%)

Table 3.8: Comparison of the solutions from CP-BB. Vehicle and crew objective values are shown inside parenthesis.

				Fixed			Semi-flexible				Flexible					
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value			Objective Value			Change (Fixed)	Objective Value			Change (Fixed)		Change (Semi-flexible)	
6	5	1,000	1,000	8,312 (	2,078 +	6,234)	7,268 (	2,078 +	5,190)	−12.6% (0.0%, −16.7%)	7,268 (	2,078 +	5,190)	−12.6% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				4,166 (	1,047 +	3,119)	4,144 (	1,047 +	3,097)	−0.5% (0.0%, −0.7%)	4,135 (	1,048 +	3,087)	−0.7% (0.1%, −1.0%)		−0.2% (0.1%, −0.3%)	
				4,098 (	1,035 +	3,063)	3,105 (	1,035 +	2,070)	−24.2% (0.0%, −32.4%)	3,089 (	1,037 +	2,052)	−24.6% (0.2%, −33.0%)		−0.5% (0.2%, −0.9%)	
				4,410 (	1,151 +	3,259)	3,453 (	1,151 +	2,302)	−21.7% (0.0%, −29.4%)	3,352 (	1,159 +	2,193)	−24.0% (0.7%, −32.7%)		−2.9% (0.7%, −4.7%)	
				6,404 (	3,202 +	3,202)	6,404 (	3,202 +	3,202)	0.0% (0.0%, 0.0%)	6,404 (	3,202 +	3,202)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	32,312 (	2,078 +	30,234)	27,268 (	2,078 +	25,190)	−15.6% (0.0%, −16.7%)	27,268 (	2,078 +	25,190)	−15.6% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				16,166 (	1,047 +	15,119)	16,144 (	1,047 +	15,097)	−0.1% (0.0%, −0.1%)	16,135 (	1,048 +	15,087)	−0.2% (0.1%, −0.2%)		−0.1% (0.1%, −0.1%)	
				16,098 (	1,035 +	15,063)	11,105 (	1,035 +	10,070)	−31.0% (0.0%, −33.1%)	11,089 (	1,037 +	10,052)	−31.1% (0.2%, −33.3%)		−0.1% (0.2%, −0.2%)	
				16,410 (	1,151 +	15,259)	11,453 (	1,151 +	10,302)	−30.2% (0.0%, −32.5%)	11,352 (	1,159 +	10,193)	−30.8% (0.7%, −33.2%)		−0.9% (0.7%, −1.1%)	
				18,404 (	3,202 +	15,202)	18,404 (	3,202 +	15,202)	0.0% (0.0%, 0.0%)	18,404 (	3,202 +	15,202)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	16,312 (	10,078 +	6,234)	15,268 (	10,078 +	5,190)	−6.4% (0.0%, −16.7%)	15,268 (	10,078 +	5,190)	−6.4% (0.0%, −16.7%)		0.0% (0.0%, 0.0%)	
				8,166 (	5,047 +	3,119)	8,144 (	5,047 +	3,097)	−0.3% (0.0%, −0.7%)	8,135 (	5,048 +	3,087)	−0.4% (0.0%, −1.0%)		−0.1% (0.0%, −0.3%)	
				8,098 (	5,035 +	3,063)	7,105 (	5,035 +	2,070)	−12.3% (0.0%, −32.4%)	7,089 (	5,037 +	2,052)	−12.5% (0.0%, −33.0%)		−0.2% (0.0%, −0.9%)	
				8,410 (	5,151 +	3,259)	7,453 (	5,151 +	2,302)	−11.4% (0.0%, −29.4%)	7,352 (	5,159 +	2,193)	−12.6% (0.2%, −32.7%)		−1.4% (0.2%, −4.7%)	
				18,404 (	15,202 +	3,202)	18,404 (	15,202 +	3,202)	0.0% (0.0%, 0.0%)	18,404 (	15,202 +	3,202)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
6	10	1,000	1,000	13,462 (	4,160 +	9,302)	13,462 (	4,160 +	9,302)	0.0% (0.0%, 0.0%)	13,462 (	4,160 +	9,302)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				8,194 (	2,064 +	6,130)	8,189 (	2,064 +	6,125)	−0.1% (0.0%, −0.1%)	7,200 (	2,068 +	5,132)	−12.1% (0.2%, −16.3%)		−12.1% (0.2%, −16.2%)	
				7,191 (	2,071 +	5,120)	6,176 (	2,071 +	4,105)	−14.1% (0.0%, −19.8%)	6,175 (	2,075 +	4,100)	−14.1% (0.2%, −19.9%)		0.0% (0.2%, −0.1%)	
				5,689 (	1,196 +	4,493)	4,590 (	1,196 +	3,394)	−19.3% (0.0%, −24.5%)	4,568 (	1,205 +	3,363)	−19.7% (0.8%, −25.2%)		−0.5% (0.8%, −0.9%)	
				13,776 (	6,380 +	7,396)	12,760 (	6,380 +	6,380)	−7.4% (0.0%, −13.7%)	12,760 (	6,380 +	6,380)	−7.4% (0.0%, −13.7%)		0.0% (0.0%, 0.0%)	
		1,000	5,000	49,462 (	4,160 +	45,302)	49,462 (	4,160 +	45,302)	0.0% (0.0%, 0.0%)	49,462 (	4,160 +	45,302)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				32,194 (	2,064 +	30,130)	32,189 (	2,064 +	30,125)	0.0% (0.0%, 0.0%)	27,200 (	2,068 +	25,132)	−15.5% (0.2%, −16.6%)		−15.5% (0.2%, −16.6%)	
				27,191 (	2,071 +	25,120)	22,176 (	2,071 +	20,105)	−18.4% (0.0%, −20.0%)	22,176 (	2,071 +	20,105)	−18.4% (0.0%, −20.0%)		0.0% (0.0%, 0.0%)	
				21,689 (	1,196 +	20,493)	16,590 (	1,196 +	15,394)	−23.5% (0.0%, −24.9%)	16,568 (	1,205 +	15,363)	−23.6% (0.8%, −25.0%)		−0.1% (0.8%, −0.2%)	
				41,776 (	6,380 +	35,396)	36,760 (	6,380 +	30,380)	−12.0% (0.0%, −14.2%)	36,760 (	6,380 +	30,380)	−12.0% (0.0%, −14.2%)		0.0% (0.0%, 0.0%)	
		5,000	1,000	29,462 (	20,160 +	9,302)	29,462 (	20,160 +	9,302)	0.0% (0.0%, 0.0%)	29,462 (	20,160 +	9,302)	0.0% (0.0%, 0.0%)		0.0% (0.0%, 0.0%)	
				16,194 (	10,064 +	6,130)	16,189 (	10,064 +	6,125)	0.0% (0.0%, −0.1%)	15,200 (	10,068 +	5,132)	−6.1% (0.0%, −16.3%)		−6.1% (0.0%, −16.2%)	
				15,191 (	10,071 +	5,120)	14,176 (	10,071 +	4,105)	−6.7% (0.0%, −19.8%)	14,176 (	10,071 +	4,105)	−6.7% (0.0%, −19.8%)		0.0% (0.0%, 0.0%)	
				9,689 (	5,196 +	4,493)	8,590 (	5,196 +	3,394)	−11.3% (0.0%, −24.5%)	8,566 (	5,204 +	3,362)	−11.6% (0.2%, −25.2%)		−0.3% (0.2%, −0.9%)	
				37,776 (	30,380 +	7,396)	36,760 (	30,380 +	6,380)	−2.7% (0.0%, −13.7%)	36,760 (	30,380 +	6,380)	−2.7% (0.0%, −13.7%)		0.0% (0.0%, 0.0%)	

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	20	1,000	1,000	31,975 (9,364 + 22,611)	28,933 (9,364 + 19,569)	−9.5% (0.0%, −13.5%)	29,971 (9,364 + 20,607)	−6.3% (0.0%, −8.9%)	3.6% (0.0%, 5.3%)
				18,358 (5,144 + 13,214)	14,336 (5,144 + 9,192)	−21.9% (0.0%, −30.4%)	15,332 (5,144 + 10,188)	−16.5% (0.0%, −22.9%)	6.9% (0.0%, 10.8%)
				16,362 (4,136 + 12,226)	14,338 (4,136 + 10,202)	−12.4% (0.0%, −16.6%)	15,360 (4,136 + 11,224)	−6.1% (0.0%, −8.2%)	7.1% (0.0%, 10.0%)
				16,215 (4,419 + 11,796)	9,917 (4,419 + 5,498)	−38.8% (0.0%, −53.4%)	9,917 (4,419 + 5,498)	−38.8% (0.0%, −53.4%)	0.0% (0.0%, 0.0%)
				24,576 (11,780 + 12,796)	23,560 (11,780 + 11,780)	−4.1% (0.0%, −7.9%)	23,560 (11,780 + 11,780)	−4.1% (0.0%, −7.9%)	0.0% (0.0%, 0.0%)
		1,000	5,000	119,969 (9,364 + 110,605)	104,927 (9,364 + 95,563)	−12.5% (0.0%, −13.6%)	124,994 (9,364 + 115,630)	4.2% (0.0%, 4.5%)	19.1% (0.0%, 21.0%)
				70,358 (5,144 + 65,214)	50,336 (5,144 + 45,192)	−28.5% (0.0%, −30.7%)	55,332 (5,144 + 50,188)	−21.4% (0.0%, −23.0%)	9.9% (0.0%, 11.1%)
				64,362 (4,136 + 60,226)	54,338 (4,136 + 50,202)	−15.6% (0.0%, −16.6%)	54,339 (4,136 + 50,203)	−15.6% (0.0%, −16.6%)	0.0% (0.0%, 0.0%)
				60,215 (4,419 + 55,796)	29,917 (4,419 + 25,498)	−50.3% (0.0%, −54.3%)	29,917 (4,419 + 25,498)	−50.3% (0.0%, −54.3%)	0.0% (0.0%, 0.0%)
				72,576 (11,780 + 60,796)	67,560 (11,780 + 55,780)	−6.9% (0.0%, −8.3%)	67,560 (11,780 + 55,780)	−6.9% (0.0%, −8.3%)	0.0% (0.0%, 0.0%)
		5,000	1,000	67,977 (45,364 + 22,613)	64,935 (45,364 + 19,571)	−4.5% (0.0%, −13.5%)	67,978 (45,364 + 22,614)	0.0% (0.0%, 0.0%)	4.7% (0.0%, 15.5%)
				38,358 (25,144 + 13,214)	34,336 (25,144 + 9,192)	−10.5% (0.0%, −30.4%)	35,338 (25,144 + 10,194)	−7.9% (0.0%, −22.9%)	2.9% (0.0%, 10.9%)
				32,362 (20,136 + 12,226)	30,338 (20,136 + 10,202)	−6.3% (0.0%, −16.6%)	30,339 (20,136 + 10,203)	−6.3% (0.0%, −16.5%)	0.0% (0.0%, 0.0%)
				32,215 (20,419 + 11,796)	25,917 (20,419 + 5,498)	−19.5% (0.0%, −53.4%)	26,951 (20,419 + 6,532)	−16.3% (0.0%, −44.6%)	4.0% (0.0%, 18.8%)
				68,576 (55,780 + 12,796)	67,560 (55,780 + 11,780)	−1.5% (0.0%, −7.9%)	67,560 (55,780 + 11,780)	−1.5% (0.0%, −7.9%)	0.0% (0.0%, 0.0%)
6	30	1,000	1,000	50,532 (14,553 + 35,979)	50,558 (14,553 + 36,005)	0.1% (0.0%, 0.1%)	55,602 (14,553 + 41,049)	10.0% (0.0%, 14.1%)	10.0% (0.0%, 14.0%)
				32,574 (9,227 + 23,347)	24,508 (9,227 + 15,281)	−24.8% (0.0%, −34.5%)	28,551 (9,227 + 19,324)	−12.4% (0.0%, −17.2%)	16.5% (0.0%, 26.5%)
				27,687 (7,270 + 20,417)	23,615 (7,270 + 16,345)	−14.7% (0.0%, −19.9%)	25,637 (7,270 + 18,367)	−7.4% (0.0%, −10.0%)	8.6% (0.0%, 12.4%)
				17,439 (4,565 + 12,874)	12,245 (4,565 + 7,680)	−29.8% (0.0%, −40.3%)	14,389 (4,565 + 9,824)	−17.5% (0.0%, −23.7%)	17.5% (0.0%, 27.9%)
				43,380 (20,157 + 23,223)	60,770 (20,157 + 40,613)	40.1% (0.0%, 74.9%)	71,975 (20,157 + 51,818)	65.9% (0.0%, 123.1%)	18.4% (0.0%, 27.6%)
		1,000	5,000	185,519 (14,553 + 170,966)	170,470 (14,553 + 155,917)	−8.1% (0.0%, −8.8%)	175,488 (14,553 + 160,935)	−5.4% (0.0%, −5.9%)	2.9% (0.0%, 3.2%)
				124,574 (9,227 + 115,347)	84,509 (9,227 + 75,282)	−32.2% (0.0%, −34.7%)	124,563 (9,227 + 115,336)	0.0% (0.0%, 0.0%)	47.4% (0.0%, 53.2%)
				107,725 (7,270 + 100,455)	87,622 (7,270 + 80,352)	−18.7% (0.0%, −20.0%)	87,644 (7,270 + 80,374)	−18.6% (0.0%, −20.0%)	0.0% (0.0%, 0.0%)
				65,439 (4,565 + 60,874)	40,232 (4,565 + 35,667)	−38.5% (0.0%, −41.4%)	40,257 (4,565 + 35,692)	−38.5% (0.0%, −41.4%)	0.1% (0.0%, 0.1%)
				126,358 (20,157 + 106,201)	206,676 (20,157 + 186,519)	63.6% (0.0%, 75.6%)	291,960 (20,157 + 271,803)	131.1% (0.0%, 155.9%)	41.3% (0.0%, 45.7%)
		5,000	1,000	106,548 (70,553 + 35,995)	105,524 (70,553 + 34,971)	−1.0% (0.0%, −2.8%)	114,648 (70,553 + 44,095)	7.6% (0.0%, 22.5%)	8.6% (0.0%, 26.1%)
				68,571 (45,227 + 23,344)	60,508 (45,227 + 15,281)	−11.8% (0.0%, −34.5%)	62,569 (45,227 + 17,342)	−8.8% (0.0%, −25.7%)	3.4% (0.0%, 13.5%)
				56,677 (35,270 + 21,407)	51,613 (35,270 + 16,343)	−8.9% (0.0%, −23.7%)	56,665 (35,270 + 21,395)	0.0% (0.0%, −0.1%)	9.8% (0.0%, 30.9%)
				33,439 (20,565 + 12,874)	28,244 (20,565 + 7,679)	−15.5% (0.0%, −40.4%)	29,314 (20,565 + 8,749)	−12.3% (0.0%, −32.0%)	3.8% (0.0%, 13.9%)
				117,330 (96,157 + 21,173)	136,725 (96,157 + 40,568)	16.5% (0.0%, 91.6%)	145,963 (96,157 + 49,806)	24.4% (0.0%, 135.2%)	6.8% (0.0%, 22.8%)

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	40	1,000	1,000	–	–	–	–	–	–
				57,886 (12,315 + 45,571)	55,850 (12,315 + 43,535)	−3.5% (0.0%, −4.5%)	58,239 (12,315 + 45,924)	0.6% (0.0%, 0.8%)	4.3% (0.0%, 5.5%)
				–	–	–	–	–	–
				47,557 (9,184 + 38,373)	72,620 (9,184 + 63,436)	52.7% (0.0%, 65.3%)	72,620 (9,184 + 63,436)	52.7% (0.0%, 65.3%)	0.0% (0.0%, 0.0%)
				51,948 (25,459 + 26,489)	118,976 (25,459 + 93,517)	129.0% (0.0%, 253.0%)	118,976 (25,459 + 93,517)	129.0% (0.0%, 253.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	–	–	–	–	–	–
				217,881 (12,313 + 205,568)	233,184 (12,315 + 220,869)	7.0% (0.0%, 7.4%)	247,890 (12,315 + 235,575)	13.8% (0.0%, 14.6%)	6.3% (0.0%, 6.7%)
				185,994 (10,378 + 175,616)	251,024 (10,378 + 240,646)	35.0% (0.0%, 37.0%)	306,049 (10,378 + 295,671)	64.5% (0.0%, 68.4%)	21.9% (0.0%, 22.9%)
				151,207 (9,184 + 142,023)	304,620 (9,184 + 295,436)	101.5% (0.0%, 108.0%)	304,620 (9,184 + 295,436)	101.5% (0.0%, 108.0%)	0.0% (0.0%, 0.0%)
				322,492 (25,459 + 297,033)	474,976 (25,459 + 449,517)	47.3% (0.0%, 51.3%)	474,976 (25,459 + 449,517)	47.3% (0.0%, 51.3%)	0.0% (0.0%, 0.0%)
		5,000	1,000	–	–	–	–	–	–
				103,880 (60,315 + 43,565)	99,846 (60,315 + 39,531)	−3.9% (0.0%, −9.3%)	107,863 (60,315 + 47,548)	3.8% (0.0%, 9.1%)	8.0% (0.0%, 20.3%)
				85,995 (50,378 + 35,617)	–	–	–	–	–
				76,454 (41,184 + 35,270)	104,620 (41,184 + 63,436)	36.8% (0.0%, 79.9%)	104,620 (41,184 + 63,436)	36.8% (0.0%, 79.9%)	0.0% (0.0%, 0.0%)
				147,946 (121,459 + 26,487)	214,976 (121,459 + 93,517)	45.3% (0.0%, 253.1%)	214,976 (121,459 + 93,517)	45.3% (0.0%, 253.1%)	0.0% (0.0%, 0.0%)
6	50	1,000	1,000	–	–	–	–	–	–
				105,181 (19,493 + 85,688)	–	–	–	–	–
				–	–	–	–	–	–
				–	115,784 (20,878 + 94,906)	–	115,784 (20,878 + 94,906)	–	0.0% (0.0%, 0.0%)
				152,153 (34,889 + 117,264)	130,081 (34,889 + 95,192)	−14.5% (0.0%, −18.8%)	130,081 (34,889 + 95,192)	−14.5% (0.0%, −18.8%)	0.0% (0.0%, 0.0%)
		1,000	5,000	–	–	–	–	–	–
				441,181 (19,493 + 421,688)	–	–	–	–	–
				–	–	–	–	–	–
				–	467,822 (20,878 + 446,944)	–	467,822 (20,878 + 446,944)	–	0.0% (0.0%, 0.0%)
				282,718 (34,868 + 247,850)	494,081 (34,889 + 459,192)	74.8% (0.1%, 85.3%)	494,081 (34,889 + 459,192)	74.8% (0.1%, 85.3%)	0.0% (0.0%, 0.0%)
		5,000	1,000	–	–	–	–	–	–
				181,181 (95,493 + 85,688)	–	–	–	–	–
				–	–	–	–	–	–
				–	191,784 (96,878 + 94,906)	–	191,784 (96,878 + 94,906)	–	0.0% (0.0%, 0.0%)
				284,153 (166,889 + 117,264)	262,081 (166,889 + 95,192)	−7.8% (0.0%, −18.8%)	262,081 (166,889 + 95,192)	−7.8% (0.0%, −18.8%)	0.0% (0.0%, 0.0%)

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	5	1,000	1,000	4,360 (1,090 + 3,270)	4,308 (1,090 + 3,218)	−1.2% (0.0%, −1.6%)	4,291 (1,093 + 3,198)	−1.6% (0.3%, −2.2%)	−0.4% (0.3%, −0.6%)
				7,440 (1,093 + 6,347)	6,405 (1,093 + 5,312)	−13.9% (0.0%, −16.3%)	6,380 (1,096 + 5,284)	−14.2% (0.3%, −16.7%)	−0.4% (0.3%, −0.5%)
				7,509 (1,111 + 6,398)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				7,395 (2,112 + 5,283)	6,336 (2,112 + 4,224)	−14.3% (0.0%, −20.0%)	5,307 (2,130 + 3,177)	−28.2% (0.9%, −39.9%)	−16.2% (0.9%, −24.8%)
				6,327 (2,109 + 4,218)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	16,360 (1,090 + 15,270)	16,308 (1,090 + 15,218)	−0.3% (0.0%, −0.3%)	16,291 (1,093 + 15,198)	−0.4% (0.3%, −0.5%)	−0.1% (0.3%, −0.1%)
				31,440 (1,093 + 30,347)	26,405 (1,093 + 25,312)	−16.0% (0.0%, −16.6%)	26,380 (1,096 + 25,284)	−16.1% (0.3%, −16.7%)	−0.1% (0.3%, −0.1%)
				31,509 (1,111 + 30,398)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				27,395 (2,112 + 25,283)	22,336 (2,112 + 20,224)	−18.5% (0.0%, −20.0%)	17,307 (2,130 + 15,177)	−36.8% (0.9%, −40.0%)	−22.5% (0.9%, −25.0%)
				22,327 (2,109 + 20,218)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	8,360 (5,090 + 3,270)	8,308 (5,090 + 3,218)	−0.6% (0.0%, −1.6%)	8,291 (5,093 + 3,198)	−0.8% (0.1%, −2.2%)	−0.2% (0.1%, −0.6%)
				11,440 (5,093 + 6,347)	10,405 (5,093 + 5,312)	−9.0% (0.0%, −16.3%)	10,380 (5,096 + 5,284)	−9.3% (0.1%, −16.7%)	−0.2% (0.1%, −0.5%)
				11,509 (5,111 + 6,398)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				15,395 (10,112 + 5,283)	14,336 (10,112 + 4,224)	−6.9% (0.0%, −20.0%)	13,307 (10,130 + 3,177)	−13.6% (0.2%, −39.9%)	−7.2% (0.2%, −24.8%)
				14,327 (10,109 + 4,218)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
10	10	1,000	1,000	9,529 (2,173 + 7,356)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)	0.0% (0.0%, 0.0%)
				10,907 (1,166 + 9,741)	10,907 (1,166 + 9,741)	0.0% (0.0%, 0.0%)	9,886 (1,174 + 8,712)	−9.4% (0.7%, −10.6%)	−9.4% (0.7%, −10.6%)
				12,744 (2,178 + 10,566)	11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)	11,659 (2,181 + 9,478)	−8.5% (0.1%, −10.3%)	−0.3% (0.1%, −0.4%)
				8,655 (2,210 + 6,445)	8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)	8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
				9,684 (2,227 + 7,457)	8,600 (2,227 + 6,373)	−11.2% (0.0%, −14.5%)	8,591 (2,230 + 6,361)	−11.3% (0.1%, −14.7%)	−0.1% (0.1%, −0.2%)
		1,000	5,000	37,529 (2,173 + 35,356)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)	0.0% (0.0%, 0.0%)
				46,907 (1,166 + 45,741)	46,907 (1,166 + 45,741)	0.0% (0.0%, 0.0%)	41,886 (1,174 + 40,712)	−10.7% (0.7%, −11.0%)	−10.7% (0.7%, −11.0%)
				52,744 (2,178 + 50,566)	47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)	47,659 (2,181 + 45,478)	−9.6% (0.1%, −10.1%)	−0.1% (0.1%, −0.1%)
				32,655 (2,210 + 30,445)	32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)	32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)	0.0% (0.0%, 0.0%)
				37,684 (2,227 + 35,457)	32,600 (2,227 + 30,373)	−13.5% (0.0%, −14.3%)	32,591 (2,230 + 30,361)	−13.5% (0.1%, −14.4%)	0.0% (0.1%, 0.0%)
		5,000	1,000	17,529 (10,173 + 7,356)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)	0.0% (0.0%, 0.0%)
				14,907 (5,166 + 9,741)	14,907 (5,166 + 9,741)	0.0% (0.0%, 0.0%)	13,886 (5,174 + 8,712)	−6.8% (0.2%, −10.6%)	−6.8% (0.2%, −10.6%)
				20,744 (10,178 + 10,566)	19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)	19,659 (10,181 + 9,478)	−5.2% (0.0%, −10.3%)	−0.2% (0.0%, −0.4%)
				16,655 (10,210 + 6,445)	16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)	16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
				17,684 (10,227 + 7,457)	16,600 (10,227 + 6,373)	−6.1% (0.0%, −14.5%)	16,591 (10,230 + 6,361)	−6.2% (0.0%, −14.7%)	−0.1% (0.0%, −0.2%)

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	20	1,000	1,000	17,931 (4,316 + 13,615)	15,851 (4,316 + 11,535)	−11.6% (0.0%, −15.3%)	17,897 (4,316 + 13,581)	−0.2% (0.0%, −0.2%)	12.9% (0.0%, 17.7%)
				15,979 (2,225 + 13,754)	14,914 (2,225 + 12,689)	−6.7% (0.0%, −7.7%)	14,914 (2,225 + 12,689)	−6.7% (0.0%, −7.7%)	0.0% (0.0%, 0.0%)
				22,234 (3,338 + 18,896)	20,176 (3,338 + 16,838)	−9.3% (0.0%, −10.9%)	20,174 (3,338 + 16,836)	−9.3% (0.0%, −10.9%)	0.0% (0.0%, 0.0%)
				15,081 (3,351 + 11,730)	14,039 (3,351 + 10,688)	−6.9% (0.0%, −8.9%)	13,062 (3,351 + 9,711)	−13.4% (0.0%, −17.2%)	−7.0% (0.0%, −9.1%)
				17,042 (4,367 + 12,675)	16,012 (4,367 + 11,645)	−6.0% (0.0%, −8.1%)	18,083 (4,367 + 13,716)	6.1% (0.0%, 8.2%)	12.9% (0.0%, 17.8%)
		1,000	5,000	69,931 (4,316 + 65,615)	59,851 (4,316 + 55,535)	−14.4% (0.0%, −15.4%)	64,904 (4,316 + 60,588)	−7.2% (0.0%, −7.7%)	8.4% (0.0%, 9.1%)
				67,979 (2,225 + 65,754)	62,914 (2,225 + 60,689)	−7.5% (0.0%, −7.7%)	62,914 (2,225 + 60,689)	−7.5% (0.0%, −7.7%)	0.0% (0.0%, 0.0%)
				94,234 (3,338 + 90,896)	84,174 (3,338 + 80,836)	−10.7% (0.0%, −11.1%)	84,194 (3,338 + 80,856)	−10.7% (0.0%, −11.0%)	0.0% (0.0%, 0.0%)
				59,081 (3,351 + 55,730)	54,039 (3,351 + 50,688)	−8.5% (0.0%, −9.0%)	49,050 (3,358 + 45,692)	−17.0% (0.2%, −18.0%)	−9.2% (0.2%, −9.9%)
				65,042 (4,367 + 60,675)	60,012 (4,367 + 55,645)	−7.7% (0.0%, −8.3%)	70,073 (4,367 + 65,706)	7.7% (0.0%, 8.3%)	16.8% (0.0%, 18.1%)
		5,000	1,000	33,931 (20,316 + 13,615)	31,854 (20,316 + 11,538)	−6.1% (0.0%, −15.3%)	33,925 (20,316 + 13,609)	0.0% (0.0%, 0.0%)	6.5% (0.0%, 17.9%)
				23,979 (10,225 + 13,754)	22,914 (10,225 + 12,689)	−4.4% (0.0%, −7.7%)	22,914 (10,225 + 12,689)	−4.4% (0.0%, −7.7%)	0.0% (0.0%, 0.0%)
				34,234 (15,338 + 18,896)	32,176 (15,338 + 16,838)	−6.0% (0.0%, −10.9%)	32,176 (15,338 + 16,838)	−6.0% (0.0%, −10.9%)	0.0% (0.0%, 0.0%)
				26,072 (15,351 + 10,721)	26,039 (15,351 + 10,688)	−0.1% (0.0%, −0.3%)	25,062 (15,351 + 9,711)	−3.9% (0.0%, −9.4%)	−3.8% (0.0%, −9.1%)
				33,042 (20,367 + 12,675)	32,012 (20,367 + 11,645)	−3.1% (0.0%, −8.1%)	32,034 (20,367 + 11,667)	−3.1% (0.0%, −8.0%)	0.1% (0.0%, 0.2%)
10	30	1,000	1,000	33,428 (7,493 + 25,935)	28,352 (7,493 + 20,859)	−15.2% (0.0%, −19.6%)	29,362 (7,493 + 21,869)	−12.2% (0.0%, −15.7%)	3.6% (0.0%, 4.8%)
				35,880 (5,513 + 30,367)	32,725 (5,513 + 27,212)	−8.8% (0.0%, −10.4%)	36,894 (5,513 + 31,381)	2.8% (0.0%, 3.3%)	12.7% (0.0%, 15.3%)
				36,817 (5,513 + 31,304)	33,716 (5,513 + 28,203)	−8.4% (0.0%, −9.9%)	35,754 (5,513 + 30,241)	−2.9% (0.0%, −3.4%)	6.0% (0.0%, 7.2%)
				31,607 (8,588 + 23,019)	26,430 (8,588 + 17,842)	−16.4% (0.0%, −22.5%)	29,571 (8,588 + 20,983)	−6.4% (0.0%, −8.8%)	11.9% (0.0%, 17.6%)
				30,780 (7,636 + 23,144)	26,625 (7,636 + 18,989)	−13.5% (0.0%, −18.0%)	27,714 (7,636 + 20,078)	−10.0% (0.0%, −13.2%)	4.1% (0.0%, 5.7%)
		1,000	5,000	123,485 (7,487 + 115,998)	118,365 (7,493 + 110,872)	−4.1% (0.1%, −4.4%)	163,512 (7,493 + 156,019)	32.4% (0.1%, 34.5%)	38.1% (0.0%, 40.7%)
				146,557 (5,436 + 141,121)	131,713 (5,513 + 126,200)	−10.1% (1.4%, −10.6%)	156,865 (5,513 + 151,352)	7.0% (1.4%, 7.2%)	19.1% (0.0%, 19.9%)
				151,810 (5,513 + 146,297)	136,718 (5,513 + 131,205)	−9.9% (0.0%, −10.3%)	156,811 (5,513 + 151,298)	3.3% (0.0%, 3.4%)	14.7% (0.0%, 15.3%)
				114,636 (8,588 + 106,048)	94,475 (8,588 + 85,887)	−17.6% (0.0%, −19.0%)	134,591 (8,588 + 126,003)	17.4% (0.0%, 18.8%)	42.5% (0.0%, 46.7%)
				113,765 (7,636 + 106,129)	93,636 (7,636 + 86,000)	−17.7% (0.0%, −19.0%)	148,781 (7,636 + 141,145)	30.8% (0.0%, 33.0%)	58.9% (0.0%, 64.1%)
		5,000	1,000	60,419 (35,493 + 24,926)	57,355 (35,493 + 21,862)	−5.1% (0.0%, −12.3%)	58,386 (35,493 + 22,893)	−3.4% (0.0%, −8.2%)	1.8% (0.0%, 4.7%)
				55,880 (25,513 + 30,367)	51,698 (25,513 + 26,185)	−7.5% (0.0%, −13.8%)	55,843 (25,513 + 30,330)	−0.1% (0.0%, −0.1%)	8.0% (0.0%, 15.8%)
				56,826 (25,513 + 31,313)	54,731 (25,513 + 29,218)	−3.7% (0.0%, −6.7%)	54,732 (25,513 + 29,219)	−3.7% (0.0%, −6.7%)	0.0% (0.0%, 0.0%)
				62,649 (40,588 + 22,061)	57,427 (40,588 + 16,839)	−8.3% (0.0%, −23.7%)	67,535 (40,588 + 26,947)	7.8% (0.0%, 22.1%)	17.6% (0.0%, 60.0%)
				57,767 (35,636 + 22,131)	54,655 (35,636 + 19,019)	−5.4% (0.0%, −14.1%)	57,760 (35,636 + 22,124)	0.0% (0.0%, 0.0%)	5.7% (0.0%, 16.3%)

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	40	1,000	1,000	61,134 (12,729 + 48,405)	–	–	–	–	–
				56,382 (8,706 + 47,676)	48,177 (8,706 + 39,471)	−14.6% (0.0%, −17.2%)	49,191 (8,706 + 40,485)	−12.8% (0.0%, −15.1%)	2.1% (0.0%, 2.6%)
				55,377 (8,661 + 46,716)	–	–	–	–	–
				45,318 (11,861 + 33,457)	–	–	–	–	–
				42,334 (10,843 + 31,491)	47,330 (10,843 + 36,487)	11.8% (0.0%, 15.9%)	57,332 (10,843 + 46,489)	35.4% (0.0%, 47.6%)	21.1% (0.0%, 27.4%)
		1,000	5,000	249,146 (12,729 + 236,417)	–	–	–	–	–
				230,287 (8,695 + 221,592)	205,212 (8,706 + 196,506)	−10.9% (0.1%, −11.3%)	205,241 (8,706 + 196,535)	−10.9% (0.1%, −11.3%)	0.0% (0.0%, 0.0%)
				220,132 (8,642 + 211,490)	225,304 (8,661 + 216,643)	2.3% (0.2%, 2.4%)	270,546 (8,661 + 261,885)	22.9% (0.2%, 23.8%)	20.1% (0.0%, 20.9%)
				188,895 (12,042 + 176,853)	–	–	–	–	–
				172,342 (10,843 + 161,499)	152,149 (10,843 + 141,306)	−11.7% (0.0%, −12.5%)	152,140 (10,843 + 141,297)	−11.7% (0.0%, −12.5%)	0.0% (0.0%, 0.0%)
		5,000	1,000	110,125 (60,729 + 49,396)	–	–	–	–	–
				89,392 (40,706 + 48,686)	80,168 (40,706 + 39,462)	−10.3% (0.0%, −18.9%)	80,181 (40,706 + 39,475)	−10.3% (0.0%, −18.9%)	0.0% (0.0%, 0.0%)
				–	–	–	–	–	–
				89,359 (55,861 + 33,498)	97,242 (55,861 + 41,381)	8.8% (0.0%, 23.5%)	117,407 (55,861 + 61,546)	31.4% (0.0%, 83.7%)	20.7% (0.0%, 48.7%)
				82,326 (50,843 + 31,483)	89,255 (50,843 + 38,412)	8.4% (0.0%, 22.0%)	–	–	–
10	50	1,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
		1,000	5,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
		5,000	1,000	–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–
				–	–	–	–	–	–

Table 3.9: Comparison of the solutions from MIP-LNS. Vehicle and crew objective values are shown inside parenthesis.

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	5	1,000	1,000	8,312 (2,078 + 6,234)	7,268 (2,078 + 5,190)	−12.6% (0.0%, −16.7%)	7,268 (2,078 + 5,190)	−12.6% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				4,166 (1,047 + 3,119)	4,144 (1,047 + 3,097)	−0.5% (0.0%, −0.7%)	4,135 (1,048 + 3,087)	−0.7% (0.1%, −1.0%)	−0.2% (0.1%, −0.3%)
				4,113 (1,035 + 3,078)	3,099 (1,035 + 2,064)	−24.7% (0.0%, −32.9%)	3,089 (1,037 + 2,052)	−24.9% (0.2%, −33.3%)	−0.3% (0.2%, −0.6%)
				4,410 (1,151 + 3,259)	3,453 (1,151 + 2,302)	−21.7% (0.0%, −29.4%)	3,352 (1,159 + 2,193)	−24.0% (0.7%, −32.7%)	−2.9% (0.7%, −4.7%)
				6,404 (3,202 + 3,202)	6,404 (3,202 + 3,202)	0.0% (0.0%, 0.0%)	6,404 (3,202 + 3,202)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	32,312 (2,078 + 30,234)	27,268 (2,078 + 25,190)	−15.6% (0.0%, −16.7%)	27,268 (2,078 + 25,190)	−15.6% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				16,166 (1,047 + 15,119)	16,144 (1,047 + 15,097)	−0.1% (0.0%, −0.1%)	16,135 (1,048 + 15,087)	−0.2% (0.1%, −0.2%)	−0.1% (0.1%, −0.1%)
				16,113 (1,035 + 15,078)	11,099 (1,035 + 10,064)	−31.1% (0.0%, −33.3%)	11,089 (1,037 + 10,052)	−31.2% (0.2%, −33.3%)	−0.1% (0.2%, −0.1%)
				16,410 (1,151 + 15,259)	11,453 (1,151 + 10,302)	−30.2% (0.0%, −32.5%)	11,352 (1,159 + 10,193)	−30.8% (0.7%, −33.2%)	−0.9% (0.7%, −1.1%)
				18,404 (3,202 + 15,202)	18,404 (3,202 + 15,202)	0.0% (0.0%, 0.0%)	18,404 (3,202 + 15,202)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	16,312 (10,078 + 6,234)	15,268 (10,078 + 5,190)	−6.4% (0.0%, −16.7%)	15,268 (10,078 + 5,190)	−6.4% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				8,166 (5,047 + 3,119)	8,144 (5,047 + 3,097)	−0.3% (0.0%, −0.7%)	8,135 (5,048 + 3,087)	−0.4% (0.0%, −1.0%)	−0.1% (0.0%, −0.3%)
				8,113 (5,035 + 3,078)	7,099 (5,035 + 2,064)	−12.5% (0.0%, −32.9%)	7,089 (5,037 + 2,052)	−12.6% (0.0%, −33.3%)	−0.1% (0.0%, −0.6%)
				8,410 (5,151 + 3,259)	7,453 (5,151 + 2,302)	−11.4% (0.0%, −29.4%)	7,352 (5,159 + 2,193)	−12.6% (0.2%, −32.7%)	−1.4% (0.2%, −4.7%)
				18,404 (15,202 + 3,202)	18,404 (15,202 + 3,202)	0.0% (0.0%, 0.0%)	18,404 (15,202 + 3,202)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
6	10	1,000	1,000	13,462 (4,160 + 9,302)	13,462 (4,160 + 9,302)	0.0% (0.0%, 0.0%)	13,455 (4,162 + 9,293)	−0.1% (0.0%, −0.1%)	−0.1% (0.0%, −0.1%)
				8,194 (2,064 + 6,130)	8,189 (2,064 + 6,125)	−0.1% (0.0%, −0.1%)	7,182 (2,073 + 5,109)	−12.4% (0.4%, −16.7%)	−12.3% (0.4%, −16.6%)
				8,230 (2,071 + 6,159)	6,213 (2,071 + 4,142)	−24.5% (0.0%, −32.7%)	6,171 (2,073 + 4,098)	−25.0% (0.1%, −33.5%)	−0.7% (0.1%, −1.1%)
				5,689 (1,196 + 4,493)	4,590 (1,196 + 3,394)	−19.3% (0.0%, −24.5%)	3,487 (1,204 + 2,283)	−38.7% (0.7%, −49.2%)	−24.0% (0.7%, −32.7%)
				13,776 (6,380 + 7,396)	12,760 (6,380 + 6,380)	−7.4% (0.0%, −13.7%)	12,760 (6,380 + 6,380)	−7.4% (0.0%, −13.7%)	0.0% (0.0%, 0.0%)
		1,000	5,000	49,462 (4,160 + 45,302)	49,462 (4,160 + 45,302)	0.0% (0.0%, 0.0%)	49,455 (4,162 + 45,293)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
				32,194 (2,064 + 30,130)	32,189 (2,064 + 30,125)	0.0% (0.0%, 0.0%)	27,196 (2,075 + 25,121)	−15.5% (0.5%, −16.6%)	−15.5% (0.5%, −16.6%)
				32,230 (2,071 + 30,159)	22,213 (2,071 + 20,142)	−31.1% (0.0%, −33.2%)	22,171 (2,073 + 20,098)	−31.2% (0.1%, −33.4%)	−0.2% (0.1%, −0.2%)
				21,689 (1,196 + 20,493)	16,590 (1,196 + 15,394)	−23.5% (0.0%, −24.9%)	11,487 (1,204 + 10,283)	−47.0% (0.7%, −49.8%)	−30.8% (0.7%, −33.2%)
				41,776 (6,380 + 35,396)	36,760 (6,380 + 30,380)	−12.0% (0.0%, −14.2%)	36,760 (6,380 + 30,380)	−12.0% (0.0%, −14.2%)	0.0% (0.0%, 0.0%)
		5,000	1,000	29,462 (20,160 + 9,302)	29,462 (20,160 + 9,302)	0.0% (0.0%, 0.0%)	29,455 (20,162 + 9,293)	0.0% (0.0%, −0.1%)	0.0% (0.0%, −0.1%)
				16,194 (10,064 + 6,130)	16,189 (10,064 + 6,125)	0.0% (0.0%, −0.1%)	15,182 (10,073 + 5,109)	−6.2% (0.1%, −16.7%)	−6.2% (0.1%, −16.6%)
				16,230 (10,071 + 6,159)	14,213 (10,071 + 4,142)	−12.4% (0.0%, −32.7%)	14,171 (10,073 + 4,098)	−12.7% (0.0%, −33.5%)	−0.3% (0.0%, −1.1%)
				9,689 (5,196 + 4,493)	8,590 (5,196 + 3,394)	−11.3% (0.0%, −24.5%)	7,487 (5,204 + 2,283)	−22.7% (0.2%, −49.2%)	−12.8% (0.2%, −32.7%)
				37,776 (30,380 + 7,396)	36,760 (30,380 + 6,380)	−2.7% (0.0%, −13.7%)	36,760 (30,380 + 6,380)	−2.7% (0.0%, −13.7%)	0.0% (0.0%, 0.0%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	20	1,000	1,000	31,980 (9,364 + 22,616)	28,933 (9,364 + 19,569)	−9.5% (0.0%, −13.5%)	28,933 (9,364 + 19,569)	−9.5% (0.0%, −13.5%)	0.0% (0.0%, 0.0%)
				16,338 (4,127 + 12,211)	13,307 (4,127 + 9,180)	−18.6% (0.0%, −24.8%)	13,307 (4,127 + 9,180)	−18.6% (0.0%, −24.8%)	0.0% (0.0%, 0.0%)
				13,341 (3,131 + 10,210)	11,319 (3,131 + 8,188)	−15.2% (0.0%, −19.8%)	11,319 (3,131 + 8,188)	−15.2% (0.0%, −19.8%)	0.0% (0.0%, 0.0%)
				8,609 (2,210 + 6,399)	6,525 (2,210 + 4,315)	−24.2% (0.0%, −32.6%)	6,504 (2,210 + 4,294)	−24.5% (0.0%, −32.9%)	−0.3% (0.0%, −0.5%)
				24,576 (11,780 + 12,796)	23,560 (11,780 + 11,780)	−4.1% (0.0%, −7.9%)	23,560 (11,780 + 11,780)	−4.1% (0.0%, −7.9%)	0.0% (0.0%, 0.0%)
		1,000	5,000	119,975 (9,364 + 110,611)	104,933 (9,364 + 95,569)	−12.5% (0.0%, −13.6%)	104,933 (9,364 + 95,569)	−12.5% (0.0%, −13.6%)	0.0% (0.0%, 0.0%)
				64,338 (4,127 + 60,211)	49,307 (4,127 + 45,180)	−23.4% (0.0%, −25.0%)	49,307 (4,127 + 45,180)	−23.4% (0.0%, −25.0%)	0.0% (0.0%, 0.0%)
				53,347 (3,131 + 50,216)	43,319 (3,131 + 40,188)	−18.8% (0.0%, −20.0%)	43,319 (3,131 + 40,188)	−18.8% (0.0%, −20.0%)	0.0% (0.0%, 0.0%)
				32,609 (2,210 + 30,399)	22,525 (2,210 + 20,315)	−30.9% (0.0%, −33.2%)	22,510 (2,213 + 20,297)	−31.0% (0.1%, −33.2%)	−0.1% (0.1%, −0.1%)
				72,576 (11,780 + 60,796)	67,560 (11,780 + 55,780)	−6.9% (0.0%, −8.3%)	67,560 (11,780 + 55,780)	−6.9% (0.0%, −8.3%)	0.0% (0.0%, 0.0%)
		5,000	1,000	67,975 (45,364 + 22,611)	64,933 (45,364 + 19,569)	−4.5% (0.0%, −13.5%)	64,933 (45,364 + 19,569)	−4.5% (0.0%, −13.5%)	0.0% (0.0%, 0.0%)
				32,338 (20,127 + 12,211)	29,307 (20,127 + 9,180)	−9.4% (0.0%, −24.8%)	29,307 (20,127 + 9,180)	−9.4% (0.0%, −24.8%)	0.0% (0.0%, 0.0%)
				25,347 (15,131 + 10,216)	23,319 (15,131 + 8,188)	−8.0% (0.0%, −19.9%)	23,308 (15,131 + 8,177)	−8.0% (0.0%, −20.0%)	0.0% (0.0%, −0.1%)
				16,609 (10,210 + 6,399)	14,525 (10,210 + 4,315)	−12.5% (0.0%, −32.6%)	14,513 (10,210 + 4,303)	−12.6% (0.0%, −32.8%)	−0.1% (0.0%, −0.3%)
				68,576 (55,780 + 12,796)	67,560 (55,780 + 11,780)	−1.5% (0.0%, −7.9%)	67,560 (55,780 + 11,780)	−1.5% (0.0%, −7.9%)	0.0% (0.0%, 0.0%)
6	30	1,000	1,000	48,500 (14,553 + 33,947)	44,432 (14,553 + 29,879)	−8.4% (0.0%, −12.0%)	44,436 (14,553 + 29,883)	−8.4% (0.0%, −12.0%)	0.0% (0.0%, 0.0%)
				23,458 (6,178 + 17,280)	19,426 (6,178 + 13,248)	−17.2% (0.0%, −23.3%)	19,424 (6,178 + 13,246)	−17.2% (0.0%, −23.3%)	0.0% (0.0%, 0.0%)
				19,457 (5,183 + 14,274)	17,431 (5,183 + 12,248)	−10.4% (0.0%, −14.2%)	17,441 (5,183 + 12,258)	−10.4% (0.0%, −14.1%)	0.1% (0.0%, 0.1%)
				10,054 (2,370 + 7,684)	7,936 (2,370 + 5,566)	−21.1% (0.0%, −27.6%)	7,918 (2,370 + 5,548)	−21.2% (0.0%, −27.8%)	−0.2% (0.0%, −0.3%)
				41,330 (20,157 + 21,173)	40,314 (20,157 + 20,157)	−2.5% (0.0%, −4.8%)	40,314 (20,157 + 20,157)	−2.5% (0.0%, −4.8%)	0.0% (0.0%, 0.0%)
		1,000	5,000	180,506 (14,553 + 165,953)	160,428 (14,553 + 145,875)	−11.1% (0.0%, −12.1%)	160,428 (14,553 + 145,875)	−11.1% (0.0%, −12.1%)	0.0% (0.0%, 0.0%)
				91,458 (6,178 + 85,280)	71,428 (6,178 + 65,250)	−21.9% (0.0%, −23.5%)	67,436 (7,189 + 60,247)	−26.3% (16.4%, −29.4%)	−5.6% (16.4%, −7.7%)
				75,457 (5,183 + 70,274)	65,432 (5,183 + 60,249)	−13.3% (0.0%, −14.3%)	65,444 (5,183 + 60,261)	−13.3% (0.0%, −14.2%)	0.0% (0.0%, 0.0%)
				38,054 (2,370 + 35,684)	27,936 (2,370 + 25,566)	−26.6% (0.0%, −28.4%)	27,918 (2,370 + 25,548)	−26.6% (0.0%, −28.4%)	−0.1% (0.0%, −0.1%)
				121,330 (20,157 + 101,173)	116,314 (20,157 + 96,157)	−4.1% (0.0%, −5.0%)	116,314 (20,157 + 96,157)	−4.1% (0.0%, −5.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	104,506 (70,553 + 33,953)	100,436 (70,553 + 29,883)	−3.9% (0.0%, −12.0%)	100,436 (70,553 + 29,883)	−3.9% (0.0%, −12.0%)	0.0% (0.0%, 0.0%)
				47,458 (30,178 + 17,280)	43,428 (30,178 + 13,250)	−8.5% (0.0%, −23.3%)	43,428 (30,178 + 13,250)	−8.5% (0.0%, −23.3%)	0.0% (0.0%, 0.0%)
				39,457 (25,183 + 14,274)	37,429 (25,183 + 12,246)	−5.1% (0.0%, −14.2%)	36,464 (25,185 + 11,279)	−7.6% (0.0%, −21.0%)	−2.6% (0.0%, −7.9%)
				18,054 (10,370 + 7,684)	15,936 (10,370 + 5,566)	−11.7% (0.0%, −27.6%)	15,914 (10,379 + 5,535)	−11.9% (0.1%, −28.0%)	−0.1% (0.1%, −0.6%)
				117,330 (96,157 + 21,173)	116,314 (96,157 + 20,157)	−0.9% (0.0%, −4.8%)	116,314 (96,157 + 20,157)	−0.9% (0.0%, −4.8%)	0.0% (0.0%, 0.0%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
6	40	1,000	1,000	64,140 (18,791 + 45,349)	62,119 (18,791 + 43,328)	−3.2% (0.0%, −4.5%)	62,106 (18,791 + 43,315)	−3.2% (0.0%, −4.5%)	0.0% (0.0%, 0.0%)
				30,619 (8,227 + 22,392)	25,558 (8,227 + 17,331)	−16.5% (0.0%, −22.6%)	25,552 (8,227 + 17,325)	−16.5% (0.0%, −22.6%)	0.0% (0.0%, 0.0%)
				26,611 (7,242 + 19,369)	22,580 (7,242 + 15,338)	−15.1% (0.0%, −20.8%)	21,583 (7,244 + 14,339)	−18.9% (0.0%, −26.0%)	−4.4% (0.0%, −6.5%)
				14,396 (3,507 + 10,889)	12,250 (3,507 + 8,743)	−14.9% (0.0%, −19.7%)	12,250 (3,507 + 8,743)	−14.9% (0.0%, −19.7%)	0.0% (0.0%, 0.0%)
				51,922 (25,446 + 26,476)	51,920 (25,446 + 26,474)	0.0% (0.0%, 0.0%)	50,892 (25,446 + 25,446)	−2.0% (0.0%, −3.9%)	−2.0% (0.0%, −3.9%)
		1,000	5,000	240,140 (18,791 + 221,349)	225,084 (18,791 + 206,293)	−6.3% (0.0%, −6.8%)	230,110 (18,791 + 211,319)	−4.2% (0.0%, −4.5%)	2.2% (0.0%, 2.4%)
				118,609 (8,227 + 110,382)	93,558 (8,227 + 85,331)	−21.1% (0.0%, −22.7%)	93,558 (8,227 + 85,331)	−21.1% (0.0%, −22.7%)	0.0% (0.0%, 0.0%)
				102,611 (7,242 + 95,369)	82,581 (7,242 + 75,339)	−19.5% (0.0%, −21.0%)	82,588 (7,245 + 75,343)	−19.5% (0.0%, −21.0%)	0.0% (0.0%, 0.0%)
				54,396 (3,507 + 50,889)	44,250 (3,507 + 40,743)	−18.7% (0.0%, −19.9%)	39,439 (3,622 + 35,817)	−27.5% (3.3%, −29.6%)	−10.9% (3.3%, −12.1%)
				156,936 (25,446 + 131,490)	156,936 (25,446 + 131,490)	0.0% (0.0%, 0.0%)	151,908 (25,446 + 126,462)	−3.2% (0.0%, −3.8%)	−3.2% (0.0%, −3.8%)
		5,000	1,000	136,140 (90,791 + 45,349)	134,102 (90,791 + 43,311)	−1.5% (0.0%, −4.5%)	134,102 (90,791 + 43,311)	−1.5% (0.0%, −4.5%)	0.0% (0.0%, 0.0%)
				62,609 (40,227 + 22,382)	56,554 (40,227 + 16,327)	−9.7% (0.0%, −27.1%)	57,566 (40,227 + 17,339)	−8.1% (0.0%, −22.5%)	1.8% (0.0%, 6.2%)
				54,611 (35,242 + 19,369)	49,581 (35,242 + 14,339)	−9.2% (0.0%, −26.0%)	50,591 (35,242 + 15,349)	−7.4% (0.0%, −20.8%)	2.0% (0.0%, 7.0%)
				26,357 (15,507 + 10,850)	24,253 (15,507 + 8,746)	−8.0% (0.0%, −19.4%)	24,250 (15,507 + 8,743)	−8.0% (0.0%, −19.4%)	0.0% (0.0%, 0.0%)
				148,936 (121,446 + 27,490)	147,920 (121,446 + 26,474)	−0.7% (0.0%, −3.7%)	146,892 (121,446 + 25,446)	−1.4% (0.0%, −7.4%)	−0.7% (0.0%, −3.9%)
6	50	1,000	1,000	87,764 (24,988 + 62,776)	79,618 (24,988 + 54,630)	−9.3% (0.0%, −13.0%)	79,628 (24,988 + 54,640)	−9.3% (0.0%, −13.0%)	0.0% (0.0%, 0.0%)
				38,782 (10,291 + 28,491)	30,697 (10,291 + 20,406)	−20.8% (0.0%, −28.4%)	31,713 (10,291 + 21,422)	−18.2% (0.0%, −24.8%)	3.3% (0.0%, 5.0%)
				36,803 (9,310 + 27,493)	29,743 (9,310 + 20,433)	−19.2% (0.0%, −25.7%)	29,728 (9,313 + 20,415)	−19.2% (0.0%, −25.7%)	−0.1% (0.0%, −0.1%)
				18,878 (4,719 + 14,159)	15,722 (4,719 + 11,003)	−16.7% (0.0%, −22.3%)	16,785 (4,719 + 12,066)	−11.1% (0.0%, −14.8%)	6.8% (0.0%, 9.7%)
				70,770 (33,847 + 36,923)	69,740 (33,847 + 35,893)	−1.5% (0.0%, −2.8%)	68,724 (33,847 + 34,877)	−2.9% (0.0%, −5.5%)	−1.5% (0.0%, −2.8%)
		1,000	5,000	331,772 (24,988 + 306,784)	291,618 (24,988 + 266,630)	−12.1% (0.0%, −13.1%)	291,614 (24,988 + 266,626)	−12.1% (0.0%, −13.1%)	0.0% (0.0%, 0.0%)
				145,782 (10,291 + 135,491)	110,695 (10,291 + 100,404)	−24.1% (0.0%, −25.9%)	115,750 (10,291 + 105,459)	−20.6% (0.0%, −22.2%)	4.6% (0.0%, 5.0%)
				139,804 (9,310 + 130,494)	109,743 (9,310 + 100,433)	−21.5% (0.0%, −23.0%)	109,743 (9,310 + 100,433)	−21.5% (0.0%, −23.0%)	0.0% (0.0%, 0.0%)
				70,880 (4,719 + 66,161)	55,696 (4,719 + 50,977)	−21.4% (0.0%, −23.0%)	60,718 (4,719 + 55,999)	−14.3% (0.0%, −15.4%)	9.0% (0.0%, 9.9%)
				210,795 (33,847 + 176,948)	210,778 (33,847 + 176,931)	0.0% (0.0%, 0.0%)	205,752 (33,847 + 171,905)	−2.4% (0.0%, −2.8%)	−2.4% (0.0%, −2.8%)
		5,000	1,000	183,769 (120,988 + 62,781)	176,630 (120,988 + 55,642)	−3.9% (0.0%, −11.4%)	175,752 (120,988 + 54,764)	−4.4% (0.0%, −12.8%)	−0.5% (0.0%, −1.6%)
				77,778 (50,291 + 27,487)	71,711 (50,291 + 21,420)	−7.8% (0.0%, −22.1%)	70,697 (50,291 + 20,406)	−9.1% (0.0%, −25.8%)	−1.4% (0.0%, −4.7%)
				71,814 (45,310 + 26,504)	65,752 (45,310 + 20,442)	−8.4% (0.0%, −22.9%)	65,747 (45,310 + 20,437)	−8.4% (0.0%, −22.9%)	0.0% (0.0%, 0.0%)
				35,861 (20,719 + 15,142)	31,684 (20,719 + 10,965)	−11.6% (0.0%, −27.6%)	31,817 (20,719 + 11,098)	−11.3% (0.0%, −26.7%)	0.4% (0.0%, 1.2%)
				198,782 (161,847 + 36,935)	198,780 (161,847 + 36,933)	0.0% (0.0%, 0.0%)	197,744 (161,847 + 35,897)	−0.5% (0.0%, −2.8%)	−0.5% (0.0%, −2.8%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	5	1,000	1,000	4,360 (1,090 + 3,270)	4,308 (1,090 + 3,218)	−1.2% (0.0%, −1.6%)	4,291 (1,093 + 3,198)	−1.6% (0.3%, −2.2%)	−0.4% (0.3%, −0.6%)
				7,440 (1,093 + 6,347)	6,405 (1,093 + 5,312)	−13.9% (0.0%, −16.3%)	6,380 (1,096 + 5,284)	−14.2% (0.3%, −16.7%)	−0.4% (0.3%, −0.5%)
				7,509 (1,111 + 6,398)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)	6,438 (1,111 + 5,327)	−14.3% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				7,395 (2,112 + 5,283)	6,336 (2,112 + 4,224)	−14.3% (0.0%, −20.0%)	5,307 (2,130 + 3,177)	−28.2% (0.9%, −39.9%)	−16.2% (0.9%, −24.8%)
				6,327 (2,109 + 4,218)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)	6,327 (2,109 + 4,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		1,000	5,000	16,360 (1,090 + 15,270)	16,308 (1,090 + 15,218)	−0.3% (0.0%, −0.3%)	16,291 (1,093 + 15,198)	−0.4% (0.3%, −0.5%)	−0.1% (0.3%, −0.1%)
				31,440 (1,093 + 30,347)	26,405 (1,093 + 25,312)	−16.0% (0.0%, −16.6%)	26,380 (1,096 + 25,284)	−16.1% (0.3%, −16.7%)	−0.1% (0.3%, −0.1%)
				31,509 (1,111 + 30,398)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)	26,438 (1,111 + 25,327)	−16.1% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				27,395 (2,112 + 25,283)	22,336 (2,112 + 20,224)	−18.5% (0.0%, −20.0%)	17,307 (2,130 + 15,177)	−36.8% (0.9%, −40.0%)	−22.5% (0.9%, −25.0%)
				22,327 (2,109 + 20,218)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)	22,327 (2,109 + 20,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
		5,000	1,000	8,360 (5,090 + 3,270)	8,308 (5,090 + 3,218)	−0.6% (0.0%, −1.6%)	8,291 (5,093 + 3,198)	−0.8% (0.1%, −2.2%)	−0.2% (0.1%, −0.6%)
				11,440 (5,093 + 6,347)	10,405 (5,093 + 5,312)	−9.0% (0.0%, −16.3%)	10,380 (5,096 + 5,284)	−9.3% (0.1%, −16.7%)	−0.2% (0.1%, −0.5%)
				11,509 (5,111 + 6,398)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)	10,438 (5,111 + 5,327)	−9.3% (0.0%, −16.7%)	0.0% (0.0%, 0.0%)
				15,395 (10,112 + 5,283)	14,336 (10,112 + 4,224)	−6.9% (0.0%, −20.0%)	13,307 (10,130 + 3,177)	−13.6% (0.2%, −39.9%)	−7.2% (0.2%, −24.8%)
				14,327 (10,109 + 4,218)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)	14,327 (10,109 + 4,218)	0.0% (0.0%, 0.0%)	0.0% (0.0%, 0.0%)
10	10	1,000	1,000	9,529 (2,173 + 7,356)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)	7,476 (2,173 + 5,303)	−21.5% (0.0%, −27.9%)	0.0% (0.0%, 0.0%)
				10,913 (1,166 + 9,747)	10,913 (1,166 + 9,747)	0.0% (0.0%, 0.0%)	9,859 (1,172 + 8,687)	−9.7% (0.5%, −10.9%)	−9.7% (0.5%, −10.9%)
				12,744 (2,178 + 10,566)	11,695 (2,178 + 9,517)	−8.2% (0.0%, −9.9%)	11,659 (2,181 + 9,478)	−8.5% (0.1%, −10.3%)	−0.3% (0.1%, −0.4%)
				8,655 (2,210 + 6,445)	8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)	8,638 (2,210 + 6,428)	−0.2% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
				9,744 (2,227 + 7,517)	8,609 (2,227 + 6,382)	−11.6% (0.0%, −15.1%)	8,609 (2,227 + 6,382)	−11.6% (0.0%, −15.1%)	0.0% (0.0%, 0.0%)
		1,000	5,000	37,529 (2,173 + 35,356)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)	27,476 (2,173 + 25,303)	−26.8% (0.0%, −28.4%)	0.0% (0.0%, 0.0%)
				46,913 (1,166 + 45,747)	46,913 (1,166 + 45,747)	0.0% (0.0%, 0.0%)	41,883 (1,175 + 40,708)	−10.7% (0.8%, −11.0%)	−10.7% (0.8%, −11.0%)
				52,744 (2,178 + 50,566)	47,695 (2,178 + 45,517)	−9.6% (0.0%, −10.0%)	47,659 (2,181 + 45,478)	−9.6% (0.1%, −10.1%)	−0.1% (0.1%, −0.1%)
				32,655 (2,210 + 30,445)	32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)	32,638 (2,210 + 30,428)	−0.1% (0.0%, −0.1%)	0.0% (0.0%, 0.0%)
				37,744 (2,227 + 35,517)	32,609 (2,227 + 30,382)	−13.6% (0.0%, −14.5%)	32,609 (2,227 + 30,382)	−13.6% (0.0%, −14.5%)	0.0% (0.0%, 0.0%)
		5,000	1,000	17,529 (10,173 + 7,356)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)	15,476 (10,173 + 5,303)	−11.7% (0.0%, −27.9%)	0.0% (0.0%, 0.0%)
				14,913 (5,166 + 9,747)	14,913 (5,166 + 9,747)	0.0% (0.0%, 0.0%)	13,859 (5,172 + 8,687)	−7.1% (0.1%, −10.9%)	−7.1% (0.1%, −10.9%)
				20,744 (10,178 + 10,566)	19,695 (10,178 + 9,517)	−5.1% (0.0%, −9.9%)	19,659 (10,181 + 9,478)	−5.2% (0.0%, −10.3%)	−0.2% (0.0%, −0.4%)
				16,655 (10,210 + 6,445)	16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)	16,638 (10,210 + 6,428)	−0.1% (0.0%, −0.3%)	0.0% (0.0%, 0.0%)
				17,744 (10,227 + 7,517)	16,609 (10,227 + 6,382)	−6.4% (0.0%, −15.1%)	16,609 (10,227 + 6,382)	−6.4% (0.0%, −15.1%)	0.0% (0.0%, 0.0%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	20	1,000	1,000	14,866 (3,271 + 11,595)	13,835 (3,271 + 10,564)	−6.9% (0.0%, −8.9%)	12,933 (3,312 + 9,621)	−13.0% (1.3%, −17.0%)	−6.5% (1.3%, −8.9%)
				15,956 (2,221 + 13,735)	15,956 (2,221 + 13,735)	0.0% (0.0%, 0.0%)	14,946 (2,229 + 12,717)	−6.3% (0.4%, −7.4%)	−6.3% (0.4%, −7.4%)
				23,281 (3,338 + 19,943)	20,176 (3,338 + 16,838)	−13.3% (0.0%, −15.6%)	20,176 (3,338 + 16,838)	−13.3% (0.0%, −15.6%)	0.0% (0.0%, 0.0%)
				13,984 (3,331 + 10,653)	13,952 (3,331 + 10,621)	−0.2% (0.0%, −0.3%)	12,950 (3,346 + 9,604)	−7.4% (0.5%, −9.8%)	−7.2% (0.5%, −9.6%)
				16,125 (3,390 + 12,735)	16,125 (3,390 + 12,735)	0.0% (0.0%, 0.0%)	15,053 (3,392 + 11,661)	−6.6% (0.1%, −8.4%)	−6.6% (0.1%, −8.4%)
		1,000	5,000	58,866 (3,271 + 55,595)	53,835 (3,271 + 50,564)	−8.5% (0.0%, −9.0%)	48,944 (3,280 + 45,664)	−16.9% (0.3%, −17.9%)	−9.1% (0.3%, −9.7%)
				67,956 (2,221 + 65,735)	67,956 (2,221 + 65,735)	0.0% (0.0%, 0.0%)	62,946 (2,229 + 60,717)	−7.4% (0.4%, −7.6%)	−7.4% (0.4%, −7.6%)
				99,281 (3,338 + 95,943)	84,174 (3,338 + 80,836)	−15.2% (0.0%, −15.7%)	84,174 (3,338 + 80,836)	−15.2% (0.0%, −15.7%)	0.0% (0.0%, 0.0%)
				53,984 (3,331 + 50,653)	53,953 (3,331 + 50,622)	−0.1% (0.0%, −0.1%)	44,881 (4,357 + 40,524)	−16.9% (30.8%, −20.0%)	−16.8% (30.8%, −19.9%)
				64,125 (3,390 + 60,735)	64,125 (3,390 + 60,735)	0.0% (0.0%, 0.0%)	59,076 (3,392 + 55,684)	−7.9% (0.1%, −8.3%)	−7.9% (0.1%, −8.3%)
		5,000	1,000	26,866 (15,271 + 11,595)	25,835 (15,271 + 10,564)	−3.8% (0.0%, −8.9%)	24,944 (15,280 + 9,664)	−7.2% (0.1%, −16.7%)	−3.4% (0.1%, −8.5%)
				23,956 (10,221 + 13,735)	23,956 (10,221 + 13,735)	0.0% (0.0%, 0.0%)	22,946 (10,229 + 12,717)	−4.2% (0.1%, −7.4%)	−4.2% (0.1%, −7.4%)
				35,281 (15,338 + 19,943)	32,174 (15,338 + 16,836)	−8.8% (0.0%, −15.6%)	32,174 (15,338 + 16,836)	−8.8% (0.0%, −15.6%)	0.0% (0.0%, 0.0%)
				25,984 (15,331 + 10,653)	25,953 (15,331 + 10,622)	−0.1% (0.0%, −0.3%)	24,950 (15,346 + 9,604)	−4.0% (0.1%, −9.8%)	−3.9% (0.1%, −9.6%)
				28,125 (15,390 + 12,735)	28,125 (15,390 + 12,735)	0.0% (0.0%, 0.0%)	27,053 (15,392 + 11,661)	−3.8% (0.0%, −8.4%)	−3.8% (0.0%, −8.4%)
10	30	1,000	1,000	27,174 (6,408 + 20,766)	24,114 (6,408 + 17,706)	−11.3% (0.0%, −14.7%)	24,113 (6,418 + 17,695)	−11.3% (0.2%, −14.8%)	0.0% (0.2%, −0.1%)
				28,405 (4,368 + 24,037)	25,261 (4,368 + 20,893)	−11.1% (0.0%, −13.1%)	25,255 (4,377 + 20,878)	−11.1% (0.2%, −13.1%)	0.0% (0.2%, −0.1%)
				34,791 (4,446 + 30,345)	31,695 (4,446 + 27,249)	−8.9% (0.0%, −10.2%)	30,497 (5,464 + 25,033)	−12.3% (22.9%, −17.5%)	−3.8% (22.9%, −8.1%)
				22,383 (5,501 + 16,882)	20,252 (5,501 + 14,751)	−9.5% (0.0%, −12.6%)	20,258 (5,501 + 14,757)	−9.5% (0.0%, −12.6%)	0.0% (0.0%, 0.0%)
				18,292 (4,432 + 13,860)	17,249 (4,432 + 12,817)	−5.7% (0.0%, −7.5%)	17,244 (4,438 + 12,806)	−5.7% (0.1%, −7.6%)	0.0% (0.1%, −0.1%)
		1,000	5,000	107,176 (6,408 + 100,768)	92,114 (6,408 + 85,706)	−14.1% (0.0%, −14.9%)	92,114 (6,408 + 85,706)	−14.1% (0.0%, −14.9%)	0.0% (0.0%, 0.0%)
				120,405 (4,368 + 116,037)	105,261 (4,368 + 100,893)	−12.6% (0.0%, −13.1%)	105,255 (4,377 + 100,878)	−12.6% (0.2%, −13.1%)	0.0% (0.2%, 0.0%)
				150,789 (4,446 + 146,343)	135,695 (4,446 + 131,249)	−10.0% (0.0%, −10.3%)	126,523 (5,460 + 121,063)	−16.1% (22.8%, −17.3%)	−6.8% (22.8%, −7.8%)
				86,379 (5,501 + 80,878)	76,258 (5,501 + 70,757)	−11.7% (0.0%, −12.5%)	76,256 (5,501 + 70,755)	−11.7% (0.0%, −12.5%)	0.0% (0.0%, 0.0%)
				70,289 (4,432 + 65,857)	65,249 (4,432 + 60,817)	−7.2% (0.0%, −7.7%)	65,249 (4,432 + 60,817)	−7.2% (0.0%, −7.7%)	0.0% (0.0%, 0.0%)
		5,000	1,000	51,176 (30,408 + 20,768)	48,114 (30,408 + 17,706)	−6.0% (0.0%, −14.7%)	48,113 (30,418 + 17,695)	−6.0% (0.0%, −14.8%)	0.0% (0.0%, −0.1%)
				44,405 (20,368 + 24,037)	41,261 (20,368 + 20,893)	−7.1% (0.0%, −13.1%)	40,417 (20,379 + 20,038)	−9.0% (0.1%, −16.6%)	−2.0% (0.1%, −4.1%)
				50,789 (20,446 + 30,343)	47,695 (20,446 + 27,249)	−6.1% (0.0%, −10.2%)	45,767 (20,457 + 25,310)	−9.9% (0.1%, −16.6%)	−4.0% (0.1%, −7.1%)
				42,383 (25,501 + 16,882)	40,252 (25,501 + 14,751)	−5.0% (0.0%, −12.6%)	40,250 (25,501 + 14,749)	−5.0% (0.0%, −12.6%)	0.0% (0.0%, 0.0%)
				34,289 (20,432 + 13,857)	33,249 (20,432 + 12,817)	−3.0% (0.0%, −7.5%)	33,244 (20,438 + 12,806)	−3.0% (0.0%, −7.6%)	0.0% (0.0%, −0.1%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis. (Continued on next page)

				Fixed	Semi-flexible		Flexible		
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	w_1	w_2	Objective Value	Objective Value	Change (Fixed)	Objective Value	Change (Fixed)	Change (Semi-flexible)
10	40	1,000	1,000	41,714 (9,621 + 32,093)	37,672 (9,621 + 28,051)	−9.7% (0.0%, −12.6%)	36,676 (9,621 + 27,055)	−12.1% (0.0%, −15.7%)	−2.6% (0.0%, −3.6%)
				41,895 (6,559 + 35,336)	37,759 (6,559 + 31,200)	−9.9% (0.0%, −11.7%)	37,752 (6,559 + 31,193)	−9.9% (0.0%, −11.7%)	0.0% (0.0%, 0.0%)
				44,083 (6,580 + 37,503)	40,980 (6,580 + 34,400)	−7.0% (0.0%, −8.3%)	39,995 (6,584 + 33,411)	−9.3% (0.1%, −10.9%)	−2.4% (0.1%, −2.9%)
				30,953 (7,693 + 23,260)	26,784 (7,693 + 19,091)	−13.5% (0.0%, −17.9%)	26,830 (7,693 + 19,137)	−13.3% (0.0%, −17.7%)	0.2% (0.0%, 0.2%)
				24,608 (6,574 + 18,034)	23,567 (6,574 + 16,993)	−4.2% (0.0%, −5.8%)	24,585 (6,574 + 18,011)	−0.1% (0.0%, −0.1%)	4.3% (0.0%, 6.0%)
		1,000	5,000	165,706 (9,621 + 156,085)	140,659 (9,621 + 131,038)	−15.1% (0.0%, −16.0%)	140,683 (9,621 + 131,062)	−15.1% (0.0%, −16.0%)	0.0% (0.0%, 0.0%)
				177,894 (6,559 + 171,335)	152,748 (6,559 + 146,189)	−14.1% (0.0%, −14.7%)	157,754 (6,564 + 151,190)	−11.3% (0.1%, −11.8%)	3.3% (0.1%, 3.4%)
				188,070 (6,580 + 181,490)	172,980 (6,580 + 166,400)	−8.0% (0.0%, −8.3%)	172,980 (6,580 + 166,400)	−8.0% (0.0%, −8.3%)	0.0% (0.0%, 0.0%)
				118,959 (7,693 + 111,266)	98,784 (7,693 + 91,091)	−17.0% (0.0%, −18.1%)	103,853 (7,693 + 96,160)	−12.7% (0.0%, −13.6%)	5.1% (0.0%, 5.6%)
				92,610 (6,574 + 86,036)	92,585 (6,574 + 86,011)	0.0% (0.0%, 0.0%)	87,613 (6,589 + 81,024)	−5.4% (0.2%, −5.8%)	−5.4% (0.2%, −5.8%)
		5,000	1,000	77,714 (45,621 + 32,093)	72,692 (45,621 + 27,071)	−6.5% (0.0%, −15.6%)	72,680 (45,627 + 27,053)	−6.5% (0.0%, −15.7%)	0.0% (0.0%, −0.1%)
				65,891 (30,559 + 35,332)	61,759 (30,559 + 31,200)	−6.3% (0.0%, −11.7%)	60,803 (30,558 + 30,245)	−7.7% (0.0%, −14.4%)	−1.5% (0.0%, −3.1%)
				68,076 (30,580 + 37,496)	64,980 (30,580 + 34,400)	−4.5% (0.0%, −8.3%)	64,023 (30,584 + 33,439)	−6.0% (0.0%, −10.8%)	−1.5% (0.0%, −2.8%)
				58,959 (35,693 + 23,266)	55,826 (35,693 + 20,133)	−5.3% (0.0%, −13.5%)	54,822 (35,693 + 19,129)	−7.0% (0.0%, −17.8%)	−1.8% (0.0%, −5.0%)
				48,608 (30,574 + 18,034)	48,575 (30,574 + 18,001)	−0.1% (0.0%, −0.2%)	44,723 (25,623 + 19,100)	−8.0% (−16.2%, 5.9%)	−7.9% (−16.2%, 6.1%)
10	50	1,000	1,000	62,486 (14,879 + 47,607)	52,382 (14,879 + 37,503)	−16.2% (0.0%, −21.2%)	53,355 (14,879 + 38,476)	−14.6% (0.0%, −19.2%)	1.9% (0.0%, 2.6%)
				52,261 (8,678 + 43,583)	48,205 (8,678 + 39,527)	−7.8% (0.0%, −9.3%)	48,157 (8,678 + 39,479)	−7.9% (0.0%, −9.4%)	−0.1% (0.0%, −0.1%)
				50,215 (7,669 + 42,546)	44,118 (7,669 + 36,449)	−12.1% (0.0%, −14.3%)	44,202 (7,671 + 36,531)	−12.0% (0.0%, −14.1%)	0.2% (0.0%, 0.2%)
				42,339 (10,880 + 31,459)	37,183 (10,880 + 26,303)	−12.2% (0.0%, −16.4%)	37,183 (10,880 + 26,303)	−12.2% (0.0%, −16.4%)	0.0% (0.0%, 0.0%)
				37,087 (8,740 + 28,347)	31,971 (8,740 + 23,231)	−13.8% (0.0%, −18.0%)	32,999 (8,758 + 24,241)	−11.0% (0.2%, −14.5%)	3.2% (0.2%, 4.3%)
		1,000	5,000	246,492 (14,879 + 231,613)	196,272 (14,879 + 181,393)	−20.4% (0.0%, −21.7%)	201,413 (14,879 + 186,534)	−18.3% (0.0%, −19.5%)	2.6% (0.0%, 2.8%)
				220,275 (8,678 + 211,597)	200,151 (8,678 + 191,473)	−9.1% (0.0%, −9.5%)	200,246 (8,678 + 191,568)	−9.1% (0.0%, −9.5%)	0.0% (0.0%, 0.0%)
				214,249 (7,669 + 206,580)	179,334 (7,669 + 171,665)	−16.3% (0.0%, −16.9%)	184,406 (7,669 + 176,737)	−13.9% (0.0%, −14.4%)	2.8% (0.0%, 3.0%)
				162,318 (10,880 + 151,438)	137,175 (10,880 + 126,295)	−15.5% (0.0%, −16.6%)	137,170 (10,880 + 126,290)	−15.5% (0.0%, −16.6%)	0.0% (0.0%, 0.0%)
				145,072 (8,740 + 136,332)	124,942 (8,740 + 116,202)	−13.9% (0.0%, −14.8%)	124,900 (8,745 + 116,155)	−13.9% (0.1%, −14.8%)	0.0% (0.1%, 0.0%)
		5,000	1,000	118,504 (70,879 + 47,625)	108,316 (70,879 + 37,437)	−8.6% (0.0%, −21.4%)	109,322 (70,879 + 38,443)	−7.7% (0.0%, −19.3%)	0.9% (0.0%, 2.7%)
				84,275 (40,678 + 43,597)	80,144 (40,678 + 39,466)	−4.9% (0.0%, −9.5%)	80,200 (40,680 + 39,520)	−4.8% (0.0%, −9.4%)	0.1% (0.0%, 0.1%)
				78,215 (35,669 + 42,546)	72,126 (35,669 + 36,457)	−7.8% (0.0%, −14.3%)	72,232 (35,669 + 36,563)	−7.6% (0.0%, −14.1%)	0.1% (0.0%, 0.3%)
				82,341 (50,880 + 31,461)	77,159 (50,880 + 26,279)	−6.3% (0.0%, −16.5%)	77,209 (50,880 + 26,329)	−6.2% (0.0%, −16.3%)	0.1% (0.0%, 0.2%)
				69,081 (40,740 + 28,341)	64,950 (40,740 + 24,210)	−6.0% (0.0%, −14.6%)	64,963 (40,740 + 24,223)	−6.0% (0.0%, −14.5%)	0.0% (0.0%, 0.1%)

Table 3.10: Comparison of the solutions from CP-LNS. Vehicle and crew objective values are shown inside parenthesis.

Chapter 4

The Vehicle Routing Problem with Location Congestion

Many real-world vehicle routing problems feature constraints that are seldom considered in academic variants. This chapter explores a rich variant named the Vehicle Routing Problem with Pickup and Delivery, Time Windows and Location Congestion (VRPPDTWLC or VRPLC for short). The VRPLC is motivated by applications in humanitarian and military logistics, where Air Force bases have limited parking spots, fuel reserve and landing and takeoff times for airplane operations. At the modeling level, the VRPLC is based on the Pickup and Delivery Problem and Time Windows (PDPTW) but extends it with one cumulative resource constraint at every location to limit the number of vehicles present and/or in service at any given time. Examples of resources can include parking bays for the first case and loading equipment for the second case. If all resources at a location are in use, incoming vehicles cannot proceed but must wait until a resource becomes available. In other words, the vehicles must be scheduled around the availability of the resources. The scheduling element leads to temporal dependencies between vehicles, which are not present in conventional vehicle routing problems. In particular, a delay on one route may entail a delay on another route if both routes visit a common location. Furthermore, the delay on the second route may cause it to become infeasible because of a time window violation, for example.

This chapter studies a mixed integer programming (MIP) model, a constraint programming (CP) model, a branch-and-price-and-check (BPC) model and a two-stage (TS) sequential model for the VRPLC. The sequential method divides the problem into a routing stage that ignores the scheduling constraints, followed by a scheduling stage that schedules the vehicles over the routes from the first stage. The BPC approach, inspired by a branch-and-cut-and-price model of the PDPTW, combines a branch-and-price algorithm that solves the PDPTW and a constraint programming subproblem that lifts the PDPTW to the VRPLC by checking the PDPTW solutions against the location resource constraints. If these constraints are violated, a combinatorial Benders cut (also known as a nogood) is added to the master problem to prohibit

As described in the Preface, the main findings of this chapter are published in the paper titled "A branch-and-price-and-check model for the vehicle routing problem with location congestion".

this PDPTW solution.

The four models are evaluated on instances with up to 300 requests (150 pickup-delivery pairs) and both types of resources. Results indicate that the BPC algorithm scales better than both the mixed integer programming and the constraint programming models by finding feasible solutions to the largest instances and optimal solutions to instances with up to 160 requests (80 pickup-delivery requests). Neither the mixed integer programming model nor the constraint programming model find feasible solutions to the large instances. The sequential method finds feasible solutions to some of the large instances but fails on the instances for which the scheduling constraints are binding, indicating that the routing and scheduling aspects must be considered simultaneously.

The remainder of this chapter is structured as follows. Section 4.1 reviews prior work on relevant problems. Section 4.2 describes the VRPLC. Sections 4.3 and 4.4 respectively present the mixed integer programming model and the constraint programming model. Section 4.5 discusses the BPC approach. Section 4.6 reports experimental results, and Section 4.7 concludes this chapter.

4.1 Literature Review

The BPC model is based on the branch-and-cut-and-price model of the PDPTW by Røpke and Cordeau (2009), which is previously discussed in Section 2.8.1. The remainder of this section reviews task scheduling problems, which are somewhat related to routing problems, and several vehicle routing problems with scheduling constraints.

In task scheduling problems, a number of tasks must be processed by various machines and the objective may involve minimizing makespan or tardiness. An example of a scheduling problem is the Resource-Constrained Project Scheduling Problem, which requires tasks of fixed duration to be scheduled on a number of machines such that precedence constraints and resource capacities are respected. Many benchmark instances of these problems are closed using constraint programming (Schutt et al. 2010).

Beck, Prosser and Selensky (2002, 2003) showed that some scheduling problems can be translated into routing problems and vice versa. However, few problems exhibit both routing and scheduling constraints simultaneously. The VRPLC is one such example: it overlays either a regular Resource-Constrained Project Scheduling Problem or a variable-duration version on top of the PDPTW. In the VRPLC, vehicles must coordinate their schedules in order to satisfy resource capacities at locations. Hence, the VRPLC is classified as a rich vehicle routing problem with synchronization (Drexl 2012). In the VRPLC, synchronization refers to the interaction between vehicles, and in particular, the transfer of information about the schedule of one vehicle to another.

Only a limited number of vehicle routing problems have appeared in the literature with resource scheduling constraints at locations. Hempsch and Irnich (2008) modeled resource constraints at destination depot locations by redefining individual routes as segments within a single long route.

Log Truck Scheduling Problems (LTSPs) are a family of forestry applications generalized from the PDPTW. El Hachemi, Gendreau and Rousseau (2013) modeled an LTSP that features location resource synchronizations similar to those in the VRPLC. Pickup and delivery requests are located at sites, each of which has a single log-loader that is required to load and unload a vehicle. A vehicle arriving at a site with an occupied log-loader must wait until the log-loader is free to service a request. Similar to the VRPLC, this LTSP optimizes the vehicle routes and the schedules of the log-loaders. The problem is solved using scheduling constraints in a constraint programming model.

4.2 The High-Level Description

This section describes the VRPLC. The VRPLC is based on the PDPTW, which is introduced in Section 2.7. The VRPLC augments the PDPTW with two major additions: the explicit modeling of locations and their resources.

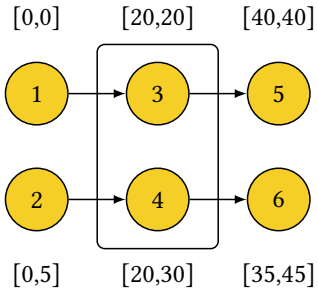
In traditional vehicle routing problems, such as the PDPTW, requests and locations are synonymous: each request is assumed to be at its own location, even if it overlaps another location. In the VRPLC, requests are grouped by location, and the locations cannot overlap. Distance costs and travel times are defined between locations. Moreover, every location features a number of cumulative resources with a fixed capacity that cannot be exceeded at any given time. Whenever all resources are in use, incoming vehicles cannot proceed until a resource becomes available for use. Two types of resources are considered: a *service* resource is used while a request is in service, and a *presence* resource is used by a vehicle from the moment it arrives at a location until it departs from the location. Service resources are fully encompassed by presence resources, since the start of a visit is necessarily before the start of service, and the end of a visit after the end of service. This chapter only considers the case where all locations have only service resources or only presence resources. However, the formulation naturally generalizes to multiple copies of these types of resources.

Contrary to traditional vehicle routing problems, delaying a route in the VRPLC may affect the feasibility of other routes since a delayed vehicle can postpone the availability of a resource required by another vehicle. The delay incurred by the second vehicle can cause a time window violation on its route. Figure 4.1 shows an example of a time window violation. Temporal interactions between vehicles require reasoning about the timing and scheduling of the vehicle visits and make the VRPLC more challenging to solve than their more conventional counterparts.

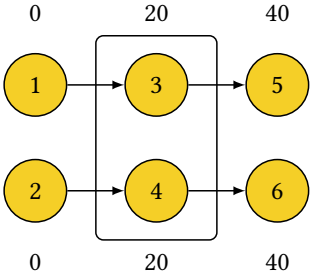
4.3 The Mixed Integer Programming Model

This section presents the mixed integer programming model of the VRPLC. The model is based on a regular three-index formulation of the PDPTW, but includes additional time variables to record the arrival and departure times at locations, and the location resource constraints.

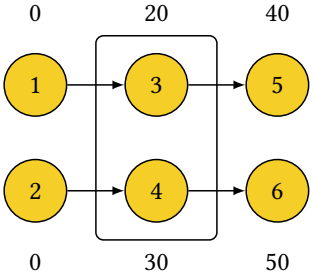
Table 4.1 lists the data and decision variables of the mixed integer programming model. The problem is solved over a time interval $\mathcal{T} = [0, T]$, where $T > 0$. The problem contains V vehicles,



(a) Consider six requests with service durations of 10 time units and time windows shown in the figure. The travel time between any two nodes is 10 time units. Requests 3 and 4 are situated at a common location with a service resource capacity of one. Also consider a vehicle that visits requests 1, 3 and 5 and another vehicle that visits requests 2, 4 and 6.



(b) The service start times shown in the figure is feasible in traditional VRPs without synchronization.



(c) With service resources, request 4 cannot be started until time 30 because the resource is occupied by request 3 from time 20 to 30. The delay invalidates the solution in Figure 4.1b at request 6.

Figure 4.1: Example of a time window violation after delaying a vehicle due to insufficient location resources.

Variable	Description
$T > 0$	Time horizon.
$\mathcal{T} = [0, T]$	Time interval.
$V \in \{1, \dots, \infty\}$	Number of vehicles.
$\mathcal{V} = \{1, \dots, V\}$	Set of vehicles.
$Q \geq 0$	Vehicle capacity.
$P \in \{1, \dots, \infty\}$	Total number of pickup-delivery pairs.
$R = 2P$	Total number of requests.
$\mathcal{P} = \{1, \dots, P\}$	Set of pickup requests/nodes.
$\mathcal{D} = \{P + 1, \dots, R\}$	Set of delivery requests/nodes.
$\mathcal{R} = \mathcal{P} \cup \mathcal{D}$	Set of all requests.
s	Start node.
e	End node.
$\mathcal{N} = \mathcal{R} \cup \{s, e\}$	Set of all nodes.
$L \in \{1, \dots, \infty\}$	Number of locations, excluding the depot location.
$\mathcal{L} = \{1, \dots, L\}$	Set of locations.
$\mathcal{R}_l = \{i \in \mathcal{R} \mid l_i = l\}$	Requests at location $l \in \mathcal{L}$.
$C_l \in \{1, \dots, \infty\}$	Resource capacity of location $l \in \mathcal{L}$.
$\mathcal{K}_l = \{1, \dots, 2\lvert\mathcal{R}_l\rvert\}$	Set of events at location $l \in \mathcal{L}$.
$l_i \in \mathcal{L}$	Location of $i \in \mathcal{R}$.
$q_i \in [-Q, Q]$	Demand at $i \in \mathcal{N}$.
$a_i \in \mathcal{T}$	Earliest start of service at $i \in \mathcal{N}$.
$b_i \in \mathcal{T}$	Latest start of service at $i \in \mathcal{N}$.
$t_i \in \mathcal{T}$	Service duration of $i \in \mathcal{N}$.
$\mathcal{A}$	Set of arcs. Defined in Equation (4.1).
$d_{i,j} \in \mathcal{T}$	Distance and travel time along the arc $(i, j) \in \mathcal{A}$.
$\text{flow}_{v,i,j} \in \{0, 1\}$	Indicates if vehicle $v \in \mathcal{V}$ traverses $(i, j) \in \mathcal{A}$.
$\text{visit}_{v,i} \in \{0, 1\}$	Indicates if vehicle $v \in \mathcal{V}$ visits $i \in \mathcal{R}$.
$\text{load}_{v,i} \in [0, Q]$	Load of vehicle $v \in \mathcal{V}$ after servicing $i \in \mathcal{N}$.
$\text{arr}_{v,i} \in \mathcal{T}$	Arrival time of vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{serv}_{v,i} \in [a_i, b_i]$	Start of service by vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{servEnd}_{v,i} \in [a_i + t_i, b_i + t_i]$	End of service by vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{dep}_{v,i} \in \mathcal{T}$	Departure time of vehicle $v \in \mathcal{V}$ at $i \in \mathcal{N}$.
$\text{start}_{l,k,i} \in \{0, 1\}$	Indicates if $k \in \mathcal{K}_l$ represents the start event of $i \in \mathcal{R}_l$.
$\text{end}_{l,k,i} \in \{0, 1\}$	Indicates if $k \in \mathcal{K}_l$ represents the end event of $i \in \mathcal{R}_l$.
$\text{use}_{l,k} \in \{0, \dots, C_l\}$	Resource use after event $k \in \mathcal{K}_l \cup \{0\}$ at $l \in \mathcal{L}$.
$\text{time}_{l,k} \in \mathcal{T}$	Time of event $k \in \mathcal{K}_l \cup \{0\}$ at $l \in \mathcal{L}$.

Table 4.1: The data and decision variables of the mixed integer programming model.

which are represented by the set $\mathcal{V} = \{1, \dots, V\}$. Every vehicle has a capacity $Q \geq 0$.

The vehicles need to pick up and deliver P requests. The pickup requests are labeled 1 to P and their corresponding deliveries are labeled $P + 1$ to $2P$. Let $R = 2P$ be the total number of requests. The pickups are represented by the set $\mathcal{P} = \{1, \dots, P\}$, and the deliveries by the set $\mathcal{D} = \{P + 1, \dots, R\}$. The set $\mathcal{R} = \mathcal{P} \cup \mathcal{D}$ represents all requests, and the set $\mathcal{N} = \mathcal{R} \cup \{s, e\}$ represents the nodes of the underlying graph, which has a node for each request $i \in \mathcal{R}$ and two nodes denoting the start s and end e of every vehicle route. Each pickup $i \in \mathcal{P}$ has a vehicle load demand of $q_i \in [0, Q]$, and its corresponding delivery $P + i$ has a demand of $q_{P+i} = -q_i \in [-Q, 0]$. The start and end nodes have zero vehicle load demand, i.e., $q_s = 0$ and $q_e = 0$. Each node $i \in \mathcal{N}$ has an earliest start time of service $a_i \in \mathcal{T}$, latest start time of service $b_i \in \mathcal{T}$ and service duration $t_i \in \mathcal{T}$.

The requests are distributed across L locations, which are contained in the set $\mathcal{L} = \{1, \dots, L\}$. Each request i is found at location $l_i \in \mathcal{L}$. The set $\mathcal{R}_l = \{i \in \mathcal{R} | l_i = l\}$ represents the set of all requests situated at location $l \in \mathcal{L}$. The problem specifies that either all locations only have service resources or all locations only have presence resources. This restriction can be removed with an easy generalization. Each location $l \in \mathcal{L}$ has a resource capacity $C_l \in \{1, \dots, \infty\}$.

The resources are scheduled using event variables (e.g., Hooker 2007, Koné et al. 2011). Every request $i \in \mathcal{R}_l$ at l is associated with two events: a start event and an end event. For service resources, the start and end events correspond to the start and end of service. For presence resources, the start and end events correspond to the arrival and departure of the servicing vehicle. The location resources are then scheduled by tracking the number of start and end events at each location. The set $\mathcal{K}_l = \{1, \dots, 2|\mathcal{R}_l|\}$ is the events at location $l \in \mathcal{L}$.

The nodes in the underlying graph are connected by the arcs

$$\mathcal{A} = \{(s, i) | i \in \mathcal{P}\} \cup \{(i, j) | i \in \mathcal{R}, j \in \mathcal{R}, i \neq j\} \cup \{(i, e) | i \in \mathcal{D}\} \cup \{(s, e)\}. \tag{4.1}$$

Each arc $(i, j) \in \mathcal{A}$ has an associated cost/travel time $d_{i,j} \in \mathcal{T}$. For any two locations $l_1, l_2 \in \mathcal{L}$, all arcs from requests at l_1 to requests at l_2 have the same cost.

The primary decision variables are the $\text{flow}_{v,i,j}$ variables, which indicate if vehicle v traverses the arc (i, j). The $\text{visit}_{v,i}$ variable indicates if vehicle v visits request i. The $\text{load}_{v,i}$, $\text{arr}_{v,i}$, $\text{serv}_{v,i}$, $\text{servEnd}_{v,i}$ and $\text{dep}_{v,i}$ variables respectively represent the vehicle load, arrival time, start time of service, end time of service and departure time at request i. The $\text{use}_{l,k}$ and $\text{time}_{l,k}$ variables contain the resource use after event k and the time of event k at location l.

Figure 4.2 depicts the three-index flow model underpinning the mixed integer programming model of the VRPLC. The symbol M denotes an appropriate big-M constant. Objective Function (4.2) minimizes the total travel distance. Constraints (4.3) to (4.5) are the flow constraints, which ensure that every vehicle follows a path beginning at the start node through to the end node. Constraint (4.6) links the flow variables to the visit indicator variables. Constraint (4.7) ensures that every request is visited. Constraints (4.8) and (4.9) are the pickup-delivery constraints, which require the corresponding delivery request of a pickup to occur after the pickup and on the same vehicle. Constraint (4.10) zeroes the load demand at the start and end nodes. Constraints (4.11) and (4.12) respectively bound the load variables at pickup and delivery nodes.

$$\min \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} d_{i,j} \text{flow}_{v,i,j} \tag{4.2}$$

subject to

$$\sum_{j:(s,j) \in \mathcal{A}} \text{flow}_{v,s,j} = 1 \qquad \forall v \in \mathcal{V}, \tag{4.3}$$

$$\sum_{h:(h,i) \in \mathcal{A}} \text{flow}_{v,h,i} = \sum_{j:(i,j) \in \mathcal{A}} \text{flow}_{v,i,j} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \tag{4.4}$$

$$\sum_{h:(h,e) \in \mathcal{A}} \text{flow}_{v,h,e} = 1 \qquad \forall v \in \mathcal{V}, \tag{4.5}$$

$$\sum_{h:(h,i) \in \mathcal{A}} \text{flow}_{v,h,i} = \text{visit}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \tag{4.6}$$

$$\sum_{v \in \mathcal{V}} \text{visit}_{v,i} = 1 \qquad \forall i \in \mathcal{P}, \tag{4.7}$$

$$\text{visit}_{v,i} = \text{visit}_{v,P+i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \tag{4.8}$$

$$\text{dep}_{v,i} + d_{i,P+i} \leq \text{arr}_{v,P+i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \tag{4.9}$$

$$\text{load}_{v,i} = 0 \qquad \forall v \in \mathcal{V}, i \in \{s, e\}, \tag{4.10}$$

$$q_i \leq \text{load}_{v,i} \leq Q \qquad \forall v \in \mathcal{V}, i \in \mathcal{P}, \tag{4.11}$$

$$0 \leq \text{load}_{v,i} \leq Q + q_i \qquad \forall v \in \mathcal{V}, i \in \mathcal{D}, \tag{4.12}$$

$$\text{load}_{v,i} + q_j - \text{load}_{v,j} \leq M(1 - \text{flow}_{v,i,j}) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}, \tag{4.13}$$

$$\text{arr}_{v,i} \leq \text{serv}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \tag{4.14}$$

$$\text{serv}_{v,i} + t_i = \text{servEnd}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \tag{4.15}$$

$$\text{servEnd}_{v,i} \leq \text{dep}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \mathcal{R}, \tag{4.16}$$

$$\text{arr}_{v,i} = \text{serv}_{v,i} = \text{servEnd}_{v,i} = \text{dep}_{v,i} \qquad \forall v \in \mathcal{V}, i \in \{s, e\}, \tag{4.17}$$

$$\text{dep}_{v,i} + d_{i,j} - \text{arr}_{v,j} \leq M(1 - \text{flow}_{v,i,j}) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}, \tag{4.18}$$

$$\text{arr}_{v,j} - \text{dep}_{v,i} - d_{i,j} \leq M(1 - \text{flow}_{v,i,j}) \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}. \tag{4.19}$$

Figure 4.2: The constraints of the three-index flow model within the mixed integer programming model.

$$\sum_{i \in \mathcal{R}_l} \text{start}_{l,k,i} + \sum_{i \in \mathcal{R}_l} \text{end}_{l,k,i} = 1 \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, \quad (4.20)$$

$$\sum_{k \in \mathcal{K}_l} \text{start}_{l,k,i} = 1 \qquad \forall l \in \mathcal{L}, i \in \mathcal{R}_l, \quad (4.21)$$

$$\sum_{k \in \mathcal{K}_l} \text{end}_{l,k,i} = 1 \qquad \forall l \in \mathcal{L}, i \in \mathcal{R}_l, \quad (4.22)$$

$$\text{use}_{l,0} = 0 \qquad \forall l \in \mathcal{L}, \quad (4.23)$$

$$\text{use}_{l,k} = \text{use}_{l,k-1} + \sum_{i \in \mathcal{R}_l} \text{start}_{l,k,i} - \sum_{i \in \mathcal{R}_l} \text{end}_{l,k,i} \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, \quad (4.24)$$

$$\text{time}_{l,0} = 0 \qquad \forall l \in \mathcal{L}, \quad (4.25)$$

$$\text{time}_{l,k-1} \leq \text{time}_{l,k} \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, \quad (4.26)$$

$$\text{time}_{l,k} - \text{serv}_{v,i} \leq M(2 - \text{start}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l, \quad (4.27)$$

$$\text{serv}_{v,i} - \text{time}_{l,k} \leq M(2 - \text{start}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l, \quad (4.28)$$

$$\text{time}_{l,k} - \text{servEnd}_{v,i} \leq M(2 - \text{end}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l, \quad (4.29)$$

$$\text{servEnd}_{v,i} - \text{time}_{l,k} \leq M(2 - \text{end}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l. \quad (4.30)$$

Figure 4.3: The constraints for service resources in the mixed integer programming model.

Constraint (4.13) is the usual load constraint. Constraints (4.14) to (4.16) order the arrival, service start, service end, and departure times at each request. Constraint (4.17) constrains the start node and end node to one common arrival/service/departure time. Constraints (4.18) and (4.19) are the travel time constraints, which together state that $\text{arr}_{v,j} = \text{dep}_{v,i} + d_{i,j}$ whenever $\text{flow}_{v,i,j} = 1$.

The novelty of the model lies in the additional constraints in Figure 4.3, which supplement the constraints of Figure 4.2 to model service resources. Constraint (4.20) requires every event to be classified as either a start event or an end event. Constraints (4.21) and (4.22) state that every request has a start event and an end event. Constraint (4.23) initializes the resource use count. Constraint (4.24) accumulates the resource use after every event. Constraint (4.25) initializes the event time variables. Constraint (4.26) removes symmetries in the index of the events by enforcing an ordering. Constraints (4.27) to (4.30) are implication constraints that link the event time variables to the route time variables. To model presence resources, Constraints (4.27) to (4.30) are replaced with

$$\text{time}_{l,k} - \text{arr}_{v,i} \leq M(2 - \text{start}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l,$$

$$\text{arr}_{v,i} - \text{time}_{l,k} \leq M(2 - \text{start}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l,$$

$$\text{time}_{l,k} - \text{dep}_{v,i} \leq M(2 - \text{end}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l,$$

$$\text{dep}_{v,i} - \text{time}_{l,k} \leq M(2 - \text{end}_{l,k,i} - \text{visit}_{v,i}) \qquad \forall l \in \mathcal{L}, k \in \mathcal{K}_l, v \in \mathcal{V}, i \in \mathcal{R}_l.$$

These constraints link the event time variables to the arrival and departure times instead of the service start and end times.

4.4 The Constraint Programming Model

This section presents the constraint programming model for the VRPLC. It is adapted from the standard constraint programming model of vehicle routing problems using successor variables, which was explained in Section 2.8.2.

Table 4.2 lists the data and decision variables of the model. The data variables are mostly same as the those of the mixed integer programming model except that many of the sets are now discrete rather than continuous. One major difference is that each vehicle $v \in \mathcal{V}$ has its own start node $s(v) = R + v$ and end node $e(v) = R + V + v$. The set $\mathcal{S} = \{R + 1, \dots, R + V\}$ represents all start nodes, and $\mathcal{E} = \{R + V + 1, \dots, R + 2V\}$ represents all end nodes.

The primary decision variables are the successor variables $\text{succ}(i)$, which determine the route of each vehicle. The successor variables trace a path from the start node of a vehicle through a number of nodes to the end node of a vehicle. For example, if $\text{succ}(s(1)) = 1$, $\text{succ}(1) = 2$, $\text{succ}(2) = e(1)$, then vehicle 1 departs its start node $s(1)$ for requests 1 and 2 and then finishes at its end node $e(1)$.

The $\text{vec}(i)$ variable contains the index of the vehicle that visits request $i \in \mathcal{N}$. The $\text{load}(i)$, $\text{arr}(i)$, $\text{serv}(i)$, $\text{dep}(i)$, and $\text{dur}(i)$ variables respectively contain the load, arrival time, service start time, departure time, and visit duration at request i.

Figure 4.4 depicts the constraints. Objective Function (4.31) minimizes the total travel distance. Constraints (4.32) to (4.34) are the domain restrictions that ensure the requests along a route are correctly ordered. Constraints (4.35) and (4.36) link the end of a route to the start of the next route, and enables the Circuit global constraint in Constraint (4.37) to eliminate subtours. Constraints (4.38) and (4.39) track the requests visited by a vehicle along its route. Constraints (4.40) and (4.41) are the pickup and delivery constraints. Constraints (4.42) to (4.45) are the usual load constraints. Constraints (4.46) and (4.47) order the arrival, service and departure times at each request. Constraint (4.48) calculates the duration of each visit. Constraint (4.49) enforces a common arrival/service/departure time at the start and end depot nodes. Constraint (4.50) implements travel times. Constraint (4.51) models the service resources. For presence resources, it can be replaced with

$$\textsc{Cumulative}(\{\text{arr}(i) | i \in \mathcal{R}_l\}, \{\text{dur}(i) | i \in \mathcal{R}_l\}, \mathbf{1}, C_l) \qquad \forall l \in \mathcal{L}. \quad (4.52)$$

The resource constraints are modeled using the $\textsc{Cumulative}(\mathbf{s}, \mathbf{d}, \mathbf{r}, C)$ global constraint (Aggoun and Beldiceanu 1993). Each component of the $\mathbf{s}$, $\mathbf{d}$ and $\mathbf{r}$ vectors respectively denotes the start time, duration and resource requirement of a task. The value C is the maximum capacity of the resource on which the tasks are to be scheduled. It is important to note that the service durations are fixed by the instance data, whereas the presence durations are variables.

Name	Description
$T \in \{1, \dots, \infty\}$	Time horizon.
$\mathcal{T} = \{0, \dots, T\}$	Set of time values.
$V \in \{1, \dots, \infty\}$	Number of vehicles.
$\mathcal{V} = \{1, \dots, V\}$	Set of vehicles.
$Q \in \{0, \dots, \infty\}$	Vehicle capacity.
$P \in \{1, \dots, \infty\}$	Total number of pickup-delivery pairs.
$R = 2P$	Total number of requests.
$\mathcal{P} = \{1, \dots, P\}$	Set of pickup requests/nodes.
$\mathcal{D} = \{P+1, \dots, R\}$	Set of delivery requests/nodes.
$\mathcal{R} = \mathcal{P} \cup \mathcal{D}$	Set of all requests.
$\mathcal{S} = \{R+1, \dots, R+V\}$	Set of vehicle start nodes.
$\mathcal{E} = \{R+V+1, \dots, R+2V\}$	Set of vehicle end nodes.
$s(v) = R + v$	Start node of vehicle $v \in \mathcal{V}$.
$e(v) = R + V + v$	End node of vehicle $v \in \mathcal{V}$.
$\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$	Set of all nodes.
$L \in \{1, \dots, \infty\}$	Number of locations, excluding the depot location.
$\mathcal{L} = \{1, \dots, L\}$	Set of locations.
$C_l \in \{1, \dots, \infty\}$	Resource capacity of location $l \in \mathcal{L}$.
$\mathcal{R}_l = \{i \in \mathcal{R} \mid l(i) = l\}$	Requests at location $l \in \mathcal{L}$.
$l(i) \in \mathcal{L}$	Location of $i \in \mathcal{R}$.
$d(i,j) \in \mathcal{T}$	Distance and travel time from $i \in \mathcal{N}$ to $j \in \mathcal{N}$.
$a(i) \in \mathcal{T}$	Earliest start of service at $i \in \mathcal{N}$.
$b(i) \in \mathcal{T}$	Latest start of service at $i \in \mathcal{N}$.
$t(i) \in \mathcal{T}$	Service duration of $i \in \mathcal{N}$.
$q(i) \in \{-Q, \dots, Q\}$	Demand at $i \in \mathcal{N}$.
$\text{succ}(i) \in \mathcal{N}$	Successor node of $i \in \mathcal{N}$.
$\text{veh}(i) \in \mathcal{V}$	Vehicle that visits $i \in \mathcal{N}$.
$\text{load}(i) \in \{0, \dots, Q\}$	Load of vehicle $\text{veh}(i)$ after servicing $i \in \mathcal{N}$.
$\text{arr}(i) \in \mathcal{T}$	Arrival time at $i \in \mathcal{N}$.
$\text{serv}(i) \in \{a(i), \dots, b(i)\}$	Start of service at $i \in \mathcal{N}$.
$\text{dep}(i) \in \mathcal{T}$	Departure time at $i \in \mathcal{N}$.
$\text{dur}(i) \in \mathcal{T}$	Visit duration at $i \in \mathcal{R}$.

Table 4.2: The data and decision variables of the constraint programming model.

$$\min \sum_{i \in \mathcal{R} \cup \mathcal{S}} d(i, \text{succ}(i)) \tag{4.31}$$

subject to

$$\text{succ}(s(v)) \in \mathcal{P} \cup \{e(v)\} \qquad \forall v \in \mathcal{V}, \tag{4.32}$$

$$\text{succ}(i) \in \mathcal{P} \cup \mathcal{D} \qquad \forall i \in \mathcal{P}, \tag{4.33}$$

$$\text{succ}(i) \in \mathcal{P} \cup \mathcal{D} \cup \mathcal{E} \qquad \forall i \in \mathcal{D}, \tag{4.34}$$

$$\text{succ}(e(v)) = s(v+1) \qquad \forall v \in \{1, \dots, V-1\}, \tag{4.35}$$

$$\text{succ}(e(V)) = s(1), \tag{4.36}$$

$$\textsc{Circuit}(\text{succ}(\cdot)), \tag{4.37}$$

$$\text{veh}(s(v)) = \text{veh}(e(v)) = v \qquad \forall v \in \mathcal{V}, \tag{4.38}$$

$$\text{veh}(i) = \text{veh}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \tag{4.39}$$

$$\text{veh}(i) = \text{veh}(P+i) \qquad \forall i \in \mathcal{P}, \tag{4.40}$$

$$\text{dep}(i) + d(i, P+i) \leq \text{arr}(P+i) \qquad \forall i \in \mathcal{P}, \tag{4.41}$$

$$\text{load}(i) = 0 \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \tag{4.42}$$

$$q(i) \leq \text{load}(i) \leq Q \qquad \forall i \in \mathcal{P}, \tag{4.43}$$

$$0 \leq \text{load}(i) \leq Q + q(i) \qquad \forall i \in \mathcal{D}, \tag{4.44}$$

$$\text{load}(i) + q(\text{succ}(i)) = \text{load}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \tag{4.45}$$

$$\text{arr}(i) \leq \text{serv}(i) \qquad \forall i \in \mathcal{R}, \tag{4.46}$$

$$\text{serv}(i) + t(i) \leq \text{dep}(i) \qquad \forall i \in \mathcal{R}, \tag{4.47}$$

$$\text{dur}(i) = \text{dep}(i) - \text{arr}(i) \qquad \forall i \in \mathcal{R}, \tag{4.48}$$

$$\text{arr}(i) = \text{serv}(i) = \text{dep}(i) \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \tag{4.49}$$

$$\text{dep}(i) + d(i, \text{succ}(i)) = \text{arr}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \tag{4.50}$$

$$\textsc{Cumulative}(\{\text{serv}(i) | i \in \mathcal{R}_l\}, \{t(i) | i \in \mathcal{R}_l\}, \mathbf{1}, C_l) \qquad \forall l \in \mathcal{L}. \tag{4.51}$$

Figure 4.4: The constraints in the constraint programming model.

4.5 The Branch-and-Price-and-Check Model

This section describes the branch-and-price-and-check (BPC) approach to the VRPLC. The BPC algorithm builds upon the ideas of the branch-and-cut-and-price model of the PDPTW by Røpke and Cordeau (2009), which employs column generation to produce routes, and separation subproblems to find cuts that forbid certain classes of infeasible routes. Although branch-and-price itself can solve the problem, the various families of cuts add problem-specific knowledge and greatly improve convergence. In other words, these cuts are implied cuts, as discussed in Section 2.3.2, since they are not necessary to correctly formulate the problem.

The BPC model follows a similar approach; the difference is that the cuts generated by the separation subproblem enforce the location resource constraints. These cuts are problem cuts, not implied cuts, since they are necessary to solve the problem correctly, and they do not appear elsewhere in the problem. In this sense, BPC integrates Benders decomposition/branch-and-check into a branch-and-price framework.

Implied cuts and problem cuts are previously discussed in Section 2.3.2, and Benders decomposition is introduced in Section 2.6.4. The rest of this section is organized as follows. Sections 4.5.1 to 4.5.3 introduce the master, pricing and separation problems. Section 4.5.4 discusses the search tree and branching rules.

4.5.1 The Master Problem

The (restricted) master problem, depicted in Figure 4.5, is the linear relaxation of the Set Partitioning problem, which is commonly seen in column generation models of vehicle routing problems. The Set Partitioning problem selects a subset of routes from a main pool Ω of routes such that this subset satisfies certain constraints. The variable $x_r \in [0, 1]$ denotes whether route $r \in \Omega$, with cost c_r, is selected.

Objective Function (4.53) minimizes the total cost of the subset. Constraint (4.54) ensures that all pickup requests are visited. The coefficient $a_{i,r}$ is equal to 1 if route r visits request $i \in \mathcal{P}$ and is equal to 0 otherwise. The constraint that every request is picked up and delivered on the same route appears in the pricing subproblem; hence, the requirement that all deliveries are fulfilled is satisfied by Constraint (4.54). Constraint (4.55) imposes the nogood cuts. Every nogood cut has an associated set B of arcs and a coefficient B_r that denotes the number of arcs in B that are traversed by route r. When a set of routes is determined to be infeasible by the separation subproblem, one nogood cut is added to the master problem, with B containing the arcs traversed by the routes in this set. Hence, the nogood cut prohibits this set of routes by allowing at most $|B| - 1$ of their arcs to be used in any feasible solution.

4.5.2 The Pricing Subproblem

The pricing subproblem solves an elementary resource-constrained shortest path problem to generate routes to add to Ω. It implements a labeling algorithm (e.g., Deo and Pang 1984, Nemani and Ahuja 2011) seen in many vehicle routing problems (e.g., Dumas, Desrosiers and Soumis 1991, Irnich and Desaulniers 2005, Røpke and Cordeau 2009). Each route must:

$$\min \sum_{r \in \Omega} c_r x_r \tag{4.53}$$

subject to

$$\sum_{r \in \Omega} a_{i,r} x_r = 1 \qquad \forall i \in \mathcal{P}, \tag{4.54}$$

$$\sum_{r \in \Omega} B_r x_r \leq |B| - 1 \qquad \forall B \in \mathcal{B}, \tag{4.55}$$

$$x_r \in [0, 1] \qquad \forall r \in \Omega. \tag{4.56}$$

Figure 4.5: The master problem of the branch-and-price-and-check model.

1. leave the start node, visit a number of requests, and terminate at the end node,
2. satisfy the service start time window, travel time, load and pickup-delivery constraints, and
3. have negative cost with respect to the reduced cost matrix $\bar{d}_{i,j}$, which is defined as

$$\bar{d}_{i,j} = \begin{cases} d_{i,j} - \pi_i + \sum_{B \in \mathcal{B}} 1_{B,i,j} \mu_B & \forall i \in \mathcal{P}, j \in \mathcal{N}, \\ d_{i,j} \qquad\quad + \sum_{B \in \mathcal{B}} 1_{B,i,j} \mu_B & \forall i \in \mathcal{N} \setminus \mathcal{P}, j \in \mathcal{N}, \end{cases}$$

   where $\mathcal{P}$ is the set of pickups, $\mathcal{N}$ is the set of all nodes (pickups, deliveries, one start node and one end node), $d_{i,j}$ is the distance cost from $i \in \mathcal{N}$ to $j \in \mathcal{N}$, π_i is the dual value of Constraint (4.54), $1_{B,i,j}$ is equal to 1 if the arc (i, j) appears in B and is equal to 0 otherwise, and μ_B is the dual value of Constraint (4.55).

The labeling algorithm begins with a label at the start node. This label represents a subpath consisting of only the start node. The label is then extended to each pickup node in turn, giving subpaths that consist of the start node and one pickup node. Provided that these subpaths are feasible with respect to the constraints mentioned above, their corresponding labels are extended to other nodes. This process is repeated until a number of subpaths reach the end node within some given time limit.

Solving the pricing subproblem using this labeling algorithm is equivalent to solving the mixed integer program in Figure 4.2 without Constraint (4.7) and with $\bar{d}_{i,j}$ in place of $d_{i,j}$ for an unspecified vehicle v.

4.5.3 The Separation Subproblem

This section describes the separation subproblem for both service and presence resource constraints. The subproblem is modeled using constraint programing and contains similar constraints to those in the constraint programming model.

Name	Description
$T \in \{1, \dots, \infty\}$	Time horizon.
$\mathcal{T} = \{0, \dots, T\}$	Set of time values.
$V \in \{1, \dots, \infty\}$	Number of vehicles.
$R \in \{1, \dots, \infty\}$	Total number of requests.
$\mathcal{R} = \{1, \dots, R\}$	Set of all requests.
$\mathcal{S} = \{R + 1, \dots, R + V\}$	Set of vehicle start nodes.
$\mathcal{E} = \{R + V + 1, \dots, R + 2V\}$	Set of vehicle end nodes.
$\mathcal{N} = \mathcal{R} \cup \mathcal{S} \cup \mathcal{E}$	Set of all nodes.
$L \in \{1, \dots, \infty\}$	Number of locations, excluding the depot location.
$\mathcal{L} = \{1, \dots, L\}$	Set of locations.
$C_l \in \{1, \dots, \infty\}$	Resource capacity of location $l \in \mathcal{L}$.
$\mathcal{R}_l = \{i \in \mathcal{R} \mid l(i) = l\}$	Requests at location $l \in \mathcal{L}$.
$l(i) \in \mathcal{L}$	Location of $i \in \mathcal{R}$.
$d(i, j) \in \mathcal{T}$	Distance and travel time from $i \in \mathcal{N}$ to $j \in \mathcal{N}$.
$a(i) \in \mathcal{T}$	Earliest start of service at $i \in \mathcal{N}$.
$b(i) \in \mathcal{T}$	Latest start of service at $i \in \mathcal{N}$.
$t(i) \in \mathcal{T}$	Service duration of $i \in \mathcal{N}$.
$\text{succ}(i) \in \mathcal{R} \cup \mathcal{E}$	Successor node of $i \in \mathcal{R} \cup \mathcal{S}$ in the routes of the master problem.
$\text{arr}(i) \in \mathcal{T}$	Arrival time at $i \in \mathcal{N}$.
$\text{serv}(i) \in \{a(i), \dots, b(i)\}$	Start of service at $i \in \mathcal{N}$.
$\text{dep}(i) \in \mathcal{T}$	Departure time at $i \in \mathcal{N}$.

Table 4.3: The data and decision variables in the separation subproblem.

Service Resources For the case of service resources, the constraints in the separation subproblem are extracted from the time and scheduling constraints of the constraint programming model of the VRPLC. Table 4.3 lists the data and decision variables. The number V of vehicles and the successor variables $\text{succ}(\cdot)$ are not decision variables; they are data variables fixed according to the routes selected by the master problem. Figure 4.6 lists the constraints. The decision variables are the arrival, service and departure times. Constraints (4.57) to (4.60) are the time constraints and serve as task precedence constraints. Constraint (4.61) schedules the resources.

Presence Resources Although it is possible to implement Constraint (4.52) in this subproblem, it is more effective to preprocess the routes selected by the master program and introduce the concept of a *location visit* to reduce the number of tasks in the subproblem. A location visit starts when a vehicle arrives at a location and lasts until the vehicle departs from the location. In other words, a location visit is a sequence of requests situated at one location. The start time of a visit is the arrival time at the first request in the visit, and the end time is the departure time at the last request in the visit. The requests in the visits are transferred to the subproblem as data, but the arrival and departure times of the visits are variables.

The visits require additional data and decision variables, which are listed in Table 4.4.

$$\text{arr}(i) \leq \text{serv}(i) \qquad \forall i \in \mathcal{R}, \quad (4.57)$$

$$\text{serv}(i) + t(i) \leq \text{dep}(i) \qquad \forall i \in \mathcal{R}, \quad (4.58)$$

$$\text{arr}(i) = \text{serv}(i) = \text{dep}(i) \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \quad (4.59)$$

$$\text{dep}(i) + d(i, \text{succ}(i)) = \text{arr}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (4.60)$$

$$\textsc{Cumulative}(\{\text{serv}(i) | i \in \mathcal{R}_l\}, \{t(i) | i \in \mathcal{R}_l\}, \mathbf{1}, C_l) \qquad \forall l \in \mathcal{L}. \quad (4.61)$$

Figure 4.6: The constraints of the separation subproblem for service resources.

Name	Description
$\mathcal{K}$	Set of visits.
$l(k) \in \mathcal{L}$	Location of visit $k \in \mathcal{K}$.
$\mathcal{K}_l = \{k \in \mathcal{K} \mid l(k) = l\}$	Set of visits to location $l \in \mathcal{L}$.
$\text{first}(k) \in \mathcal{R}$	First request of the visit $k \in \mathcal{K}$.
$\text{last}(k) \in \mathcal{R}$	Last request of the visit $k \in \mathcal{K}$.
$\text{start}(k) \in \mathcal{T}$	Start time of visit $k \in \mathcal{K}$.
$\text{end}(k) \in \mathcal{T}$	End time of visit $k \in \mathcal{K}$.
$\text{dur}(k) \in \mathcal{T}$	Duration of visit $k \in \mathcal{K}$.

Table 4.4: Additional data and decision variables for presence resources in the separation subproblem.

The subproblem is shown in Figure 4.7. Constraints (4.62) to (4.65) are the time constraints. Constraints (4.66) and (4.67) link the start and end times of a visit to the arrival and departure times at the first and last requests of the visit. Constraint (4.68) calculates the duration of a visit, and Constraint (4.69) is the cumulative resource constraint. Unlike for service resources, the durations of the visits are variables.

4.5.4 The BPC Search Algorithm

This section presents the BPC algorithm, which integrates the components presented earlier. The algorithm, described in Figure 4.8, begins by choosing an open node to solve (step 1). Nodes are chosen using two alternating selection rules. The first rule chooses the node with the largest number of flow variables fixed by branching, and the second rule selects the node with the smallest lower bound. By alternating between these two node selection rules, the BPC algorithm attempts to (1) find easy solutions by choosing nodes almost entirely fixed, giving the algorithm good upper bounds in the early stages of the search, and (2) find better solutions that improve substantially upon the current upper bound in the later stages.

In the chosen node, the BPC algorithm solves the master problem (step 2). If there is at least one fractional route and the percentage change in the objective value over the past several iterations does not exceed some value given as a parameter, the algorithm proceeds to branch

$$\text{arr}(i) \leq \text{serv}(i) \qquad \forall i \in \mathcal{R}, \quad (4.62)$$

$$\text{serv}(i) + t(i) \leq \text{dep}(i) \qquad \forall i \in \mathcal{R}, \quad (4.63)$$

$$\text{arr}(i) = \text{serv}(i) = \text{dep}(i) \qquad \forall i \in \mathcal{S} \cup \mathcal{E}, \quad (4.64)$$

$$\text{dep}(i) + d(i, \text{succ}(i)) = \text{arr}(\text{succ}(i)) \qquad \forall i \in \mathcal{R} \cup \mathcal{S}, \quad (4.65)$$

$$\text{start}(k) = \text{arr}(\text{first}(k)) \qquad \forall k \in \mathcal{K}, \quad (4.66)$$

$$\text{end}(k) = \text{dep}(\text{last}(k)) \qquad \forall k \in \mathcal{K}, \quad (4.67)$$

$$\text{dur}(k) = \text{end}(k) - \text{start}(k) \qquad \forall k \in \mathcal{K}, \quad (4.68)$$

$$\textsc{Cumulative}(\{\text{start}(k) | k \in \mathcal{K}_l\}, \{\text{dur}(k) | k \in \mathcal{K}_l\}, \mathbf{1}, C_l) \qquad \forall l \in \mathcal{L}. \quad (4.69)$$

Figure 4.7: Additional constraints for presence resources in the separation subproblem.

early (step 3), i.e., before the restricted master problem is solved to optimality, or equivalently, prior to the pricing subproblem reporting that no routes with negative reduced cost exist (e.g., Lübbecke and Desrosiers 2005). Early branching can be beneficial because an optimal fractional master problem solution is rarely useful and because fixing some arcs may fix other arcs by reasoning about the load, time and pickup-delivery constraints.

If early branching is not completed, the algorithm then counts the number of integer routes, i.e., routes r with $x_r = 1$. If there are $c > \min_{l \in \mathcal{L}} C_l$ integer routes, the BPC algorithm solves the separation subproblem on these routes (step 4). It is unnecessary to solve the separation subproblem if there are fewer than c integer x_r variables as the location constraints are automatically satisfied. Note that the solution to the master problem may consist of fractional values for some routes, but only the integer routes are transferred to the separation subproblem.

If no feasible schedule exists for these routes, a nogood cut is added to the master problem, which is reoptimized. Otherwise, if all routes are integral, then the routes and the schedules form a solution (step 5). Since the algorithm may iterate through this step multiple times in a node, it is possible, though unlikely, that multiple solutions are found within one node.

The algorithm then proceeds to the pricing subproblem to generate new routes (step 6). If new routes are found, they are added to the master problem, which is reoptimized. The node is completed when no new routes are found.

Branching occurs if any route in the master problem is fractional and the node is not suboptimal, i.e., the lower bound of the node is lower than the incumbent solution (step 7). The BPC algorithm selects a fractional route $r' = (i_1, i_2, \ldots, i_{n-1}, i_n)$ of length n and creates n children, in which certain arcs must or must not be used. The jth child, $j = 1, \ldots, n$, corresponding to the prefix $(i_1, i_2, \ldots, i_j)$, has the edges $(i_1, i_2), \ldots, (i_{j-1}, i_j)$ present and the edge (i_j, i_{j+1}) absent. This branching scheme is easily implemented by removing all incompatible edges in the graph within the pricing subproblem and all incompatible routes in the master problem. Similar branching rules have previously appeared in vehicle routing problems (e.g., Desrosiers, Soumis

1. **Node Selection:** Select an open node, otherwise, terminate if no open nodes remain.
2. **Master:** Solve the master problem in Figure 4.5.
3. **Early Branching:** If there is at least one route r with $0 < x_r < 1$, and the percentage change in the objective value has plateaued over the last several iterations, go to step 7.
4. **Separation:** If $\sum_{r \in \Omega}(x_r = 1) > \min_{l \in \mathcal{L}} C_l$, solve the separation subproblem described in Figure 4.6 (or Figure 4.7). If the separation subproblem is infeasible, generate a nogood cut, and go back to step 2.
5. **Feasible Solution:** If all x_r variables are integral, a feasible solution is found.
6. **Pricing:** Solve the pricing subproblem described in Section 4.5.2. If at least one new route is generated, go back to step 2.
7. **Branching:** If the lower bound given by the master problem is less than the upper bound, and there is at least one route r with $0 < x_r < 1$, select one such route $r' = (i_1, i_2, \ldots, i_{n-1}, i_n)$. Create n open nodes, one for each (possibly empty or full) prefix of r. In the open node corresponding to prefix $(i_1, i_2, \ldots, i_j)$, require all edges in the prefix and exclude the edge (i_j, i_{j+1}). Go to step 1.

Figure 4.8: The branch-and-price-and-check algorithm.

and Desrochers 1984, Dumas, Desrosiers and Soumis 1991) and traveling salesmen problems (e.g., Bellmore and Malone 1971, Carpaneto and Toth 1980). The children nodes are added to the set of open nodes.

4.6 Experimental Results

This section reports experimental results for the three approaches.

The Instances The algorithms are tested on three sets of randomly generated instances. The time horizon is set at 100. Eight and eleven locations are generated on a Euclidean grid up to 40 units wide. Next, pickup-delivery pairs are generated and randomly distributed among the locations. Service durations vary between 5 and 20, and load demands vary between 1 and 5. The time windows of requests are randomly chosen. The requests are repeatedly generated until 150 pickup-delivery pairs are feasible with respect to the time windows. Smaller instances are created by randomly discarding some of the requests. Every instance is then duplicated with a resource capacity of one to fifteen for every location. In total, there are 1,620 instances.

The Implementations The mixed integer programming model is solved using Gurobi. The constraint programming model is solved using Chuffed. The master problem of the BPC model is solved using Gurobi. The pricing subproblem is solved using a custom implementation of a standard labeling algorithm. The separation subproblem is solved using Chuffed. The three integrated models are also compared against a two-stage model to evaluate the feasibility of sequential routing and scheduling. The sequential model is based on the BPC implementation. It uses branch-and-price to find optimal vehicle routes that ignore the scheduling constraints in

the first stage, and then schedules the vehicles on these routes in the second stage using the separation subproblem. The two-stage model is only a heuristic since it does not explore the entire search space. All four models are run for twelve hours on an Intel Xeon E5-2660 V3 CPU at 2.6 GHz.

Summary of the Results All solutions for service resources are reported in Table 4.5 at the end of this chapter on pages 126 to 130. The main findings are summarized as follows.

1. All three integrated models find optimal solutions to the instances with 10 pickup-delivery requests. The CP and BPC models prove optimality to all of these instances while the MIP formulation proves optimality to two of the three sets of instances. The two-stage model finds all of the same optimal solution except the three instances with eight locations and one resource at the locations because the optimal vehicle routes are infeasible with respect to the scheduling constraints.
2. The BPC model continues to find optimal solutions to the instances with 15 pickup-delivery requests, but the CP and MIP models begin to struggle. CP finds feasible solutions that can be much worse than BPC, and MIP fails to even find feasible solution to many of these instances. Nevertheless, CP proves optimality identically to BPC for the second set of instances with eight locations, and MIP performs similarly for the same instance set with eleven locations. TS fails to find feasible solutions to seven instances with one to two resources because the optimal routes are infeasible with respect to the scheduling constraints. Contrastingly, BPC proves optimality or infeasibility to these seven instances. In particular, the routes for the instance with eight locations, fifteen request pairs and one resource in the third set of instances is different to the routes of the instance with two resources, despite having the same optimal value.
3. The BPC method finds optimal solutions to all but one of the feasible instances with 20 request pairs. On this instance, it closes the optimality gap to 0.6%. The CP model performs significantly worse than BPC. However, it proves infeasibility to three of these instances, which is one more than BPC. The MIP approach is also worse than BPC; it finds very few feasible solutions. For the instances with 11 locations, 20 request pairs and 3, 5, 14 and 15 resources, MIP finds solutions known to be optimal but fails to close the optimality gap. Since relaxing the scheduling constraints cannot invalidate a solution, the same solution must be feasible for resource capacities greater than three. However, the solver fails to explicitly find these solutions within the given time limit due to the choices made by branching and/or the different dual solutions to the master problem.
4. Overall, there are 50 instances for which TS fails but BPC finds feasible solutions. Naturally, these instances have tight scheduling constraints, which results in solutions with greater optimal values or the same optimal value but with different routes.
5. BPC dominates TS up to 60 request pairs. Whenever TS is feasible, BPC finds the same or better solutions, and proves their optimality or otherwise closes the optimality gap to less than 0.8%.
6. TS finds better solutions than BPC to nine of the larger instances with 80 or more pickup-

delivery pairs. This outcome occurs at the interface where BPC first finds feasible solutions as the scheduling constraints relax, and it arises from the choices made by branching. In BPC, the cuts induce different, and usually more, fractionalities, which lead the solver to explore different areas of the search space by branching. As a result, BPC finds different solutions to TS. Of course, some of these solutions are better and some are worse. This effect will disappear if the solver is allowed to finish exploring the search tree beyond the time limit. Nonetheless, BPC is only 0.5% worse on these nine instances.

7. BPC finds better solutions than TS on 18 instances. This, again, is due to the interaction between the cuts and the branches chosen.
8. The CP model proves infeasibility to 69 of the instances, including some of the largest with 150 pickup-delivery pairs. Having few resources and fixed task durations greatly improves the propagation of the scheduling constraints, allowing the learning CP solver to make strong deductions about the feasibility of the resource schedules.
9. The BPC model is able to prove infeasibility only to three of the smallest instances with 15 and 20 pairs of requests. The master problem in the BPC model has no knowledge about the scheduling constraints, and therefore, must exclude an exponential number of combinations of arcs before proving infeasibility.
10. The scheduling constraints in the MIP model are even weaker, leading it to not proving infeasibility on any instance.
11. The objective values and the number of nogoods from the BPC solutions generally decrease as the resource constraints are relaxed, allowing for some minor variation due to the choices made by branching and the fixed time limit. This result indicates that the BPC algorithm deals with looser capacity constraints effectively. It is able to schedule the vehicles around the resources and maintain the same or similar routing.
12. The BPC method dominates the CP and MIP models on all instances except the 66 instances for which CP proves infeasibility but BPC does not.
13. BPC finds feasible solutions to 695 of the 810 instances. On these instances, its average optimality gap is 1.1%, and over 90% of these instances have 5.0% or smaller gap.
14. There are 46 instances for which no model proves feasibility or infeasibility.

The solutions for presence resources are shown in Table 4.6 on pages 135 to 139. Many of the findings discussed in the previous analysis for service resources remain valid for presence resources and are not repeated here. Several remarks are discussed below:

1. The performance of the four models are very similar to service resources. The models cannot scale beyond similar instance sizes.
2. The trend of improving optimal values and fewer nogoods holds for presence resources as for service resources.
3. The BPC method again dominates both the MIP and CP approaches besides the proofs of infeasibility by the CP model.
4. BPC finds feasible solutions to 680 of the 810 instances. On these instances, its average optimality gap is 1.1%, and over 91% of these instances have 5.0% or smaller gap.

5. There are 51 instances for which BPC is feasible but TS is infeasible. There are 8 instances for which TS is better than BPC. On these instances, BPC only performs 0.7% worse on average. There are 21 instances for which TS is worse than BPC.

An obvious drawback of the BPC method compared to the CP model is that without the scheduling constraints, or some relaxed form of the scheduling constraints, in the master problem, it is unable to prove infeasibility prior to discovering an exponential number of cuts forbidding all possible combinations of arcs.

The BPC model compares favorably against the two-stage approach. It dominates the sequential method for almost all instances by finding the same or better solutions and by finding solutions to the tighter instances for which the optimal vehicle routes found by the two-stage method are infeasible with respect to the scheduling constraints. For the easier instances, BPC generates no nogoods; therefore, it finds the same feasible solutions as TS, but unlike TS, BPC guarantees their optimality. For a few of the largest instances, BPC performs worse than TS due to the branching decisions. Given that TS is simply a heuristic, it is possible to run TS and initialize the root node of BPC with the routes found by TS; thereby erasing this effect.

Overall, the results demonstrate that the BPC model outperforms both the mixed integer programming and constraint programming models on almost all instances by finding better solutions. The BPC approach nicely capitalizes on the orthogonal strengths of mathematical programming and constraint programming by using column generation to produce routes and dual bounds, and constraint programming to check the routes for feasibility of the resource constraints.

4.7 Conclusion

This chapter presents the Vehicle Routing Problem with Location Congestion, which overlays the Resource-Constrained Project Scheduling Problem on a traditional vehicle routing problem. Two types of location resources are considered: service resources are used while requests are in service, and presence resources are used whenever a vehicle is present at a location. The problem is formulated as a mixed integer programming model, a constraint programming model, a branch-and-price-and-check model and a sequential model. In the branch-and-price-and-check model, the pricing subproblem generates routes for the master problem, and the separation subproblem verifies the feasibility of the routes selected by the master problem against the location resource constraints. In the sequential model, optimal vehicle routes ignoring the scheduling constraints are first found and then fixed in the following scheduling stage.

Empirical results indicate that the branch-and-price-and-check approach finds optimal solutions to instances with up to 160 requests (80 pickup-delivery pairs) and feasible solutions with small optimality gaps to instances with up to 300 requests (150 pickup-delivery pairs). In general, the branch-and-price-and-check method outperforms both the mixed integer programming and constraint programming formulations, which are unable to consistently scale to instances larger than ten pickup-delivery requests. The branch-and-price-and-check method also dominates the sequential approach on almost all instances by finding the same or better solutions and

by finding feasible solutions to instances for which the two-stage method fails. The two-stage method performs better than branch-and-price-and-check on a few of the large instances due to the incomplete search conducted over the given time limit. Despite these minor shortcomings, the branch-and-price-and-check method is the best performer overall.

A drawback of the branch-and-price-and-check model is that it does not explicitly contain the scheduling constraints, and hence, must reason about the resources via the prohibition of certain arcs. In contrast, the constraint programming model includes the scheduling constraints, and therefore, can prove infeasibility rather quickly. The next chapter develops a technique that remediates this limitation by further integrating the unique strengths of mathematical programming and constraint programming.

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	10	1	–	**146**	146	0.0%	**146**	**146**	146	0.0%	27	–	**249**	249	0.0%	**249**	**249**	249	0.0%	72	–	84	44	47.6%	**84**	**84**	84	0.0%	11
		2	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		3	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		4	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	61	22.8%	**79**	**79**	79	0.0%	0
		5	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	55	30.4%	**79**	**79**	79	0.0%	0
		6	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	57	27.8%	**79**	**79**	79	0.0%	0
		7	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	57	27.8%	**79**	**79**	79	0.0%	0
		8	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	54	31.6%	**79**	**79**	79	0.0%	0
		9	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		10	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		11	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		12	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		13	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		14	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		15	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	58	26.6%	**79**	**79**	79	0.0%	0
11	10	1	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	62	26.2%	**84**	**84**	84	0.0%	0
		2	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		3	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	67	20.2%	**84**	**84**	84	0.0%	0
		4	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		5	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	62	26.2%	**84**	**84**	84	0.0%	0
		6	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	62	26.2%	**84**	**84**	84	0.0%	0
		7	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		8	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0
		9	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		10	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0
		11	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		12	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		13	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0
		14	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		15	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	15	1	–	–	45	–	204	**196**	196	0.0%	705	–	–	170	–	**359**	**359**	359	0.0%	634	–	–	24	–	150	**118**	118	0.0%	75
		2	–	–	47	–	212	**180**	180	0.0%	6	277	277	129	53.4%	**277**	**277**	277	0.0%	0	118	–	19	–	156	**118**	118	0.0%	0
		3	179	–	44	–	208	**179**	179	0.0%	0	277	277	159	42.6%	**277**	**277**	277	0.0%	0	118	–	21	–	143	**118**	118	0.0%	0
		4	179	208	46	77.9%	181	**179**	179	0.0%	0	277	277	204	26.4%	**277**	**277**	277	0.0%	0	118	–	21	–	175	**118**	118	0.0%	0
		5	179	–	47	–	181	**179**	179	0.0%	0	277	277	137	50.5%	**277**	**277**	277	0.0%	0	118	–	19	–	175	**118**	118	0.0%	0
		6	179	–	43	–	181	**179**	179	0.0%	0	277	277	210	24.2%	**277**	**277**	277	0.0%	0	118	133	23	82.7%	175	**118**	118	0.0%	0
		7	179	–	48	–	181	**179**	179	0.0%	0	277	277	230	17.0%	**277**	**277**	277	0.0%	0	118	–	20	–	175	**118**	118	0.0%	0
		8	179	191	45	76.4%	181	**179**	179	0.0%	0	277	277	206	25.6%	**277**	**277**	277	0.0%	0	118	–	24	–	175	**118**	118	0.0%	0
		9	179	207	44	78.7%	181	**179**	179	0.0%	0	277	277	194	30.0%	**277**	**277**	277	0.0%	0	118	–	18	–	175	**118**	118	0.0%	0
		10	179	–	47	–	181	**179**	179	0.0%	0	277	277	160	42.2%	**277**	**277**	277	0.0%	0	118	–	20	–	175	**118**	118	0.0%	0
		11	179	186	46	75.3%	181	**179**	179	0.0%	0	277	277	160	42.2%	**277**	**277**	277	0.0%	0	118	–	22	–	175	**118**	118	0.0%	0
		12	179	–	47	–	181	**179**	179	0.0%	0	277	277	159	42.6%	**277**	**277**	277	0.0%	0	118	–	16	–	175	**118**	118	0.0%	0
		13	179	–	47	–	181	**179**	179	0.0%	0	277	277	162	41.5%	**277**	**277**	277	0.0%	0	118	–	20	–	175	**118**	118	0.0%	0
		14	179	191	44	77.0%	181	**179**	179	0.0%	0	277	277	159	42.6%	**277**	**277**	277	0.0%	0	118	–	20	–	175	**118**	118	0.0%	0
		15	179	199	44	77.9%	181	**179**	179	0.0%	0	277	277	191	31.0%	**277**	**277**	277	0.0%	0	118	–	20	–	175	**118**	118	0.0%	0
11	15	1	–	245	100	59.2%	234	**223**	223	0.0%	20	–	–	374	–	×	×	–	–	74	–	124	42	66.1%	149	**114**	114	0.0%	290
		2	219	227	77	66.1%	243	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	45	–	161	**107**	107	0.0%	0
		3	219	230	90	60.9%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	45	–	173	**107**	107	0.0%	0
		4	219	227	93	59.0%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	44	–	173	**107**	107	0.0%	0
		5	219	222	88	60.4%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	44	–	173	**107**	107	0.0%	0
		6	219	235	96	59.1%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	41	–	173	**107**	107	0.0%	0
		7	219	244	89	63.5%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	47	–	173	**107**	107	0.0%	0
		8	219	258	92	64.3%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	44	–	173	**107**	107	0.0%	0
		9	219	228	93	59.2%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	120	42	65.0%	173	**107**	107	0.0%	0
		10	219	228	93	59.2%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	43	–	173	**107**	107	0.0%	0
		11	219	228	93	59.2%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	44	–	173	**107**	107	0.0%	0
		12	219	228	93	59.2%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	43	–	173	**107**	107	0.0%	0
		13	219	228	92	59.6%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	143	42	70.6%	173	**107**	107	0.0%	0
		14	219	228	92	59.6%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	143	42	70.6%	173	**107**	107	0.0%	0
		15	219	219	102	53.4%	272	**219**	219	0.0%	0	302	**302**	302	0.0%	415	**302**	302	0.0%	0	107	–	43	–	173	**107**	107	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	20	1	–	–	50	–	×	–	282	–	12,145	–	–	107	–	×	×	–	–	6,377	–	–	16	–	213	159	158	0.6%	6,669
		2	–	–	52	–	514	**247**	247	0.0%	115	–	–	106	–	580	**375**	375	0.0%	1,087	148	–	17	–	282	**148**	148	0.0%	0
		3	246	–	52	–	385	**246**	246	0.0%	0	369	–	110	–	580	**369**	369	0.0%	0	148	–	15	–	295	**148**	148	0.0%	0
		4	246	–	51	–	410	**246**	246	0.0%	0	369	–	100	–	580	**369**	369	0.0%	0	148	–	18	–	268	**148**	148	0.0%	0
		5	246	–	52	–	410	**246**	246	0.0%	0	369	–	116	–	457	**369**	369	0.0%	0	148	–	16	–	248	**148**	148	0.0%	0
		6	246	–	53	–	410	**246**	246	0.0%	0	369	–	125	–	457	**369**	369	0.0%	0	148	–	19	–	248	**148**	148	0.0%	0
		7	246	–	54	–	410	**246**	246	0.0%	0	369	–	132	–	457	**369**	369	0.0%	0	148	–	16	–	248	**148**	148	0.0%	0
		8	246	–	53	–	410	**246**	246	0.0%	0	369	–	103	–	457	**369**	369	0.0%	0	148	–	18	–	248	**148**	148	0.0%	0
		9	246	–	53	–	410	**246**	246	0.0%	0	369	–	113	–	457	**369**	369	0.0%	0	148	–	14	–	248	**148**	148	0.0%	0
		10	246	–	50	–	410	**246**	246	0.0%	0	369	–	107	–	457	**369**	369	0.0%	0	148	–	17	–	248	**148**	148	0.0%	0
		11	246	–	52	–	410	**246**	246	0.0%	0	369	–	102	–	457	**369**	369	0.0%	0	148	–	14	–	248	**148**	148	0.0%	0
		12	246	–	52	–	410	**246**	246	0.0%	0	369	–	101	–	457	**369**	369	0.0%	0	148	–	18	–	248	**148**	148	0.0%	0
		13	246	–	49	–	410	**246**	246	0.0%	0	369	–	100	–	459	**369**	369	0.0%	0	148	–	18	–	248	**148**	148	0.0%	0
		14	246	–	52	–	410	**246**	246	0.0%	0	369	–	103	–	459	**369**	369	0.0%	0	148	–	17	–	248	**148**	148	0.0%	0
		15	246	–	52	–	410	**246**	246	0.0%	0	369	–	103	–	459	**369**	369	0.0%	0	148	–	19	–	248	**148**	148	0.0%	0
11	20	1	–	–	83	–	396	**312**	312	0.0%	8,390	–	–	164	–	×	×	–	–	420	–	–	27	–	281	**147**	147	0.0%	771
		2	283	–	78	–	422	**283**	283	0.0%	0	–	–	164	–	478	**361**	361	0.0%	26	143	–	27	–	287	**143**	143	0.0%	0
		3	283	–	78	–	372	**283**	283	0.0%	0	359	359	183	49.0%	546	**359**	359	0.0%	0	143	–	27	–	265	**143**	143	0.0%	0
		4	283	–	80	–	326	**283**	283	0.0%	0	359	376	207	44.9%	574	**359**	359	0.0%	0	143	–	26	–	244	**143**	143	0.0%	0
		5	283	–	78	–	326	**283**	283	0.0%	0	359	359	202	43.7%	529	**359**	359	0.0%	0	143	–	28	–	244	**143**	143	0.0%	0
		6	283	–	76	–	326	**283**	283	0.0%	0	359	393	174	55.7%	529	**359**	359	0.0%	0	143	–	27	–	244	**143**	143	0.0%	0
		7	283	–	75	–	326	**283**	283	0.0%	0	359	–	130	–	529	**359**	359	0.0%	0	143	–	26	–	244	**143**	143	0.0%	0
		8	283	–	77	–	326	**283**	283	0.0%	0	359	–	170	–	529	**359**	359	0.0%	0	143	–	32	–	244	**143**	143	0.0%	0
		9	283	–	81	–	326	**283**	283	0.0%	0	359	–	154	–	529	**359**	359	0.0%	0	143	–	28	–	244	**143**	143	0.0%	0
		10	283	–	78	–	326	**283**	283	0.0%	0	359	–	168	–	529	**359**	359	0.0%	0	143	–	26	–	244	**143**	143	0.0%	0
		11	283	–	78	–	326	**283**	283	0.0%	0	359	361	165	54.3%	529	**359**	359	0.0%	0	143	–	27	–	244	**143**	143	0.0%	0
		12	283	–	78	–	326	**283**	283	0.0%	0	359	–	141	–	529	**359**	359	0.0%	0	143	–	29	–	244	**143**	143	0.0%	0
		13	283	–	78	–	326	**283**	283	0.0%	0	359	–	135	–	529	**359**	359	0.0%	0	143	–	28	–	244	**143**	143	0.0%	0
		14	283	–	78	–	326	**283**	283	0.0%	0	359	359	168	53.2%	529	**359**	359	0.0%	0	143	–	26	–	244	**143**	143	0.0%	0
		15	283	–	74	–	326	**283**	283	0.0%	0	359	359	168	53.2%	529	**359**	359	0.0%	0	143	–	27	–	244	**143**	143	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality.
(Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	30	1	–	–	30	–	×	–	354	–	10,783	–	–	83	–	×	–	514	–	35,636	–	–	10	–	×	–	210	–	3,324
		2	–	–	30	–	–	356	352	1.1%	8,985	–	–	79	–	–	594	514	13.5%	30,328	–	–	8	–	–	**206**	206	0.0%	1,468
		3	351	–	30	–	–	**351**	351	0.0%	0	517	–	80	–	–	**517**	517	0.0%	0	204	–	8	–	460	**204**	204	0.0%	0
		4	351	–	30	–	–	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		5	351	–	29	–	–	**351**	351	0.0%	0	517	–	79	–	–	**517**	517	0.0%	0	204	–	7	–	–	**204**	204	0.0%	0
		6	351	–	30	–	–	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		7	351	–	30	–	–	**351**	351	0.0%	0	517	–	80	–	–	**517**	517	0.0%	0	204	–	8	–	–	**204**	204	0.0%	0
		8	351	–	30	–	–	**351**	351	0.0%	0	517	–	80	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		9	351	–	30	–	–	**351**	351	0.0%	0	517	–	79	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		10	351	–	30	–	–	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		11	351	–	30	–	–	**351**	351	0.0%	0	517	–	79	–	–	**517**	517	0.0%	0	204	–	7	–	–	**204**	204	0.0%	0
		12	351	–	31	–	–	**351**	351	0.0%	0	517	–	80	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		13	351	–	30	–	–	**351**	351	0.0%	0	517	–	84	–	–	**517**	517	0.0%	0	204	–	8	–	–	**204**	204	0.0%	0
		14	351	–	30	–	–	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		15	351	–	30	–	–	**351**	351	0.0%	0	517	–	77	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
11	30	1	–	–	63	–	×	–	383	–	17,216	–	–	132	–	×	–	529	–	29,892	–	–	20	–	–	–	184	–	4,995
		2	387	–	64	–	–	**387**	387	0.0%	124	–	–	129	–	869	**544**	544	0.0%	14,011	184	–	11	–	–	**184**	184	0.0%	0
		3	387	–	62	–	–	**387**	387	0.0%	0	–	–	132	–	–	**538**	538	0.0%	344	184	–	20	–	433	**184**	184	0.0%	0
		4	387	–	63	–	–	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	21	–	426	**184**	184	0.0%	0
		5	387	–	63	–	–	**387**	387	0.0%	0	536	–	128	–	816	**536**	536	0.0%	0	184	–	10	–	–	**184**	184	0.0%	0
		6	387	–	63	–	–	**387**	387	0.0%	0	536	–	129	–	–	**536**	536	0.0%	0	184	–	8	–	–	**184**	184	0.0%	0
		7	387	–	62	–	–	**387**	387	0.0%	0	536	–	130	–	–	**536**	536	0.0%	0	184	–	22	–	–	**184**	184	0.0%	0
		8	387	–	62	–	–	**387**	387	0.0%	0	536	–	130	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		9	387	–	63	–	–	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		10	387	–	62	–	–	**387**	387	0.0%	0	536	–	127	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		11	387	–	64	–	–	**387**	387	0.0%	0	536	–	130	–	–	**536**	536	0.0%	0	184	–	11	–	–	**184**	184	0.0%	0
		12	387	–	64	–	–	**387**	387	0.0%	0	536	–	30	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		13	387	–	64	–	–	**387**	387	0.0%	0	536	–	35	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		14	387	–	64	–	–	**387**	387	0.0%	0	536	–	130	–	–	**536**	536	0.0%	0	184	–	6	–	–	**184**	184	0.0%	0
		15	387	–	63	–	–	**387**	387	0.0%	0	536	–	130	–	–	**536**	536	0.0%	0	184	–	10	–	–	**184**	184	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	40	1	–	–	16	–	×	–	426	–	17,780	–	–	0	–	×	–	625	–	38,889	–	–	0	–	×	–	237	–	4,164
		2	–	–	16	–	–	503	428	14.9%	12,608	–	–	8	–	×	–	625	–	37,803	–	–	0	–	–	–	235	–	6,928
		3	431	–	0	–	–	**431**	431	0.0%	6	634	–	0	–	–	**634**	634	0.0%	308	–	–	0	–	–	252	235	6.7%	2,254
		4	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		5	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		6	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		7	431	–	5	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		8	431	–	16	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		9	431	–	6	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		10	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		11	431	–	14	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	3	–	–	**239**	239	0.0%	0
		12	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		13	431	–	10	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		14	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		15	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
11	40	1	–	–	32	–	×	–	505	–	2,773	–	–	66	–	×	–	682	–	33,456	–	–	0	–	×	–	236	–	11,685
		2	–	–	14	–	–	551	496	10.0%	3,364	–	–	54	–	×	–	682	–	33,474	–	–	0	–	–	**238**	238	0.0%	163
		3	499	–	32	–	–	**499**	499	0.0%	0	–	–	56	–	–	755	682	9.7%	22,868	238	–	0	–	–	**238**	238	0.0%	1
		4	499	–	38	–	–	**499**	499	0.0%	0	–	–	51	–	–	716	688	3.9%	19,032	238	–	0	–	–	**238**	238	0.0%	0
		5	499	–	17	–	–	**499**	499	0.0%	0	696	–	51	–	–	**696**	696	0.0%	30	238	–	3	–	–	**238**	238	0.0%	0
		6	499	–	14	–	–	**499**	499	0.0%	0	696	–	54	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		7	499	–	5	–	–	**499**	499	0.0%	0	696	–	53	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		8	499	–	14	–	–	**499**	499	0.0%	0	696	–	54	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		9	499	–	14	–	–	**499**	499	0.0%	0	696	–	51	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		10	499	–	14	–	–	**499**	499	0.0%	0	696	–	55	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		11	499	–	35	–	–	**499**	499	0.0%	0	696	–	51	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		12	499	–	32	–	–	**499**	499	0.0%	0	696	–	54	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		13	499	–	14	–	–	**499**	499	0.0%	0	696	–	53	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		14	499	–	20	–	–	**499**	499	0.0%	0	696	–	56	–	–	**696**	696	0.0%	0	238	–	4	–	–	**238**	238	0.0%	0
		15	499	–	32	–	–	**499**	499	0.0%	0	696	–	54	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	60	1	–	–	0	–	×	–	606	–	15,782	–	–	0	–	×	–	903	–	31,754	–	–	0	–	×	–	326	–	11,343
		2	–	–	0	–	×	–	603	–	12,807	–	–	0	–	×	–	903	–	31,669	–	–	0	–	–	–	325	–	7
		3	–	–	0	–	–	690	606	12.2%	12,962	–	–	0	–	×	–	903	–	31,754	–	–	0	–	–	403	326	19.1%	9,284
		4	–	–	0	–	–	635	611	3.8%	1,883	–	–	0	–	–	992	913	8.0%	22,466	–	–	0	–	–	352	327	7.1%	746
		5	619	–	0	–	–	619	614	0.8%	232	922	–	0	–	–	922	918	0.4%	1,604	332	–	0	–	–	332	330	0.6%	9
		6	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	394	332	–	0	–	–	332	330	0.6%	0
		7	619	–	0	–	–	619	616	0.5%	0	922	–	0	–	–	**922**	922	0.0%	1	332	–	0	–	–	332	330	0.6%	0
		8	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		9	619	–	0	–	–	619	616	0.5%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		10	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		11	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		12	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		13	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		14	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		15	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
11	60	1	–	–	4	–	×	–	699	–	13,973	–	–	0	–	×	–	985	–	27,952	–	–	0	–	×	–	322	–	14,545
		2	–	–	4	–	–	–	699	–	17,424	–	–	0	–	×	–	985	–	28,667	–	–	0	–	–	–	322	–	2,378
		3	–	–	4	–	–	746	704	5.6%	2,056	–	–	0	–	–	–	985	–	26,176	324	–	0	–	–	**324**	324	0.0%	0
		4	706	–	4	–	–	**706**	706	0.0%	0	–	–	0	–	–	1,049	988	5.8%	18,506	324	–	0	–	–	**324**	324	0.0%	0
		5	706	–	4	–	–	**706**	706	0.0%	0	–	–	0	–	–	992	989	0.3%	7,556	324	–	0	–	–	**324**	324	0.0%	0
		6	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		7	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		8	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		9	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		10	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		11	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		12	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		13	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		14	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		15	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	80	1	–	–	0	–	×	–	746	–	3,171	–	–	0	–	×	–	1,176	–	28,124	–	–	0	–	×	–	390	–	2,865
		2	–	–	0	–	×	–	746	–	3,169	–	–	0	–	×	–	1,176	–	27,082	–	–	0	–	–	–	390	–	17
		3	–	–	0	–	–	–	746	–	13,897	–	–	0	–	×	–	1,176	–	28,660	–	–	0	–	–	–	391	–	2,730
		4	–	–	0	–	–	845	755	10.7%	2,971	–	–	0	–	–	–	1,176	–	28,040	393	–	0	–	–	**393**	393	0.0%	111
		5	794	–	0	–	–	799	755	5.5%	1,903	–	–	0	–	–	1,370	1,182	13.7%	20,805	393	–	0	–	–	**393**	393	0.0%	0
		6	794	–	0	–	–	795	761	4.3%	149	1,195	–	0	–	–	1,195	1,182	1.1%	2,565	393	–	0	–	–	**393**	393	0.0%	0
		7	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,194	1,184	0.8%	1,729	393	–	0	–	–	**393**	393	0.0%	0
		8	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,184	0.9%	2,242	393	–	0	–	–	**393**	393	0.0%	0
		9	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		10	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		11	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		12	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		13	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		14	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		15	794	–	0	–	–	793	763	3.8%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
11	80	1	–	–	4	–	×	–	876	–	8,763	–	–	0	–	×	–	1,260	–	24,022	–	–	0	–	×	–	411	–	10,388
		2	–	–	4	–	×	–	872	–	12,877	–	–	0	–	×	–	1,260	–	25,102	–	–	0	–	–	–	407	–	12
		3	–	–	4	–	–	–	875	–	9,282	–	–	0	–	×	–	1,260	–	24,733	–	–	0	–	–	–	410	–	780
		4	–	–	4	–	–	–	881	–	914	–	–	0	–	–	–	1,260	–	25,974	413	–	0	–	–	413	411	0.5%	184
		5	883	–	4	–	–	**883**	883	0.0%	32	–	–	0	–	–	1,340	1,260	6.0%	11,902	413	–	0	–	–	413	412	0.2%	0
		6	883	–	4	–	–	**883**	883	0.0%	0	–	–	0	–	–	1,319	1,266	4.0%	5,011	413	–	0	–	–	413	412	0.2%	0
		7	883	–	4	–	–	**883**	883	0.0%	0	–	–	0	–	–	1,312	1,269	3.3%	2,083	413	–	0	–	–	413	412	0.2%	0
		8	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,270	0.8%	2,165	413	–	0	–	–	413	412	0.2%	0
		9	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		10	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		11	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		12	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		13	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		14	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	411	0.5%	0
		15	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	100	1	–	–	0	–	×	–	899	–	2,645	–	–	0	–	×	–	1,454	–	26,079	–	–	0	–	×	–	459	–	22,399
		2	–	–	0	–	×	–	902	–	18,689	–	–	0	–	×	–	1,454	–	26,217	–	–	0	–	–	–	459	–	20
		3	–	–	0	–	–	–	899	–	0	–	–	0	–	×	–	1,454	–	26,633	–	–	0	–	–	–	459	–	0
		4	–	–	0	–	–	–	904	–	1,789	–	–	0	–	×	–	1,454	–	26,568	492	–	0	–	–	493	462	6.3%	512
		5	–	–	0	–	–	999	906	9.3%	950	–	–	0	–	–	–	1,454	–	11	492	–	0	–	–	492	464	5.7%	22
		6	–	–	0	–	–	982	909	7.4%	111	–	–	0	–	–	1,603	1,462	8.8%	2,133	492	–	0	–	–	486	464	4.5%	0
		7	970	–	0	–	–	970	910	6.2%	2	–	–	0	–	–	1,513	1,463	3.3%	2,927	492	–	0	–	–	492	464	5.7%	0
		8	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,513	1,464	3.2%	874	492	–	0	–	–	486	464	4.5%	0
		9	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		10	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		11	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		12	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	492	464	5.7%	0
		13	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		14	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		15	970	–	0	–	–	964	910	5.6%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	492	464	5.7%	0
11	100	1	–	–	4	–	×	–	1,038	–	2,461	–	–	0	–	×	–	1,535	–	24,071	–	–	0	–	×	–	475	–	19,758
		2	–	–	4	–	×	–	1,038	–	2,454	–	–	0	–	×	–	1,535	–	22,294	–	–	0	–	–	–	475	–	3
		3	–	–	4	–	–	–	1,042	–	8,694	–	–	0	–	×	–	1,535	–	23,391	–	–	0	–	–	–	476	–	1,237
		4	–	–	4	–	–	1,136	1,047	7.8%	747	–	–	0	–	–	–	1,535	–	22,478	–	–	0	–	–	–	476	–	1,240
		5	1,060	–	4	–	–	1,068	1,048	1.9%	1,239	–	–	0	–	–	–	1,536	–	25,235	480	–	0	–	–	479	477	0.4%	20
		6	1,060	–	4	–	–	1,060	1,051	0.8%	1	–	–	0	–	–	1,627	1,539	5.4%	2,141	480	–	0	–	–	481	477	0.8%	0
		7	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,588	1,547	2.6%	2,364	480	–	0	–	–	480	477	0.6%	0
		8	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	3	480	–	0	–	–	480	477	0.6%	0
		9	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		10	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		11	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		12	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		13	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		14	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		15	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	150	1	–	–	–	–	×	–	1,280	–	14,611	–	–	–	–	×	–	2,153	–	18,786	–	–	–	–	×	–	0	–	3,281
		2	–	–	–	–	×	–	1,280	–	14,626	–	–	–	–	×	–	2,153	–	18,777	–	–	–	–	×	–	0	–	12
		3	–	–	–	–	–	–	1,280	–	14,715	–	–	–	–	×	–	2,153	–	18,952	–	–	–	–	–	–	0	–	1
		4	–	–	–	–	–	–	1,280	–	14	–	–	–	–	×	–	2,153	–	18,992	–	–	–	–	–	–	0	–	0
		5	–	–	–	–	–	–	1,280	–	0	–	–	–	–	–	–	2,153	–	18,992	–	–	–	–	–	–	619	–	843
		6	–	–	–	–	–	1,441	1,284	10.9%	690	–	–	–	–	–	–	2,153	–	0	667	–	–	–	–	667	620	7.0%	0
		7	–	–	–	–	–	1,391	1,285	7.6%	284	–	–	–	–	–	–	2,153	–	0	667	–	–	–	–	667	620	7.0%	0
		8	1,364	–	–	–	–	1,364	1,286	5.7%	1	–	–	–	–	–	–	2,153	–	0	667	–	–	–	–	667	620	7.0%	0
		9	1,364	–	–	–	–	1,364	1,286	5.7%	0	–	–	–	–	–	–	2,153	–	0	667	–	–	–	–	667	620	7.0%	0
		10	1,364	–	–	–	–	1,364	1,286	5.7%	0	–	–	–	–	–	2,245	2,157	3.9%	2,624	667	–	–	–	–	667	620	7.0%	0
		11	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
		12	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
		13	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
		14	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
		15	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
11	150	1	–	–	0	–	×	–	1,435	–	3,612	–	–	0	–	×	–	2,220	–	13,583	–	–	–	–	×	–	0	–	13,118
		2	–	–	0	–	×	–	1,435	–	3,611	–	–	0	–	×	–	2,220	–	13,559	–	–	–	–	×	–	0	–	15
		3	–	–	0	–	×	–	1,435	–	3	–	–	0	–	×	–	2,220	–	13,575	–	–	–	–	–	–	0	–	0
		4	–	–	0	–	–	–	1,435	–	0	–	–	0	–	×	–	2,220	–	13,887	–	–	–	–	–	–	683	–	940
		5	–	–	0	–	–	1,551	1,441	7.1%	692	–	–	0	–	–	–	2,220	–	13,882	–	–	–	–	–	–	683	–	863
		6	1,506	–	0	–	–	1,531	1,439	6.0%	579	–	–	0	–	–	–	2,220	–	13,905	–	–	–	–	–	–	684	–	206
		7	1,506	–	0	–	–	1,511	1,445	4.4%	0	–	–	0	–	–	–	2,221	–	16,117	–	–	–	–	–	734	684	6.8%	77
		8	1,506	–	–	–	–	1,506	1,444	4.1%	0	–	–	0	–	–	–	2,221	–	3,312	734	–	–	–	–	734	684	6.8%	0
		9	1,506	–	4	–	–	1,506	1,444	4.1%	0	2,266	–	0	–	–	2,266	2,222	1.9%	1,283	734	–	–	–	–	734	684	6.8%	0
		10	1,506	–	0	–	–	1,506	1,444	4.1%	0	2,266	–	–	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		11	1,506	–	4	–	–	1,506	1,444	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		12	1,506	–	4	–	–	1,506	1,444	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		13	1,506	–	4	–	–	1,506	1,444	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		14	1,506	–	0	–	–	1,506	1,445	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		15	1,506	–	0	–	–	1,506	1,445	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0

Table 4.5: Solutions to the instances with service resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality.

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	10	1	–	**146**	146	0.0%	**146**	**146**	146	0.0%	27	–	**249**	249	0.0%	**249**	**249**	249	0.0%	107	–	88	41	53.4%	**84**	**84**	84	0.0%	11
		2	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	65	17.7%	**79**	**79**	79	0.0%	0
		3	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		4	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		5	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	57	27.8%	**79**	**79**	79	0.0%	0
		6	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	57	27.8%	**79**	**79**	79	0.0%	0
		7	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	56	29.1%	**79**	**79**	79	0.0%	0
		8	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	55	30.4%	**79**	**79**	79	0.0%	0
		9	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	57	27.8%	**79**	**79**	79	0.0%	0
		10	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		11	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		12	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		13	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		14	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	53	32.9%	**79**	**79**	79	0.0%	0
		15	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	205	**205**	205	0.0%	**205**	**205**	205	0.0%	0	79	79	54	31.6%	**79**	**79**	79	0.0%	0
11	10	1	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	62	26.2%	**84**	**84**	84	0.0%	0
		2	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		3	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	68	19.0%	**84**	**84**	84	0.0%	0
		4	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	62	26.2%	**84**	**84**	84	0.0%	0
		5	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0
		6	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		7	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		8	146	146	130	11.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		9	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		10	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		11	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		12	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	65	22.6%	**84**	**84**	84	0.0%	0
		13	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0
		14	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	63	25.0%	**84**	**84**	84	0.0%	0
		15	146	**146**	146	0.0%	**146**	**146**	146	0.0%	0	231	**231**	231	0.0%	**231**	**231**	231	0.0%	0	84	84	64	23.8%	**84**	**84**	84	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	15	1	–	–	47	–	239	**206**	206	0.0%	2,172	–	–	160	–	×	×	–	–	1,367	–	–	17	–	150	**120**	120	0.0%	119
		2	–	192	59	69.3%	196	**180**	180	0.0%	6	277	277	173	37.5%	**277**	**277**	277	0.0%	0	118	–	26	–	151	**118**	118	0.0%	0
		3	179	180	48	73.3%	196	**179**	179	0.0%	0	277	277	195	29.6%	**277**	**277**	277	0.0%	0	118	–	20	–	144	**118**	118	0.0%	0
		4	179	186	52	72.0%	196	**179**	179	0.0%	0	277	277	204	26.4%	**277**	**277**	277	0.0%	0	118	–	19	–	144	**118**	118	0.0%	0
		5	179	191	50	73.8%	196	**179**	179	0.0%	0	277	277	203	26.7%	**277**	**277**	277	0.0%	0	118	–	22	–	160	**118**	118	0.0%	0
		6	179	–	51	–	196	**179**	179	0.0%	0	277	277	146	47.3%	**277**	**277**	277	0.0%	0	118	–	22	–	160	**118**	118	0.0%	0
		7	179	191	49	74.3%	196	**179**	179	0.0%	0	277	277	188	32.1%	**277**	**277**	277	0.0%	0	118	–	23	–	160	**118**	118	0.0%	0
		8	179	196	52	73.5%	196	**179**	179	0.0%	0	277	277	189	31.8%	**277**	**277**	277	0.0%	0	118	–	20	–	160	**118**	118	0.0%	0
		9	179	194	49	74.7%	196	**179**	179	0.0%	0	277	277	177	36.1%	**277**	**277**	277	0.0%	0	118	–	21	–	160	**118**	118	0.0%	0
		10	179	188	47	75.0%	196	**179**	179	0.0%	0	277	277	207	25.3%	**277**	**277**	277	0.0%	0	118	–	20	–	160	**118**	118	0.0%	0
		11	179	205	52	74.6%	196	**179**	179	0.0%	0	277	277	207	25.3%	**277**	**277**	277	0.0%	0	118	–	19	–	160	**118**	118	0.0%	0
		12	179	213	48	77.5%	196	**179**	179	0.0%	0	277	277	225	18.8%	**277**	**277**	277	0.0%	0	118	–	15	–	160	**118**	118	0.0%	0
		13	179	220	49	77.7%	196	**179**	179	0.0%	0	277	277	213	23.1%	**277**	**277**	277	0.0%	0	118	–	19	–	160	**118**	118	0.0%	0
		14	179	180	51	71.7%	196	**179**	179	0.0%	0	277	277	230	17.0%	**277**	**277**	277	0.0%	0	118	–	20	–	160	**118**	118	0.0%	0
		15	179	–	45	–	196	**179**	179	0.0%	0	277	277	237	14.4%	**277**	**277**	277	0.0%	0	118	–	18	–	160	**118**	118	0.0%	0
11	15	1	–	244	63	74.2%	241	**223**	223	0.0%	20	–	–	354	–	×	×	–	–	263	–	–	44	–	157	**114**	114	0.0%	263
		2	219	231	98	57.6%	248	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	41	–	143	**107**	107	0.0%	0
		3	219	230	93	59.6%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	42	–	147	**107**	107	0.0%	0
		4	219	225	78	65.3%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	41	–	147	**107**	107	0.0%	0
		5	219	225	92	59.1%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	45	–	147	**107**	107	0.0%	0
		6	219	219	84	61.6%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	125	44	64.8%	147	**107**	107	0.0%	0
		7	219	229	101	55.9%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	46	–	147	**107**	107	0.0%	0
		8	219	248	102	58.9%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	46	–	147	**107**	107	0.0%	0
		9	219	219	95	56.6%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	44	–	147	**107**	107	0.0%	0
		10	219	219	99	54.8%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	121	42	65.3%	147	**107**	107	0.0%	0
		11	219	219	99	54.8%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	45	–	147	**107**	107	0.0%	0
		12	219	219	98	55.3%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	126	44	65.1%	147	**107**	107	0.0%	0
		13	219	219	98	55.3%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	42	–	147	**107**	107	0.0%	0
		14	219	219	95	56.6%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	42	–	147	**107**	107	0.0%	0
		15	219	240	102	57.5%	250	**219**	219	0.0%	0	302	**302**	302	0.0%	**302**	**302**	302	0.0%	0	107	–	42	–	147	**107**	107	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	20	1	–	–	51	–	×	–	259	–	19,850	–	–	105	–	×	×	–	–	18,990	–	–	14	–	228	161	156	3.1%	9,169
		2	–	–	53	–	477	**247**	247	0.0%	115	–	–	119	–	580	**375**	375	0.0%	762	148	–	16	–	297	**148**	148	0.0%	0
		3	246	–	51	–	449	**246**	246	0.0%	0	369	–	101	–	566	**369**	369	0.0%	0	148	–	16	–	293	**148**	148	0.0%	0
		4	246	–	51	–	455	**246**	246	0.0%	0	369	–	99	–	448	**369**	369	0.0%	0	148	–	18	–	266	**148**	148	0.0%	0
		5	246	–	55	–	437	**246**	246	0.0%	0	369	–	116	–	551	**369**	369	0.0%	0	148	–	18	–	243	**148**	148	0.0%	0
		6	246	–	51	–	437	**246**	246	0.0%	0	369	–	95	–	551	**369**	369	0.0%	0	148	–	16	–	217	**148**	148	0.0%	0
		7	246	–	51	–	437	**246**	246	0.0%	0	369	–	107	–	551	**369**	369	0.0%	0	148	–	18	–	242	**148**	148	0.0%	0
		8	246	–	52	–	437	**246**	246	0.0%	0	369	–	127	–	551	**369**	369	0.0%	0	148	–	14	–	242	**148**	148	0.0%	0
		9	246	–	52	–	437	**246**	246	0.0%	0	369	–	104	–	551	**369**	369	0.0%	0	148	–	17	–	242	**148**	148	0.0%	0
		10	246	–	53	–	437	**246**	246	0.0%	0	369	–	120	–	551	**369**	369	0.0%	0	148	–	20	–	242	**148**	148	0.0%	0
		11	246	–	53	–	437	**246**	246	0.0%	0	369	–	123	–	551	**369**	369	0.0%	0	148	–	17	–	242	**148**	148	0.0%	0
		12	246	–	54	–	437	**246**	246	0.0%	0	369	–	106	–	551	**369**	369	0.0%	0	148	–	15	–	242	**148**	148	0.0%	0
		13	246	–	52	–	437	**246**	246	0.0%	0	369	–	112	–	551	**369**	369	0.0%	0	148	–	15	–	242	**148**	148	0.0%	0
		14	246	–	48	–	437	**246**	246	0.0%	0	369	–	105	–	551	**369**	369	0.0%	0	148	–	19	–	242	**148**	148	0.0%	0
		15	246	–	52	–	437	**246**	246	0.0%	0	369	–	99	–	551	**369**	369	0.0%	0	148	–	17	–	242	**148**	148	0.0%	0
11	20	1	–	–	81	–	374	321	308	4.0%	13,641	–	–	153	–	×	×	–	–	7,966	–	–	33	–	295	**147**	147	0.0%	984
		2	283	–	79	–	403	**283**	283	0.0%	0	–	–	188	–	564	**361**	361	0.0%	26	143	–	26	–	267	**143**	143	0.0%	0
		3	283	–	82	–	449	**283**	283	0.0%	0	359	–	158	–	522	**359**	359	0.0%	0	143	–	29	–	290	**143**	143	0.0%	0
		4	283	–	88	–	458	**283**	283	0.0%	0	359	–	164	–	551	**359**	359	0.0%	0	143	–	28	–	267	**143**	143	0.0%	0
		5	283	–	75	–	458	**283**	283	0.0%	0	359	–	127	–	516	**359**	359	0.0%	0	143	–	28	–	247	**143**	143	0.0%	0
		6	283	–	79	–	458	**283**	283	0.0%	0	359	359	176	51.0%	516	**359**	359	0.0%	0	143	–	31	–	247	**143**	143	0.0%	0
		7	283	–	76	–	458	**283**	283	0.0%	0	359	–	128	–	516	**359**	359	0.0%	0	143	–	28	–	247	**143**	143	0.0%	0
		8	283	–	75	–	458	**283**	283	0.0%	0	359	–	140	–	516	**359**	359	0.0%	0	143	–	29	–	247	**143**	143	0.0%	0
		9	283	–	81	–	458	**283**	283	0.0%	0	359	–	152	–	516	**359**	359	0.0%	0	143	–	25	–	247	**143**	143	0.0%	0
		10	283	–	79	–	458	**283**	283	0.0%	0	359	–	151	–	516	**359**	359	0.0%	0	143	–	36	–	247	**143**	143	0.0%	0
		11	283	–	75	–	458	**283**	283	0.0%	0	359	–	166	–	516	**359**	359	0.0%	0	143	–	27	–	247	**143**	143	0.0%	0
		12	283	–	79	–	458	**283**	283	0.0%	0	359	–	144	–	516	**359**	359	0.0%	0	143	–	27	–	247	**143**	143	0.0%	0
		13	283	–	79	–	458	**283**	283	0.0%	0	359	–	155	–	516	**359**	359	0.0%	0	143	–	37	–	247	**143**	143	0.0%	0
		14	283	–	80	–	458	**283**	283	0.0%	0	359	372	171	54.0%	516	**359**	359	0.0%	0	143	–	27	–	247	**143**	143	0.0%	0
		15	283	–	80	–	458	**283**	283	0.0%	0	359	–	156	–	516	**359**	359	0.0%	0	143	–	28	–	247	**143**	143	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	30	1	–	–	30	–	×	–	348	–	16,505	–	–	84	–	×	–	514	–	35,193	–	–	7	–	×	–	207	–	2,952
		2	–	–	30	–	–	356	352	1.1%	6,963	–	–	78	–	–	686	514	25.1%	30,880	–	–	7	–	–	239	204	14.6%	6,143
		3	351	–	30	–	725	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	–	–	6	–	–	**205**	205	0.0%	95
		4	351	–	30	–	–	**351**	351	0.0%	0	517	–	76	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		5	351	–	30	–	–	**351**	351	0.0%	0	517	–	75	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		6	351	–	30	–	–	**351**	351	0.0%	0	517	–	82	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		7	351	–	30	–	–	**351**	351	0.0%	0	517	–	80	–	–	**517**	517	0.0%	0	204	–	9	–	–	**204**	204	0.0%	0
		8	351	–	30	–	–	**351**	351	0.0%	0	517	–	76	–	–	**517**	517	0.0%	0	204	–	9	–	–	**204**	204	0.0%	0
		9	351	–	30	–	–	**351**	351	0.0%	0	517	–	78	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		10	351	–	28	–	–	**351**	351	0.0%	0	517	–	74	–	–	**517**	517	0.0%	0	204	–	6	–	–	**204**	204	0.0%	0
		11	351	–	30	–	–	**351**	351	0.0%	0	517	–	75	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		12	351	–	30	–	–	**351**	351	0.0%	0	517	–	77	–	–	**517**	517	0.0%	0	204	–	7	–	–	**204**	204	0.0%	0
		13	351	–	30	–	–	**351**	351	0.0%	0	517	–	76	–	–	**517**	517	0.0%	0	204	–	7	–	–	**204**	204	0.0%	0
		14	351	–	30	–	–	**351**	351	0.0%	0	517	–	79	–	–	**517**	517	0.0%	0	204	–	10	–	–	**204**	204	0.0%	0
		15	351	–	30	–	–	**351**	351	0.0%	0	517	–	76	–	–	**517**	517	0.0%	0	204	–	7	–	–	**204**	204	0.0%	0
11	30	1	–	–	63	–	×	–	383	–	7,617	–	–	30	–	×	–	529	–	30,038	–	–	21	–	–	–	183	–	6,821
		2	387	–	62	–	–	**387**	387	0.0%	124	–	–	130	–	–	**544**	544	0.0%	13,160	184	–	6	–	–	**184**	184	0.0%	0
		3	387	–	62	–	–	**387**	387	0.0%	0	–	–	131	–	–	**538**	538	0.0%	344	184	–	21	–	–	**184**	184	0.0%	0
		4	387	–	62	–	798	**387**	387	0.0%	0	536	–	131	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		5	387	–	62	–	798	**387**	387	0.0%	0	536	–	131	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		6	387	–	62	–	798	**387**	387	0.0%	0	536	–	131	–	–	**536**	536	0.0%	0	184	–	14	–	–	**184**	184	0.0%	0
		7	387	–	63	–	798	**387**	387	0.0%	0	536	–	32	–	–	**536**	536	0.0%	0	184	–	10	–	–	**184**	184	0.0%	0
		8	387	–	62	–	798	**387**	387	0.0%	0	536	–	32	–	–	**536**	536	0.0%	0	184	–	21	–	–	**184**	184	0.0%	0
		9	387	–	63	–	798	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		10	387	–	62	–	798	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		11	387	–	62	–	798	**387**	387	0.0%	0	536	–	133	–	–	**536**	536	0.0%	0	184	–	10	–	–	**184**	184	0.0%	0
		12	387	–	62	–	798	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		13	387	–	62	–	798	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0
		14	387	–	62	–	798	**387**	387	0.0%	0	536	–	32	–	–	**536**	536	0.0%	0	184	–	16	–	–	**184**	184	0.0%	0
		15	387	–	22	–	798	**387**	387	0.0%	0	536	–	132	–	–	**536**	536	0.0%	0	184	–	20	–	–	**184**	184	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	40	1	–	–	14	–	×	–	426	–	18,661	–	–	0	–	×	–	625	–	36,911	–	–	0	–	×	–	234	–	12,487
		2	–	–	0	–	–	–	427	–	8,693	–	–	0	–	×	–	625	–	27,030	–	–	0	–	–	–	235	–	3,647
		3	431	–	0	–	–	**431**	431	0.0%	6	–	–	0	–	–	645	625	3.1%	16,350	–	–	0	–	–	266	235	11.7%	2,353
		4	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	–	–	0	–	–	239	238	0.4%	882
		5	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		6	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		7	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		8	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		9	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		10	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		11	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		12	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		13	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		14	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
		15	431	–	0	–	–	**431**	431	0.0%	0	634	–	0	–	–	**634**	634	0.0%	0	239	–	0	–	–	**239**	239	0.0%	0
11	40	1	–	–	14	–	×	–	489	–	12,301	–	–	50	–	×	–	682	–	32,708	–	–	4	–	×	–	236	–	10,368
		2	–	–	14	–	–	556	495	11.0%	6,782	–	–	28	–	×	–	682	–	34,062	–	–	0	–	–	**238**	238	0.0%	250
		3	499	–	14	–	–	**499**	499	0.0%	0	–	–	42	–	–	772	682	11.7%	24,610	238	–	0	–	–	**238**	238	0.0%	1
		4	499	–	14	–	–	**499**	499	0.0%	0	–	–	28	–	–	716	688	3.9%	20,026	238	–	0	–	–	**238**	238	0.0%	0
		5	499	–	4	–	–	**499**	499	0.0%	0	696	–	40	–	–	**696**	696	0.0%	30	238	–	0	–	–	**238**	238	0.0%	0
		6	499	–	5	–	–	**499**	499	0.0%	0	696	–	28	–	–	**696**	696	0.0%	0	238	–	2	–	–	**238**	238	0.0%	0
		7	499	–	14	–	–	**499**	499	0.0%	0	696	–	49	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		8	499	–	14	–	–	**499**	499	0.0%	0	696	–	34	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		9	499	–	14	–	–	**499**	499	0.0%	0	696	–	56	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		10	499	–	14	–	–	**499**	499	0.0%	0	696	–	36	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		11	499	–	14	–	–	**499**	499	0.0%	0	696	–	44	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		12	499	–	14	–	–	**499**	499	0.0%	0	696	–	45	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		13	499	–	14	–	–	**499**	499	0.0%	0	696	–	42	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		14	499	–	14	–	–	**499**	499	0.0%	0	696	–	40	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0
		15	499	–	14	–	–	**499**	499	0.0%	0	696	–	41	–	–	**696**	696	0.0%	0	238	–	0	–	–	**238**	238	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	60	1	–	–	0	–	×	–	604	–	23,518	–	–	0	–	×	–	903	–	30,509	–	–	0	–	×	–	326	–	6,081
		2	–	–	0	–	×	–	606	–	18,480	–	–	0	–	×	–	903	–	31,406	–	–	0	–	–	–	326	–	13,068
		3	–	–	0	–	–	700	607	13.3%	3,294	–	–	0	–	–	–	903	–	33,219	–	–	0	–	–	–	327	–	1,792
		4	–	–	0	–	–	638	611	4.2%	2,062	–	–	0	–	–	–	912	–	19,396	–	–	0	–	–	363	327	9.9%	1,048
		5	619	–	0	–	–	619	613	1.0%	273	922	–	0	–	–	925	913	1.3%	8,769	–	–	0	–	–	348	327	6.0%	883
		6	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	922	918	0.4%	1,427	332	–	0	–	–	332	330	0.6%	0
		7	619	–	0	–	–	619	616	0.5%	0	922	–	0	–	–	**922**	922	0.0%	182	332	–	0	–	–	332	330	0.6%	0
		8	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		9	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		10	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		11	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		12	619	–	0	–	–	619	618	0.2%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		13	619	–	0	–	–	619	618	0.2%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		14	619	–	0	–	–	619	617	0.3%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
		15	619	–	0	–	–	619	618	0.2%	0	922	–	0	–	–	**922**	922	0.0%	0	332	–	0	–	–	332	330	0.6%	0
11	60	1	–	–	4	–	×	–	699	–	15,099	–	–	0	–	×	–	985	–	26,490	–	–	0	–	×	–	322	–	14,559
		2	–	–	4	–	–	–	699	–	3,119	–	–	0	–	×	–	985	–	27,884	–	–	0	–	–	–	322	–	3,965
		3	–	–	4	–	–	729	704	3.4%	2,418	–	–	0	–	–	–	985	–	23,220	–	–	0	–	–	**324**	324	0.0%	119
		4	706	–	4	–	–	**706**	706	0.0%	0	–	–	0	–	–	1,049	988	5.8%	17,678	324	–	0	–	–	**324**	324	0.0%	0
		5	706	–	4	–	–	**706**	706	0.0%	0	–	–	0	–	–	992	989	0.3%	7,205	324	–	0	–	–	**324**	324	0.0%	0
		6	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		7	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		8	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		9	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		10	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		11	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		12	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		13	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		14	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0
		15	706	–	4	–	–	**706**	706	0.0%	0	991	–	0	–	–	**991**	991	0.0%	0	324	–	0	–	–	**324**	324	0.0%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	80	1	–	–	0	–	×	–	746	–	3,216	–	–	0	–	×	–	1,176	–	28,861	–	–	0	–	×	–	390	–	14,770
		2	–	–	0	–	×	–	746	–	20,083	–	–	0	–	×	–	1,176	–	28,390	–	–	0	–	–	–	390	–	8,810
		3	–	–	0	–	–	–	746	–	16,948	–	–	0	–	×	–	1,176	–	27,177	–	–	0	–	–	–	391	–	1,219
		4	–	–	0	–	–	866	754	12.9%	4,357	–	–	0	–	–	–	1,176	–	29,646	393	–	0	–	–	**393**	393	0.0%	111
		5	–	–	0	–	–	811	757	6.7%	1,049	–	–	0	–	–	–	1,180	–	25,328	393	–	0	–	–	**393**	393	0.0%	0
		6	794	–	0	–	–	795	761	4.3%	210	–	–	0	–	–	–	1,183	–	12,903	393	–	0	–	–	**393**	393	0.0%	0
		7	794	–	0	–	–	794	762	4.0%	0	–	–	0	–	–	1,243	1,183	4.8%	5,410	393	–	0	–	–	**393**	393	0.0%	0
		8	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,197	1,186	0.9%	928	393	–	0	–	–	**393**	393	0.0%	0
		9	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,184	0.9%	421	393	–	0	–	–	**393**	393	0.0%	0
		10	794	–	0	–	–	794	763	3.9%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		11	794	–	0	–	–	794	761	4.2%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		12	794	–	0	–	–	794	763	3.9%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		13	794	–	0	–	–	794	762	4.0%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		14	794	–	0	–	–	775	762	1.7%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
		15	794	–	0	–	–	775	762	1.7%	0	1,195	–	0	–	–	1,195	1,186	0.8%	0	393	–	0	–	–	**393**	393	0.0%	0
11	80	1	–	–	4	–	×	–	871	–	2,463	–	–	0	–	×	–	1,260	–	24,496	–	–	0	–	×	–	410	–	9,774
		2	–	–	4	–	–	–	872	–	14,985	–	–	0	–	×	–	1,260	–	24,553	–	–	0	–	–	–	410	–	3,488
		3	–	–	4	–	–	–	875	–	16,354	–	–	0	–	×	–	1,260	–	24,637	–	–	0	–	–	–	410	–	772
		4	–	–	4	–	–	965	880	8.8%	2,201	–	–	0	–	–	–	1,260	–	25,370	–	–	0	–	–	448	411	8.3%	464
		5	883	–	4	–	–	**883**	883	0.0%	338	–	–	0	–	–	1,450	1,260	13.1%	20,788	413	–	0	–	–	413	411	0.5%	379
		6	883	–	4	–	–	**883**	883	0.0%	0	–	–	0	–	–	–	1,266	–	7,696	413	–	0	–	–	413	412	0.2%	0
		7	883	–	4	–	–	**883**	883	0.0%	0	–	–	0	–	–	1,315	1,267	3.7%	4,107	413	–	0	–	–	413	412	0.2%	0
		8	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,281	1,271	0.8%	1,193	413	–	0	–	–	413	412	0.2%	0
		9	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,274	0.5%	0	413	–	0	–	–	413	412	0.2%	0
		10	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		11	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		12	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		13	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		14	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0
		15	883	–	4	–	–	**883**	883	0.0%	0	1,280	–	0	–	–	1,280	1,272	0.6%	0	413	–	0	–	–	413	412	0.2%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	100	1	–	–	0	–	×	–	902	–	10,416	–	–	0	–	×	–	1,454	–	27,115	–	–	0	–	×	–	459	–	10,591
		2	–	–	0	–	×	–	902	–	11,969	–	–	0	–	×	–	1,454	–	26,938	–	–	0	–	–	–	461	–	16,475
		3	–	–	0	–	–	–	902	–	15,531	–	–	0	–	×	–	1,454	–	26,476	–	–	0	–	–	–	462	–	1,105
		4	–	–	0	–	–	–	904	–	1,792	–	–	0	–	–	–	1,454	–	27,561	492	–	0	–	–	497	463	6.8%	372
		5	–	–	0	–	–	1,007	906	10.0%	972	–	–	0	–	–	–	1,454	–	27,020	492	–	0	–	–	492	464	5.7%	22
		6	–	–	0	–	–	982	909	7.4%	197	–	–	0	–	–	–	1,456	–	19,567	492	–	0	–	–	492	464	5.7%	0
		7	970	–	0	–	–	957	910	4.9%	4	–	–	0	–	–	–	1,461	–	3,429	492	–	0	–	–	486	464	4.5%	0
		8	970	–	0	–	–	964	910	5.6%	0	–	–	0	–	–	–	1,459	–	299	492	–	0	–	–	486	464	4.5%	0
		9	970	–	0	–	–	964	910	5.6%	0	–	–	0	–	–	1,513	1,463	3.3%	46	492	–	0	–	–	486	464	4.5%	0
		10	970	–	0	–	–	970	910	6.2%	0	1,504	–	0	–	–	1,504	1,462	2.8%	16	492	–	0	–	–	486	464	4.5%	0
		11	970	–	0	–	–	958	910	5.0%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	492	464	5.7%	0
		12	970	–	0	–	–	958	907	5.3%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		13	970	–	0	–	–	957	910	4.9%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	464	4.5%	0
		14	970	–	0	–	–	958	910	5.0%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	486	462	4.9%	0
		15	970	–	0	–	–	958	910	5.0%	0	1,504	–	0	–	–	1,504	1,465	2.6%	0	492	–	0	–	–	492	464	5.7%	0
11	100	1	–	–	4	–	×	–	1,038	–	8,505	–	–	0	–	×	–	1,535	–	22,180	–	–	0	–	×	–	475	–	19,951
		2	–	–	4	–	×	–	1,042	–	2,982	–	–	0	–	×	–	1,535	–	22,159	–	–	0	–	–	–	475	–	8,959
		3	–	–	4	–	–	–	1,040	–	9,742	–	–	0	–	×	–	1,535	–	22,162	–	–	0	–	–	–	476	–	1,149
		4	–	–	4	–	–	1,185	1,046	11.7%	4,482	–	–	0	–	–	–	1,535	–	22,267	–	–	0	–	–	–	476	–	1,212
		5	1,060	–	4	–	–	1,083	1,047	3.3%	1,245	–	–	0	–	–	–	1,535	–	22,225	480	–	0	–	–	478	477	0.2%	23
		6	1,060	–	4	–	–	1,060	1,051	0.8%	47	–	–	0	–	–	–	1,537	–	4,176	480	–	0	–	–	478	477	0.2%	0
		7	1,060	–	4	–	–	1,060	1,051	0.8%	0	–	–	0	–	–	1,624	1,539	5.2%	2,162	480	–	0	–	–	480	477	0.6%	0
		8	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,582	1,545	2.3%	2,548	480	–	0	–	–	480	477	0.6%	0
		9	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	3	480	–	0	–	–	480	477	0.6%	0
		10	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		11	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		12	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		13	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		14	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0
		15	1,060	–	4	–	–	1,060	1,051	0.8%	0	1,585	–	0	–	–	1,585	1,544	2.6%	0	480	–	0	–	–	480	477	0.6%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality. (Continued on next page)

			Instance Set 1									Instance Set 2									Instance Set 3								
			TS	MIP			CP	BPC				TS	MIP			CP	BPC				TS	MIP			CP	BPC			
$\|\mathcal{L}\|$	$\|\mathcal{P}\|$	C_l	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG	UB	UB	LB	Gap	UB	UB	LB	Gap	NG
8	150	1	–	–	–	–	×	–	1,280	–	14,294	–	–	–	–	×	–	2,153	–	18,698	–	–	–	–	×	–	0	–	9,322
		2	–	–	–	–	×	–	1,280	–	14,471	–	–	–	–	×	–	2,153	–	18,686	–	–	–	–	–	–	0	–	2,982
		3	–	–	–	–	×	–	1,280	–	14,392	–	–	–	–	×	–	2,153	–	18,504	–	–	–	–	–	–	0	–	8,982
		4	–	–	–	–	–	–	1,283	–	8,329	–	–	–	–	×	–	2,153	–	18,531	–	–	–	–	–	–	0	–	2,227
		5	–	–	–	–	–	–	1,283	–	1,525	–	–	–	–	–	–	2,153	–	20,621	–	–	–	–	–	–	619	–	759
		6	–	–	–	–	–	1,506	1,283	14.8%	656	–	–	–	–	–	–	2,153	–	7,961	667	–	–	–	–	667	620	7.0%	0
		7	–	–	–	–	–	–	1,283	–	1,438	–	–	–	–	–	–	2,155	–	2,322	667	–	–	–	–	667	620	7.0%	0
		8	–	–	–	–	–	1,407	1,285	8.7%	375	–	–	–	–	–	–	2,155	–	2,929	667	–	–	–	–	667	620	7.0%	0
		9	1,364	–	–	–	–	1,364	1,285	5.8%	182	–	–	–	–	–	–	2,156	–	3,017	667	–	–	–	–	667	620	7.0%	0
		10	1,364	–	–	–	–	1,364	1,286	5.7%	0	–	–	–	–	–	–	2,153	–	429	667	–	–	–	–	667	620	7.0%	0
		11	1,364	–	–	–	–	1,364	1,286	5.7%	0	–	–	–	–	–	–	2,153	–	286	667	–	–	–	–	667	620	7.0%	0
		12	1,364	–	–	–	–	1,364	1,286	5.7%	0	–	–	–	–	–	–	2,155	–	18	667	–	–	–	–	667	620	7.0%	0
		13	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,206	2,155	2.3%	0	667	–	–	–	–	667	620	7.0%	0
		14	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
		15	1,364	–	–	–	–	1,364	1,286	5.7%	0	2,217	–	–	–	–	2,217	2,160	2.6%	0	667	–	–	–	–	667	620	7.0%	0
11	150	1	–	–	0	–	×	–	1,435	–	3,596	–	–	0	–	×	–	2,220	–	13,790	–	–	–	–	×	–	0	–	14,792
		2	–	–	4	–	×	–	1,435	–	9,132	–	–	0	–	×	–	2,220	–	13,840	–	–	–	–	×	–	0	–	22,348
		3	–	–	4	–	–	–	1,439	–	5,639	–	–	0	–	×	–	2,220	–	13,847	–	–	–	–	–	–	683	–	984
		4	–	–	4	–	–	–	1,440	–	1,219	–	–	0	–	–	–	2,220	–	13,830	–	–	–	–	–	–	683	–	962
		5	–	–	0	–	–	–	1,440	–	1,268	–	–	0	–	–	–	2,220	–	13,889	–	–	–	–	–	–	683	–	842
		6	1,506	–	4	–	–	1,533	1,441	6.0%	737	–	–	0	–	–	–	2,220	–	13,930	–	–	–	–	–	–	684	–	212
		7	1,506	–	0	–	–	1,506	1,442	4.2%	321	–	–	–	–	–	–	2,220	–	13,916	–	–	–	–	–	736	683	7.2%	599
		8	1,506	–	4	–	–	1,506	1,444	4.1%	0	–	–	0	–	–	–	2,220	–	4,821	734	–	–	–	–	734	684	6.8%	0
		9	1,506	–	4	–	–	1,506	1,444	4.1%	0	–	–	0	–	–	2,285	2,221	2.8%	952	734	–	–	–	–	734	684	6.8%	0
		10	1,506	–	4	–	–	1,506	1,444	4.1%	0	2,266	–	0	–	–	2,269	2,223	2.0%	2,141	734	–	–	–	–	734	684	6.8%	0
		11	1,506	–	4	–	–	1,506	1,445	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		12	1,506	–	4	–	–	1,506	1,445	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		13	1,506	–	0	–	–	1,506	1,444	4.1%	0	2,266	–	–	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		14	1,506	–	4	–	–	1,506	1,445	4.1%	0	2,266	–	–	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0
		15	1,506	–	4	–	–	1,506	1,445	4.1%	0	2,266	–	0	–	–	2,266	2,223	1.9%	0	734	–	–	–	–	734	684	6.8%	0

Table 4.6: Solutions to the instances with presence resources. The upper bound (UB), lower bound (LB), optimality gap (Gap) and the number of nogoods (NG) are reported for each solver when available. The symbol × denotes a proof of infeasibility and values in bold indicate a proof of optimality.

Chapter 5

Branch-and-Check with Explanations

Strong objective bounds are necessary for pruning large sections of the search space in hard optimization problems. The linear relaxation bounds in mixed integer programming are generally stronger than the bounds found by propagators in constraint programming because propagators usually have a narrow view of the problem (e.g., Focacci, Lodi and Milano 1999, Hooker 2006, Milano 2010), whereas the constraints in a linear relaxation can directly influence the objective function via elementary row operations. Even though constraint programming lacks strong bounds (e.g., Benchimol et al. 2012, Focacci, Lodi and Milano 2002, 2004), it excels at finding feasible solutions in satisfaction problems (e.g., Milano 2010), and particularly, in scheduling problems (e.g., Schutt et al. 2009, 2010, 2013). The previous chapter shows that it is possible to take advantage of the distinct strengths of mixed integer programming and constraint programming by hybridization. This concept is developed further in the present chapter.

This chapter proposes an exact method named *branch-and-check with explanations* (BCE). BCE represents an optimization problem in terms of a master problem and a checking subproblem. The master problem is a mixed integer program that ignores a number of difficult constraints. The feasibility of these constraints is checked in the constraint programming checking subproblem. More precisely, BCE uses constraint programming for three purposes: (1) to fix variables in the master problem through propagation and inference, (2) to generate cuts in the master problem via conflict analysis, and (3) to probe the feasibility of candidate solutions from the linear relaxation and to derive additional cuts through conflict analysis if the probing fails. Hence, the constraint programming model can be viewed, in some way, as equivalent to an advanced separation algorithm and primal heuristic in conventional branch-and-cut.

BCE opens some interesting opportunities unavailable in branch-and-cut. First, it has the advantage of relying on a general-purpose constraint programming engine for inference and cut separation, bypassing the need for problem-specific separation algorithms. Second, it permits

As described in the Preface, the main findings of this chapter are published in the paper titled "Branch-and-Check with Explanations for the Vehicle Routing Problem with Time Windows".

conflict-based branching rules in addition to traditional branching rules (e.g., fractional or pseudo-cost branching). Finally, BCE can recognize special classes of cuts after conflict analysis and then strengthen them using well-known techniques from the mathematical programming literature. As a result, BCE offers a natural integration of not only mathematical programming and constraint programming but also Boolean satisfiability via conflict analysis.

The BCE method is evaluated on the Vehicle Routing Problem with Time Windows (VRPTW). Experimental results indicate that BCE outperforms a branch-and-cut algorithm: it proves optimality on more instances and finds significantly better solutions to instances for which branch-and-cut cannot prove optimality. The results also show that a conflict-based branching rule is particularly effective in BCE and that cut strengthening produces interesting improvements to the lower bounds.

The rest of this chapter is structured as follows. Section 5.1 formalizes the BCE method for the VRPTW. Section 5.2 discusses several cut strengthenings. Section 5.3 presents experimental results that compare the BCE model of the VRPTW to the branch-and-cut model. Section 5.4 discusses limitations and potential improvements of the BCE approach for the VRPTW and the relevance of BCE to branch-and-price. Section 5.5 presents concluding remarks.

5.1 The Branch-and-Check Model of the VRPTW

This section proposes the BCE model of the VRPTW. The model is organized around a mixed integer programming master problem and a constraint programming checking subproblem.

The Master Problem The master problem is the standard two-index flow model mentioned in Section 2.7. Its data and decision variables are listed in Table 5.1. The problem is modeled over a time interval $\mathcal{T} = [0, T]$, where $T > 0$. It contains an unlimited number of vehicles with capacity $Q \geq 0$. The set $\mathcal{Q} = [0, Q]$ denotes the range of vehicle load. The vehicles need to service R requests, which are grouped in the set $\mathcal{R} = \{1, \ldots, R\}$. The set $\mathcal{N} = \mathcal{R} \cup \{s, e\}$ represents the nodes of the underlying graph, which has a node for each request $i \in \mathcal{R}$ and two additional nodes denoting the start depot $s = 0$ and end depot $e = R + 1$. The nodes in the underlying graph are connected by the arcs

$$\mathcal{A} = \{(s, i) | i \in \mathcal{R}\} \cup \{(i, j) | i, j \in \mathcal{R}, i \neq j, a_i + t_{i,j} \leq b_j, q_i + q_j \leq Q\} \cup \{(i, e) | i \in \mathcal{R}\}. \tag{5.1}$$

Each arc $(i, j) \in \mathcal{A}$ has an associated cost $c_{i,j} \in \mathbb{Z}_+$ and travel time $t_{i,j} \in \mathcal{T}$. Each request $i \in \mathcal{R}$ has a vehicle load demand $q_i \in \mathcal{Q}$, while the start and end nodes have zero demand, i.e., $q_s = 0$ and $q_e = 0$. Each node $i \in \mathcal{N}$ has an earliest start time of service $a_i \in \mathcal{T}$ and latest start time of service $b_i \in \mathcal{T}$, with $a_s = b_s = 0$ and $a_e = b_e = T$. The service durations of the requests, which are present in the models from the previous chapters, are unnecessary in this model as they can be folded into the travel times. The $x_{i,j} \in \{0, 1\}$ decision variable indicates if a vehicle traverses the arc $(i, j) \in \mathcal{A}$.

The initial constraints of the model are shown in Figure 5.1. Objective Function (5.2) minimizes the total travel distance. Constraints (5.3) and (5.4) require every request to be visited

Name	Description
$T > 0$	Time horizon.
$\mathcal{T} = [0, T]$	Time interval.
$Q \geq 0$	Vehicle capacity.
$\mathcal{Q} = [0, Q]$	Range of vehicle load.
$R \in \{1, \dots, \infty\}$	Number of requests.
$\mathcal{R} = \{1, \dots, R\}$	Set of requests.
$s = 0$	Start node.
$e = R + 1$	End node.
$\mathcal{N} = \mathcal{R} \cup \{s, e\}$	Set of all nodes.
$\mathcal{A}$	Arcs of the network. Defined in Equation (5.1).
$c_{i,j} \in \mathbb{Z}_+$	Distance cost along arc $(i, j) \in \mathcal{A}$.
$t_{i,j} \in \mathcal{T}$	Travel time along arc $(i, j) \in \mathcal{A}$.
$q_i \in \mathcal{Q}$	Vehicle load demand of $i \in \mathcal{N}$.
$a_i \in \mathcal{T}$	Earliest service start time at $i \in \mathcal{N}$.
$b_i \in \mathcal{T}$	Latest service start time at $i \in \mathcal{N}$.
$x_{i,j} \in \{0, 1\}$	Decision variable indicating if a vehicle traverses $(i, j) \in \mathcal{A}$.

Table 5.1: The data and decision variables of the mixed integer programming master problem.

exactly once. Notice that the subtour elimination, vehicle capacity and time window constraints are omitted from the master problem.

The Checking Subproblem The decision variables of the constraint programming model are listed in Table 5.2. Instead of successor and predecessor variables commonly seen in constraint programming models of vehicle routing problems, the checking subproblem uses the $y_{i,j} \in \{0, 1\}$ variable to indicate whether a vehicle traverses the arc $(i, j) \in \mathcal{A}$. Using these binary variables provides a one-to-one mapping between the y variables and the x variables of the master problem. The $l_i \in [q_i, Q] \subseteq \mathcal{Q}$ and $t_i \in [a_i, b_i] \subseteq \mathcal{T}$ variables respectively contain the vehicle load and time at request $i \in \mathcal{N}$.

$$\min \sum_{(i,j) \in \mathcal{A}} c_{i,j} x_{i,j} \tag{5.2}$$

subject to

$$\sum_{h:(h,i) \in \mathcal{A}} x_{h,i} = 1 \qquad \forall i \in \mathcal{R}, \tag{5.3}$$

$$\sum_{j:(i,j) \in \mathcal{A}} x_{i,j} = 1 \qquad \forall i \in \mathcal{R}. \tag{5.4}$$

Figure 5.1: Initial constraints of the mixed integer programming master problem.

Name	Description
$y_{i,j} \in \{0, 1\}$	Decision variable indicating if a vehicle traverses $(i, j) \in \mathcal{A}$.
$l_i \in [q_i, Q] \subseteq \mathcal{Q}$	Vehicle load after service at request $i \in \mathcal{N}$.
$t_i \in [a_i, b_i] \subseteq \mathcal{T}$	Time that a vehicle begins service at request $i \in \mathcal{N}$.

Table 5.2: The decision variables of the constraint programming checking subproblem.

$$\bigvee_{h:(h,i)\in\mathcal{A}} y_{h,i} \qquad \forall i \in \mathcal{R}, \quad (5.5)$$

$$\bigvee_{j:(i,j)\in\mathcal{A}} y_{i,j} \qquad \forall i \in \mathcal{R}, \quad (5.6)$$

$$\neg y_{h,i} \vee \neg y_{h,j} \qquad \forall h, i, j \in \mathcal{N} : (h, i) \in \mathcal{A}, (h, j) \in \mathcal{A}, i \neq j, \quad (5.7)$$

$$\neg y_{h,j} \vee \neg y_{i,j} \qquad \forall h, i, j \in \mathcal{N} : (h, j) \in \mathcal{A}, (i, j) \in \mathcal{A}, h \neq i, \quad (5.8)$$

$$\textsc{NoSubtour}(y), \quad (5.9)$$

$$y_{i,j} \longrightarrow l_j \geq l_i + q_j \qquad \forall (i, j) \in \mathcal{A}, \quad (5.10)$$

$$y_{i,j} \longrightarrow t_j \geq t_i + t_{i,j} \qquad \forall (i, j) \in \mathcal{A}. \quad (5.11)$$

Figure 5.2: Initial constraints of the constraint programming checking subproblem.

Figure 5.2 presents the initial constraints, i.e., without the nogoods. Constraint (5.5) requires an arc entering every node i, i.e., ensures that i has at least one predecessor. Constraint (5.6) requires an arc exiting every node i, i.e., ensures that i has at least one successor. Constraint (5.7) states that every node h can be a predecessor to at most one other node. Similarly, Constraint (5.8) states that every node j can be a successor to at most one other node. Constraints (5.5) to (5.8) is a binarization of the AllDifferent global constraint, and together, the four constraints ensure that every request is visited exactly once. Constraint (5.9) is a global constraint, proposed by Pesant et al. (1998), that prevents subtours. Its propagator is a simple algorithm that prevents the head of a partial path from connecting to its tail. Constraints (5.10) and (5.11) enforce the vehicle capacity and travel time constraints.

Communication between the Two Models The two models communicate in three ways: (1) variable assignments in the constraint programming model are transmitted to the mixed integer programming model, (2) candidate solutions from the linear relaxation are probed using the constraint programming model to determine if they are valid for the VRPTW, and (3) nogoods found by conflict analysis in the constraint programming model are translated into cuts in the mixed integer programming model.

Extended Conflict Analysis When a failure occurs in the constraint programming solver, conflict analysis derives a First Unique Implication Point (1UIP) nogood that is added to the constraint programming subproblem. This constraint should also be added to the master problem but sometimes it cannot be translated into a cut for the master problem because it contains variables that do not appear in the master problem (i.e., the load and time variables). As a result, the BCE algorithm features an extended conflict analysis that continues explaining the failure until the nogood only contains variables in master problem. This nogood has the form

$$\bigvee_{(i,j)\in\mathcal{C}_1} y_{i,j} \vee \bigvee_{(i,j)\in\mathcal{C}_2} \neg y_{i,j},$$

where $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{A}$ are sets of arcs. This nogood can be rewritten as the cut

$$\sum_{(i,j)\in\mathcal{C}_1} x_{i,j} + \sum_{(i,j)\in\mathcal{C}_2} (1 - x_{i,j}) \geq 1.$$

It is always possible to obtain these cuts since the solver only branches on variables in the master problem. *Observe that the BCE algorithm provides a general-purpose mechanism to separate cuts in the master problem via the extended conflict analysis. These cuts, which we call MIP-1UIP nogoods, are automatically generated and do not rely on specialized separation algorithms.*

Probing the Linear Relaxation The BCE algorithm probes whether the current linear relaxation solution is feasible with respect to the subtour elimination, vehicle capacity and time constraints. It temporarily assigns every $y_{i,j}$ variable to the value of the corresponding $x_{i,j}$ variable in the linear relaxation, provided that this value is integral. The resulting tentative assignments can then be propagated by the constraint programming solver. If a failure occurs,

conflict analysis generates nogoods for both the constraint programming and mixed integer programming models. The mixed integer programming cut will exclude the current linear relaxation solution, forcing it to find another candidate solution, and improve the lower bound.

The Search Algorithm The BCE algorithm, detailed in Figure 5.3, includes the components described earlier. It blends depth-first and best-first search since best-first search is experimentally evaluated to be effective for many optimization problems (e.g., Achterberg 2007, Achterberg, Berthold et al. 2008, IBM 2015), but complicates the implementation of constraint programming solvers with conflict analysis, as discussed in Section 2.4.2. The node selection strategy selects the node with the lowest lower bound from the set of open nodes and then explores the node subtree using depth-first search until it reaches a limit on the maximum number of open nodes per subtree. Once it reaches this limit, all unsolved siblings in the subtree are moved into the set of open nodes, and then the algorithm starts a new depth-first search from the node with the next lowest lower bound. Section 5.4 explains the rationale behind this search procedure.

Once a node is selected (step 1), the constraint programming subproblem infers the implications of the decision (step 2). In the case of failure, the constraint programming solver generates nogoods for both models and then backtracks (step 5b). If the test succeeds, the BCE algorithm checks for suboptimality using the linear relaxation (step 3). If the node is suboptimal, it backtracks (step 5b). Otherwise, the BCE algorithm checks the linear relaxation solution against the omitted constraints and separates cuts using conflict analysis if necessary (step 4). The BCE algorithm iterates between the linear relaxation and the feasibility test until no cuts are generated. Then, if the node is fractional and not suboptimal, the BCE algorithm executes a branching step (step 5a). Two branching rules are implemented. The first selects the most fractional variable and the second selects the variable with the highest activity, which is defined as the number of nogoods in which the variable has previously appeared. This branching rule, known as activity-based search or variable state independent decaying sum (VSIDS), guides the search tree towards subtrees that can be quickly pruned due to infeasibility (Moskewicz et al. 2001). The activities mirror the statistics collected in pseudocost branching but for infeasibility instead of suboptimality (e.g., Achterberg, Koch and Martin 2005).

Illustrating the Extended Conflict Analysis The following discussion illustrates the extended conflict analysis procedure. Figure 5.4 shows an example of a network. Next to every request is its time window. The travel time across any arc is 10 units of time. Figure 5.5 depicts the implication graph for one run of the BCE algorithm. Literals shown in a grey are fixed by the data at the root level, and hence, are always true. They are discarded in the explanations but are shown for clarity.

The BCE solver first branches on $\neg y_{4,6}$, removing the arc $(4, 6)$. The travel time constraint, Constraint (5.11), propagates $[\![t_6 \geq 30]\!]$ with the reason

$$[\![t_3 \geq 25]\!] \wedge [\![c_{3,6} = 10]\!] \wedge \neg y_{4,6} \wedge [\![t_5 \geq 20]\!] \wedge [\![c_{5,6} = 10]\!] \longrightarrow [\![t_6 \geq 30]\!]$$

because the predecessor of request 6 must be either 3 or 5, and the earliest time to reach 6

1. **Node Selection:** Select an open node. Terminate if no open nodes remain.
2. **Feasibility Check:** Propagate the constraint programming model to determine the implications of the branching decision of the node. If the propagation fails, perform conflict analysis, add the 1UIP and the MIP-1UIP nogoods to the constraint programming and mixed integer programming models, and go to step 5b. Otherwise, fix $x_{i,j}$ in the mixed integer programming model to the values assigned to the $y_{i,j}$ variables.
3. **Suboptimality Check:** Solve the linear relaxation. If the objective value is worse than the incumbent solution, go to step 5b.
4. **Candidate Solution Probing:** For all $x_{i,j}$ variables with a value of 0 or 1 in the linear relaxation, temporarily fix the $y_{i,j}$ variables in the constraint programming model to the same value. Propagate the constraint programming model. If it fails, perform conflict analysis, generate the 1UIP and the MIP-1UIP nogoods and go back to step 3.
5. **Branching and Backtracking:** If all $x_{i,j}$ variables are integral, store the linear relaxation solution as the incumbent solution and go to step 5b. Otherwise, go to step 5a because the node is fractional.
   (a) **Branching:** Create two children nodes from a fractional $x_{i,j}$ variable. Fix the variable to 0 in one child node and to 1 in the other.
   (b) **Backtracking:** If the number of nodes in the current subtree exceeds the limit or if the subtree is entirely solved, move all unsolved siblings in the subtree to the set of open nodes and go back to step 1. Otherwise, backtrack to an ancestor with an unsolved child node, select the child node and go to step 2.

Figure 5.3: The branch-and-check with explanations search algorithm.

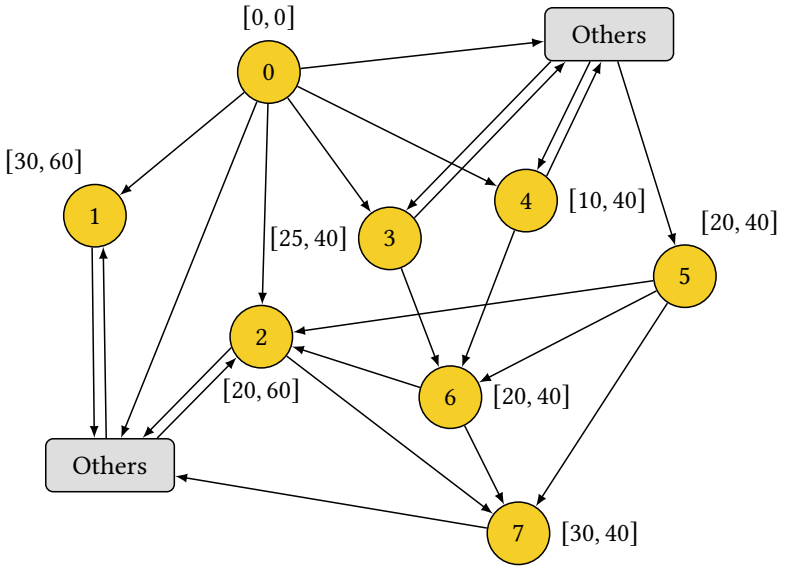


Figure 5.4: Example of a network. Next to every request is its time window. Travel across any arc incurs ten units of time. The load demands are not shown as they are not relevant to the discussion.

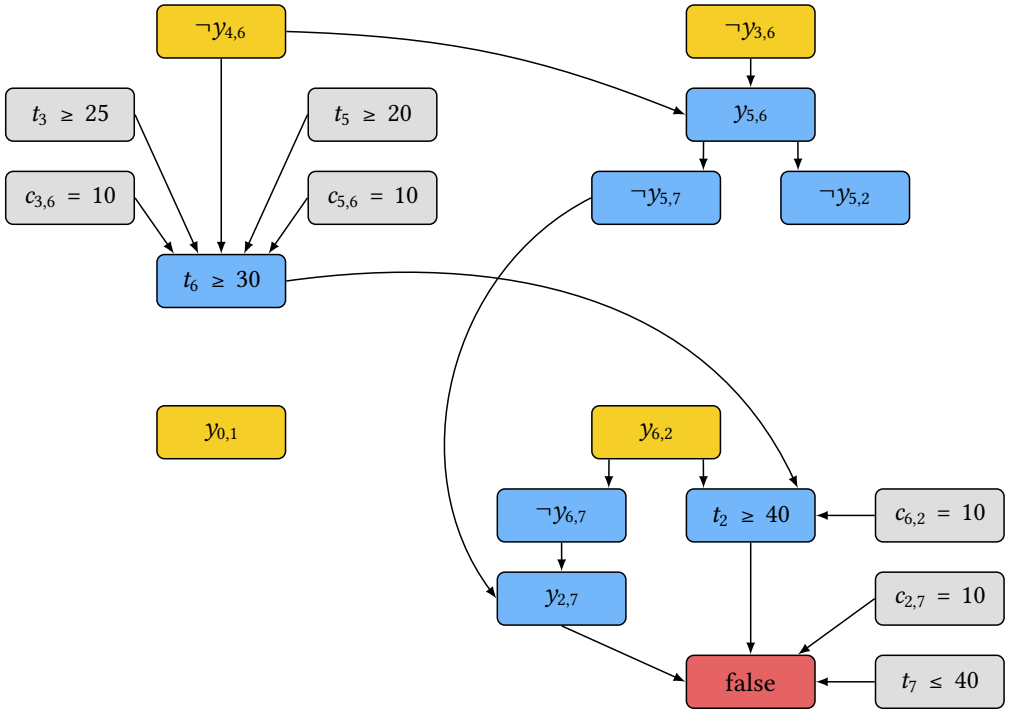


Figure 5.5: Example of an implication graph after making the decisions $\neg y_{4,6}$, $\neg y_{3,6}$, $y_{0,1}$ and $y_{6,2}$ on the network in Figure 5.4. Yellow literals are branching decisions. Blue literals are propagations. Grey literals are propagated at the root level, and hence, can be excluded from the nogoods since they are always true.

is at time $\min(\min(t_3) + c_{3,6}, \min(t_5) + c_{5,6}) = 30$. The BCE solver then branches on $\neg y_{3,6}$. Constraint (5.5) requires every request to have a predecessor, which leads to the assignment of $y_{5,6}$ with the reason

$$\neg y_{3,6} \wedge \neg y_{4,6} \longrightarrow y_{5,6}.$$

Constraint (5.8) then propagates

$$y_{5,6} \longrightarrow \neg y_{5,2}$$

and

$$y_{5,6} \longrightarrow \neg y_{5,7}.$$

The BCE solver then branches on $y_{0,1}$, which does not produce any inference, and then branches on $y_{6,2}$, which produces the following inferences:

$$y_{6,2} \longrightarrow \neg y_{6,7},$$

$$\neg y_{6,7} \wedge \neg y_{5,7} \longrightarrow y_{2,7},$$

$$y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!] \longrightarrow [\![t_2 \geq 40]\!].$$

Then, the travel time propagator fails with

$$y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] \longrightarrow \text{false}.$$

Conflict analysis deduces the following:

$$\begin{aligned}
y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] &\longrightarrow \text{false} \\
y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge \text{true} \wedge \text{true} &\longrightarrow \text{false} \\
(\neg y_{6,7} \wedge \neg y_{5,7}) \wedge (y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!]) &\longrightarrow \text{false} \\
y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] \wedge \text{true} &\longrightarrow \text{false} \\
y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] &\longrightarrow \text{false}. \qquad (5.12)
\end{aligned}$$

This explanation contains exactly one literal ($y_{6,2}$) at the current depth, and hence, is rewritten as the 1UIP clause

$$\neg y_{6,2} \vee y_{5,7} \vee [\![t_6 < 30]\!],$$

which is added to the constraint programming model. Conflict analysis must continue because the nogood contains a time literal. It explains $[\![t_6 \geq 30]\!]$ in Equation (5.12), which results in the MIP-1UIP explanation

$$y_{6,2} \wedge \neg y_{5,7} \wedge \neg y_{4,6} \longrightarrow \text{false}.$$

This explanation is rewritten into the disjunction

$$\neg y_{6,2} \vee y_{5,7} \vee y_{4,6}, \qquad (5.13)$$

and then into the cut

$$(1 - x_{6,2}) + x_{5,7} + x_{4,6} \geq 1.$$

Note that the literal $y_{5,7}$ was not assigned by the search.

5.2 Nogood Strengthening

The BCE algorithm presented so far uses a completely general-purpose mechanism for cut separation. Despite its generality, conflict analysis routinely discovers classical cuts. For instance, when the subtour elimination propagator fails, conflict analysis frequently derives a subtour elimination cut. These cuts can be strengthened using proven techniques whenever they are recognized. This section presents a post-processing step that recognizes then strengthens several types of cuts. In general, strengthening cuts is relatively simple but may have a significant impact on the quality of the lower bounds.

Infeasible Path Cuts Infeasible partial path cuts arise from the failure of the load or time constraints (Constraints (5.10) and (5.11)). Let $P = i_1, i_2, \dots, i_k$, with all $i_1, \dots, i_k \in \mathcal{N}$ distinct, be a partial path. The partial path P is infeasible with respect to the load constraint if $\sum_{u=1}^{k} q_{i_u} > Q$, and it is infeasible with respect to the time constraint if $t_{i_k} > b_{i_k}$, where $t_{i_1} = a_{i_1}$ and $t_{i_u} = \max(a_{i_u}, t_{i_{u-1}} + t_{i_{u-1},i_u})$ for $u = 2, \dots, k$. When a load or time window constraint fails, conflict analysis will usually produce the nogood

$$\bigvee_{(i,j)\in A(P)} \neg y_{i,j}, \tag{5.14}$$

where $A(P) = \{(i_1, i_2), \dots, (i_{k-1}, i_k)\}$ is the arcs of P. This nogood requires one arc of P to be unused. It can be stated equivalently as requiring at least one arc that exits P, i.e.,

$$\bigvee_{(i,j)\in\Delta^+(P)} y_{i,j},$$

where $\Delta^+(P) = \bigcup_{u=1}^{k-1}\{(i_u, j) \in \mathcal{A} | j \neq i_{u+1}\}$. This nogood can be translated into the cut

$$\sum_{(i,j)\in\Delta^+(P)} x_{i,j} \geq 1.$$

Kallehauge, Boland and Madsen (2007) showed that this cut can be strengthened into

$$\sum_{(i,j)\in\widetilde{\Delta}^+(P)} x_{i,j} \geq 1,$$

where

$$\widetilde{\Delta}^+(P) = \bigcup_{u=1}^{k-1}\Bigg(\Big\{(i_u, j) \in \mathcal{A} | i_u \in \mathcal{R}, j \in \mathcal{R}, j \neq i_1, \dots, i_{u+1}, \sum_{v=1}^{u} q_{i_v} + q_j \leq Q, t_{i_u} + t_{i_u,j} \leq b_j\Big\} \cup \Big\{(i_u, e) \in \mathcal{A}\Big\}\Bigg)$$

is the arcs that branch off P to a feasible request. In other words, the strengthening discards arcs that are not feasible when taking into account the load and time window constraints.

Subtour Elimination Cuts The propagator of Constraint (5.9) will fail if the solution contains a subtour $S = i_1, i_2, \dots, i_k$, where $i_1 = i_k$ and all $i_1, i_2, \dots, i_{k-1} \in \mathcal{R}$ are distinct. Conflict analysis will usually find the nogood

$$\bigvee_{(i,j)\in A(S)} \neg y_{i,j}, \tag{5.15}$$

where $A(S) = \{(i_1, i_2), \dots, (i_{k-1}, i_k)\}$ is the arcs of S. Using the same reasoning as for the infeasible path cuts, this nogood can be rewritten as the cut

$$\sum_{(i,j)\in\Delta^+(S)} x_{i,j} \geq 1. \tag{5.16}$$

If $a_j + c_{j,i} > b_i$, then no vehicle can depart j for i while respecting the time windows. Hence, i must precede j with respect to time, written as $i \prec j$. Let $\pi(j) = \{i \in \mathcal{N} | i \prec j\}$ be the set of requests that precedes j with respect to time and let $\sigma(i) = \{j \in \mathcal{N} | i \prec j\}$ be the set of requests that i precedes. Kallehauge, Boland and Madsen (2007) proved that Constraint (5.16) can be strengthened in two ways by reasoning about the precedence relations. Proposition 5.1 strengthens it to *weak predecessor cuts*, and Proposition 5.2 strengthens it to *weak successor cuts*.

Proposition 5.1. Let $\bar{S} = \mathcal{N} \setminus S$ be the nodes not in a subtour S, then for any $u \in S$, Constraint (5.16) can be strengthened to

$$\sum_{\substack{(i,j)\in\mathcal{A}:\\ i\in S\setminus\pi(u),\\ j\in\bar{S}\setminus\pi(u)}} x_{i,j} \geq 1. \tag{5.17}$$

Proof. Consider Figure 5.6, which shows a subtour S and a feasible path F that visits the request u. Let $v \in \mathcal{R}$ be the last request of F visited by S. By definition, v is visited by S, i.e., $v \in S$. Furthermore, since F is a feasible path, v cannot precede u with respect to time, i.e., $v \notin \pi(u)$. Hence, $v \in S \setminus \pi(u)$. Now consider the successor of v, denoted by $\text{succ}(v) \in \mathcal{N}$. By the definition of v, $\text{succ}(v)$ cannot be visited by S, i.e., $\text{succ}(v) \notin S$. Again, $\text{succ}(v)$ cannot precede u with respect to time since F is a feasible path. Hence, $\text{succ}(v) \in \bar{S} \setminus \pi(u)$. Considering every request in S as v results in the proposition. □

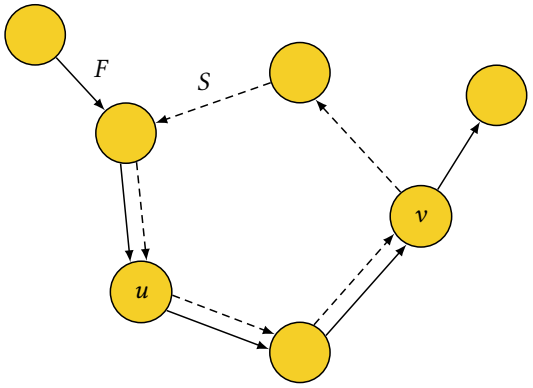


Figure 5.6: Example of a subtour S in dashed arrows and a feasible path F in solid arrows. The request u is visited by both S and F. The request v is the last request of F visited by S.

Proposition 5.2. Let $\bar{S} = \mathcal{N} \setminus S$ be the nodes not in a subtour S, then for any $u \in S$, Constraint (5.16) can be strengthened to

$$\sum_{\substack{(i,j)\in\mathcal{A}:\\ i\in S\setminus\sigma(u),\\ j\in\bar{S}\setminus\sigma(u)}} x_{i,j} \geq 1. \tag{5.18}$$

Proof. Similar to Proposition 5.1. □

General Cuts Conflict analysis can find cuts that are do not have the form of Constraint (5.14) nor Constraint (5.15). These cuts contain both true literals and false literals, such as those of Constraint (5.13). They originate from fixing an arc to be unused (i.e., setting $x_{i,j} = 0$ for some $(i,j) \in \mathcal{A}$), which can result in tightening the bounds of a time or load variable. Consequently, an assigned arc can become infeasible. Hence, the originating nogood will contain both true and false literals. We are not aware of vehicle routing cuts in the literature that mix true literals and false literals. This is possibly because tightening bounds is too costly for every call to a separation algorithm. Constraint programming maintains the bounds internally as part of propagation, and hence, the bounds are readily available. Because of this, these cuts seem to be fundamentally linked to constraint programming. It is an open research issue to understand whether these cuts can be strengthened.

5.3 Experimental Results

The Solvers The BCE algorithm includes a small constraint programming solver with nogood learning and calls Gurobi 6.5.2 to solve the linear relaxations. The algorithm presented in Figure 5.3 has a limit of 500 nodes for the depth-first search. This number was chosen experimentally as it was superior to limits of 100, 1,000, 5,000, and 10,000 nodes, as shown later. The experiments consider four versions of the BCE algorithm: with and without cut strengthening, and with the two branching rules presented earlier. In the strengthened cuts of Equations (5.17) and (5.18), the request u is repeated for all valid $u \in S$. The four versions of

the solver are compared against published results for the branch-and-cut model by Kallehauge, Boland and Madsen (2007), as well as a pure constraint programming model and a pure mixed integer programming model. The constraint programming model is the standard VRPTW model based on successor variables (e.g., Kilby, Prosser and Shaw 2000, Rousseau, Gendreau and Pesant 2002) and is solved using Chuffed. The mixed integer programming model is the three-index flow model (e.g., Vigo and Toth 2014) and is solved using Gurobi. The reported results for the branch-and-cut model are given one hour of execution time on an Intel Pentium III CPU at 600 MHz. For a fair comparison, our solvers are run for ten minutes on an Intel Xeon E5-2660 V3 CPU at 2.6 GHz. Additional results for shorter time-outs of 1 minute and 5 minutes are provided in Appendix A.

The Test Instances The solvers are tested on all the Solomon benchmarks with 100 requests (see Section 2.7). In order to compare BCE against results from the branch-and-cut publication, the instance data are rounded using the same procedure. Let (x'_i, y'_i) be the original coordinates of node i, and let q'_i, s'_i, a'_i and b'_i be the original load demand, service duration, earliest service start time and latest service start time of node i. Define $x_i = 10x'_i$ and $y_i = 10y'_i$, and then set $a_i = 10a'_i$, $b_i = 10b'_i$, $c_{i,j} = \lfloor\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\rfloor$ and $t_{i,j} = 10s'_i + c_{i,j}$. This procedure essentially rounds the distances, travel times, time windows and vehicle demands to one decimal place and the floored distance costs restore the triangle inequality.

The Results The results are reported in Table 5.3. The pure constraint programming model failed to find any feasible solution and is omitted from the table. The pure mixed integer programming model proves optimality on only one instance and finds poor-quality solutions to three other instances. These results are expected and are given to confirm the need for the other approaches. The rest of this section compares the branch-and-cut and BCE approaches.

Upper Bounds The four BCE methods are able to find the same or better solutions than the branch-and-cut algorithm for all instances except C204. Of the best solutions found, all but two (R201 and C204) can be found using the activity-based branching rule (with or without strengthening). For the C instances, branch-and-cut and BCE with activity-based search and cut strengthening are comparable since they both dominate on seven of the eight instances. For the R and RC instances, BCE with activity-based search improves upon the branch-and-cut method, which generally finds solutions with costs about five times higher.

Lower Bounds First observe that BCE with activity-based search and cut strengthening proves optimality on one more instance (RC201) than branch-and-cut, which is quite remarkable. The bounds found by the branch-and-cut model are superior to those from all BCE methods except for instance RC201, on which BCE with activity-based search and cut strengthening finds a tighter bound. This is not surprising since the branch-and-cut algorithm implements families of cuts not present in the BCE model. These families of cuts capture logic that the constraints in the checking subproblem do not. Stronger objective bounds should be available once the BCE model is expanded with global optimization constraints (to be discussed in the next section).

	Branch-and-Check – Most-Fractional						Branch-and-Check – Activity-based											
	No Strengthening			With Strengthening			No Strengthening			With Strengthening			Branch-and-Cut			Mixed Integer Program		
Instance	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time
R201	1055.8	1198.0	–	1117.7	**1143.3**	–	1054.7	1177.6	–	1114.3	1149.9	–	1132.7	1155.6	–	975.6	–	–
R202	762.8	1213.0	–	852.1	1219.6	–	763.2	1133.4	–	850.7	**1109.3**	–	888.6	4980.0	–	715.3	–	–
R203	660.1	1244.8	–	709.8	1253.6	–	659.9	**1025.2**	–	707.7	1052.2	–	748.1	4980.0	–	620.3	–	–
R204	625.3	1166.7	–	639.2	1193.3	–	625.8	**858.4**	–	638.3	887.4	–	661.9	4980.0	–	584.9	–	–
R205	796.3	1222.0	–	889.6	1069.9	–	794.3	1091.3	–	876.9	**1052.5**	–	900.0	4980.0	–	732.3	–	–
R206	686.3	1171.6	–	751.4	1157.4	–	686.0	1040.1	–	745.3	**1018.9**	–	783.6	4980.0	–	644.8	–	–
R207	648.1	1187.5	–	681.5	1168.5	–	647.5	**940.7**	–	685.8	941.4	–	714.8	4980.0	–	603.1	–	–
R208	623.2	1097.4	–	633.5	1187.7	–	623.4	855.0	–	635.3	**832.5**	–	651.8	4980.0	–	577.2	–	–
R209	687.5	1238.1	–	756.6	1172.5	–	686.5	**1046.6**	–	753.1	1073.8	–	785.8	4980.0	–	648.2	–	–
R210	679.7	1225.6	–	749.9	1240.3	–	679.8	1105.6	–	750.8	**1024.9**	–	798.3	4980.0	–	636.6	–	–
R211	621.2	1335.5	–	633.0	1355.9	–	621.2	**1004.2**	–	632.1	1065.1	–	645.1	4980.0	–	577.2	4224.9	–
C201	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	11.5	589.1	**589.1**	15.2
C202	548.7	679.8	–	589.1	**589.1**	131.2	548.2	629.9	–	589.1	**589.1**	12.6	589.1	**589.1**	202.9	524.3	–	–
C203	526.5	948.3	–	563.4	672.2	–	524.7	686.5	–	565.9	**601.2**	–	586.0	632.3	–	507.3	–	–
C204	516.3	946.7	–	552.9	1086.7	–	514.7	884.5	–	555.9	660.9	–	584.4	**597.1**	–	488.3	–	–
C205	546.9	685.8	–	586.4	**586.4**	0.2	546.5	613.1	–	586.4	**586.4**	16.3	586.4	**586.4**	334.4	511.4	–	–
C206	539.9	776.9	–	586.0	**586.0**	11.8	538.2	702.6	–	586.0	**586.0**	10.6	586.0	**586.0**	419.0	504.7	4997.5	–
C207	542.7	851.1	–	585.8	**585.8**	20.6	538.3	635.2	–	585.8	**585.8**	8.3	585.8	**585.8**	527.5	503.9	–	–
C208	534.5	857.2	–	585.8	**585.8**	60.0	533.1	652.4	–	585.8	**585.8**	11.2	585.8	**585.8**	569.7	500.3	–	–
RC201	1086.5	1403.6	–	1245.8	**1261.8**	–	1081.0	1338.3	–	1261.8	**1261.8**	44.5	1250.1	1288.2	–	938.3	–	–
RC202	704.5	1465.9	–	912.7	1418.6	–	699.2	1204.2	–	916.9	**1152.3**	–	940.1	6609.4	–	641.0	–	–
RC203	615.0	1402.7	–	750.2	1359.6	–	610.8	1149.0	–	748.8	**1117.6**	–	781.6	6609.4	–	563.1	–	–
RC204	583.9	1410.2	–	657.8	1352.4	–	581.0	1007.1	–	657.0	**923.5**	–	692.7	6609.4	–	532.4	–	–
RC205	822.5	1511.6	–	1075.5	1307.0	–	818.8	1249.8	–	1055.8	**1240.9**	–	1081.7	6609.4	–	746.3	–	–
RC206	785.4	1485.4	–	964.3	1273.9	–	784.9	1270.2	–	950.7	**1202.8**	–	974.8	6609.4	–	698.2	–	–
RC207	647.3	1486.3	–	794.6	1424.5	–	642.9	1193.5	–	800.9	**1172.1**	–	832.4	6609.4	–	594.9	–	–
RC208	572.7	1629.8	–	624.0	1776.1	–	573.9	**1039.5**	–	624.3	1078.4	–	647.7	6609.4	–	527.1	5299.5	–

Table 5.3: Solutions to the Solomon instances with 100 requests. The table reports the lower bound, upper bound and time to prove optimality for each of the solvers. The best upper bound for each instance is shown in bold. The constraint programming model is omitted as it is unable to find feasible solutions to any instance.

The Number of Cuts Table 5.4 presents the number of cuts found in the version of the solver that uses activity-based branching with cut strengthening. Cuts that have the form of Constraints (5.14) and (5.15) are respectively labeled infeasible path and subtour cuts. These two families of cuts can be strengthened according to the methods presented previously. The remaining cuts have neither of these two forms and are directly added to the linear relaxation. On average, 12% of the cuts, and on one occasion, 40% of the cuts, are these general cuts, which are believed to be previously unconsidered in the operations research literature.

The Impact of Branching Rules Activity-based branching performs significantly better than most-fractional branching. Without cut strengthening, activity-based branching finds solutions better than most-fractional branching on all instances except C201, on which all four BCE methods prove optimality. With cut strengthening, activity-based search performs better on 19 of the 27 instances and worse on only one instance.

The Impact of Cut Strengthening Cut strengthening improves the lower bounds for both branching rules. For the C instances, cut strengthening is critical for proving optimality. For the RC instances except RC208, BCE with activity-based branching and cut strengthening finds solutions better than the other methods. Cut strengthening interferes with the activity-based branching rule for about half of the R instances. The cause of this interference is not yet understood.

The Impact of Best-first Node Selection on the Upper Bounds A higher frequency of best-first node selection is used to emulate true best-first search. Frequently performing best-first node selection is expected to perform better in hard optimization problems since the solver can explore disparate parts of the search tree. However, this must be balanced with exploring deep into the tree because conflict analysis naturally depends on depth-first search, as explained previously in Section 2.4.2. Table 5.5 lists the upper bounds found by several versions of the solver that vary the number of nodes solved using depth-first search before executing one best-first node selection. The table shows that solving 500 nodes using depth-first search then choosing the next node using best-first search better balances optimization and nogood learning by finding the best solution to nine instances.

The Impact of Best-first Node Selection on the Lower Bounds Best-first search favors nodes with the least lower bound first, and as such, is expected to find better lower bounds. Table 5.6 lists the lower bounds found by the same versions of the solver mentioned above. Coincidentally, executing best-first node selection after every 500 nodes performs better than after every 100 nodes. With this parameter, the solver finds the best lower bound to 13 instances. This is probably due to inefficiencies of the implementation, which incurs significant overhead during each restart (e.g., recreating all variables, literals and constraints).

The results indicate that BCE is an interesting avenue for solving hard vehicle routing problems. BCE finds superior primal solutions despite its simplicity and the fact that it ignores

many families of cuts and that the checking subproblem does not reason about optimality nor variables with fractional values in the linear relaxation solution. For practitioners without the expertise in branch-and-cut, BCE provides an interesting and practically appealing alternative.

5.4 Future Research Directions

The BCE algorithm, presented here as a proof-of-concept, can be improved in many ways. This section explores some potential improvements.

Branching The branching rules implemented are extremely simple: assign a fractional variable to 0 in one child and to 1 in the other. This branching rule makes the search tree highly unbalanced, considerably degrading the performance of the solver. Furthermore, always branching on the most fractional variable is known to perform worse than randomly selecting a fractional variable (Achterberg, Koch and Martin 2005). However, this branching rule was tested in order to maintain a deterministic solver. Future implementations should test branching on cutsets, which is a branching rule commonly seen in vehicle routing problems (e.g., Lysgaard, Letchford and Eglese 2004, Naddef and Rinaldi 2002). It would also be interesting to test branching on variables in the constraint programming model (e.g., branching on time windows) by propagating these decisions and enforcing the implications in the mixed integer programming model.

Search Strategy Tables 5.5 and 5.6 show that the VRPTW greatly benefits from best-first search. For simplicity, the BCE implementation uses depth-first search, which allows literals and explanations to be stored in a stack data structure. It is obviously possible to implement conflict analysis in best-first search but an efficient implementation is non-trivial and is an open research question in itself. As explained in Section 5.1, the search strategy of the BCE implementation blends depth-first search with periodic best-first selection to explore attractive parts of the search tree.

Subtour Elimination The propagator of Constraint (5.9) is extremely simple and only eliminates assignments that would create a cycle. This contrasts with separation algorithms, which are able to separate cuts using fractional solutions. It would be highly desirable to study the impact of more advanced propagators and explanations for subtour elimination in constraint programming.

Global Optimization Constraints The objective function has been omitted from the checking subproblem because propagators for linear functions are known to be weak. Sophisticated propagators for the WeightedCircuit constraint (Benchimol et al. 2012) should be implemented, as they may produce considerably stronger nogoods.

Application to Branch-and-Price As discussed earlier in Section 2.3.4, branch-and-cut-and-price is the current state-of-the-art exact method for solving classical vehicle routing problems

	Number of Cuts			Percentage		
Instance	Infeasible Path	Subtour	Other	Infeasible Path	Subtour	Other
R201	112	70	15	57%	36%	8%
R202	671	189	104	70%	20%	11%
R203	933	285	199	66%	20%	14%
R204	910	375	213	61%	25%	14%
R205	606	237	97	64%	25%	10%
R206	707	246	95	67%	23%	9%
R207	741	306	318	54%	22%	23%
R208	639	372	265	50%	29%	21%
R209	739	230	223	62%	19%	19%
R210	652	232	133	64%	23%	13%
R211	1281	387	178	69%	21%	10%
C201	0	0	0	–	–	–
C202	86	53	0	62%	38%	0%
C203	494	200	29	68%	28%	4%
C204	588	337	274	49%	28%	23%
C205	141	55	1	72%	28%	1%
C206	128	67	1	65%	34%	1%
C207	10	76	0	12%	88%	0%
C208	33	70	0	32%	68%	0%
RC201	85	76	6	51%	46%	4%
RC202	449	159	81	65%	23%	12%
RC203	789	284	197	62%	22%	16%
RC204	670	311	647	41%	19%	40%
RC205	448	155	46	69%	24%	7%
RC206	711	207	91	70%	21%	9%
RC207	666	200	139	66%	20%	14%
RC208	1733	456	490	65%	17%	18%

Table 5.4: The number of cuts found and the proportion of each family of cuts in the solver that uses activity-based branching with cut strengthening.

Instance	100	500	1000	5000	10000
R201	1153.7	1149.9	1145.5	**1143.6**	1145.4
R202	1110.1	**1109.3**	1143.9	1120.4	1136.7
R203	1055.8	1052.2	**1019.5**	1059.9	1053.3
R204	896.0	**887.4**	888.7	921.9	948.9
R205	1068.5	1052.5	**1041.4**	1045.0	1058.8
R206	1044.1	1018.9	**1010.7**	1022.5	1051.0
R207	959.2	**941.4**	1011.5	993.1	1008.0
R208	**814.7**	832.5	848.4	876.7	820.9
R209	1079.8	1073.8	1019.6	1015.9	**1008.3**
R210	**1022.9**	1024.9	1064.7	1037.8	1033.2
R211	1121.6	**1065.1**	1107.1	1097.0	1135.4
C201	589.1	589.1	589.1	589.1	589.1
C202	589.1	589.1	589.1	589.1	589.1
C203	631.2	**601.2**	648.5	668.0	680.8
C204	**648.1**	660.9	655.2	654.9	719.8
C205	**586.4**	**586.4**	**586.4**	**586.4**	608.8
C206	586.0	586.0	586.0	586.0	586.0
C207	585.8	585.8	585.8	585.8	585.8
C208	585.8	585.8	585.8	585.8	585.8
RC201	1269.2	**1261.8**	**1261.8**	**1261.8**	**1261.8**
RC202	1182.0	**1152.3**	1187.7	1195.1	1219.6
RC203	**1073.8**	1117.6	1084.6	1130.3	1173.0
RC204	897.9	923.5	**891.3**	937.0	937.0
RC205	1272.6	1240.9	1230.5	**1218.0**	1219.7
RC206	1256.9	1202.8	1231.4	**1154.4**	1222.9
RC207	1182.4	1172.1	**1168.0**	1169.7	1183.3
RC208	1153.7	**1078.4**	1217.4	1146.4	1158.0

Table 5.5: Upper bounds from variants of the solver with different numbers of nodes solved using depth-first search before performing best-first node selection. Best to worst solutions in each row are colored from green to red.

Instance	100	500	1000	5000	10000
R201	1115.2	1114.3	1115.2	**1116.1**	1114.7
R202	848.1	**850.7**	843.4	810.1	803.8
R203	707.1	**707.7**	703.6	688.6	683.3
R204	636.4	**638.3**	637.9	633.5	631.3
R205	**878.5**	876.9	872.0	854.5	844.0
R206	**745.9**	745.3	743.2	726.3	719.3
R207	682.8	**685.8**	682.8	669.0	664.3
R208	**635.3**	**635.3**	634.2	629.3	627.8
R209	**755.7**	753.1	742.0	718.9	703.8
R210	**753.7**	750.8	740.5	725.9	722.4
R211	**632.6**	632.1	631.9	630.0	629.2
C201	589.1	589.1	589.1	589.1	589.1
C202	589.1	589.1	589.1	589.1	589.1
C203	**570.6**	565.9	567.5	550.3	547.2
C204	550.3	**555.9**	552.2	531.2	526.7
C205	**586.4**	**586.4**	**586.4**	**586.4**	585.6
C206	586.0	586.0	586.0	586.0	586.0
C207	585.8	585.8	585.8	585.8	585.8
C208	585.8	585.8	585.8	585.8	585.8
RC201	1242.9	**1261.8**	**1261.8**	**1261.8**	**1261.8**
RC202	912.7	**916.9**	893.6	862.0	848.7
RC203	**757.7**	748.8	739.1	691.1	670.7
RC204	653.3	**657.0**	649.7	624.9	618.8
RC205	1051.2	**1055.8**	1052.6	985.3	978.1
RC206	941.5	950.7	**951.2**	905.8	892.3
RC207	797.7	**800.9**	792.9	762.6	742.4
RC208	623.9	**624.3**	623.7	611.6	610.8

Table 5.6: Lower bounds from variants of the solver with different numbers of nodes solved using depth-first search before performing best-first node selection. Best to worst bounds in each row are colored from green to red.

by incorporating both column generation and cut generation. Preliminary experiments with an existing branch-and-price solver show that BCE is not beneficial with column generation for the VRPTW as nogoods will not be generated in step 4 of Figure 5.3 because this step will not fail since the paths already respect the time and capacity constraints. However, step 2 can fail due to incompatibility between the branching decisions of a node. This infeasibility cannot be detected by the pricing problem because it has no knowledge of the global problem, nor detected by the master problem until all paths are generated because artificial variables satisfy the constraints in the interim. Incompatible branching decisions can induce nogoods but this seldom occurs in branch-and-price because its linear relaxation bound is asymptotically tight, allowing it to discard nodes due to suboptimality much earlier than infeasibility. Hence, branch-and-price-and-check is unlikely to prove useful in solving classical vehicle routing problems. It is, however, useful for rich vehicle routing problems with inter-route constraints as shown in the previous chapter. This is because inter-route constraints are difficult to consider within the pricing subproblem because the pricing subproblem, being a shortest path problem, has no knowledge of the interactions between routes in the parent problem.

Automatic Cut Strengthening The constraint programming model contains all the omitted constraints; namely, the subtour elimination, vehicle capacity and time window constraints. As a result, conflict analysis can deduce nogoods based on the combined infeasibility of multiple constraints. In contrast, separation algorithms only reason about one family of cuts. It is an open question whether conflict analysis can strengthen the cuts from reasoning about a conjunction of constraints. Automatically deducing strengthened cuts will be highly advantageous as it will enable strengthenings to be declared in a model instead of manually implemented in a solver.

Automatic Dantzig-Wolfe Decomposition A long-sought goal of combinatorial optimization is to let a user declare a model then leave its solution to a software system that selects an appropriate solver. A full unification of BCE with automatic Dantzig-Wolfe decomposition (Bergner et al. 2015, Puchinger et al. 2011) and automatic cut strengthening will be a major breakthrough since a software system will be able to detect problem structures and seamlessly apply Benders decomposition and Dantzig-Wolfe decomposition to construct stronger branch-and-cut-and-price models. Considering that many state-of-the-art problem-specific solvers are based on branch-and-cut-and-price, this reformulation can potentially rival the latest advances.

5.5 Conclusion

This chapter proposed the framework of branch-and-check with explanations (BCE) as a step towards unifying linear programming, constraint programming and Boolean satisfiability. BCE finds cuts using general-purpose conflict analysis instead of specialized separation algorithms. The method features a master problem, which ignores a number of constraints, and a checking subproblem, which uses inference to check the feasibility of the omitted constraints and conflict analysis to derive nogood cuts. It also leverages conflict-based branching rules and can

strengthen cuts using traditional insights from branch-and-cut in a post-processing step.

Experimental results on the Vehicle Routing Problem with Time Windows show that BCE is a viable alternative to branch-and-cut. In particular, BCE dominates branch-and-cut, both in proving optimality (with cut strengthening) and in finding high-quality solutions.

BCE offers an interesting alternative to existing branch-and-cut approaches. By using a general-purpose constraint programming solver to derive cuts, BCE can greatly simplify the modeling of problems that traditionally use branch-and-cut. This, in turn, avoids the need for dedicated separation algorithms. BCE is also capable of identifying well-known classes of cuts and strengthening them in a post-processing step. Finally, BCE significantly benefits from conflict-based branching rules, opening further opportunities typically not available in branch-and-cut.

Chapter 6

Conclusion

This chapter summarizes the main findings previously presented in this thesis and provides several recommendations for future research.

Introduction Chapter 1 introduced vehicle routing problems and the field of combinatorial optimization. It laid out two combinatorial optimization technologies for modeling and solving vehicle routing problems; namely, mixed integer programming and constraint programming. It outlined their weaknesses and proposed hybridization for mitigating these weaknesses.

Background Chapter 2 reviewed background material to the main chapters of this thesis. It formalized linear programming and generalized it to mixed integer programming, and then considered Boolean satisfiability and extended it to constraint programming. It also presented four basic vehicle routing problems.

Joint Vehicle and Crew Routing and Scheduling Problem Chapter 3 saw the development of the Joint Vehicle and Crew Routing and Scheduling Problem (JVCRSP). In the JVCRSP, crews must drive vehicles to service requests at a number of locations and crews have a limit on their driving duration. Crews are able to interchange vehicles at all locations, making the vehicle routes and crew routes tightly coupled since two vehicles must be present at a location for a crew to switch vehicles. The problem involves simultaneously routing and scheduling vehicles and crews, which are traditionally solved in stages. A mixed integer programming model and a constraint programming model were developed for the JVCRSP. The constraint programming model included a global optimization constraint that partially calculates objective bounds using a linear relaxation. The results showed that executing a custom-built large neighborhood search on the constraint programming model performed better than the other approaches.

A shortcoming of the constraint programming model is the crew maximum driving duration constraint, which calculates the maximum driving duration by subtracting the start time of a crew from its end time. This is especially ineffective as the start and end times are available only at the final stages of the search when the routes are almost fully specified.

Both the mixed integer programming and constraint programming model can be further improved with better neighborhoods. The four existing neighborhoods isolated vehicle object-

ives and crew objectives. Since the vehicles and crews are interdependent, implementing a neighborhood that simultaneously reasoned about vehicle and crew objectives and constraints should have a significant impact on the solutions.

The Vehicle Routing Problem with Location Congestion Chapter 4 presented the Vehicle Routing Problem with Location Congestion (VRPLC). In addition to designing minimal-cost routes, the problem involves scheduling vehicles around the availabilities of a limited number of resources. The scheduling interferes with the routing, making the two aspects highly interdependent. The problem was solved using a branch-and-price-and-check method, which uses column generation to find high-quality routes that disregard the scheduling requirements, and a constraint programming subproblem to verify whether the routes adhere to the scheduling constraints. The branch-and-price-and-check method was compared against a pure mixed integer programming model, a pure constraint programming model and a two-stage sequential model. The results indicated that the hybrid branch-and-price-and-check method performed better than the other three approaches.

A major fault of the branch-and-price-and-check model is its inability to translate temporal nogoods in the checking subproblem to arc nogoods in the master problem. Currently, the scheduling constraints are completely absent in the master problem; they are enforced by forbidding combinations of arcs. Hence, all routes must be rejected by cuts in order to prove infeasibility. Equipping the master problem with a relaxation of the scheduling constraints would enable it to prove infeasibility without an exponential number of cuts. Whether this is possible is presently not known.

Branch-and-Check with Explanations Chapter 5 formally develops the preliminary ideas from the previous chapter into an exact method named branch-and-check with explanations (BCE). The BCE method unifies mixed integer programming, constraint programming and Boolean satisfiability. It begins with a linear relaxation that omits difficult constraints. Solving the linear relaxation produces an objective bound and a linearly feasible solution. The solution is checked in the constraint programming subproblem for the feasibility of the omitted constraints. If the solution is infeasible, the solution is passed through a Boolean satisfiability solver, which performs conflict analysis to learn nogoods. The nogoods are added as constraints in the linear relaxation and in the constraint programming subproblem. The method was assessed on the Vehicle Routing Problem with Time Windows, and the results demonstrated that this fully hybrid method performed better than an existing branch-and-cut method.

An obvious avenue for future research is to the develop the BCE method into a general black-box solver. The method is currently implemented in a proof-of-concept solver custom-tailored to the Vehicle Routing Problem with Time Windows. The implementation is inefficient and does not make use of recent advances in constraint programming and Boolean satisfiability solvers. Building a generic solver would enable the BCE method to be evaluated on problems traditionally avoided by the constraint programming community.

A thorough investigation into compromises between depth-first search and best-first search

is also warranted. The impact of tradeoffs between depth-first search and best-first search are significant since the results improved when the node selection strategy moved away from depth-first search towards best-first search. However, conflict analysis is most practical with depth-first search because the nogoods can only be built by exploring deep into the search tree. Instead of implementing conflict analysis with best-first search, it may be possible to use other node selection rules that emulate best-first search. For example, this can include pausing the linear relaxation and then diving deeper into the search tree with constraint programming.

Integrating BCE with automatic Dantzig-Wolfe decomposition and automatic cut strengthening will be a major advance for the field. Currently, both branch-and-cut-and-price and cut strengthening must be manually implemented as they exploit problem-specific knowledge and combinatorial substructures. Having an automatic system for detecting problem structures and creating branch-and-cut-and-price reformulations can enable models to be stated and solved efficiently using the latest methods in optimization.

The Future This dissertation presented several advances towards the grand unification of mixed integer programming, constraint programming and Boolean satisfiability. Even though the hybrid techniques were developed specifically for vehicle routing problems, they are general in the sense that they are applicable to many other problems. It would be interesting to see these techniques applied in other problem domains, and particularly, in generic black-box solvers.

Bibliography

Achterberg, T. (2007). 'Constraint Integer Programming'. PhD thesis. Technische Universität Berlin.

Achterberg, T., T. Berthold et al. (2008). 'Constraint Integer Programming: A New Approach to Integrate CP and MIP'. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 5th International Conference, CPAIOR 2008 Paris, France, May 20-23, 2008 Proceedings*. Ed. by L. Perron and M. A. Trick. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 6–20.

Achterberg, T., T. Koch and A. Martin (2005). 'Branching rules revisited'. In: *Operations Research Letters* 33.1, pp. 42–54.

Aggoun, A. and N. Beldiceanu (1993). 'Extending chip in order to solve complex scheduling and placement problems'. In: *Mathematical and Computer Modelling* 17.7, pp. 57–73.

Applegate, D. et al. (2003). 'Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems'. In: *Mathematical programming* 97.1-2, pp. 91–153.

Backer, B. et al. (2000). 'Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics'. In: *Journal of Heuristics* 6.4, pp. 501–523.

Baldacci, R., N. Christofides and A. Mingozzi (2008). 'An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts'. In: *Mathematical Programming* 115.2, pp. 351–385.

Baldacci, R., A. Mingozzi and R. Roberti (2011). 'New route relaxation and pricing strategies for the vehicle routing problem'. In: *Operations Research* 59.5, pp. 1269–1283.

Bard, J. F., G. Kontoravdis and G. Yu (2002). 'A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows'. In: *Transportation Science* 36.2, pp. 250–269.

Barnhart, C., E. L. Johnson et al. (1998). 'Branch-and-Price: Column Generation for Solving Huge Integer Programs'. In: *Operations Research* 46.3, pp. 316–329.

Barnhart, C., F. Lu and R. Shenoi (1998). 'Integrated Airline Schedule Planning'. In: ed. by G. Yu. Operations Research in the Airline Industry. Springer US. Chap. 13, pp. 384–403.

Beck, J. C. (2010). 'Checking-Up on Branch-and-Check'. In: *Principles and Practice of Constraint Programming – CP 2010*. Ed. by D. Cohen. Vol. 6308. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 84–98.

Beck, J. C., P. Prosser and E. Selensky (2002). 'On the Reformulation of Vehicle Routing Problems and Scheduling Problems'. In: *Abstraction, Reformulation, and Approximation*. Ed. by

S. Koenig and R. Holte. Vol. 2371. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 282–289.

Beck, J. C., P. Prosser and E. Selensky (2003). 'Vehicle Routing and Job Shop Scheduling: What's the Difference?' In: *Proceedings of the Thirteenth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS'03. AAAI Press, pp. 267–276.

Bellmore, M. and J. C. Malone (1971). 'Pathology of Traveling-Salesman Subtour-Elimination Algorithms'. In: *Operations Research* 19.2, pp. 278–307.

Benchimol, P. et al. (2012). 'Improved filtering for weighted circuit constraints'. In: *Constraints* 17.3, pp. 205–233.

Benders, J. F. (1962). 'Partitioning procedures for solving mixed-variables programming problems'. In: *Numerische mathematik* 4.1, pp. 238–252.

Benson, H. Y. (2011). 'Interior-Point Linear Programming Solvers'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Bent, R. and P. Van Hentenryck (2004). 'A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows'. In: *Transportation Science* 38.4, pp. 515–530.

Bent, R. and P. Van Hentenryck (2006). 'A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows'. In: *Computers & Operations Research* 33.4, pp. 875–893.

Bergman, D., A. A. Cire and W.-J. van Hoeve (2015). 'Improved Constraint Propagation via Lagrangian Decomposition'. In: *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31 – September 4, 2015, Proceedings*. Ed. by G. Pesant. Springer International Publishing, pp. 30–38.

Bergner, M. et al. (2015). 'Automatic Dantzig–Wolfe reformulation of mixed integer programs'. In: *Mathematical Programming* 149.1, pp. 391–424.

Bessiere, C. (2010). 'Basic CP Theory: Consistency And Propagation (Advanced)'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Bixby, R. E. (2002). 'Solving real-world linear programs: a decade and more of progress'. In: *Operations Research* 50.1, pp. 3–15.

Bramel, J. and D. Simchi-Levi (1997). 'On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows'. In: 45.2, pp. 295–301.

Bräysy, O. and G. Hasle (2014). 'Software Tools and Emerging Technologies for Vehicle Routing and Intermodal Transportation'. In: ed. by D. Vigo and P. Toth. Second Edition. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics. Chap. 12, pp. 351–380.

Carpaneto, G. and P. Toth (1980). 'Some New Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem'. In: *Management Science* 7, p. 736.

Castro, P. M., I. E. Grossmann and L.-M. Rousseau (2011). 'Decomposition Techniques for Hybrid MILP/CP Models applied to Scheduling and Routing Problems'. In: *Hybrid Optimization*. Ed. by P. Van Hentenryck and M. Milano. Vol. 45. Springer Optimization and Its Applications. Springer New York, pp. 135–167.

Christofides, N. and S. Eilon (1969). 'An Algorithm for the Vehicle-dispatching Problem'. In: *Journal of the Operational Research Society* 20.3, pp. 309–318.

Chung, W. (2010). 'Dantzig–Wolfe Decomposition'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Cochran, J. J. (2010). 'An Introduction to Linear Programming'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Cordeau, J.-F. et al. (2001). 'Benders Decomposition for Simultaneous Aircraft Routing and Crew Scheduling'. In: *Transportation Science* 35.4, pp. 375–388.

Dantzig, G. B. and J. Ramser (1959). 'The Truck Dispatching Problem'. In: *Management Science* 6.1, p. 80.

Dantzig, G. B. and P. Wolfe (1960). 'Decomposition Principle for Linear Programs'. In: *Operations Research* 8.1, pp. 101–111.

Dantzig, G. B., R. Fulkerson and S. Johnson (1954). 'Solution of a large-scale traveling-salesman problem'. In: *Journal of the Operations Research Society of America* 2.4, pp. 393–410.

Davis, M., G. Logemann and D. Loveland (1962). 'A Machine Program for Theorem-proving'. In: *Communications of the ACM* 5.7, pp. 394–397.

Davis, M. and H. Putnam (1960). 'A Computing Procedure for Quantification Theory'. In: *Journal of the ACM* 7.3, pp. 201–215.

Deo, N. and C.-Y. Pang (1984). 'Shortest-Path Algorithms: Taxonomy and Annotation'. In: *Networks* 14.2, pp. 275–323.

Desaulniers, G., J. Desrosiers and M. M. Solomon (2005). *Column Generation*. Springer US.

Desaulniers, G., O. B. Madsen and S. Røpke (2014). 'The Vehicle Routing Problem with Time Windows'. In: ed. by D. Vigo and P. Toth. Second Edition. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics. Chap. 5, pp. 119–159.

Desrochers, M., J. Desrosiers and M. Solomon (1992). 'A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows'. In: *Operations Research* 40.2, pp. 342–354.

Desrochers, M. and G. Laporte (1991). 'Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints'. In: *Operations Research Letters* 10.1, pp. 27–36.

Desrosiers, J. and M. E. Lübbecke (2010). 'Branch-Price-and-Cut Algorithms'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Desrosiers, J., F. Soumis and M. Desrochers (1984). 'Routing with time windows by column generation'. In: *Networks* 14.4, pp. 545–565.

Drexl, M. (2007). 'On some generalized routing problems'. PhD thesis. RWTH Aachen University.

Drexl, M. (2012). 'Synchronization in Vehicle Routing—A Survey of VRPs with Multiple Synchronization Constraints'. In: *Transportation Science* 46.3, pp. 297–316.

Drexl, M. (2013). 'Applications of the vehicle routing problem with trailers and transshipments'. In: *European Journal of Operational Research* 227.2, pp. 275–283.

Drexl, M. (2014). 'Branch-and-cut algorithms for the vehicle routing problem with trailers and transshipments'. In: *Networks* 63.1, pp. 119–133.

Drexl, M. et al. (2013). 'Simultaneous Vehicle and Crew Routing and Scheduling for Partial- and Full-Load Long-Distance Road Transport'. In: *BuR - Business Research* 6.2, pp. 242–264.

Dreyfus, S. E. (1969). 'An Appraisal of Some Shortest-Path Algorithms'. In: *Operations Research* 17.3, pp. 395–412.

Dumas, Y., J. Desrosiers and F. Soumis (1991). 'The pickup and delivery problem with time windows'. In: *European Journal of Operational Research* 54.1, pp. 7–22.

Eén, N. and N. Sörensson (2004). 'An Extensible SAT-solver'. In: *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers*. Ed. by E. Giunchiglia and A. Tacchella. Springer Berlin Heidelberg, pp. 502–518.

El Hachemi, N., M. Gendreau and L.-M. Rousseau (2013). 'A heuristic to solve the synchronized log-truck scheduling problem'. In: *Computers & Operations Research* 40.3, pp. 666–673.

Feillet, D. (2010). 'A tutorial on column generation and branch-and-price for vehicle routing problems'. In: *4OR: A Quarterly Journal of Operations Research* 8.4, pp. 407–424.

Feillet, D., M. Gendreau and L.-M. Rousseau (2007). 'New Refinements for the Solution of Vehicle Routing Problems with Branch and Price'. In: *INFOR: Information Systems and Operational Research* 45.4, pp. 239–256.

Feydy, T. and P. J. Stuckey (2009). 'Lazy Clause Generation Reengineered'. In: *Principles and Practice of Constraint Programming – CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings*. Ed. by I. P. Gent. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 352–366.

Focacci, F., A. Lodi and M. Milano (1999). 'Cost-Based Domain Filtering'. In: *Principles and Practice of Constraint Programming – CP'99: 5th International Conference, CP'99, Alexandria, VA, USA, October 11-14, 1999. Proceedings*. Ed. by J. Jaffar. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 189–203.

Focacci, F., A. Lodi and M. Milano (2000). 'Cutting Planes in Constraint Programming: An Hybrid Approach'. In: *Principles and Practice of Constraint Programming – CP 2000*. Ed. by R. Dechter. Vol. 1894. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 187–201.

Focacci, F., A. Lodi and M. Milano (2002). 'Optimization-Oriented Global Constraints'. In: *Constraints* 7.3-4, pp. 351–365.

Focacci, F., A. Lodi and M. Milano (2004). 'Exploiting Relaxations in CP'. In: *Constraint and Integer Programming*. Ed. by M. Milano. Vol. 27. Operations Research/Computer Science Interfaces Series. Springer US, pp. 137–167.

Fontaine, D., L. Michel and P. Van Hentenryck (2014). 'Constraint-based Lagrangian Relaxation'. In: *Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by B. O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer International Publishing, pp. 324–339.

Francis, K. G. and P. J. Stuckey (2014). 'Explaining circuit propagation'. In: *Constraints* 19.1, pp. 1–29.

Franco, J. and J. Martin (2009). 'A History of Satisfiability'. In: ed. by A. Biere et al. Handbook of Satisfiability. IOS Press. Chap. 1, pp. 3–74.

Freling, R., D. Huisman and A. P. Wagelmans (2001). 'Applying an Integrated Approach to Vehicle and Crew Scheduling in Practice'. In: *Computer-Aided Scheduling of Public Transport*.

Ed. by S. Voß and J. R. Daduna. Vol. 505. Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, pp. 73–90.

Freling, R., D. Huisman and A. P. Wagelmans (2003). 'Models and Algorithms for Integration of Vehicle and Crew Scheduling'. In: *Journal of Scheduling* 6.1, pp. 63–85.

Freling, R., A. P. Wagelmans and J. M. P. Paixão (1999). 'An Overview of Models and Techniques for Integrating Vehicle and Crew Scheduling'. In: *Computer-Aided Transit Scheduling*. Ed. by N. H. Wilson. Vol. 471. Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, pp. 441–460.

Fukasawa, R. et al. (2006). 'Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem.' In: *Mathematical Programming* 106.3, pp. 491–511.

Gilmore, P. C. and R. E. Gomory (1961). 'A Linear Programming Approach to the Cutting-Stock Problem'. In: *Operations Research* 9.6, pp. 849–859.

Gilmore, P. C. and R. E. Gomory (1963). 'A Linear Programming Approach to the Cutting Stock Problem—Part II'. In: *Operations Research* 11.6, pp. 863–888.

Golden, B. L., A. A. Assad and E. A. Wasil (2001). 'Routing vehicles in the real world: applications in the solid waste, beverage, food, dairy, and newspaper industries'. In: ed. by P. Toth and D. Vigo. The Vehicle Routing Problem. Society for Industrial and Applied Mathematics. Chap. 10, pp. 245–286.

Gomory, R. E. (1958). 'Outline of an Algorithm for Integer Solutions to Linear Programs'. In: *Bulletin Of the American Mathematical Society* 64, pp. 275–278.

Gomory, R. E. (1960). *An algorithm for the mixed integer problem*. Tech. rep. Rand Corporation.

Gomory, R. E. (1963). 'An algorithm for integer solutions to linear programs'. In: *Recent Advances in Mathematical Programming* 64, pp. 260–302.

Gondzio, J. (2012). 'Interior point methods 25 years later'. In: *European Journal of Operational Research* 218.3, pp. 587–601.

Gualandi, S. and F. Malucelli (2013). 'Constraint Programming-based Column Generation'. In: *Annals of Operations Research* 204.1, pp. 11–32.

Haase, K., G. Desaulniers and J. Desrosiers (2001). 'Simultaneous Vehicle and Crew Scheduling in Urban Mass Transit Systems'. In: *Transportation Science* 35.3, pp. 286–303.

Harvey, W. D. and M. L. Ginsberg (1995). 'Limited discrepancy search'. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. IJCAI'95, pp. 607–615.

Hempsch, C. and S. Irnich (2008). 'Vehicle Routing Problems with Inter-tour Resource Constraints'. In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by B. Golden, S. Raghavan and E. Wasil. Vol. 43. Operations Research/Computer Science Interfaces. Springer US, pp. 421–444.

Hollis, B., M. Forbes and B. Douglas (2006). 'Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post'. In: *European Journal of Operational Research* 173.1, pp. 133–150.

Hooker, J. N. (1994). 'Logic-based methods for optimization'. In: *Principles and Practice of Constraint Programming: Second International Workshop, PPCP '94 Rosario, Orcas Island, WA,*

USA, May 2–4, 1994 Proceedings. Ed. by A. Borning. Vol. 874. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 336–349.

Hooker, J. N. (2006). 'Operations Research Methods in Constraint Programming'. In: *Handbook of Constraint Programming*. Ed. by P. v. B. Francesca Rossi and T. Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier. Chap. 15, pp. 527–570.

Hooker, J. N. (2007). *Integrated Methods for Optimization*. Springer.

Hooker, J. N. (2010). 'Formulating Good MILP Models'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Hooker, J. N. and W.-J. van Hoeve (2018). 'Constraint programming and operations research'. In: *Constraints* 23.2, pp. 172–195.

Ibaraki, T. (1973). 'Algorithms for Obtaining Shortest Paths Visiting Specified Nodes'. In: *SIAM Review* 15.2, pp. 309–317.

IBM (2015). *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual (Version 12, Release 6)*. URL: `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/usrcplex.pdf` (visited on 12/04/2018).

Irnich, S. (2008). 'A Unified Modeling and Solution Framework for Vehicle Routing and Local Search-Based Metaheuristics'. In: *INFORMS Journal on Computing* 20.2, pp. 270–287.

Irnich, S. and G. Desaulniers (2005). 'Shortest Path Problems with Resource Constraints'. In: ed. by G. Desaulniers, J. Desrosiers and M. M. Solomon. Column generation. Springer. Chap. 2.

Irnich, S., P. Toth and D. Vigo (2014). 'The Family of Vehicle Routing Problems'. In: ed. by D. Vigo and P. Toth. Second Edition. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics. Chap. 1, pp. 1–33.

Jaffar, J. and J.-L. Lassez (1987). 'Constraint Logic Programming'. In: *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*. ACM, pp. 111–119.

Jaffar, J., S. Michaylov et al. (1992). 'The CLP(R) Language and System'. In: *ACM Transactions on Programming Languages and Systems* 14.3, pp. 339–395.

Jepsen, M. et al. (2008). 'Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows'. In: *Operations Research* 56.2, pp. 497–511.

Jünger, M. et al. (2009). *50 years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media.

Junker, U. et al. (1999). 'A Framework for Constraint Programming Based Column Generation'. In: *Principles and Practice of Constraint Programming – CP'99: 5th International Conference, CP'99, Alexandria, VA, USA, October 11-14, 1999. Proceedings*. Ed. by J. Jaffar. Springer, pp. 261–274.

Jussien, N. and O. Lhomme (2002). 'Local search with constraint propagation and conflict-based heuristics'. In: *Artificial Intelligence* 139.1, pp. 21–45.

Kallehauge, B., N. Boland and O. B. G. Madsen (2007). 'Path inequalities for the vehicle routing problem with time windows'. In: *Networks* 49.4, pp. 273–293.

Karmarkar, N. (1984). 'A new polynomial-time algorithm for linear programming'. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. ACM, pp. 302–311.

Khachiyan, L. G. (1979). 'A polynomial algorithm in linear programming'. In: vol. 20. Soviet Mathematics Doklady, pp. 191–194.

Kianfar, K. (2010). 'Branch-and-Bound Algorithms'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Kilby, P., P. Prosser and P. Shaw (2000). 'A Comparison of Traditional and Constraint-based Heuristic Methods on Vehicle Routing Problems with Side Constraints'. In: *Constraints* 5.4, pp. 389–414.

Kim, B.-I., J. Koo and J. Park (2010). 'The combined manpower-vehicle routing problem for multi-staged services'. In: *Expert Systems with Applications* 37.12, pp. 8424–8431.

Koné, O. et al. (2011). 'Event-based MILP models for resource-constrained project scheduling problems'. In: *Computers & Operations Research* 38.1. Project Management and Scheduling, pp. 3–13.

Kuhn, H. W. (1955). 'The Hungarian method for the assignment problem'. In: *Naval Research Logistics (NRL)* 2.1-2, pp. 83–97.

Lahyani, R., M. Khemakhem and F. Semet (2015). 'Rich vehicle routing problems: From a taxonomy to a definition'. In: *European Journal of Operational Research* 241.1, pp. 1–14.

Land, A. H. and A. G. Doig (1960). 'An automatic method of solving discrete programming problems'. In: *Econometrica: Journal of the Econometric Society*, pp. 497–520.

Laporte, G., H. Mercure and Y. Norbert (1984). 'Optimal tour planning with specified nodes'. In: *RAIRO-Operations Research* 18.3, pp. 203–210.

Lodi, A. (2010). 'Mixed Integer Programming Computation'. In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. by M. Jünger et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 619–645.

Lübbecke, M. E. (2010). 'Column Generation'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Lübbecke, M. E. and J. Desrosiers (2005). 'Selected topics in column generation'. In: *Operations Research* 53.6, pp. 1007–1023.

Lysgaard, J., A. N. Letchford and R. W. Eglese (2004). 'A new branch-and-cut algorithm for the capacitated vehicle routing problem'. In: *Mathematical Programming* 100.2, pp. 423–445.

Mak, V. (2001). 'On the Asymmetric Travelling Salesman Problem with Replenishment Arcs'. PhD thesis. University of Melbourne.

Marques Silva, J. P. and K. A. Sakallah (1996). 'GRASP–A New Search Algorithm for Satisfiability'. In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*. ICCAD '96. San Jose, California, USA: IEEE Computer Society, pp. 220–227.

Mercier, A., J.-F. Cordeau and F. Soumis (2005). 'A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem'. In: *Computers & Operations Research* 32.6, pp. 1451–1476.

Mercier, A. and F. Soumis (2007). 'An integrated aircraft routing, crew scheduling and flight retiming model'. In: *Computers & Operations Research* 34.8, pp. 2251–2265.

Mesquita, M. and A. Paias (2008). 'Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem'. In: *Computers & Operations Research* 35.5, pp. 1562–1575.

Michel, L. and P. Van Hentenryck (2010). 'Basic CP Theory: Search'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Milano, M. (2010). 'Constraint Programming Links with Math Programming'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Miller, C. E., A. W. Tucker and R. A. Zemlin (1960). 'Integer Programming Formulation of Traveling Salesman Problems'. In: *Journal of the ACM* 7.4, pp. 326–329.

Mitchell, J. E. (2010). 'Branch and Cut'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Moskewicz, M. W. et al. (2001). 'Chaff: Engineering an efficient SAT solver'. In: *Proceedings of the 38th annual Design Automation Conference*. ACM, pp. 530–535.

Murty, K. G. (2010). 'History of LP Development'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Naddef, D. and G. Rinaldi (2002). 'Branch-and-Cut Algorithms for the Capacitated VRP'. In: *The Vehicle Routing Problem*. Ed. by P. Toth and D. Vigo. The Vehicle Routing Problem. Society for Industrial and Applied Mathematics. Chap. 3, pp. 53–84.

Nemani, A. K. and R. K. Ahuja (2011). 'Shortest Path Problem Algorithms'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Nemhauser, G. L. and L. A. Wolsey (1999). *Integer and Combinatorial Optimization*. Wiley-Interscience.

Ohrimenko, O., P. J. Stuckey and M. Codish (2009). 'Propagation via lazy clause generation'. In: *Constraints* 14.3, pp. 357–391.

Padberg, M. and G. Rinaldi (1991). 'A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems'. In: *SIAM Review* 33.1, pp. 60–100.

Pecin, D. et al. (2014). 'Improved Branch-Cut-and-Price for Capacitated Vehicle Routing'. In: *Integer Programming and Combinatorial Optimization: 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*. Ed. by J. Lee and J. Vygen. Springer International Publishing, pp. 393–403.

Peng, J. and M. Salahi (2011). 'Interior Point Methods for Nonlinear Programs'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Pesant, G. et al. (1998). 'An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows'. In: *Transportation Science* 32.1, pp. 12–29.

Poggi, M. and E. Uchoa (2014). 'New Exact Algorithms for the Capacitated Vehicle Routing Problem'. In: ed. by D. Vigo and P. Toth. Second Edition. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics. Chap. 3, pp. 59–86.

Prabhu, N. (2010). 'The Simplex Method and Its Complexity'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Puchinger, J. et al. (2011). 'Dantzig-Wolfe decomposition and branch-and-price solving in G12'. In: *Constraints* 16.1, pp. 77–99.

Rader, D. J. (2010). *Deterministic operations research: models and methods in linear optimization*. John Wiley & Sons.

Røpke, S. (2012). *The Solomon instances are solved!* Presentation at the International Workshop on Column Generation 2012.

Røpke, S. and J.-F. Cordeau (2009). 'Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows'. In: *Transportation Science* 43.3, pp. 267–286.

Røpke, S., J.-F. Cordeau and G. Laporte (2007). 'Models and branch-and-cut algorithms for pickup and delivery problems with time windows'. In: *Networks* 49.4, pp. 258–272.

Rossi, F., P. Van Beek and T. Walsh (2006). *Handbook of constraint programming*. Elsevier.

Rousseau, L.-M., M. Gendreau and G. Pesant (2002). 'Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows'. In: *Journal of Heuristics* 8.1, pp. 43–58.

Rousseau, L.-M., M. Gendreau, G. Pesant and F. Focacci (2004). 'Solving VRPTWs with Constraint Programming Based Column Generation'. In: *Annals of Operations Research* 130.1, pp. 199–216.

Schutt, A. et al. (2009). 'Why Cumulative Decomposition Is Not as Bad as It Sounds'. In: *Principles and Practice of Constraint Programming - CP 2009*. Ed. by I. P. Gent. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 746–761.

Schutt, A. et al. (2010). 'Explaining the cumulative propagator'. In: *Constraints* 16.3, pp. 250–282.

Schutt, A. et al. (2013). 'Solving RCPSP/max by lazy clause generation'. In: *Journal of Scheduling* 16.3, pp. 273–289.

Sellmann, M. (2004). 'Theoretical Foundations of CP-Based Lagrangian Relaxation'. In: *Principles and Practice of Constraint Programming – CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004. Proceedings*. Ed. by M. Wallace. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 634–647.

Semet, F., P. Toth and D. Vigo (2014). 'Classical Exact Algorithms for the Capacitated Vehicle Routing Problem'. In: ed. by D. Vigo and P. Toth. Second Edition. Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics. Chap. 2, pp. 37–57.

Solomon, M. M. (1987). 'Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints'. In: *Operations Research* 35.2, pp. 254–265.

Stuckey, P. J. et al. (2014). 'The MiniZinc challenge 2008–2013'. In: *AI Magazine* 35.2, pp. 55–60.

Terlaky, T. (2010). 'Introduction to Polynomial Time Algorithms for LP'. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Thorsteinsson, E. (2001). 'Branch-and-Check: A Hybrid Framework Integrating Mixed Integer Programming and Constraint Logic Programming'. In: *Principles and Practice of Constraint Programming – CP 2001*. Ed. by T. Walsh. Vol. 2239. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 16–30.

Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. MIT Press Cambridge.

Van Hentenryck, P. and T. Graf (1992). 'Standard forms for rational linear arithmetic in constraint logic programming'. In: *Annals of Mathematics and Artificial Intelligence* 5.2, pp. 303–319.

Van Hentenryck, P. and L. Michel (2013). 'The Objective-CP Optimization System'. In: *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*. Ed. by C. Schulte. Vol. 8124. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 8–29.

Van Hentenryck, P. and M. Milano (2011). *Hybrid Optimization: The Ten Years of CPAIOR*. Vol. 45. Springer Optimization and Its Applications. Springer.

Vanderbei, R. J. (2014). *Linear Programming: Foundations and Extensions*. 4th. Vol. 196. International Series in Operations Research & Management Science. Springer.

Vigo, D. and P. Toth (2014). *Vehicle Routing: Problems, Methods, and Applications*. Second Edition. Society for Industrial and Applied Mathematics.

Volgenant, T. and R. Jonker (1987). 'On Some Generalizations of the Travelling-Salesman Problem'. In: *The Journal of the Operational Research Society* 38.11, pp. 1073–1079.

Winston, W. L. (2004). *Operations Research: Applications and Algorithms*. 4th. Duxbury Press.

Wright, S. J. (1997). *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics.

Zhang, L. et al. (2001). 'Efficient Conflict Driven Learning in a Boolean Satisfiability Solver'. In: *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*. ICCAD '01. San Jose, California: IEEE Press, pp. 279–285.

Appendix A

Supplementary Results for Branch-and-Check with Explanations

Tables A.1 and A.2 on the next two pages contain additional experimental results for shorter runs (1 minute and 5 minutes respectively) of the Branch-and-Check with Explanations algorithm seen in Table 5.3. These results show that there is no significant change to the conclusions made in Chapter 5 for shorter time-outs.

	Branch-and-Check – Most-Fractional						Branch-and-Check – Activity-based								
	No Strengthening			With Strengthening			No Strengthening			With Strengthening			Branch-and-Cut		
Instance	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time
R201	1055.5	1211.6	–	1114.3	1160.5	–	1053.4	1181.5	–	1114.3	**1149.9**	–	1132.7	1155.6	–
R202	763.6	1254.3	–	825.2	1302.4	–	764.6	1133.4	–	812.3	**1112.8**	–	888.6	4980.0	–
R203	659.3	1273.7	–	683.6	1416.4	–	659.2	**1025.2**	–	688.8	1058.3	–	748.1	4980.0	–
R204	624.7	1236.7	–	633.7	1193.3	–	625.3	**868.2**	–	633.5	949.8	–	661.9	4980.0	–
R205	796.7	1229.1	–	859.1	1207.1	–	794.2	**1091.3**	–	852.6	1096.2	–	900.0	4980.0	–
R206	684.9	1237.4	–	737.3	1338.7	–	684.4	**1040.1**	–	726.3	1068.0	–	783.6	4980.0	–
R207	646.9	1193.8	–	665.6	1260.6	–	648.4	**940.7**	–	669.1	957.1	–	714.8	4980.0	–
R208	623.2	1113.8	–	628.6	1246.5	–	623.4	855.0	–	629.3	**852.4**	–	651.8	4980.0	–
R209	685.1	1302.9	–	726.5	1313.3	–	685.8	**1049.4**	–	718.9	1073.8	–	785.8	4980.0	–
R210	680.1	1318.0	–	729.2	1246.7	–	679.0	1108.9	–	728.8	**1036.4**	–	798.3	4980.0	–
R211	621.2	1335.5	–	630.6	1529.8	–	621.6	**1004.2**	–	630.2	1068.8	–	645.1	4980.0	–
C201	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	11.5
C202	549.8	764.6	–	583.8	642.2	–	548.7	629.9	–	589.1	**589.1**	10.0	589.1	**589.1**	202.9
C203	524.4	1122.8	–	560.6	722.6	–	524.8	704.9	–	553.2	**610.2**	–	586.0	632.3	–
C204	514.2	1012.1	–	534.2	1112.4	–	515.6	884.5	–	533.4	661.2	–	584.4	**597.1**	–
C205	543.8	783.1	–	586.4	**586.4**	0.2	546.0	613.1	–	586.4	**586.4**	15.8	586.4	**586.4**	334.4
C206	539.9	851.7	–	586.0	**586.0**	12.2	539.2	702.6	–	586.0	**586.0**	10.5	586.0	**586.0**	419.0
C207	539.2	867.8	–	585.8	**585.8**	19.7	538.3	637.1	–	585.8	**585.8**	8.4	585.8	**585.8**	527.5
C208	535.6	867.0	–	585.8	**585.8**	54.2	532.5	665.8	–	585.8	**585.8**	10.4	585.8	**585.8**	569.7
RC201	1080.2	1458.2	–	1242.5	1262.9	–	1080.4	1338.3	–	1261.8	**1261.8**	45.5	1250.1	1288.2	–
RC202	702.5	1508.1	–	885.2	1550.0	–	699.2	1204.2	–	862.0	**1152.3**	–	940.1	6609.4	–
RC203	615.7	1473.8	–	695.3	1444.6	–	611.3	**1149.0**	–	691.1	1151.5	–	781.6	6609.4	–
RC204	582.6	1420.8	–	645.2	1645.7	–	582.2	1007.1	–	628.9	**937.0**	–	692.7	6609.4	–
RC205	823.0	1621.3	–	1035.9	1366.6	–	819.0	**1249.8**	–	996.3	1300.3	–	1081.7	6609.4	–
RC206	784.1	1551.5	–	926.6	1392.7	–	785.4	1270.2	–	909.7	**1258.1**	–	974.8	6609.4	–
RC207	644.7	1522.3	–	758.8	1686.0	–	642.8	1193.5	–	762.6	**1172.1**	–	832.4	6609.4	–
RC208	572.7	1779.6	–	621.7	1799.7	–	574.0	**1065.8**	–	613.5	1080.3	–	647.7	6609.4	–

Table A.1: Experimental results for 1-minute runs of Branch-and-Check Explanations. The table reports the lower bound, upper bound and time to prove optimality for each of the solvers. The best upper bound for each instance is shown in bold.

	Branch-and-Check – Most-Fractional						Branch-and-Check – Activity-based								
	No Strengthening			With Strengthening			No Strengthening			With Strengthening			Branch-and-Cut		
Instance	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time
R201	1055.4	1198.0	–	1117.6	**1143.3**	–	1055.9	1177.6	–	1115.4	1149.9	–	1132.7	1155.6	–
R202	763.3	1224.0	–	845.1	1251.1	–	764.0	1133.4	–	847.3	**1109.3**	–	888.6	4980.0	–
R203	659.4	1244.8	–	706.5	1283.2	–	658.9	**1025.2**	–	703.6	1052.2	–	748.1	4980.0	–
R204	625.3	1166.7	–	637.7	1193.3	–	625.6	**858.4**	–	637.0	899.1	–	661.9	4980.0	–
R205	797.2	1222.7	–	878.0	1161.5	–	794.3	1091.3	–	874.2	**1076.9**	–	900.0	4980.0	–
R206	687.1	1214.9	–	749.6	1157.4	–	684.2	1040.1	–	741.8	**1031.3**	–	783.6	4980.0	–
R207	647.2	1193.8	–	678.8	1178.0	–	647.5	**940.7**	–	682.8	951.0	–	714.8	4980.0	–
R208	622.6	1097.4	–	633.1	1214.5	–	623.4	855.0	–	633.9	**832.5**	–	651.8	4980.0	–
R209	687.5	1249.3	–	749.8	1172.5	–	685.4	**1046.6**	–	745.9	1073.8	–	785.8	4980.0	–
R210	682.1	1225.6	–	745.4	1240.3	–	679.6	1105.6	–	742.9	**1024.9**	–	798.3	4980.0	–
R211	621.2	1335.5	–	632.5	1355.9	–	620.7	**1004.2**	–	632.2	1065.1	–	645.1	4980.0	–
C201	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	11.5
C202	550.0	687.9	–	589.1	**589.1**	124.2	548.2	629.9	–	589.1	**589.1**	11.6	589.1	**589.1**	202.9
C203	526.4	1007.3	–	563.2	680.0	–	524.7	686.5	–	562.6	**605.3**	–	586.0	632.3	–
C204	514.3	946.7	–	553.2	1106.7	–	514.7	884.5	–	552.2	660.9	–	584.4	**597.1**	–
C205	545.3	724.1	–	586.4	**586.4**	0.2	546.3	613.1	–	586.4	**586.4**	14.7	586.4	**586.4**	334.4
C206	538.8	806.0	–	586.0	**586.0**	12.2	538.2	702.6	–	586.0	**586.0**	9.8	586.0	**586.0**	419.0
C207	538.2	851.1	–	585.8	**585.8**	18.7	538.3	635.2	–	585.8	**585.8**	7.1	585.8	**585.8**	527.5
C208	534.1	867.0	–	585.8	**585.8**	52.1	532.5	652.4	–	585.8	**585.8**	12.5	585.8	**585.8**	569.7
RC201	1083.7	1421.1	–	1245.2	**1261.8**	–	1081.0	1338.3	–	1261.8	**1261.8**	56.9	1250.1	1288.2	–
RC202	702.8	1480.0	–	902.0	1452.5	–	699.2	1204.2	–	905.7	**1152.3**	–	940.1	6609.4	–
RC203	616.1	1408.2	–	742.9	1359.6	–	611.3	1149.0	–	740.6	**1125.7**	–	781.6	6609.4	–
RC204	583.4	1419.9	–	656.0	1352.4	–	582.2	1007.1	–	652.1	**926.6**	–	692.7	6609.4	–
RC205	823.1	1511.6	–	1071.8	1321.2	–	818.8	1249.8	–	1054.3	**1245.3**	–	1081.7	6609.4	–
RC206	789.4	1485.4	–	959.9	1309.6	–	784.9	1270.2	–	947.1	**1204.5**	–	974.8	6609.4	–
RC207	646.5	1496.7	–	786.7	1606.1	–	642.8	1193.5	–	797.4	**1172.1**	–	832.4	6609.4	–
RC208	576.0	1737.3	–	624.0	1776.1	–	573.9	**1039.5**	–	624.2	1078.4	–	647.7	6609.4	–

Table A.2: Experimental results for 5-minutes runs of Branch-and-Check Explanations.